RWTH Aachen

Department of Computer Science
Technical Report

Controlling Development Processes

Thomas Heer

ISSN 0935–3232 · Aachener Informatik-Berichte · AIB-2012-02

RWTH Aachen · Department of Computer Science · January 2012

Controlling Development Processes

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der RWTH Aachen University zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

Diplom-Informatiker
Thomas Heer

aus Vechta

Berichter: Universitätsprofessor Dr.-Ing. Dr.h.c. Manfred Nagl
Universitätsprofessor Dr.rer.pol. Matthias Jarke

Tag der mündlichen Prüfung: 19. Juli 2011

Die Druckfassung dieser Dissertation
ist unter der ISBN 978-3-8440-0509-7 erschienen.

Thomas Heer
Lehrstuhl Informatik 3
`heer@se.rwth-aachen.de`

Aachener Informatik Bericht AIB-2012-02

Herausgeber: Fachgruppe Informatik
RWTH Aachen University
Ahornstr. 55
52074 Aachen
GERMANY

ISSN 0935-3232

Abstract

The development of an innovative product is a complex and highly dynamic process which has to be performed in a controlled way. Several dependencies exist between the defined tasks, the assigned resources, and the artifacts to be produced. In a development project, which is planned and executed according to a process definition, the time, budget, and available resources are limited. Controlling a development process involves monitoring the actual performance and analyzing whether it conforms to the plan. Poor performance, changing requirements, the detection of errors, and the creation or modification of key artifacts may require plan changes at process runtime. As a consequence of the inherent complexity of the task, software tool support is essential for controlling development processes.

Different insufficient solutions are nowadays applied in practice for this purpose. Project management systems support project planning and to some degree project controlling, but they do not support the execution of predefined processes. Workflow management systems on the other hand are commonly applied for process execution. However, they do not support the scheduling of tasks in a project, and they are not flexible enough for the management of development processes. As a consequence, both types of systems are insufficient when it comes to controlling development processes. Attempts for their integration fell short with respect to representing execution states in project plans and scheduling workflow instances.

This thesis describes a new concept for a process management system, which combines the strengths of the aforementioned tools and eliminates their deficiencies by substantial extensions. Starting point of the research were results of the collaborative research center (SFB) 476 IMPROVE. An integrated approach for the management of development processes has been extended with respect to task scheduling, progress measurement, and change management in development projects. In particular, an algorithm for the automatic generation of a project schedule has been developed which takes the execution states of the tasks into account. Subprocesses of a development process can be executed by a workflow engine, which interprets predefined workflow definitions. With respect to monitoring, specific progress measures for the degree of completion of tasks have been defined which rely on elements of the process model. In the case of plan changes at process runtime, the consistency of the plan with the execution state of the process is ensured.

The concepts have been implemented in the extension module PROCEED of the commercial life cycle asset information management system Comos of Siemens Industry Software. Comos is widely used in the plant engineering industries. Therefore, this thesis combines fundamental research results with a proof of concept implementation in an industrial context. The realization of PROCEED based on an industrial platform offers great opportunities for further evaluation of the provided functionalities in plant design projects in the plant engineering industries.

Danksagung

Viele Menschen haben zum Gelingen dieser Arbeit und zum erfolgreichen Abschluss meiner Promotion beigetragen. Ihnen möchte ich an dieser Stelle herzlich danken.

Mein besonderer Dank gilt Prof. Dr.-Ing Dr.h.c. Manfred Nagl für die Möglichkeit an seinem Lehrstuhl zu promovieren, das spannende und nach wie vor aktuelle Thema, die stets konstruktive Kritik in Vorträgen und Diskussionen, und sein vorbildliches Arbeitsethos an dem man sich orientieren konnte. Auch über die rein fachlichen Dinge hinaus habe ich in den vergangenen Jahren viel von ihm gelernt.

Prof. Dr. Matthias Jarke danke ich für die Erstellung des Zweitgutachtens. Prof. Dr. Berthold Vöcking und Prof. Dr. Thomas Seidl danke ich für ihre Bereitschaft als Prüfer zu fungieren. Allen Mitgliedern der Promotionskommission danke ich für die angenehme Prüfungsatmosphäre. Dem neuen Lehrstuhlinhaber Prof. Dr. Bernhard Rumpe gilt mein Dank für seine Unterstützung in der Endphase meiner Promotion.

Diese Arbeit ist im Rahmen eines DFG Transferprojektes entstanden und wäre ohne die Förderung der Deutschen Forschungsgemeinschaft nicht möglich gewesen. Den Mitarbeitern des Industriepartners im Transferprojekt Siemens Industry Software, vormals innotec, danke ich für die fruchtbare Zusammenarbeit. Herrn Weller danke ich für die Bereitschaft, mit dem Lehrstuhl zum Thema Prozessmanagment zu kooperieren, Herrn Kokkelink danke ich für die vielen konstruktiven Gespräche in Bonn, Alexander Wojciekowski danke ich für seine tatkräftige Unterstützung in den ersten Monaten des Projektes, Frau Nestler und Herrn Schimmang danke ich für die gute Weiterführung der Zusammenarbeit.

Auf Seiten des Lehrstuhls waren mehrere Mitarbeiter, Diplomanden und Hilfswissenschaftler am Projekt beteiligt. Mein besonderer Dank gilt Galina Volkova und Jochen Hormes für ihre Beiträge zur Entwicklung des Softwareprototyps. Die Diplomanden und Masterstudenten Christoph Briem, Sheryl Leong, Christoph Außem, Emily Pu, Michael Dreher und Ventsislava Vasileva haben im Rahmen ihrer Abschlussarbeiten maßgeblich zu den in dieser Arbeit vorgestellten Konzepten und zur Entwicklung des Softwareprototyps beigetragen. Zeitweilig unterstützten uns Christoph Briem, Florian Meyer und Ghislain Manib Mbogos als Hilfswissenschaftler. Allen Teammitgliedern bin ich für ihre Unterstützung sehr dankbar.

In den Jahren während meiner Promotion durfte ich mit großartigen Kollegen zusammenarbeiten und hatte mit ihnen viel Spaß am Lehrstuhl und darüber hinaus. Daraus sind auch einige feste Freundschaften entstanden. Bodo Kraft, mein Diplomarbeitsbetreuer, hat mir gezeigt wie man ein Promotionsprojekt effektiv managed. Markus Heller war mir ein Vorbild als gewissenhafter Forscher. Er hat wesentliche Vorarbeiten zu dieser Arbeit geliefert. Mit René Wörzberger konnte ich im Rahmen des Transferprojekts sehr gut zusammenarbeiten. Auf den Konferenzen, Arbeitskreisen, und Kundengesprächen, die wir zusammen besuchten, gab es dank ihm viele heitere Momente. Theresa Körtgen danke ich für ihre moralische Unterstützung und ihre wertvollen Verbesserungsvorschläge zu dieser Arbeit sowie zu meinem Promotionsvortrag. Sie war mir immer ein Vorbild in Bezug auf Disziplin und Zielstrebigkeit. Daniel Retkowitz habe ich immer bewundert für seine stoische Gelassenheit. Von

ihm habe ich gelernt, die Promotion richtig einzuordnen und gelassen zu bleiben. Ibrahim Armac hat mir gezeigt, wie man stilvoll promoviert und dabei immer eine gute Figur macht. Mein lieber Bürokollege der letzten Tage Cem Mengi war immer hilfsbereit und herzensgut, und war mir damit eine moralische Stütze. Allen alten Kollegen möchte ich danken für die schöne Zeit in den ersten Jahren. Neben den zuvor genannten sind dies Simon Becker, Thomas Haase, Christian Fuß, Christoph Mosler, Ulrike Ranger, Erhard Weinell, Marita Breuer, Ulrich Norbisrath und Boris Böhlen. Mit Simon verbinden mich auch einige nette Bierabende in Aachens Kneipen. Bernhard Westfechtel hatte zwar den Lehrstuhl zu meiner Zeit schon verlassen, mir jedoch bei einem Besuch noch eine gute Motivationsspritze verpasst. Den neuen Kollegen, die mit Prof. Rumpe an den Lehrstuhl kamen, danke ich für die schöne Zeit in den letzten Jahren und die vielfältige Unterstützung. Zu nennen sind hier Ingo Weisemöller, Jan Oliver Ringert, Thomas Kurpick, Claas Pinkernell, Arne Haber, Christoph Herrmann, Antonio Navarro-Perez, Markus Look, Minh Tran, Rim Jnidi, Tim Gülke, Roland Hildebrandt, Martin Schindler, Christian Berger, Steven Völkel, Hans Grönniger, Holger Rendel und Holger Krahn. Rim Jnidi war mir für einige Zeit eine liebe Bürokollegin. Besonderer Dank gilt unseren Sekretärinnen Angelika Fleck, Silke Cormann und Sylvia Gunder.

Ich möchte auch allen meinen Freunden und Bekannten in Aachen und darüber hinaus danken, die mich moralisch immer unterstützt haben. Einen besonderen Motivationsschub verdanke ich meinem Cousin Patrick gegen Ende der Promotion. Schließlich danke ich meiner Familie für ihre Unterstützung in all den Jahren, meinem Bruder Christoph, meinen Schwestern Radegunde und Walburga, und besonders meinen Eltern Resi und Josef. Ohne euch hätte ich das nicht geschafft.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Introduction

Processes are inherent in the work of people in virtually every organization. A *process* can be defined as the logical organization of people, materials, energy, equipment, and procedures into work activities designed to produce a specified end result [Pal87]. *Process models* can be applied to capture all of these aspects. A process model defines the activities to be executed, their sequence, their responsible roles, the artifacts which are required for the activities and those which shall be produced [Lon93]. Additional properties regarding the expected durations of activities, their required effort, their priorities, etc. can be incorporated as well. The explicit definition of processes has several advantages [Kra98]. A process model allows the *communication* about the process. Process models can be *reused* for the definition of new processes. An explicitly defined process can be analyzed and *improved* subsequently. Finally, the explicit definition of a process enables *process controlling* during its performance. Process controlling subsumes all activities which are performed to ensure that the results of a process are delivered on time and meet the required quality standards. This includes monitoring the process and taking corrective measures if necessary.

A specific type of processes are *development processes*. They define how different human actors have to cooperate to develop a new product. Development processes have several specific characteristics [NW94]. First, they define multiple interrelationships between tasks, human resources, and products, which have to be managed at process runtime. Products subsume all documents and other artifacts produced in the process. Second, there is a high degree of uncertainty, and dynamic changes to the defined tasks, resources, products, and their mutual relationships occur frequently at process runtime. Finally, the tasks which have to be executed in a development process and their relationships cannot be completely defined in advance before the start of the process. The creation of key artifacts is often the precondition for the definition of further tasks.

The term *design process* is often used synonymously for development process [NM08]. However, one can also regard a design process as a special case or subprocess of a development process. The result of the former is the design of a new product, e.g. a car or a chemical plant, while the latter also includes the actual construction of the product. For example, the result of a plant design project is the design of a chemical plant while the actual construction has to take place subsequently or simultaneously. In contrast, the result of a software development process includes the software itself and not merely its design.

Process management includes all activities regarding the modeling, performance, controlling, and improvement of processes. It is a means to improve the performance of individual process participants, teams, and whole organizations. The specific characteristics of development processes and their complexity demand for software tool support for process management. A *process management system* can support the process responsible by keeping track of all interdependencies between tasks, resources, and products, thereby maintaining a consistent state of the management data. The human resources who execute the defined tasks can be guided in carrying out their work. The compliance of the actual process performance to the defined process model can be enforced. Finally, a process management system can support the partial automation of development processes.

1.1 Motivation

A process management system which addresses the specific characteristics of development processes was realized in the project *Adaptable and Human-Centered Environment for the Administration of Development Processes* (AHEAD) [HJK$^+$08] at the Department of Computer Science 3 of RWTH Aachen University. The aim of the AHEAD project was to provide new concepts and tool functionalities for the management of development processes. In particular, the aspects of process modeling, enactment, and improvement, as well as interorganizational collaboration were addressed. The AHEAD prototype was developed to evaluate the applicability of the developed concepts for the management of development processes.

Besides a formal definition of the process to be performed, a process management system needs an internal representation of a *process instance*, i.e. the current state of a performed process at runtime. In [Sch02], the following terminology has been introduced, which will also be used in this thesis.

Process Model Definition. A process model definition defines the process independent of any specific instance.

Process Model Instance. A process model instance represents a process in its current state of performance.

In both cases the term *model* is used to emphasize that the definition of the process as well as the process instance are both represented by according models in a process management system. The term *process model enactment* is used for the mechanical interpretation of a process model by a process management system and is thereby distinguished from process performance [Lon93]. *Performance* of a process means the execution of the defined tasks by process participants according to the process model and thereby refers to the actions of the process participants in the real world. In contrast, enactment refers to the state changes of the corresponding process model instance in a process management system.

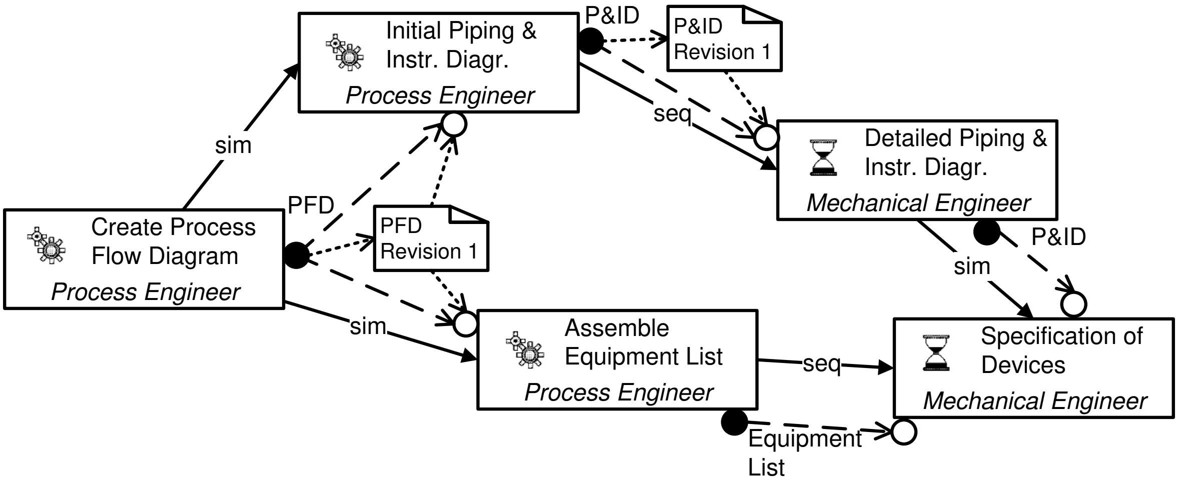


Abbildung 1.1: Example for a dynamic task net.

For the representation of process model instances in AHEAD, the concept of a *dynamic task net* was introduced [Kra98]. A dynamic task net integrates the current state of planning with the enactment state of the corresponding development process. In Figure 1.1, an example of a dynamic task net is depicted showing the currently defined tasks to be executed, their required roles, their execution states, and the actual data flow, i.e. the document revisions which have been produced within the scope of the tasks. The available entities, properties, and relationships for modeling dynamic task nets in AHEAD are defined in the DYNAMITE meta-model [Kra98], which also defines *structural and behavioral invariants* constraining the allowed change operations to ensure the consistency of the management data.

In the majority of cases, development processes are executed in the form of *development projects*. According to the ISO 9000 standard, a *project* is a *unique process*, consisting of a set of coordinated and controlled activities with *start and finish dates*, undertaken to achieve an objective conforming to specific requirements including constraints of *time, cost and resources* [ISO05]. The connection between the concepts of a process and a project has been illustrated in [Dow91]. A process model definition cannot make any assertion about fixed date constraints in a calendar or about actual human resources in a project team. These aspects can only be determined in a concrete project where one has to deal with time, cost and resource constraints. On the other hand, a process model instance of a development process should incorporate all aspects of a project plan.

Nowadays, complex development projects are usually carried out according to a process model definition which defines the project phases, milestones, tasks, functional roles, and the artifacts to be produced. The project plan of such a development project has to comply to restrictions defined in the process model definition concerning the order of tasks, the assignment of qualified human resources to tasks, and the produced artifacts [HJK$^+$08, GDMR04]. However, a project plan stored in a conventional project management system does not capture the current enactment state. For this purpose, dynamic task nets integrate the information contained in project plans with information about the current performance state of a development process, namely the execution states of tasks and the actual data flow.

Controlling the performance of a development process refers to controlling a development project. *Project controlling* subsumes all activities for *measuring* the actual performance of the defined tasks, *comparing* the actual performance to the plan, *identifying deviations* from the plan, and taking *corrective measures* to get back on track if necessary [DIN09]. If corrective measures are unsuccessful or the project goals have changed, *plan changes* may be required at project runtime. The project controlling activities can be divided into *monitoring* and *steering*. Monitoring subsumes the activities from measuring the actual performance to identifying deviations while corrective measures and plan changes are means to control or steer a project.

The capabilities of the AHEAD system for project controlling are limited. Monitoring the performance of a development process is only possible based on task execution states and released document revisions. No degree of completion is computed to determine the exact progress of tasks during execution. Tasks in a dynamic task net are not scheduled for particular dates in the AHEAD system. As a consequence, it cannot be decided whether the resource assignments are feasible. Without a baseline schedule, it is not possible to assess the performance of a development process with respect to deadlines and budget limits. The AHEAD approach includes a conceptual framework for process management on several modeling layers which allows for continuous process improvement [Sch02]. However, process improvement with respect to timing data like average task durations is generally only possible when development processes are monitored at runtime. Consequently, it was required to extend the AHEAD approach with respect to project scheduling and controlling capabilities.

A prerequisite for project controlling is the existence of a *baseline schedule* which serves as a reference for assessing the actual performance. For example, the tasks depicted in Figure 1.1 have an estimated duration and resource requirements in terms of working hours. They have to be scheduled for particular dates, so that resources can be allocated according to the requirements. An executable schedule has to be time- and resource-feasible, i.e. it has to comply to all defined task dependencies, and the resource usage may not exceed the defined limits for any date. The complexity of development projects demands for software tool support for the automatic generation of such time- and resource-feasible schedules. Research on *project scheduling* has been conducted for several decades with considerable success [DH02]. However, there are still several open research questions with respect to scheduling [Smi03]. One of these problems is *managing change*. A project is never executed exactly as planned, so that plan changes occur frequently at project runtime. Dynamic plan changes require a partial or full rescheduling of the defined tasks. Most established approaches for project scheduling assume that the scheduling environment is predictable and that the generated schedule can be executed as planned, while practical applications tend to be highly unpredictable, so that scheduling is an ongoing process of responding to unexpected and evolving circumstances. Algorithms which are able to reschedule the tasks in a project fast upon dynamic plan changes are still rare.

Based on a baseline schedule, *project monitoring* can be performed. The first step in project monitoring is to measure the actual performance of the defined tasks. For this purpose, the *degree of completion* of a task has to be determined. The degree of completion reflects to what extent the goal of the task has been reached. In this way, quantitative statements can be made about the progress of a running task. For example in Figure 1.1, it could be determined that the task Create Process Flow Diagram is 60% complete after creating the first revision of the process flow diagram. For the computation of the degree of completion, different *progress measures* can be applied, which quantify the actual progress of running tasks. A progress measure has to be sufficiently *accurate* in order to base management decisions on the computed values. At the same time, the *measuring effort* should be as low as possible in order to avoid a disproportionate measurement overhead which negatively affects the performance of the development process. Accuracy and measuring effort are mutually dependent, and a good trade-off has to be found by selecting appropriate progress measures. Furthermore, it is important to be able to determine the current project status in a timely manner, i.e. it should be possible to derive an accurate degree of completion for a task at any time. In large and complex projects, the *timeliness* of progress measurement can only be achieved with software tool support. But even with software tool support, the actuality of the computed values is degraded if progress measures rely on user inputs which are not provided on a regular basis. In project monitoring theory and practice, several different measures for the degree of completion of a task have been proposed and are used [PR05]. These measures differ with respect to their accuracy, the required measurement effort, and the actuality of the measured values. For different tasks in a project, different progress measures are appropriate. However, no integrated measurement framework exists which combines all common measures and which can be tailored to a specific development process.

Besides monitoring the actual performance, controlling also involves the application of *corrective measures* and *plan changes*. The integration of planning and scheduling has been identified in [Smi03] as another open problem in the scheduling research field. When tasks, dependencies, or resources are created, modified, or deleted at project runtime, then the schedule has to be updated. For example in Figure 1.1, the duration of the task Initial Piping and Instrumentation Diagram may be increased or the deadline for the task Assemble Equipment List may be antedated. Individual plan changes may lead to an inconsistent state of the management data with respect to timing properties and may even render scheduling impossible. A process management system has to ensure that dynamic changes eventually enable successful rescheduling. However, it is not possible to ensure the consistency of the timing data at any time during replanning because several operations may be required to obtain a new consistent state.

When it comes to planning and replanning, several additional issues have to be considered. If project planning is performed by means of a software tool which is used by several users with different permissions, then the authorization for change operations has to be decided for every user individually. When a user is authorized to perform plan changes, he may require additional information to decide on the best option. For example, he may want to inspect the currently planned usage of all resources with a certain functional role to decide which resource to assign to a task. Finally, the history of plan changes should be traceable for two reasons. First, the effect of unforeseen disruptions at project runtime can be analyzed. Second, bad estimates of the required time and effort for tasks specified in a process model definition, which have been used for project planning, can be identified and corrected after project completion.

1.2 Research Context

The research presented in this thesis has been conducted in the context of the transfer project T6 *Dynamic Process Management based upon Existing Systems* at the Department of Computer Science 3 of RWTH Aachen University [HNWH08]. The project was part of the Transfer Center 61 succeeding the Collaborative Research Center (CRC) 476 IMPROVE [NM08], which were both funded by the DFG (Deutsche Forschungsgemeinschaft). The application domain of the CRC and the Transfer Center was the domain of plant engineering. The goal of the interdisciplinary research was to provide better software tool support for plant design processes. Software prototypes should demonstrate the applicability of the developed concepts and functionalities for the management of dynamic development processes.

Besides the advancement of the AHEAD approach, the transfer of the research results to industrial practice was the second goal of the transfer project T6. The AHEAD system was a purely academic prototype. It was generated from a formal specification of a graph rewriting system [SWZ99]. In contrast, the new concepts and algorithms presented in this thesis have been implemented as an extension module of the commercial life cycle asset information management system *Comos*, a product of *Siemens Industry Software* formerly known as *innotec* [Sie10]. Siemens Industry Software was the industrial partner in the transfer project T6. The Comos system was extended by new functionality for the management of dynamic development processes. The cooperation with the industry partner Siemens Industry Software also gave rise to additional requirements. In particular, workflow support for individual subprocesses in a plant design project and an adequate visualization of the project management data for project status reports were demanded by the industry partner.

Two other research projects were conducted in the Transfer Center 61 which were closely related to the transfer project T6. In the project T2 *Computer-Assisted Work Process Modeling* [THM08], the modeling of work processes based on ontologies was investigated. The project T3 *Simulation-Supported Workflow Optimization in Process Engineering* [KSSL08] was concerned with the simulation of development processes in order to optimize the usage of resources and tools. The software prototypes developed in the projects T2, T3, and T6 were coupled in a joint cooperation, so that process models of the T2 project could be simulated by the simulation tool of the T3 project, and both, the models and the simulation results could be imported into Comos in the form of workflow templates. The resulting integrated tool framework has been presented in [HTT09, Pu09] but will not be treated in this thesis.

In the CRC 476 IMPROVE, another research project was conducted which was concerned with process management. In the project B1 *Experience-Based Development Processes*, direct process support was integrated in the software-tools which are used by engineers in a plant design project. The results of this project will be reviewed as related work in Chapter 6 of this thesis.

1.3 Solution Approach

In this thesis an innovative approach for controlling development processes by means of a process management system is presented. The developed solution exceeds state-of-the-art project and workflow management systems with respect to the management of development processes. Project management systems do not allow to connect the defined tasks in a project plan with technical products. Furthermore, they do not allow to enact predefined processes. On the other hand, workflow management systems do not support planning and scheduling. The deficiencies of the two types of systems are overcome by integrating the two paradigms and thereby providing integrated support for project planning and process enactment.

The AHEAD approach for process management has been advanced in this thesis to cover the aspects of *project planning, scheduling and controlling*. Planning refers to the initial definition of a dynamic task net including estimates for the required effort and duration of the defined tasks. In the process management literature, the terms *task* and *activity* are sometimes used synonymously. In this thesis, only the term task is used to refer to the portions of work defined in a process model. After planning, the defined tasks are scheduled. Controlling subsumes monitoring the actual performance as well as changing a dynamic task net at project runtime. A task net can be replanned and subsequently rescheduled at project runtime.

Established approaches for project planning, scheduling, and controlling from theory and practice have been analyzed, adapted, extended and integrated with the AHEAD approach for process management. The developed algorithms and tool functionalities have been implemented in the *Process Management Environment for Engineering Design Processes* (PROCEED) which is an extension module of the commercial life cycle asset information management system Comos. An exemplary plant design project has been defined and the enactment of the process model instance has been simulated in PROCEED to verify the correctness of the implemented algorithms and to evaluate the applicability of the provided tool functionality.

The solution approach can generally be divided into two parts. First, the DYNAMITE meta-model for modeling dynamic task nets has been extended by new entities, properties, relationships, and constraints to enable the modeling of timing and progress measurement aspects, resulting in the new TNT meta-model for *Timed Dynamic Task Nets*. Second, algorithms and tool functionalities have been developed

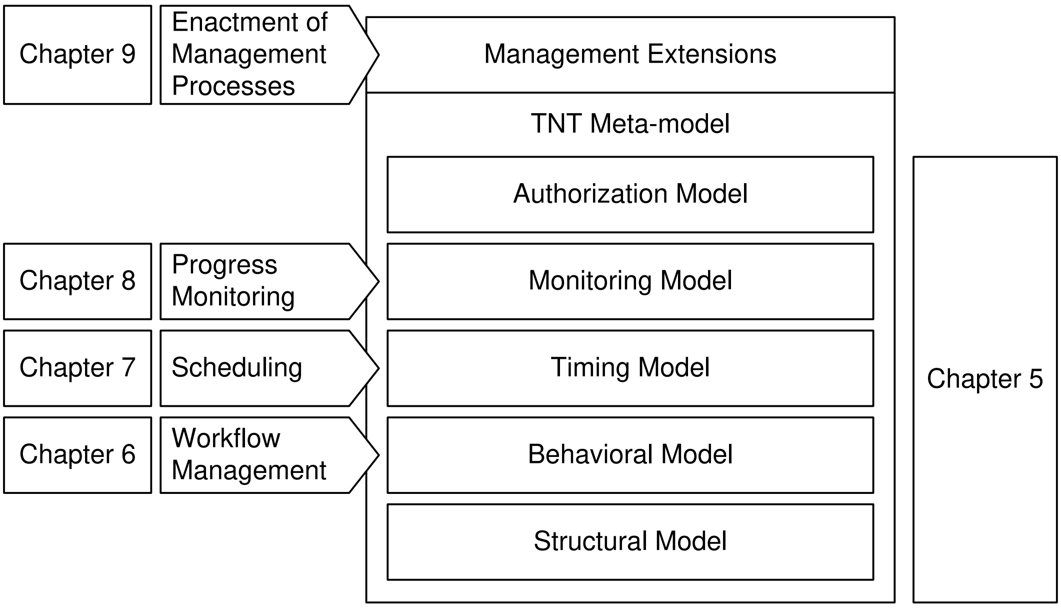


Abbildung 1.2: Overview over solution approach.

which build on the extended modeling capabilities and support the scheduling and controlling of process model instances. Figure 1.2 shows the structure of the TNT meta-model together with the corresponding algorithms and tool functionalities. Furthermore, it shows which aspects of the solution approach are treated in which chapter of this thesis.

Workflow support for technical subprocesses has been a specific requirement in the industrial context of the transfer project T6. *Workflow management systems* (WfMS) [JB96, Jab95, Wor95] are a popular type of process management systems. WfMS put an emphasis on the automation of processes. Previous work in the AHEAD project [Hel08a] regarding the integration of workflow instances into dynamic task nets could be continued [HBW09]. A workflow engine has been integrated in PROCEED to partially automate the enactment of dynamic task nets. If a task in a process is refined by several subtasks which together define another process, then the latter process is called a subprocess of the former. Workflow definitions serve as process model definitions for subprocesses of a development process. This tool functionality is related to the behavioral model of the TNT meta-model.

The timing model of the TNT meta-model defines *entities, properties, and relationships* to enable the scheduling of tasks in a dynamic task net. Tasks can be planned with respect to required workload and budget, expected duration, constraint dates, and planned dates. *Timing consistency constraints* have been defined which ensure the consistency of a dynamic task net with respect to all timing properties. The classical *critical path analysis* [KW59] has been adapted for hierarchically structured dynamic task nets to compute the earliest and latest possible start and end times of the defined tasks. A *heuristic algorithm* for *resource-constrained scheduling* of dynamic task nets has been developed based on a general parallel scheduling scheme.

The heuristic yields a good time- and resource-feasible schedule which is flexible in the sense that dynamic changes to the task net at runtime can be incorporated into the schedule. The heuristic can be applied for initial schedule generation as well as for dynamic rescheduling at project runtime.

The monitoring model of the TNT meta-model defines additional entities and properties which capture the actual performance of the enacted process. Several established progress measures from practice have been combined in an *integrated measuring framework*. These measures are complemented by new progress measures which build on the modeling capabilities of dynamic task nets [HW11]. In particular, the connection of tasks and products is exploited to determine the degree of completion of tasks. Furthermore, the information contained in process model definitions is used to determine the progress of subprocesses which are automatically enacted by the workflow engine. Earned value analysis [Anb03] is applied to derive *performance indices* for running tasks which allow a comparison of the actual process performance with the plan. The expected end times and costs of tasks can be *forecasted* and *deviations* from the plan can be detected early.

In the AHEAD system, different environments are designated to be used by the process manager and the process performers, respectively. In PROCEED, all process participants use the same environment. This required the development of an *authorization model* as part of the TNT meta-model to determine which user of the system is allowed to view and modify which parts of a dynamic task net. The authorization of a user depends on his personal permissions, his assigned tasks, and his position in the project team.

The five partial models amount to the core of the TNT meta-model. They enable the modeling, enactment, and controlling of development processes and regulate the access to the management data. However, a process model instance of a development process does not capture the management activities which are performed to monitor and control the development process. These activities include reporting, quality management, change management, replanning, and rescheduling. The solution approach for the controlling of development processes presented in this thesis also provides tool support for these activities. First, a general *change management procedure* has been defined which regulates the process of replanning and rescheduling. This procedure ensures that inconsistencies introduced during replanning are eventually resolved during rescheduling. Second, management processes can be enacted in the same way as technical processes in PROCEED. The workflow approach has been applied for the modeling and enactment of management processes. Specific workflow definitions can be defined by an organization, which define how change management cases, quality management procedures, and reporting have to be performed in a project of the organization. To enable the modeling of connections between management processes in a project and the corresponding development process, management extensions have been defined for the TNT meta-model.

All developed concepts and algorithms presented in this thesis have been implemented in the Process Management Environment for Engineering Design Processes to verify their correctness and applicability. PROCEED provides the main functionalities of the AHEAD system for the modeling and enactment of dynamic task nets and complements them by functionality for project scheduling and controlling. Several views are provided to view and modify the management data which is stored in the Comos database, including a task list view and a task net view. For the visualization of a scheduled dynamic task net, a Gantt chart is preferable to a network diagram, because the temporal extent of tasks is directly visible. Therefore, PROCEED has been *coupled* with the widely used *project management system MS Project* [Mic10a] in order to use its various diagram types for the presentation of the management data. PROCEED additionally comprises a dedicated user interface for project status control which also provides decision support for replanning. The development of this user interface has been a requirement of the industrial partner in the research project and has been motivated by the specific characteristics of plant design processes. A *multidimensional visualization approach* has been developed to provide a condensed overview over the tasks in a project, their workload and their current status [HBW09]. The current project status can be analyzed from different perspectives. The processing of the management data is performed in a *project data warehouse* to which the management data is exported in regular intervals.

The developed concepts and algorithms and their implementation in PROCEED have been evaluated by means of an example scenario. An example plant design project has been modeled and scheduled. The enactment of the design process has been simulated. The monitoring functionalities provided by PROCEED have been used to determine the expected effects of task delays and other disruptions. Predefined management workflows have been enacted for controlled change management. Replanning and rescheduling of the task net have been performed according to the general change management procedure.

1.4 Contributions

This thesis makes several contributions which can be viewed from three different perspectives. First, research results have been transferred to the commercial software-tool Comos which is widely used in the plant engineering industries. Second, the AHEAD approach for the management of dynamic development processes has been extended by project management functionality, which is required to enact process model instances in the form of development projects. Third, new concepts and algorithms enabling software tool support for project scheduling and controlling have been developed which represent significant contributions in these fields of research.

Process management in Comos The commercial life cycle asset information system Comos has been extended by new functionalities for the explicit management of engineering design processes.

Task management The Comos system has been extended to support the management of tasks in addition to the engineering data and the resources, i.e. the users of the system. Tasks can now be explicitly defined and stored as objects in the Comos database. They can be connected with the required engineering data and the assigned resources. This enables the integrated management of tasks, resources and products as it is supported by the AHEAD system.

Workflow management A workflow engine and client applications for the definition and monitoring of workflows have been integrated into Comos as part of PROCEED. The workflow management functionality enables the enactment of partially automated processes to guide engineers in performing their work. The underlying workflow technology offers interfaces for the integration of Comos with external applications.

Project management Based on the explicit representation of tasks, the Comos system now allows to plan and control plant design projects not only regarding the engineering data but also regarding time, resources and cost.

Project management with dynamic task nets The concepts for process management underlying the AHEAD prototype have been advanced. Several enhancements and extensions have been made regarding the meta-model for dynamic task nets and the algorithms for planning and controlling process model instances.

Scheduling The tasks in a dynamic task net can be scheduled automatically based on task assignments and workload estimates. Thereby, the semantics of the specific control flow types available in dynamic task nets are taken into account as well as the hierarchical structure of dynamic task nets. Task nets can also be rescheduled at runtime in case of dynamic changes. The generated schedule is robust in the sense that modifications to a dynamic task net at runtime do not necessarily affect the whole schedule and can often be handled locally.

Monitoring The degree of completion of tasks in a dynamic task net can be measured by means of several different progress measures. Specific progress measures have been developed for dynamic task nets which use the actual data flow and the knowledge contained in process model definitions. Earned value analysis is applied to detect and quantify deviations of the actual performance from the plan.

Integration with workflow management An approach for integrating workflow instances into dynamic task nets which was developed in the AHEAD project has been advanced, so that arbitrary subprocesses in a development project can now be automatically enacted by the workflow engine which has been developed as part of PROCEED.

Access control A new authorization model has been implemented in PROCEED which enables various ways of collaboration in contrast to the strict distinction between the roles of manager and engineer in the AHEAD system. All process participants use the same user interface while their access to the management data is restricted depending on their permissions, assigned tasks, and respective positions in the project team.

Controlling of development processes Scheduling and controlling of development projects are still open fields of research. This thesis makes several contributions to these research topics.

Timing consistency constraints for executable process model The formally defined TNT meta-model for dynamic task nets incorporates timing consistency constraints. These constraints cover all dependencies between the structure of a process model instance, its enactment state, and the values of the timing properties. An executable process model can be timed, enacted, and dynamically changed at runtime whereby the consistency of the management data is ensured by the process management system.

Heuristic for resource-constrained scheduling A heuristic algorithm for the generalized resource-constrained project scheduling problem [DH02] has been developed, which has several distinguishing features. A general parallel scheduling scheme has been extended to support hierarchically structured task nets. Furthermore, the execution states of the defined tasks are taken into account during scheduling which enables the rescheduling of task nets at project runtime. Scheduling can be performed locally for only a part of a task net. Scheduling is performed directly based on the work calendars of the assigned resources. The durations of the defined tasks can be derived from workload estimates. If workload and duration buffers are defined for complex tasks, the resulting schedule is robust in the sense that the creation of additional subtasks at project runtime does not necessarily affect the whole project schedule.

Integrated approach for progress measurement Different progress measures have been combined in an integrated measuring framework. Common measures from theory and practice are complemented by two new progress measures which rely on the actual data flow modeled in a dynamic task net and the enactment state of workflow instances, respectively. For every task in a process model instance, the most appropriate progress measure can be selected in order to achieve the best trade-off between measurement accuracy and effort. Aggregation methods which are used to combine the progress degrees of several subtasks at a common parent task yield aggregated progress degrees which remain stable even in case of dynamic changes to the task net.

Data warehousing and visualization for project status analysis The combination of a data warehouse with visualization techniques enables visual project status analysis which complements the monitoring capabilities based on earned value analysis. Multidimensional visual analysis enables the assessment of the project status even in case of plant design projects with a vast number of engineering tasks and a high degree of simultaneous engineering. The analysis views are tightly integrated with the management views of PROCEED which enables direct intervention when delays or resource bottlenecks are detected.

Authorization model covering process changes The authorization model implemented in the PROCEED system covers structural changes to a dynamic task net at runtime. Users are authorized to make changes to a process model instance depending on their assigned tasks. This approach goes one step further than common access control mechanisms for process management systems which only regulate the assignment of resources to tasks and the execution of tasks in a process.

Explicit modeling of management processes Technical processes and management processes in development projects are distinguished in PROCEED, and their mutual dependencies are explicitly modeled in the system. While engineering tasks are part of the project plan, the enactment of management processes involves changes to the project plan. In related work, management processes are either not covered at all or handled independently of the development process.

1.5 Structure of the Thesis

This thesis is structured as follows. In Chapter 2, the application domain of this thesis is described. The general plant design process is introduced, the main functionalities of the Comos system are described, and the example scenario is presented, which is used throughout this thesis. In Chapter 3, some fundamental concepts and established approaches for project management, scheduling, and controlling, are explained, which are required for a better understanding of the main chapters. The previous research achievements of the AHEAD project are reviewed in Chapter 4.

Chapter 5 to Chapter 10 are the main chapters of this thesis. In Chapter 5, the TNT meta-model is presented which includes the adopted and adapted modeling elements and constraints of the DYNAMITE meta-model. Chapter 6 describes how process knowledge including timing data can be defined in PROCEED using task types and process templates, and how it can be reused for subsequent projects. Furthermore, the workflow management capabilities of PROCEED are described in Chapter 6. The algorithms for temporal analysis and resource-constrained scheduling of dynamic task nets are described in Chapter 7. In particular, the solution for the scheduling of workflow instances is presented. The approach to project monitoring implemented in PROCEED is described in Chapter 8, including the integrated framework for progress measurement and the multidimensional visualization of project management data. Chapter 9 deals with the controlled enactment of management processes. It is explained how domain and organization specific management workflows can be defined, parameterized, and enacted. Furthermore, the general change management procedure for replanning and rescheduling is described. Related work on the different subtopics of this thesis is reviewed in the respective chapters. In Chapter 10, the prototypical implementation of the PROCEED system is presented. Chapter 11 concludes the thesis.

Kapitel 2

Application Context

In the Collaborative Research Center 476 IMPROVE, the application domain of the AHEAD process management system was the domain of process plant engineering [NM08]. Also the Comos system, which has been extended by process management functionality in this thesis, is commonly used in the plant engineering industries. Therefore, this chapter introduces the general process model for plant design projects in Section 2.1. The purpose and main functionalities of the Comos system are described in Section 2.2. Finally in Section 2.3, a concrete example process is introduced which will be used as the reference scenario throughout this thesis.

2.1 The General Plant Design Process

This section introduces the engineering phases and main activities in a plant design project. Furthermore, the central documents and the different functional roles of project team members are described. The presentation is a synthesis of the descriptions given in [For94, PTW03, Mad00, Ull83, Lan00, Bön99, Mar06, DIN06, Hel08b, Wag03, Hir99, NM08]

A *process plant* is an industrial plant in which raw materials are processed and converted into products. Source materials and products may be gaseous, liquid or solid substances or mixtures of these states. Inside a process plant, a chemical or physical process takes place which converts intermediate products step by step by means of several unit operations. Different process plants exist for the various products in different business domains like chemical engineering, pharmaceutics, petroleum industry, food industry and construction industry. In a chemical plant, different substances react with each other to yield the final product. The required conditions regarding temperature, pressure, material flow, etc. are established by according devices.

Process design The *chemical process* which takes place inside a chemical plant has to be distinguished from the design process which yields the design of the chemical plant. During the design of a chemical process, decisions have to be made regarding the processing mode, the raw materials, products, and by-products, and the operations required to achieve the desired product. The processing mode can be either batch or continuous. Regarding the products, the flow rate is of primary interest which affects the flow rate of the raw materials. Furthermore, the composition, phase, form, temperatures, and pressures of all raw material and product streams are determined. The processing operations to convert the raw material to products are inserted into the process flow sheet. Several alternative configurations are generated which are compared in order to select the one that is best suited for the existing conditions. During the design of the chemical process, several different process designs are created, which can be classified depending on the accuracy and detail they provide [PTW03].

Quick estimating procedures include so-called *order-of-magnitude designs* and *study or factored designs* which are both not really process designs, but are merely used for preliminary cost estimation.

Preliminary designs are used as a basis for deciding whether further work should be done on a proposed process or if an alternative solution is to be preferred.

Detailed-estimate designs are used to determine the cost-and-profit potential of an established process by detailed analyzes and calculations; exact specifications for the equipment, however, are not given and piping and layout work is minimized.

Final process designs are developed as the final step before developing the construction plans for the plant; complete specifications are presented for all components of the plant, and accurate costs based on quoted prices are obtained.

Life cycle of a chemical plant In the following, the *design process* is described which yields the design of a chemical plant. It is divided into several subsequent phases. Parallel to all process phases, project management and controlling have to be performed in a plant design project. The design process is embedded in the overall *life cycle of a chemical plant* which also includes its construction, operation and maintenance. The life cycle of a process plant can be divided into several phases [Bön99, Mar06, DIN06]. Figure 2.1 shows the most common division into phases and the most important activities of these phases.

The first phase in a plant design project is the *preliminary planning*. A *feasibility study* shows, if the plant can be realized given the general conditions. The main task in the preliminary planning phase is the process design. A general process concept is developed, mass balances are calculated and the general process data of the main devices is specified. It is usually the case, that the basic chemical process is not designed from scratch at this point, but that an existing process is adapted for the specific process plant to be developed. Therefore, extensive literature research and the use of according databases is an important step during the preliminary design phase. The central document which is created in this phase is the *block diagram* of the chemical process which specifies the basic operations and the material flow between them. An example of a block diagram is given in Figure 2.2. In this example, the raw materials and intermediate products pass through several process steps like Reaction, Concentration and Distillation. The material streams are annotated

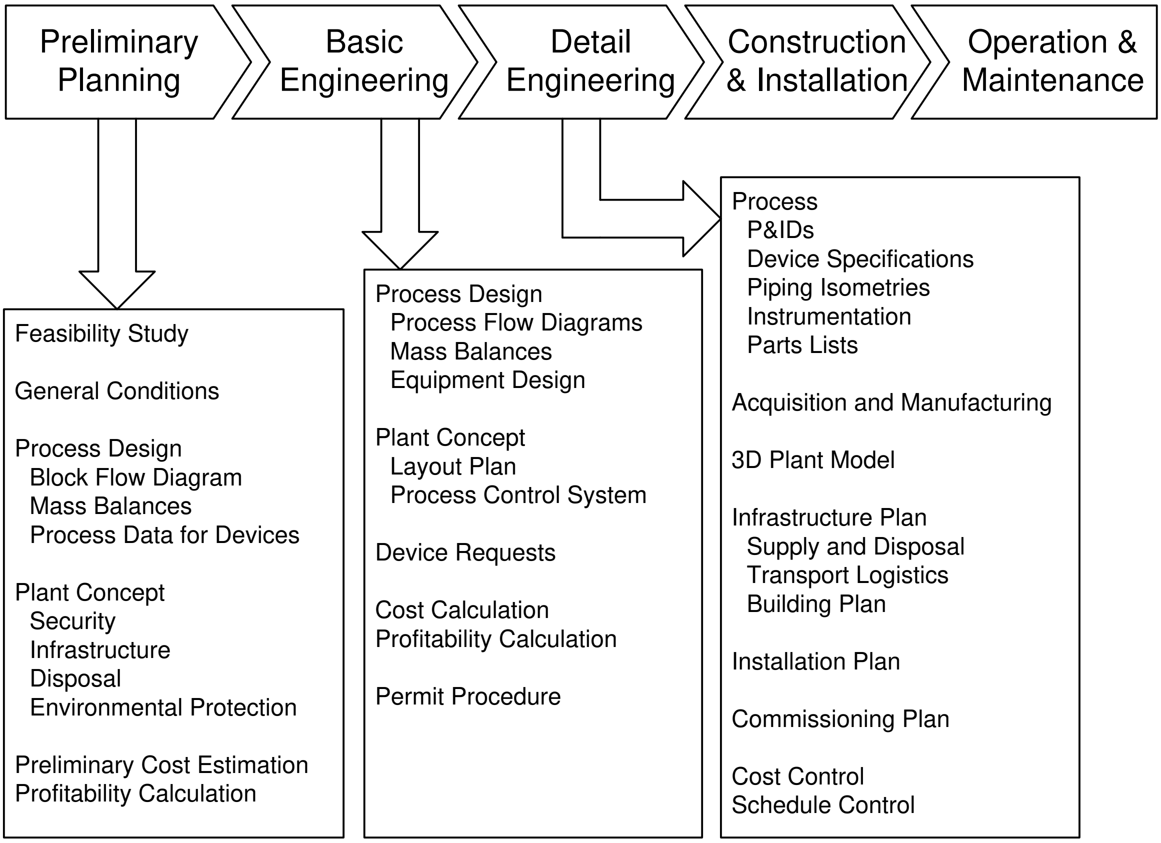


Abbildung 2.1: Phases and main activities of the general plant design process.

with the corresponding (intermediate) products or by-products. Besides the process design, a general plant concept is developed regarding security, infrastructure, disposal of waste material and issues of environmental protection. A preliminary cost estimation and a profitability calculation can be conducted at the end of the preliminary planning phase.

The purpose of the *basic engineering* phase is to elaborate the process design and the plant concept so far that the permit procedure can be carried out after which the government decides about the permission for the construction of the plant. The central documents in this phase are the *process flow diagrams* (PFDs) for the different plant parts. It is common that every operation of the block flow diagram is detailed as a separate PFD to handle the complexity of the overall design. Figure 2.3 shows an example for a PFD. The PFD shows the main devices in the chemical plant like pumps (`P1A`, `P2A`, `P3A`), heat exchangers (`W1`-`W5`), vessels (`B2`) and columns (`K1`). In the basic engineering phase, the mass balances are elaborated and the equipment designs are specified. Devices which take a long time to be delivered are already requested from the manufacturers. The plant concept is elaborated in the form of a layout plan, and the process control system is specified. Based on the technical documents produced in the basic engineering phase, a cost calculation

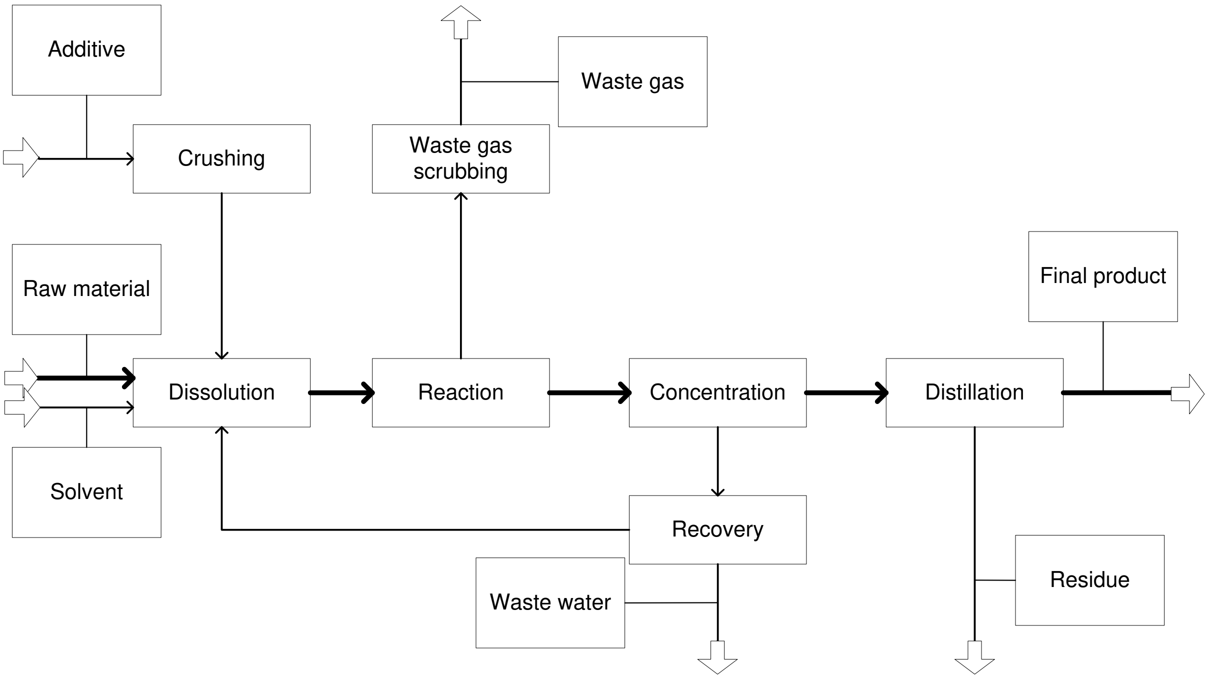


Abbildung 2.2: Block diagram with basic information [ISO01].

and a profitability calculation can be performed.

In the *detail engineering* phase, the flow diagrams, specifications, layout plans, etc. are elaborated in full detail to enable the manufacturing of devices and the construction of the process plant. Based on the PFDs, *piping and instrumentation diagrams* (P&IDs) are created which incorporate additional information about the pipes between devices and all the instrumentation devices like measuring points, valves, etc. An example P&ID is depicted in Figure 2.4. In the detail engineering phase, the devices are designed in detail, and either standard products are ordered from the manufacturers or special devices are fabricated for the process plant. The same holds for the instrumentation devices. The layout of the pipes and other connections between devices are specified in isometries, and a three-dimensional model of the plant is generated. The infrastructure of the plant is specified in detail as well. Plans are established for the installation and the commissioning of the plant. During the whole detail engineering phase, the costs have to be monitored which includes the costs for the design work as well as the expected costs for the construction and installation work and the material costs of the chemical plant. Finally, schedule control is important so that the design process is performed as planned and no delays are introduced which could not be compensated in the following phases.

The *construction and installation* of the process plant already starts during the detail engineering phase. Basic and detail engineering are executed in sequence because the governmental permission is a prerequisite for proceeding with the detailed planning. However, the foundations can be built before P&IDs are completed,

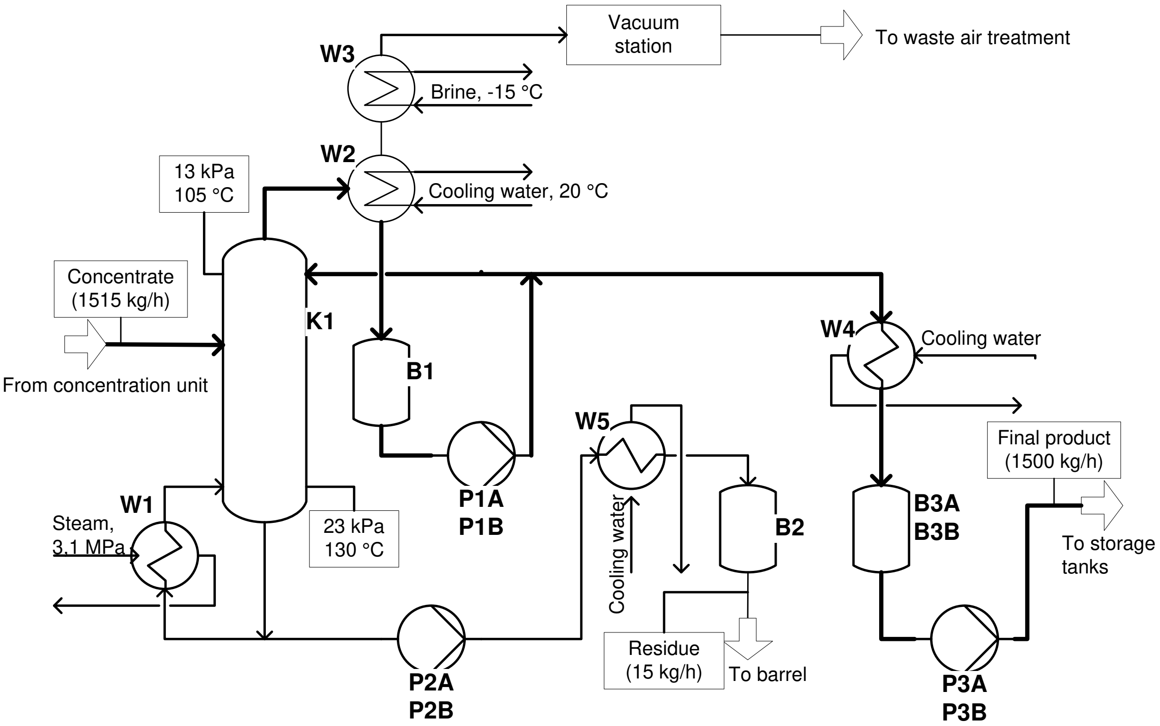


Abbildung 2.3: Process flow diagram with basic information [ISO01].

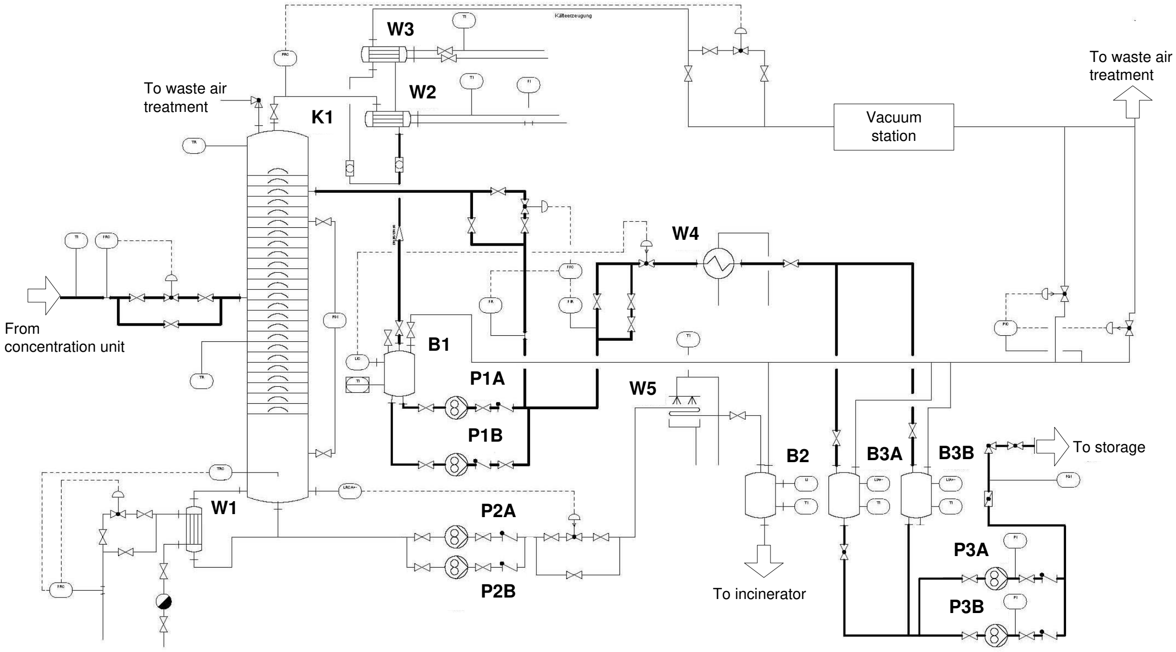


Abbildung 2.4: Piping and instrumentation diagram with basic information [ISO01].

and available devices can be installed before all devices have been delivered. Hence, the phases of detail engineering and construction and installation overlap in time. Besides the foundations, the superstructure, the installed devices, the pipes, and the instrumentation, the construction also includes roads and infrastructure facilities. At the end of the construction and installation phase, there are several test runs of the plant before it can be commissioned.

During *operation and maintenance*, the correct functioning of the plant has to be ensured which requires the monitoring of according measured values. If the plant operates in batch mode, operation includes starting and stopping the plant and cleaning it between two batches. Also in the continuous operation mode a plant may have to be shut down temporarily for maintenance activities. Since a shutdown time is always associated with an income loss, it should be as short as possible. This is the goal of *shutdown management*. For all maintenance activities, the flow diagrams, specifications, reports, etc., which have been created during the different design phases, are of great importance and should be available to the plant operator. Therefore, software tools for the management of the engineering data like Comos (cf. Section 2.2) are used throughout the whole life cycle of a process plant. The final step in the life cycle of a process plant is its *decommissioning*.

All phases of the life cycle of a process plant have to be managed. However, in this thesis the focus lies on the design process which includes preliminary planning, basic and detail engineering. It is common, that different companies are responsible for the design and the construction of a plant. These contractors have to cooperate, but also manage their internal processes independently. The customer, which is the future owner of the plant, takes over after the commissioning and becomes the plant operator. All required documents from the design phases should be handed over to the plant operator.

Functional roles In plant design projects, several different functional roles are required [Lan00]. On the organizational level, the general management defines targets and verifies their implementation. This includes human resource management, product management, quality management, technical management and commercial management. Part of the technical management are *controlling* and *engineering* which are important roles in plant design projects. The engineering is responsible for the creation of the technical drawings, layout and installation plans. *Process engineers* are involved in the process design and the creation of the basic flow sheets. For the design of devices, engineers from different fields collaborate including *mechanical* and *electrical engineers*. Building plans and layout plans are created by *architects* and *construction engineers*. For the design of the process control system, specific know how regarding process automation is required. These and many other functional roles are required to successfully carry out the design and construction of a chemical plant. Tasks in a plant design project are usually assigned based on the required qualifications specified in terms of functional roles in the process model definition.

Software support Software tools have been used to support many activities of the plant design process [PTW03]. The traditional areas where software is used in the process include the following.

Research Electronic publications and databases for literature research about the latest data, flow diagrams, equipment, and simulation models,

Process flow diagrams Generation, evaluation and selection of process flow diagrams,

Simulation Selection of operating conditions, unit operation and sizing, component simulation, dynamic/steady-state simulation, and overall simulation of the process, scenario investigation,

Economics Cost databases for chemicals, utilities, and equipment for process economics evaluations,

Optimization Optimization of equipment type and size, processing order, etc. by means of mathematical models,

Layout Generation of 2D/3D isometrics for piping and material transfer equipment as well as maintenance access.

Life cycle asset information systems like Comos cover several of these functionalities, inter alia the creation of process flow diagrams and layouts. The traditional areas for software support in plant design projects do not cover the management of the design process. However, software tools for process management are required in large design projects [NM08] as well.

2.2 The Life Cycle Asset Information System Comos

The PROCEED prototype has been realized as an extension module of the commercial life cycle asset information management system Comos [Sie10]. This section introduces the Comos system, its purpose and main functionalities.

The Comos system supports the design, construction and maintenance of industrial plants. Engineers in a plant design project use Comos to create and maintain the engineering data including flow diagrams, device specifications, and piping isometries [Sie10]. All engineering data is stored in the Comos database. Figure 2.5 shows screenshots of the Comos user interface. The main functionalities of the Comos system are the following.

- Creation and revisioning of flow diagrams and other technical drawings (cf. Figure 2.5 front),
- Design and specification of devices and instrumentation, e.g. vessels, heaters, pumps (cf. Figure 2.5 middle),
- Three-dimensional piping planning and visualization (cf. Figure 2.5 back),

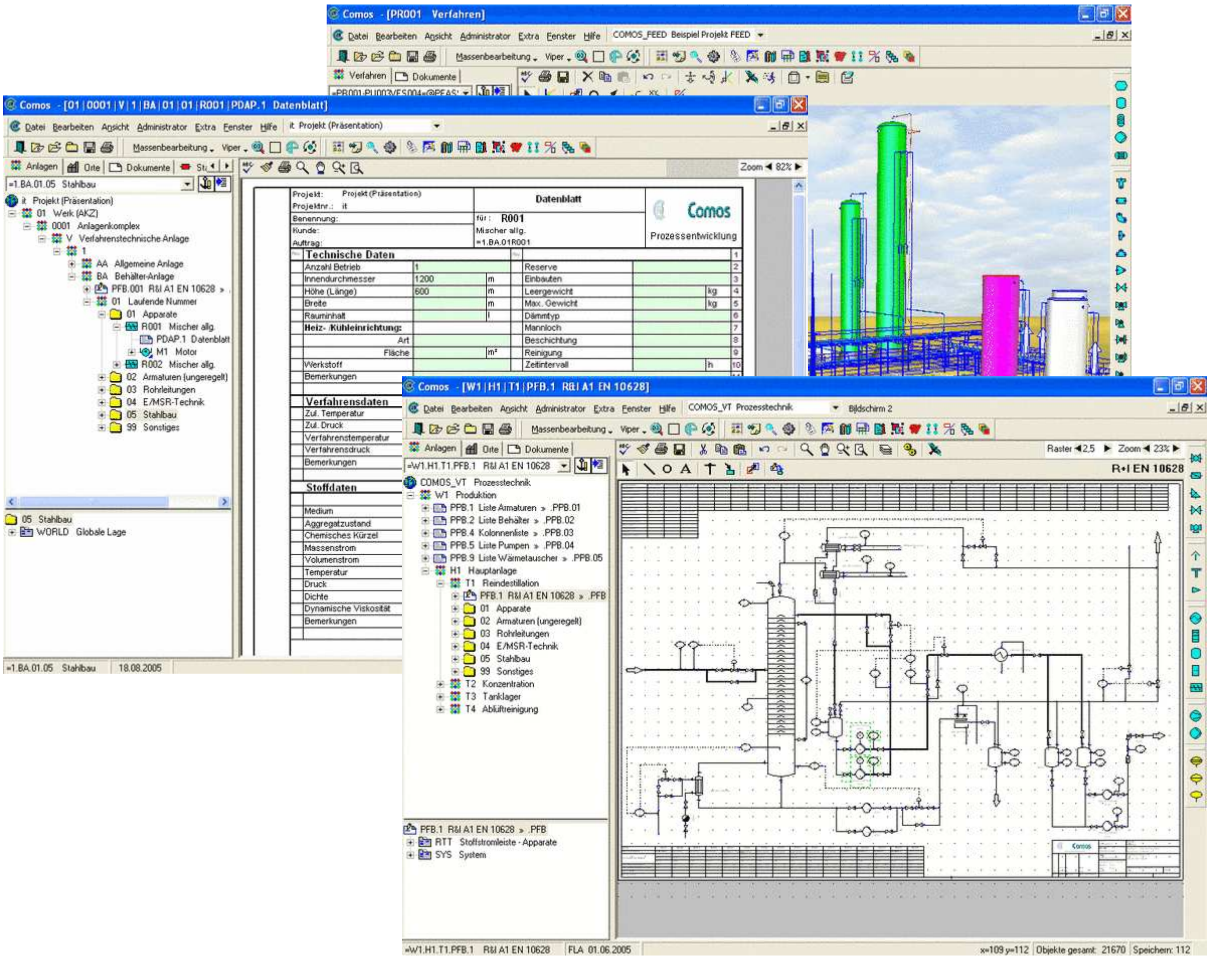


Abbildung 2.5: Screenshots of the Comos user interface.

- Automatic generation of parts lists depending on the planned devices and instrumentation,
- Management and revisioning of all required documents in a plant design process.

Comos stores all data in an object-oriented database. Documents created by means of external programs are stored in the file system and are referenced by according objects in the database. There are different types of objects in Comos.

Planning objects represent the actual engineering data in a project. Comos users work with planning objects.

Base objects are templates for planning objects and define their attributes.

Attributes can be specified for base objects. Attributes are also represented as objects in the database. Attributes can have textual, numerical or boolean values, or can link other objects in the database.

Documents can either be flow diagrams created with Comos or placeholder objects for external documents in the file system.

Comos provides basic functionality for collaborative work. All engineers in a project work on the same database. A user of the system can create a so-called *working layer* for the project data. All modifications done in a working layer do not affect the data in other working layers. The working layers are structured hierarchically. Changes made in one working layer can be released to the next higher working layer. Rudimentary support is provided for resolving conflicts between changes released from different working layers.

In Comos, the information about human resources in a company which belong to different departments is maintained in the so-called *users project*. In this project, multiple locations of a company can be defined which are subdivided into departments. Employees of the company are modeled as person objects contained in the department objects. The person objects can be linked to user accounts of the Comos system. Thereby, human resources and users of the Comos system are unified.

Until the development of the PROCEED prototype, Comos did not provide any functionality for explicit process management. The provided functionality for collaborative work can be regarded as implicit process support at best. Tasks could not be explicitly defined in Comos and no support was available for engineers to follow the defined procedures in a project. An integration of the product data contained in the Comos database with the project management data in a plant design project was not given. However, this integration is of great value for project monitoring and control. The available functionality for progress control in the released version of Comos was limited to status management of individual documents. For this reason, Comos has been extended by customers to allow for progress measurement of a whole plant design project based on the engineering data in the Comos database [LGB$^+$05]. In this thesis, Comos has been extended by explicit process management support, integrating product, task and resource management and allowing for task-based project status analysis.

2.3 Example Scenario

While Section 2.1 introduced the general plant design process, this section describes the concrete example scenario which will be used in this thesis. While the AHEAD system has been applied to support the early phases of the plant design process [NM08], the Comos system is applied to all process phases, in particular to the detail engineering phase. Therefore, the example scenario covers preliminary planning, basic engineering, and detail engineering in a plant design project. The scenario has been developed based on thorough literature research [For94, PTW03, Mad00, Ull83, Lan00, Bön99, Mar06, DIN06, Hel08b, Wag03, Hir99, NM08] and discussions with representatives from industry [Sie10, Tec10]. The scenario has been designed to include the typical tasks which have to be executed in a plant design project. Although the scenario is quite elaborate, it cannot be complete. A plant design project in practice comprises several hundreds of tasks. Therefore, several tasks, documents and dependencies have been omitted to reduce the complexity of the process model instance. However, the most important task types which occur in plant design projects are represented in the example. Several concepts used in this section and the notation for dynamic task nets will be introduced in the following chapters. The description in this section shall only provide a first glance on the example scenario to the reader. And it may serve as a reference for the examples which will be given in the following chapters.

The main part of the example is a dynamic task net which defines the phases, work packages and tasks in a two-year plant design project. The estimated workload, the durations, and the budget are defined for all tasks in the example task net. In this section, the defined tasks, the structure of the task net, and the planning data are described. Different planning and enactment states of the task net will be presented in the following chapters to explain the developed concepts and algorithms. The tasks in the example task net will be be scheduled. The progress of the defined tasks will be measured during the enactment of the process. Changes to the task net will be made, and the tasks will be rescheduled to incorporate these changes. Figure 2.6 gives an overview over the part of the hierarchically structured dynamic task net which covers the work breakdown structure of the example project. Tasks are represented as boxes and control flow dependencies as labeled arrows between these boxes. The labels indicate the semantics of the control flows which will be introduced in Section 5.2. For some control flows, lag times are defined. The subtasks of a complex task are positioned in a box below the task, which is connected by a double-headed arrow with the complex task.

The root node of the dynamic task net represents the whole design project. The typical project phases Preliminary Planning, Basic Engineering, and Detail Engineering are defined as subtasks of the root task. The subtasks of the project phases define the work packages in the project. In the preliminary planning phase, on the one hand, the chemical process is defined by performing a simulation and creating a block flow diagram (task BFD in Figure 2.6). On the other hand, a plant concept is determined which allows for a preliminary cost estimation. In the basic engineering phase, the cost estimation is concretized which is a prerequisite for the decision about the realization of the chemical plant and the continuation of the project. The task Realization Approval is the corresponding milestone in the project. In the task PFDs, the process flow diagrams are created, which are elaborated in the form of piping and instrumentation diagrams in the following task Initial P&IDs. The piping and instrumentation diagrams are elaborated in the detail engineering phase. Based on these P&IDs, the machines and devices, instruments, and pipes can be specified in detail in the corresponding tasks. A long-running task in the detail engineering phase is the procurement of the machines, devices and instruments. It starts as soon as the first devices have been specified and ends at the end of the detail engineering phase.

Figure 2.7 shows the subtasks of the work packages which are not part of the work breakdown structure of the project but are taken into account during project scheduling. An example for a workflow-managed task is included in the example: The task Specify Pump 037 is executed according to a predefined procedure. The tasks to determine the type of the pump have to be executed alternatively, i.e. only

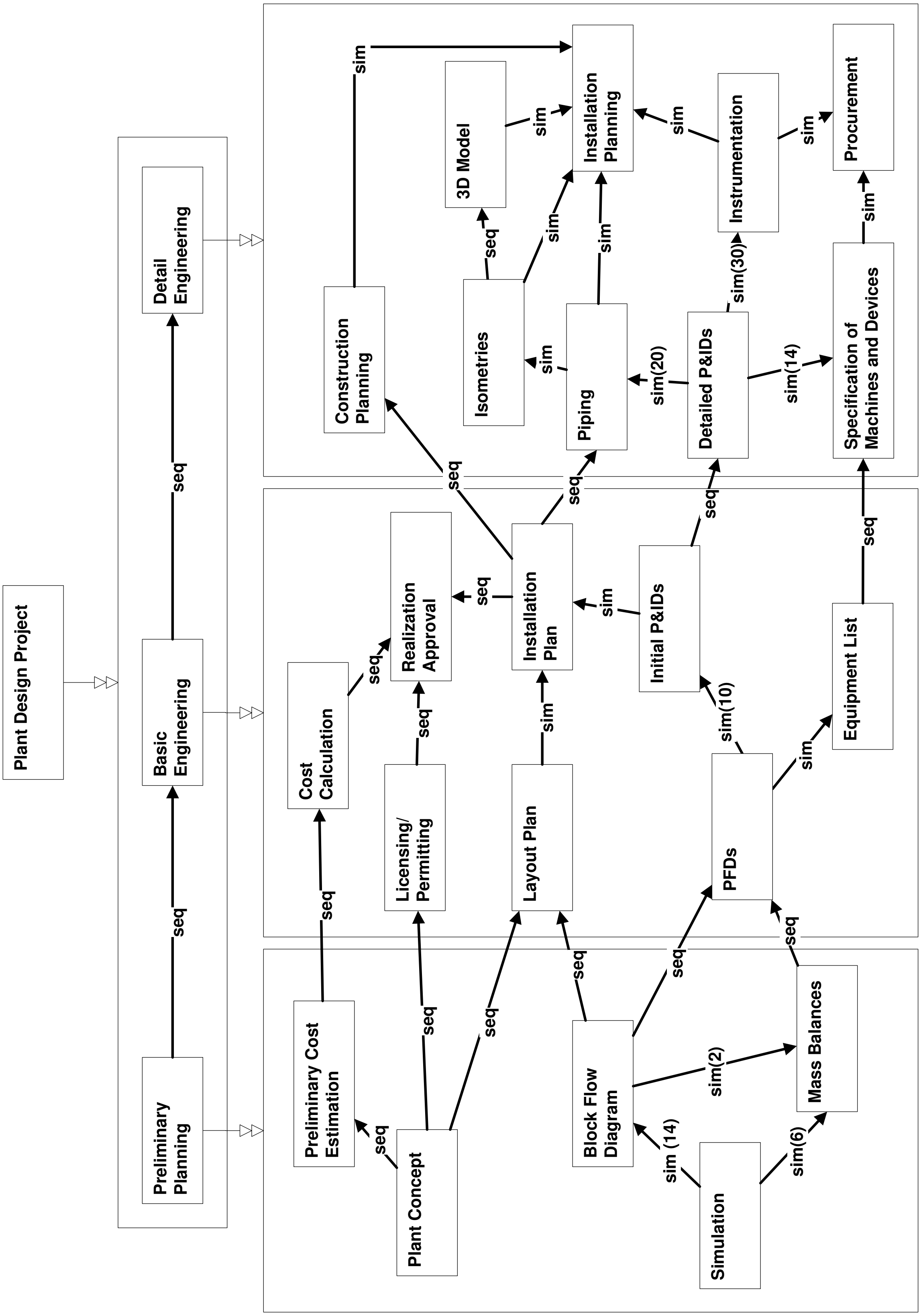


Abbildung 2.6: Overview over the tasks of the work breakdown structure.

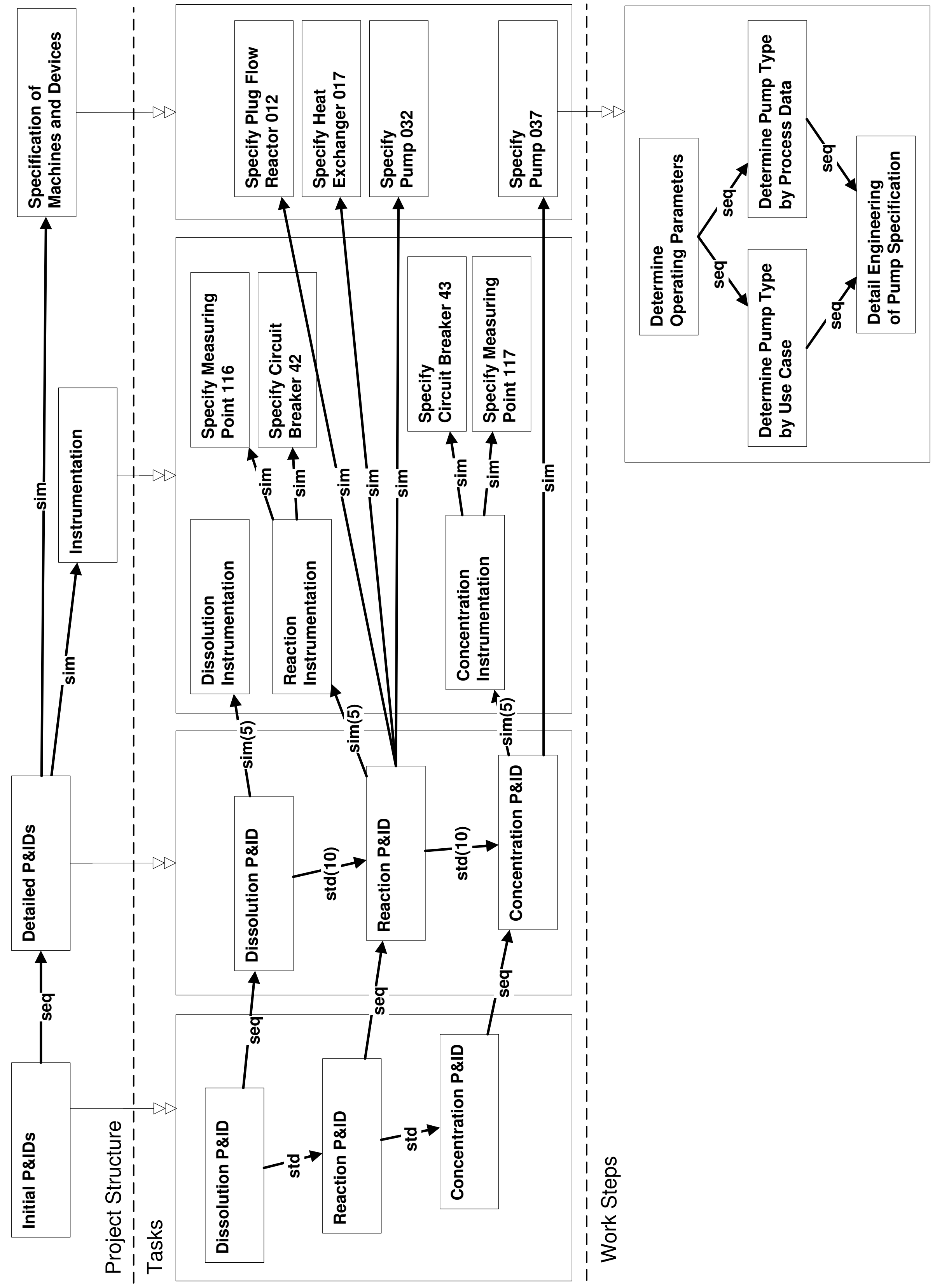


Abbildung 2.7: Overview over technical tasks and work steps.

one of the tasks is executed at project runtime. All subtasks of Specify Pump 037 are work steps which together specify the procedure the engineer has to follow to complete the task. Work steps of individual engineers are not taken into account during project scheduling.

The planning data of the defined tasks in the example are listed in Table 2.1. For every task, the required workload in man hours is estimated. Furthermore, every task has budgeted costs in the currency Euros. Finally, the total duration of a task can be estimated. The workload and budget of a task include the respective values of all subtasks.

Task	**Workload (man hours)**	**Budget (Euros)**	**Duration (work days)**
Project	14278	759589	586
Preliminary Planning	1678	89269	76
Simulation	800	42560	50
Plant Concept	240	12768	25
Preliminary Cost Estimation	250	13300	15
Block Flow Diagram	196	10427	25
Mass Balances	40	2128	10
Basic Engineering	3272	174070	160
Layout Plan	160	8512	20
Process Flow Diagrams	1760	93632	110
Cost Calculation	200	10640	20
Licensing and Permitting	80	4256	40
Initial P&IDs	316	16811	22
Dissolution P&ID	40	2128	7
Reaction P&ID	40	2128	8
Concentration P&ID	40	2128	5
Installation Plan	352	18726	44
Equipment List	64	3404	8
Realization Approval	20	1064	5
Detail Engineering	8156	433899	318
Construction Planning	480	25536	40
Detailed P&IDs	820	43624	110
Dissolution P&ID	200	10640	25
Reaction P&ID	200	10640	25
Concentration P&ID	200	10640	34
Instrumentation	872	46390	110
Reaction Instrumentation	120	6384	20
Dissolution Instrumentation	120	6384	15
Concentration Instrumentation	160	8512	20
Specify Circuit Breaker 42	8	425	2
Specify Measuring Point 117	8	425	1
Specify Circuit Breaker 43	8	425	1

Specify Measuring Point 116	8	425	1
Piping	2112	112358	88
Procurement	2072	110230	259
Specification of Machines and Devices	216	11491	66
Specify Pump 032	20	1064	10
Specify Pump 037	20	1064	10
Specify Heat Exchanger 017	20	1064	10
Specify Plug Flow Reactor 012	24	1276	12
Isometries	352	18726	22
3D Model	176	9363	22
Installation Planning	396	21067	22

Tabelle 2.1: Planning data of tasks in the example scenario.

The complete example task net has been scheduled, enacted, monitored, replanned and rescheduled to show the applicability of the concepts, algorithms and software tools which are presented in this thesis. However, only a part of the overall scenario can be displayed in detail in this thesis. Figure 2.8 shows the cutout of the dynamic task net which will be used to demonstrate the algorithms for scheduling and progress measurement. The figure contains additional information compared to the overview diagrams. For every task, the assigned resources are depicted in boxes with rounded edges. Multiple resources can be assigned to a task. The roles which are required for the task assignments are shown in brackets. In the example project, the project team members can play different functional roles which are structured in a generalization hierarchy. For example, the roles Process Engineer and Mechanical Engineer are both specializations of the general role Engineer. Roles are defined on the organizational level for all projects in the company. Depending on the composition of the project team, a subset of the organizational roles are available in a project. The structure of the project team will be introduced in Section 5.1.3. Figure 2.8 shows a state of the enacted development process at project runtime. The execution states of tasks are visualized by pictograms: gears for active tasks, check marks for terminated tasks, and pencil and paper for tasks which have not been started yet. The arrows connecting different tasks are labeled with their execution semantics which constrain the allowed execution states of the connected tasks. For the tasks Initial P&IDs and Detailed P&IDs, input and output parameters for documents and connecting data flows are defined. A revision of a document has been produced which is visualized by the paper icon. Dynamic task nets and their graphical representation will be described in detail in Chapter 5.

The cutout of the example shows the main project phases Basic Engineering and Detail Engineering and some of their subtasks. Among others, the following steps are executed in the example scenario.

1. The tasks Basic and Detail Engineering are defined as well as their main subtasks.

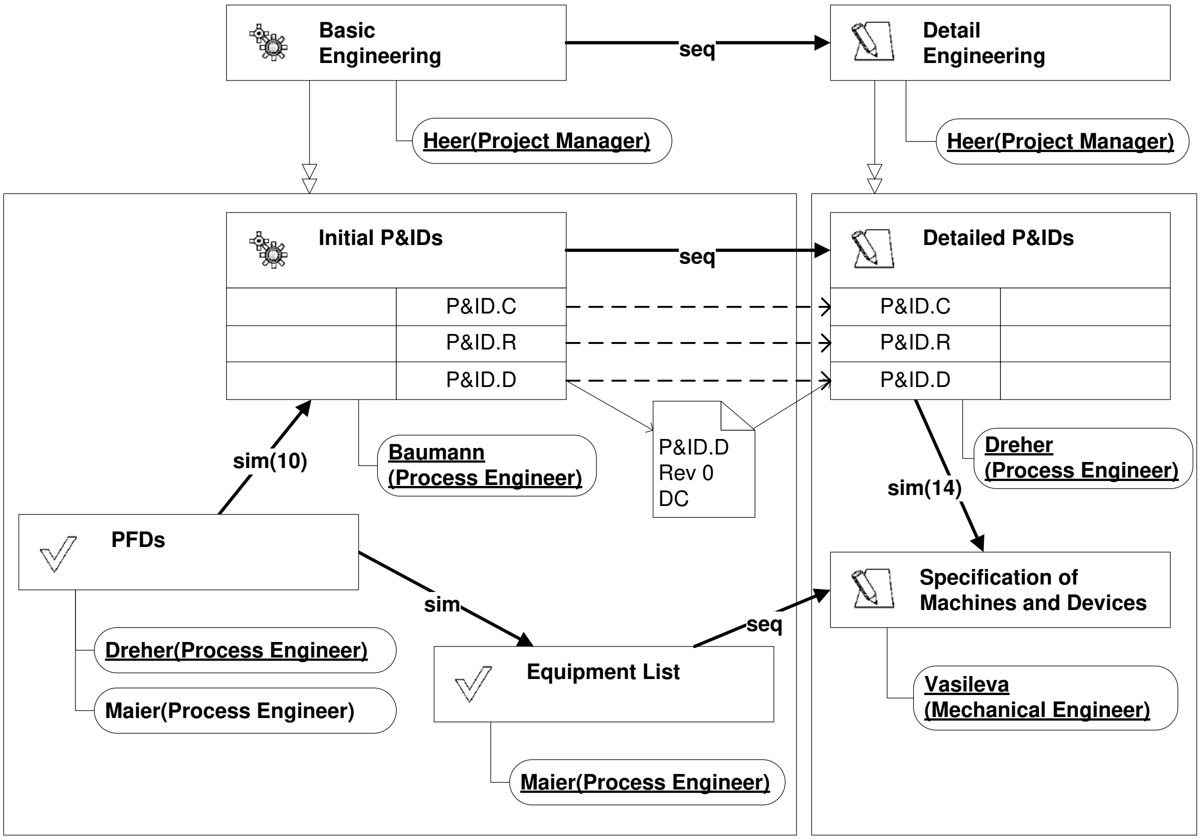


Abbildung 2.8: Cutout of dynamic task net at runtime.

Required roles are defined for the tasks. Resources are manually assigned to the tasks.

2. Basic Engineering and Initial P&IDs are started.

3. A delay of the task PFDs is detected.

4. The tasks PFDs and Equipment List are finally terminated.

5. A first revision of the document P&ID.D is created.

6. After the previous steps, the development process has reached the depicted state of Figure 2.8. The scenario proceeds with a change management case. In the task Initial P&IDs, problems are encountered which require changes to process flow diagrams. The process manager makes the following changes to the task net.
   - A new version of the terminated task PFDs is created.
   - A feedback flow relation is defined indicating that feedback is given from Initial P&IDs to the new task version.
   - Furthermore, a new version of the task Equipment List is created because it was already terminated.

The described enactment and modifications of the dynamic task net could already be performed in the AHEAD prototype. With respect to this example, the functionality of PROCEED for managing a process model instance exceeds the process management support provided by the AHEAD system in the following ways, where the numbers refer to the previous enumeration.

1. The workload for the tasks and their durations are estimated and planned. Based on this information, the tasks are scheduled. During scheduling, eligible resources are automatically assigned to the defined tasks based on the specified required roles.

2. The degree of completion of the running tasks is measured by means of different progress measures. Progress measurement allows to forecast the expected duration and end time of a task by means of earned value analysis. In this way, the delay of the task PFDs can already be detected while it is still executing.

3. The resource Dreher which is responsible for the task PFDs reports the delay to the project manager. Since there is still time buffer for the task Basic Engineering available, prolonging task PFDs will prolong Basic Engineering only marginally and the project deadline will still be met. Therefore, the project manager decides to adapt the plan with respect to the task PFDs to reflect the actual performance.

4. Several changes are automatically performed for the tasks PFDs and Equipment List.
    - The planned end time is set to the actual end time, and the total duration is adapted accordingly.
    - Remaining planned workload is deleted.
    - The planned budget is aligned to the actual budget.
    - The degree of completion is set to 100%.

5. The first released revision of the P&ID.D represents the document state *Devices Complete* (DC). This results in an increased degree of completion of the task Initial P&IDs.

6. The plan changes are implemented in the course of a change management workflow which is requested by the person responsible for the task Initial P&IDs. The creation of the new version of the task PFDs leads to violations of timing consistency constraints with respect to its successors and its parent task. The inconsistencies are resolved by rescheduling the part of the dynamic task net. The change management case is closed after new revisions of the erroneous process flow diagrams have been approved.

Altogether, the extensions of the AHEAD approach with respect to the presented example scenario cover the scheduling of tasks, the progress measurement and detection of delays, dynamic changes to a dynamic task net at runtime including rescheduling, the evaluation and enforcement of timing consistency constraints, and the explicit modeling and enactment of management processes.

Kapitel 3

Fundamentals

The enactment of a development process always takes place in the form of a development project. In particular the problem of task scheduling, which will be introduced in Section 3.2, has been addressed mostly in the context of project management. Therefore, Section 3.1 gives an overview over the main concepts and techniques for project management. Section 3.2 deals with different scheduling problems and general solution approaches. Section 3.3 is concerned with project controlling, in particular with project monitoring, i.e. the determination of the current state and the progress of a running project. Finally, Section 3.4 introduces the workflow paradigm, which has been applied in this thesis for supporting subprocesses in development projects.

3.1 Project Management

According to the ISO 9000 standard, a *project* is a *unique process*, consisting of a set of coordinated and controlled activities and *start and finish dates*, undertaken to achieve an objective conforming to specific requirements including constraints of *time, cost and resources* [ISO05]. In [DH02] the main attributes of a project are summed up as follows.

A project has a goal or objective A definable end product, result or output that is defined in terms of cost, quality and timing.

Uniqueness A project is not a repetitive undertaking. Even projects of the same type, e.g. plant design projects, are essentially different.

Complexity The relationships between the defined tasks may be very complex.

Temporary nature Projects have defined start and end dates.

Uncertainty Projects are planned before they are executed, but due to uncertainties, i.e. unforeseen events and developments, no project is carried out exactly as planned.

Life cycle A project passes through a life cycle that consists of several phases.

Kerzner [Ker98] offers a reduced list for the general attributes of a project comprising: a specific objective, defined start and end dates, funding limits and the consumption of resources. Thereby, he omits one of the most significant characteristics of a project, namely its uniqueness. No two projects are run in exactly the same way. Therefore, it is not possible to provide a precise and fine-grained process model definition to be enacted in several different projects. However, different projects in a common domain may be run in a similar way when coarse grained process models are applied to support project management activities.

3.1.1 Project Management Activities

According to [Ker98], *project management* consists of all activities regarding project planning and project monitoring where monitoring includes steering the project. A similar definition can be found in [PR05] where the proper term controlling is used instead of monitoring. The 2004 edition of the PMBOK [PMI04] defines project management as äpplication of knowledge, skills, tools and techniques to project activities to meet project requirements. Project management is accomplished through the application and integration of the project management processes of initiating, planning, executing, monitoring and controlling, and closing". This definition emphasizes the use of skills, tools and techniques and defines the different processes of project management. Demeulemeester and Herroelen stress in [DH02] that project management "basically involves the planning, scheduling and control of project activities to achieve performance, cost and time objectives for a given scope of work, while using resources efficiently and effectively". Thereby, a distinction is made between planning and scheduling of a project since the focus of [DH02] lies on project scheduling.

Altogether, many different definitions of project management exist which put emphasis on different aspects, be it soft skills or techniques. However, they all agree on the necessity of project planning and controlling. Furthermore, project management is commonly considered to be successful if the project objectives are achieved within time and cost limits and the product has the desired quality and is accepted by the customer.

The different constraints on the project are strongly related to each other, which is usually illustrated by the project management triangle [Ker98, PR05] depicted in Figure 3.1. The overall goal is to optimize all three aspects, i.e. complete the project as soon as possible with as little money as necessary and with the best possible results. However, reducing the makespan of the project may only be possible with higher resource usage and consequently higher costs. If the project duration and/or the resource usage should be reduced this is usually only possible by reducing the scope of the project or producing a product at a lower quality.

The definitions of project management determine the main responsibilities and processes. These can be refined by *project management activities* like the following list shows which is an excerpt of the list presented in [PR05].

- Project planning

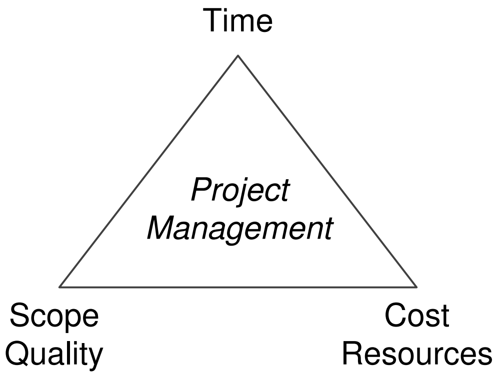


Abbildung 3.1: The project management triangle [PR05].

  - Project definition
  - Identification of project risks
  - Definition of quantity and quality of work
  - Scheduling
  - Resource planning
  - Cost planning and budgeting
- Organization, communication and coordination
  - Definition of roles
  - Task assignments
  - Organization of the information flow
- Leadership
  - Resource selection
  - Promoting the clarity and acceptance of the project goals
  - Promoting the collaboration of the project team members
  - Initiation of changes
  - Decision making
- Controlling
  - Measuring and analyzing the project's progress
  - Integrated controlling of quality, time, resources, costs and budget
  - Ordering corrective measures

In contrast to classical management, project management does not include staffing [Ker98, DH02]. As a matter of fact, a project manager does not staff his project. Staffing is a responsibility of the line managers in a company. The project manager merely has the right to request specific resources. While the activities listed under

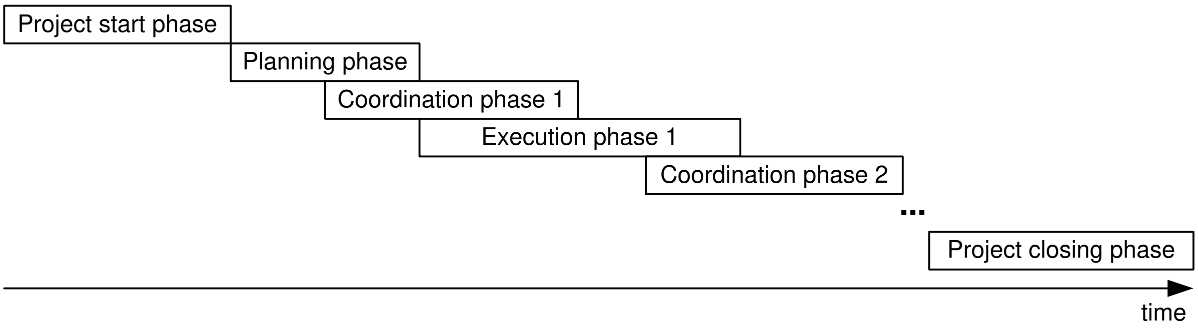


Abbildung 3.2: Project management phases according to [PR05].

leadership describe soft skills of a project manager, well established techniques can be applied to the activities listed under project planning, organization, and controlling. The latter can be supported by appropriate software tools.

3.1.2 Project Management Phases

The project management activities can be associated with different phases which define the life cycle of a project. As for the definitions of the terms project and project management, different models for the general project life cycle exist.

In [PR05], four types of project management phases are distinguished: project start phase, planning and execution phases, coordination and change phases, project closing phase. This is illustrated in Figure 3.2. In the project start phase, the project is initiated. In the planning phase all project documents are created including plans and schedules. Afterwards, coordination and execution phases alternate until the project closing phase is reached. In the execution phases the actual technical work is performed which finally yields the product or outcome of the project. The coordination phases constitute the transitions between the planning, execution and closing phases and allow for changes to the project parameters and in particular the project schedule.

The definition of project management phases can be regarded as a coarse-grained, domain-independent process model for a project, which is even independent of the type of project. Process models for certain domains or specific project types usually also define distinct phases. For example, in plant design projects the design process is commonly divided into preliminary planning, basic and detail engineering as described in Section 2.1. These specific process phases should not be confused with the project management phases. However, there are connections and overlaps between the two perspectives on a project. The transitions between preliminary planning, basic and detail engineering constitute coordination phases in which milestones and intermediate results are evaluated. For software development projects, several elaborated process models exist. Prominent examples are the Unified Software Development Process [JBR99] and the V-Model [Bun06]. The Unified Software Development Process divides the control/execution phase into several execution phases with well defined transitions for coordination and change as described in [PR05] for projects in general. These technical project phases are the inception, elaboration, construction and transition.

In [DH02], a different process model for project management is presented which comprises the following six project management phases.

Concept phase The need for a project is identified, proposals are made for solving the problem together with preliminary estimates of cost and preliminary schedules. Often, a feasibility study is conducted.

Definition phase The proposed solution to the need or problem is exactly defined in terms of the project's objectives, its scope and its strategy.

Planning phase The project activities are identified, the time and resource requirements are estimated, relationships and dependencies are identified, as well as schedule constraints.

Scheduling phase The project base plan is constructed which specifies resource feasible start and end dates for the activities, their resource requirements, and as a result the budget.

Control phase The work is performed according to the plan. The actual progress of the tasks is measured and compared to the planned progress. If a delay, budget overrun or underperformance is detected, corrective actions must be taken to get the project back on track.

Termination phase The project termination phase should include a thorough follow-up to learn from mistakes and to improve the defined processes.

Here, the execution phase is not divided into multiple phases and is called the control phase. The project start phase is subdivided into two phases as is the project planning phase where scheduling is distinguished from planning. The differences of this definition of project management phases compared to [PR05] are probably due to the focus of [DH02] on research about project scheduling while [PR05] addresses the project management practitioner.

Common to all general process models for project management is a planning phase in the beginning. In some cases, scheduling is considered as part of the planning process, in other cases planning is understood as the prerequisite for scheduling. In any case, planning of activities, dependencies, roles and resources has to be conducted before a schedule can be computed. Project planning can be described by the process depicted in Figure 3.3. In this case, a distinction is made between structural planning, i.e. the definition of tasks and their dependencies, and scheduling which includes the estimation of all data that is required to calculate a schedule (cf. Section 3.2).

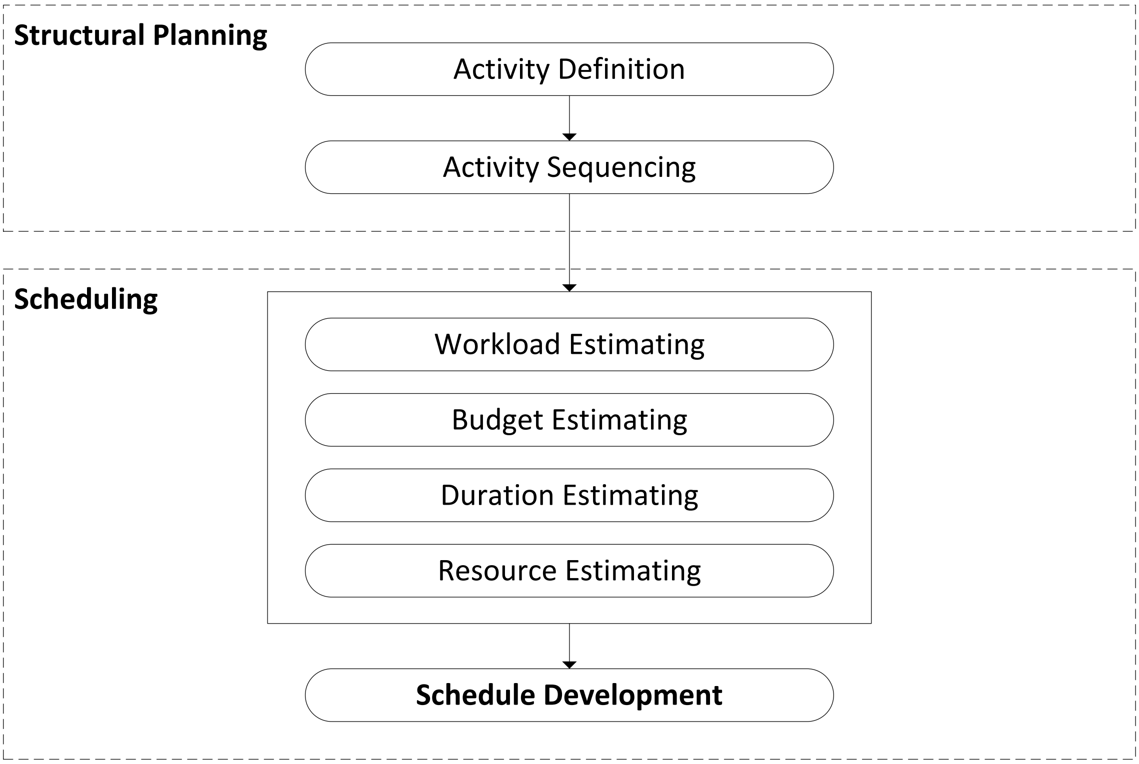


Abbildung 3.3: The process of project planning adapted from [Ela08].

3.1.3 Work Breakdown Structure and Project Plan

To manage the complexity of a project, the work to be done in the project is broken down into manageable portions which results in the so-called *work breakdown structure* (WBS). The project is divided into major pieces which are further refined into tasks and subtasks, until finally the level of *work packages* is reached.

The work packages form the lowest level of the WBS. A work package describes the work to be accomplished by a specific performing group and serves as a vehicle for monitoring and reporting progress of work [Ker98]. Work packages should be uniquely associated with a certain organizational unit to ensure clear responsibilities, they should not extend across project phase boundaries, and they should not refer to different product parts [Bur00].

The defined tasks together comprise the full amount of work that has to be conducted in the project. The costs of the project are distributed over the WBS. Each node in the WBS subsumes the costs and the need for resources of all subtasks. Therefore, also project management is incorporated as a task in the WBS because it occasions costs.

There are different ways to define a WBS, i.e. it can be structured according to different aspects. Three different types of work breakdown structures are distinguished [Bur00].

- Product oriented WBS
- Function oriented WBS
- Process oriented WBS

In a *product oriented WBS*, the main tasks and the work packages are divided according to the product structure, e.g. according to the parts of a chemical plant or a car which shall be designed in the project. As a consequence, the work breakdown structure becomes very similar to the product breakdown structure which is the hierarchical representation of the final product. A *function oriented WBS* structures the project according to the *functional roles* in a project. For a plant design project this would be among others civil engineering, process design, piping and instrumentation. A *process oriented WBS* defines the main tasks according to the enacted development process. The top level of a process oriented WBS therefore resembles the main process steps, e.g. preliminary planning, basic engineering, and detail engineering in a plant design project. Besides the three distinct types of WBS there may be mixed forms, e.g. a combination of function oriented WBS on the top level and a process oriented WBS on the lower levels [Bur00, p. 144].

A WBS is always project specific. There can be no standardized WBS which applies for several projects. It is however possible to use a standard WBS for a class of development projects which has to be tailored to the specific project at hand by removing irrelevant parts and adding new branches or work packages.

The following list is an example of a concrete work breakdown structure for a process plant construction project. The example is a reduced version of the one given in [PMI06]. It is a mixed type of a function and process oriented WBS.

1. Plant System Design
    1.1 Business requirements
    1.2 Process Models
        1.2.1 System Engineering
        1.2.2 Site Development
        1.2.3 Civil Structures
        ...
2. Construction
3. Legal and Regulatory Issues
4. Testing
    4.1 System Test
    4.2 Acceptance Test
5. Startup

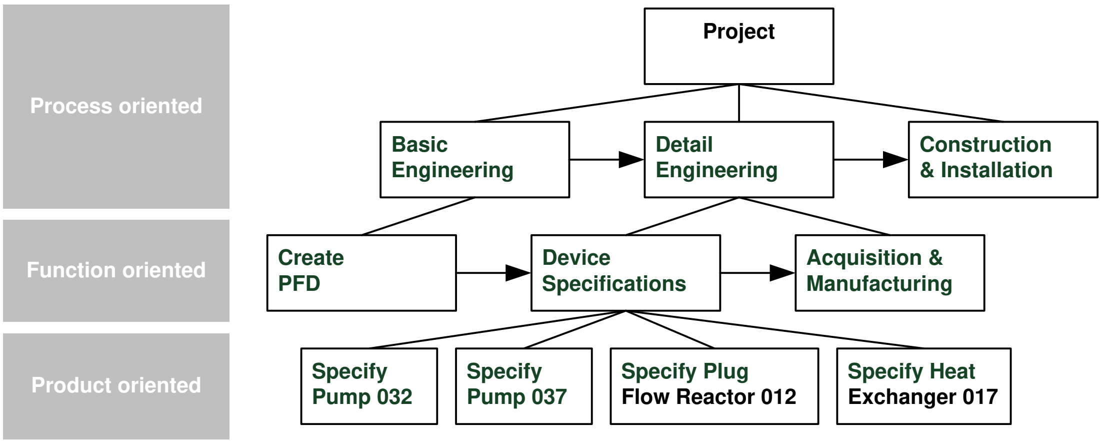


Abbildung 3.4: Mixed type work breakdown structure of example scenario.

6. Project Management

The example for a dynamic task net used in this thesis incorporates a work breakdown structure which is a combination of a process oriented WBS on the top level, a function oriented WBS on the lower levels and a product oriented WBS on the lowest levels (cf. Figures 2.6, 2.7 and 3.4). The first division into subtasks has been made according to the phases of the development process. On the next level, the tasks are defined according to the different functions, e.g. process definition, specification of devices, piping, instrumentation, and procurement. Finally, on the lowest level, tasks are defined according to the structure of the chemical plant to be designed. Individual tasks are defined for the flow diagrams of different plant parts and for the specification of devices.

The WBS of a project constitutes the basis for the responsibility matrix, network scheduling, costing, risk analysis, organizational structure, coordination of objectives and controlling [Ker98]. For detailed project planning, the work packages are further refined into subtasks. For every work package, the specific tasks are identified that need to be performed in order to accomplish the project's objective [DH02]. When the subtasks of the work packages have been identified, their interdependencies can be defined. The tasks can then be visualized in the form of a project network. While the WBS defines the organization of the work in the project, the project network defines the control flow between the tasks and thereby the inherent processes in the project. In addition to the tasks and their dependencies, time estimates for the task durations are made, and the required resources are defined. Based on these data, a project schedule can be calculated. The work breakdown structure, the task dependencies, and the scheduled dates of tasks are all part of the *project plan* which is used to monitor and control a project at runtime.

For the visualization of project plans, several representations have emerged over time where the most commonly used are the *Gantt chart* and the *network diagram* [PR05]. A Gantt chart is a horizontal bar chart in which every bar represents a task in the project. The width of the bar indicates the duration of the task and its horizontal position indicates the start time of the task. The tasks in a Gantt chart can be connected by dependency relations with several different semantics (cf. Section 3.2). However, Gantt charts are often used without task dependencies, since their representation quickly becomes too complex and confusing. A network diagram depicts the tasks in a project as nodes of a graph and the task dependencies as edges. The representation of a graph node and its position do not transport any information about the task duration and start time. Consequently, even in complex project plans the tasks can be neatly arranged, so that the logical task dependencies can be easily comprehended.

A work breakdown structure is not fixed after the planning phase of a project. It is continuously modified and adapted during the execution of the project [Bur00]. In large projects, it is common that work packages of later project phases are not subdivided into subtasks until project runtime. This practice is called *rolling-wave planning* [Ela08]. Work to be accomplished in the near term is planned in detail at a low level of the WBS, while work far in the future is planned for WBS components that are at a relatively high level of the WBS. As a consequence, the whole project plan is continuously modified and adapted at project runtime.

3.1.4 Organizational Breakdown Structure and Resources

In addition to the WBS, an organizational chart is defined for a project which is called the *organizational breakdown structure* (OBS) [DH02]. The OBS comprises the various organizational units that are going to work for the project, i.e. it defines the hierarchical structure of the project team. The members of the project team are called *human resources*. A project team can be structured into several subteams.

Figure 3.5 shows an example of an OBS for a plant design project. The depicted boxes represent the organizational units which may be subteams or positions for individuals. For every organizational unit, there is a person responsible. The connecting lines define the hierarchy of responsibility and authority.

The organizational units of the OBS are linked to the WBS of a project, i.e. the lowest units in the OBS are assigned to work packages of the WBS. For example, the subteam Piping is assigned to the piping work package in a plant design project. The members of the subteam are assigned to subtasks of the work package. The head of the subteam is responsible for the whole work package.

The members of an organizational unit in the OBS do not necessarily have the same qualifications. The qualifications of a human resource are usually defined by his *functional roles*. Role definitions can be used for the assignment of resources to tasks [NW94]. In a first step, the required role is defined for a task. In a second step, an eligible resource which can play the required role is selected for the task. Although it is not necessarily the case, membership to an organizational unit often coincides with certain roles, so that tasks which require the same qualifications are assigned to resources which are in the same organizational unit.

In project management, several different categories of resources can be distin-

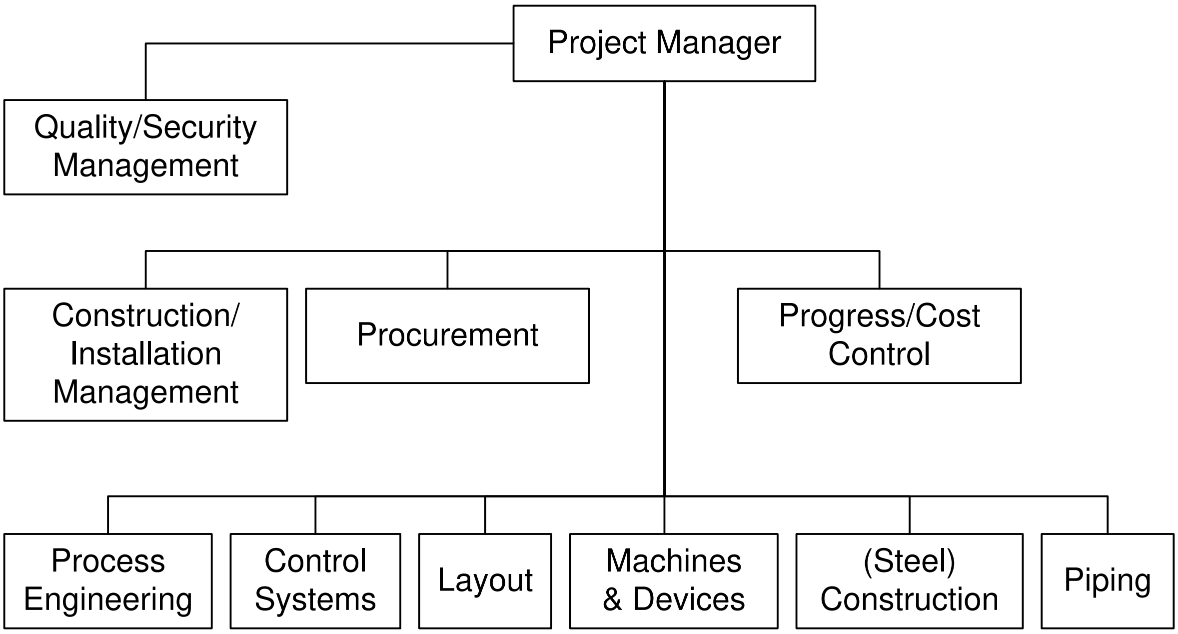


Abbildung 3.5: Example for an organizational breakdown structure [EDL10].

guished [DH02], including human and non-human resources. The main distinction is made with respect to whether resources are renewable. Renewable resources are available on a period-by-period basis. The available amount is renewable from period to period while the total resource used at every time instant is constrained. Typical examples are manpower, machines, tools, equipment and space. Nonrenewable resources are available on a total project basis, with a limited consumption availability for the entire project. The most prominent example is money. Doubly-constrained resources are constrained per period as well as for the overall project, e.g. man hours per day in combination with a constraint on the total number of man hours for the project.

Another category contains partially (non)renewable resources whose availability is defined for a specific time interval. With this kind of resources it is possible to model a project team member who may work on either Saturday or Sunday but not both. This is not possible with renewable resources. As a matter of fact, the concept of partially (non)renewable resources includes renewable, nonrenewable and doubly-constrained resources.

A resource can be continuously divisible or discretely divisible. For a continuously divisible resource, the constant availability in each (real-valued) time instant may not be exceeded. Examples are power and energy. Human resources however are discretely divisible resources since the availability is constrained for each discrete time instant, e.g. a day.

In project scheduling, the most challenging task is to assign the available human resources to the defined tasks. A human resource, i.e. an employee of a company who is a member of the project team, can be categorized as a renewable or doubly-constrained, discretely divisible resource. The time unit is usually a workday and the resource can spend the available man hours of a workday on several tasks.

3.2 Project Scheduling

The results of structural planning are the definitions of tasks and their dependencies. Effective project controlling requires the defined tasks to be scheduled over the project duration. Thereby, planned start and end times are assigned to the tasks. The schedule which is calculated in the project planning phase is called the baseline schedule and is used as a reference for project monitoring. It should satisfy precedence and resource constraints and meet as much as possible the objectives set forward by project management.

Several different objectives can be distinguished [DH02]. The most common objective is the time-based objective to *minimize the project's makespan*, i.e. its duration. Another time-based objective is to minimize the total tardiness, i.e. the sum of all delays of tasks in the project should be minimized.

An important example of a resource-based objective is to *minimize the resource availability costs* while a given project deadline has to be met. In this case, the optimal capacities of renewable resources have to be determined. Another resource-based objective is *resource leveling* where one tries to level the usage of a resource over all time units.

Furthermore, financial and quality oriented objectives can be pursued. In the multi-objective case, scheduling involves the use of different objectives which are weighted and combined. Finally, *multi-project scheduling* is concerned with scheduling several projects in parallel which share common resources. This problem class can be solved by combining the project networks in one super-network and applying the techniques for single project scheduling, although this approach is not always feasible.

Estimation of task durations For the calculation of a baseline schedule, information about the required resources and workload of the defined tasks as well as their expected durations is needed. The estimation of task durations is difficult and error-prone. It depends on human judgment which may be biased, e.g. overestimating task durations is common to add extra safety as a protection against future time-cuts imposed by the upper management.

Two types of methods for workload and cost estimation can be distinguished [PR05]. The *global estimation* allows to derive the overall workload and cost of the project from only few key figures like, e.g., in plant engineering the type and size of the chemical plant. In software development projects, common global estimation methods are the function point method or the Cocomo approach [Bal00]. For global estimation methods, experience values of the company about comparable projects are required. Global estimation methods are useful for the decision whether the project should be executed. They are, however, too coarse to be useful for cost controlling at project runtime.

The *analytical estimation* derives the workload and costs from the corresponding values of the defined tasks and subtasks in the work breakdown structure in a bottom-up fashion. The cost estimation on project level finally should also incorporate indirect costs. Since the workload and costs are defined for the individual tasks and work packages in the project, they form a basis for project scheduling and controlling.

If several similar projects are run in a company which have similar goals, scope, objectives and underlying process models, knowledge from previous projects can be used for estimating the workload, resource requirements, and task durations for a new project. This is often the case in plant engineering, when several chemical plants for the same chemical product are designed by a company. Parameters like the size of the chemical plant, the location, legal and political issues may differ, but the type and number of devices and the general architecture of the plant are comparable and can be scaled accordingly. Therefore, it is often possible to estimate the required effort for the plant design based on the size of the plant and by comparing it to the size of previously designed chemical plants.

There are several different approaches for estimating task durations [DH02]. The most common technique is based on *single-time estimates*. In this case, the duration estimate made for a task is the mean or average time the task will probably take. The duration of a task should not include unexpected disruptions at project runtime. Furthermore, it should not depend on the duration of preceding or succeeding tasks.

Besides the estimation of deterministic task durations, there also exist approaches which deal with uncertainty. In the PERT model, stochastic task durations are used [MRCF59]. The duration of an activity is defined by a probability density function whose parameters are derived from *three-time-estimates*, i.e. a project manager estimates the optimistic, pessimistic and most likely duration of a task. During scheduling, the task durations are derived as values of random variables thereby incorporating a certain degree of uncertainty.

Task net representation After the work breakdown structure and task dependencies have been defined, and the required workload, resources and costs for the defined tasks have been estimated, the project schedule can be calculated which defines planned start and end times for all tasks in the project. Starting point for the schedule calculation is the project network. There are two possible formal representations of a project network.

In the so-called activity-on-arc (AoA) representation, tasks are represented by edges of a directed graph and the nodes of the graph represent events. AoA networks form the basis of the two best-known project networking techniques, namely PERT (Project Evaluation and Review Technique) [MRCF59] and CPM (Critical Path Method) [KW59]. However, AoA networks have the significant drawback that it is not possible to define control flow relationships between tasks with semantics different from the finish-start relationship [DH02].

In the activity-on-node (AoN) representation, a project network is represented by a directed graph $G = (V, E)$ where the tasks are represented by the vertices in V

and the precedence relations by the edges in $E \subseteq N \times N$. This representation allows for the so-called generalized precedence relations (GPRs). Four types of GPRs can be distinguished: start-start (SS), start-finish (SF), finish-start (FS) and finish-finish (FF). A GPR represents a maximal or minimal time-lag between the corresponding events of the source and target tasks. For example, a minimal time-lag $FS_{ij}^{min}(x)$ specifies that activity j can only start x time units after activity i has finished. A GPR of $FF_{ij}^{min}(x)$ is common in design projects where activity j may not be finished before x time units after the preceding activity i has finished so that the final results of the preceding activity can be incorporated. Combined GPRs are also possible, e.g. the combination of $SS_{ij}^{min}(x)$ and $FF_{ij}^{min}(x)$ to demand a time-lag of x time units between the start and the end events of tasks i and j. In [Haj97] an additional GPR was introduced for this specific combination. The use of AoN networks with GPRs with only minimal time-lags but no maximal time-lags is called the *Precedence Diagramming Method* (PDM) [Wie81, DH02, Kle00].

3.2.1 Temporal Analysis

Early techniques for scheduling tasks in a project network like the PERT [MRCF59], CPM [KW59] and the Metra-Potential Method (MPM) [Dib70, KS75] can be subsumed under the term *temporal analysis* [DH02]. With these techniques, the minimal makespan of a project, earliest possible start and end times of tasks and slack times allowing for task delays can be calculated. However, temporal analysis does not take resource requirements and availabilities into account. Therefore, it is not sufficient for computing a resource-feasible project schedule. In this thesis, temporal analysis is performed as a preprocessing step for resource-constrained scheduling. For this reason, the critical path method for AoN-networks is presented in the following.

Critical Path Method

In its basic form, the critical path method (CPM) assumes deterministic task durations and only finish-start precedence relations with zero time-lag [KW59]. The goal is to compute and analyze the critical path, i.e. the set of tasks which cause a project deadline violation if they are delayed. The CPM is divided into two phases, the forward termination (forward pass calculation) and the backward termination (backward pass calculation).

The *forward termination* determines for every task i its earliest possible start time $EPST_i$ and its earliest possible end time $EPET_i$. Let the project network be represented by a graph $G = (V, E)$ with $|V| = n$ and the nodes in V representing the tasks are numbered in topological order so that each arc leads from a smaller to a higher node number. The duration of a task i is denoted by d_i, and P_i denotes the set of immediate predecessors of i. Without loss of generality one can assume a unique starting node for the project network and a unique end node, each with duration zero. The forward termination proceeds according to Algorithm 3.1.

From the earliest possible start times the so-called early schedule can be derived, i.e. every task is scheduled at its earliest possible start time.

Algorithm 3.1 CPM Forward Termination [DH02]

```
$EPST_1 = EPET_1 = 0$
for j:=2 to n do
    $EPST_j := max\{EPET_i | i \in P_j\};$
    $EPET_j := EPST_j + d_j;$
end for
```

In the *backward termination* phase, the latest possible start times (LPST) and latest possible end times (LPET) of all tasks are calculated. The algorithm proceeds analogously to Algorithm 3.1 in reverse-topological order where the latest possible end time of a task is set to the minimal latest possible start time of all immediate successors, starting from the project deadline as the latest possible end time of the unique end node in the network. The overall time complexity of the forward and backward pass calculations is $O(n^2)$ [DH02].

From the earliest and latest possible start and end times the float (or slack time) of tasks can be calculated. The *total float* of a task indicates the number of time units for which the task can be delayed without delaying the project end time. For a task i, the total float TF_i is defined as

$$TF_i = LPST_i - EPST_i = LPET_i - EPET_i$$

The *free float* of a task indicates the number of time units for which the task can be delayed without delaying an immediate successor. For a task i, the free float FF_i is defined as

$$FF_i = min\{EPST_j | j \in S_i\} - EPET_i$$

where S_i denotes the set of immediate successors of task i.

A task with a total float of zero is called a *critical task* since a delay of this task would delay the whole project. In practice however, the delay of a critical task can often be compensated by a faster completion of a succeeding task. A path from the start to the end node of the project network, which only contains critical tasks is called a *critical path*. There may exist several critical paths in a project network.

Temporal Analysis with Generalized Precedence Constraints

Temporal analysis can also be performed on project networks with GPRs (generalized precedence relations). This problem class was originally addressed by the Metra-Potential Method (MPM) [KS75]. The GPRs between two tasks i and j have the form:

$$\begin{array}{ll} s_i + SS_{ij}^{min} \leq s_j \leq s_i + SS_{ij}^{max} & s_i + SF_{ij}^{min} \leq f_j \leq s_i + SF_{ij}^{max} \\ f_i + FS_{ij}^{min} \leq s_j \leq f_i + FS_{ij}^{max} & f_i + FF_{ij}^{min} \leq f_j \leq_i + FF_{ij}^{max} \end{array}$$

Where s_i denotes the start event of task i and f_i its finish event. FS_{ij}^{min} denotes for example the minimal lag time x imposed by a generalized precedence constraint $FS_{ij}^{min}(x)$. A GPR with a maximal time lag can be represented as a GPR with a minimal time lag in the opposite direction [RH96]. Consequently, task nets with GPRs may contain cycles [RH96, DH02]. For example, two tasks can be connected in one direction by a start-finish precedence relation with a minimal time lag and in the other direction by a start-finish precedence relation with a maximal time lag.

GPRs can be represented in a standardized from, e.g. by transforming them to minimal start-start precedence relations of the form $s_i + l_{ij} \leq s_j$ with $l_{ij} \in \Re$. The interval $[s_i + l_{ij}, s_i - l_{ij}]$ is then called the time window of s_j relative to s_i (cf. [DH02, p.116] for details). This results in the standardized representation of the network $G = (V, E)$. The transformation requires that the durations of the tasks are deterministic and fixed. A schedule $S = (s_1, s_2, \ldots, s_n)$ is called time-feasible, if the task start times s_i satisfy the following conditions:

$$\begin{aligned} s_i &\geq 0 & \forall i \in V \\ s_i + l_{ij} &\leq s_j & \forall (i,j) \in E \end{aligned}$$

A time feasible earliest start schedule (ESS) is a solution to the resource-unconstrained project scheduling problem with GPRs under the minimum makespan objective. The ESS can be computed efficiently by using the *Modified Label Correcting Algorithm* which is of time complexity $O(|V||E|)$ [DH02].

In a non-standardized network with GPRs, a critical path may not be a path in the strict sense but rather a tree structure because it may include a task connected with a minimal and maximal time-lag in the same direction, ensuring that the start or completion of the task should be exactly equal to the start or finish of a predecessor plus a certain time-lag. An example for this case is given in [DH02, p.122].

Also the notion of criticality has to be refined for task networks with GPRs. As mentioned earlier, in traditional CPM terminology a task is critical if delaying that task causes a project delay. Likewise, an increase of the duration of a critical task results in an increase of the project duration. In task nets with GPRs, delaying a critical task will also increase the project duration but a duration increase of a critical task may not. This is for example the case if only the start event of the task imposes constraints on other tasks in the network. Therefore, different criticality types are described in [DH02]. A task is labeled:

Critical if it is located on a critical path, which is the longest path from the unique start node to the unique end node,

Start-critical if (a) it is critical, and (b) if the project duration increases when the start time of the task is delayed,

Finish-critical if (a) it is critical, and (b) if the project duration increases when the end time of the task is delayed,

Forward-critical if (a) it is start-critical, and (b) when the project duration increases when the task's duration is increased,

Backward-critical if (a) it is finish-critical, and (b) when the project duration increases when the task's duration is decreased,

Bi-critical if (a) it is start- and finish-critical, and (b) when the project duration increases when the task's duration is either increased or decreased.

The effects of decreasing the duration of a backward-critical task have already been described in [Wie81]. Splitting of tasks is proposed as a solution to the problem that increasing the duration of a backward-critical task may be required to decrease the duration of the project. In fact, this seems to be a practical solution, in which the task is simply paused which has to wait for another task to start or finish before itself may finish. When resources are taken into account, pausing a task means scheduling zero workload for a number of time units. The workload per day could also be reduced to a smaller value but still greater than zero to increase the duration of a task. These considerations however, are not possible in the context of traditional CPM and PDM. Furthermore, the anomalous effects of backward-critical tasks mainly affect the project planning phase. At project runtime, if a backward-critical task has already been started, the decrease of its duration cannot affect its start time anymore. The possible inconsistency regarding a constraint imposed on the end time of a started task can only be resolved by pausing, i.e. splitting, the task and thereby keeping the late end time.

The discussion of different types of task criticality is concerned with the different effects of a task delay or duration increase or decrease on the overall project duration. If, however, the project deadline must not be violated and the latest possible end time of the last task in the network is fixed, the question arises whether the delay of a task or a change of its duration still results in a time-feasible network for the fixed project deadline. Therefore, the notion of *flexibility* has been introduced in [EK92] to denote the freedom to manipulate the task duration to achieve feasibility. A task is said to be

Forward inflexible if extending the task duration results in a time-infeasible schedule,

Backward inflexible if shortening the task duration results in a time-infeasible schedule,

Bi-inflexible if it is forward- and backward-inflexible.

The notion of task floats for task nets with GPRs are similar to those of standard CPM networks. However, in task nets with GPRs, only late starts are used for computing task float, since task duration increases may not lead to a project delay [DH02].

3.2.2 Resource-Constrained Project Scheduling

To obtain a resource-feasible schedule, resource requirements and availabilities have to be considered in addition to precedence constraints and duration estimates. If two tasks require the same resource, they have to share the resource or they have to be executed in sequence. The problem to find a time and resource feasible schedule for a task net with only finish-start relationships and a given set of renewable resources which is optimal with respect to the project's makespan is called the *resource-constrained project scheduling problem* (RCPSP).

There are different approaches for solving the RCPSP [DH02]. The problem of finding the optimal solution to the RCPSP is NP-hard in the strong sense. Hence, exact methods for solving the RCPSP have an exponential runtime complexity in the worst case. Exact methods for solving the RCPSP include linear programming based approaches and branch-and-bound procedures. Besides exact methods, there are heuristic approaches which do not necessarily provide an optimal solution for the RCPSP but allow for an efficient and fast computation. Especially in the case of projects with high uncertainty and many disruptions at runtime, it is reasonable to work with sub-optimal schedules which can be (re-)computed efficiently. The heuristic methods can be divided into *constructive heuristics* and *improvement heuristics*.

Constructive Heuristics

Constructive heuristics start with an empty schedule and add tasks step by step until a time-feasible schedule has been generated. Priority rules are used to select the next task to be scheduled if there are several possibilities. Five different categories for priority rules can be distinguished.

Task based priority rules e.g. the task with the longer duration has higher priority,

Network based priority rules e.g. the task with more immediate successors has higher priority,

Critical path based priority rules e.g. the task with earlier latest possible start time or the smaller slack time has higher priority,

Resource based priority rules e.g. the task with greater resource demand has higher priority,

Composite priority rules define a weighted average of several priority rules.

Based on the defined priority rules, a priority list is computed in which the tasks to be scheduled are ordered according to their priority starting with the highest priority.

Two different scheduling schemes are distinguished. The *serial scheduling scheme* goes through the tasks according to the priority list and determines the earliest start time for each task whilst taking limited resource availabilities into account.

The *parallel scheduling scheme* goes through the time units from the start to the end of the scheduling time frame (in the planning phase this usually equals the whole project duration) and selects for each decision point the tasks which can be started. Decision points are those points in time at which a task is terminated. If several tasks can be started at a decision point which compete for resources, the selection is made according to the priority list.

Both, the serial and the parallel scheduling scheme have a runtime complexity of $O(m \cdot n^2)$ where m is the number of resources and n the number of tasks [KH98]. A schedule obtained by the serial scheduling scheme is a so-called active schedule [DH02, Kle00]. Among the active schedules for a given task net, there is at least one optimal solution. A schedule obtained by the parallel scheduling scheme is a so-called non-delay schedule. There may be no optimal solution among the non-delay schedules of a task net. This means, that for a given example the parallel scheme may not find an optimal solution no matter which priority rules are used. However, according to [Kle00], the two scheduling schemes do not dominate each other. Examples are given for which the parallel scheduling scheme yields better results for backward planning, where the scheduling schemes are applied starting from the project deadline. Furthermore, the serial scheduling scheme performs slightly worse than the parallel scheduling scheme considering the average deviation from optimality but manages to find more optimal solutions.

Since constructive heuristics are efficiently computable, it is common to combine the different variants in a *multi-pass method*. Thereby, different priority rules can be used in each pass and different combinations of serial or parallel scheduling with forwards, backwards or bi-directional scheduling can be applied. In this way, several good solutions can be computed from which the best one can be selected.

Improvement Heuristics

Improvement heuristics start with a given time-feasible schedule, e.g. obtained using a constructive heuristic, and try to improve it by applying change operations to the schedule until a locally optimal solution is reached [DH02]. One example is the *steepest descent* approach which computes the neighborhood of a solution which consists of all schedules which result from applying an atomic change operation to the solution. Then, the best solution in the neighborhood is selected. This procedure is repeated until a solution is reached which is the optimal solution in its neighborhood. In the *fastest descent* approach, not the complete neighborhood of a given solution is generated, but as soon as a better solution than the current one is generated, this is selected for the next round. The *iterated descent* approach executes a descent method several times for randomly generated schedules as starting points. This way, several locally optimal solutions are generated from which the best one is selected.

For the generation of a locally optimal resource-feasible schedule, other metaheuristic approaches can be applied as well [DH02, KH98]. These include *tabu search*, *simulated annealing* and *genetic algorithms*. The latter two have also been applied for schedule repair in case of disruptions [Wan05].

The Generalized Resource-Constrained Scheduling Problem

The RCPSP is restricted to scheduling tasks subject to minimal finish-start precedence constraints. The *generalized resource-constrained project scheduling problem* (GRCPSP) also allows for the additional precedence relations of the precedence diagramming method (start-start, finish-finish and start-finish relations with minimal time lags) [Kle00]. Furthermore, release and due dates can be specified for tasks which define time windows for the tasks. Finally, the resource availabilities may vary for different time units.

Just like the RCPSP, the GRCPSP is NP-hard. Even the decision problem of detecting whether a feasible schedule exists for the GRCPSP is NP-complete [DH02].

In [Kle00] it is described, how the heuristic approaches for the RCPSP would have to be adapted to handle the generalized precedence relations of the GRCPSP. A brief discussion is given showing how priority rules can be adapted to the GRCPSP. The serial scheduling scheme can be immediately transferred to the GRCPSP while the parallel scheduling scheme has to be slightly modified. The computation of the next decision point has to be adapted because a task can not only be scheduled when another task is finished but possibly before because of generalized precedence constraints. A time-feasible schedule for the GRCPSP can be computed using forward planning. Backward planning, however, is generally not possible for the GRCPSP since different schedules are obtained for different project end dates due to varying resource availabilities [Kle00].

The *resource-constrained project scheduling problem with generalized precedence relations* (RCPSP-GPR) allows for all GPRs including maximal time lags. In [DH02], exact branch-and-bound solution procedures are presented for the RCPSP-GPR. However, it is not described, how the heuristic approaches for the RCPSP would have to be adapted to handle generalized precedence relations with maximal time lags. No information can be found in literature, how maximal time lags affect the applicability of the scheduling schemes and priority rules for constructive heuristics.

3.2.3 Disruption Management

The baseline schedule is generated before the execution of the project. However, no project is executed exactly as planned. Due to uncertainties, unexpected disruptions occur frequently at project runtime which often require changes to the project schedule. Changes to the baseline schedule are eventually required when corrective measures have failed to align the actual performance with the plan.

According to [HL05], different methods for schedule generation under uncertainty can be followed. One possibility is to not generate a baseline schedule at all. In this case, scheduling policies are applied to assign tasks to resources on the fly at project runtime. However, without a baseline schedule effective project monitoring is not possible. The second possibility is to generate a baseline schedule with no anticipation of variability. In this case, reactive scheduling is required to repair the schedule in case of disruptions. In the third case, information about the particular variability characteristics, e.g. probability distributions for activity durations, is used.

This approach is called proactive (robust) scheduling. In this case, disruptions do not necessarily require schedule repair.

A schedule is called *robust* if it is rather insensitive to disruptions [HL05]. The term quality robustness is used when referring to the insensitivity of the schedule performance in terms of the objective value, e.g. the resource availability costs. The term solution robustness or *stability* is used to refer to the insensitivity of the task start times to disruptions, i.e. the schedule itself does not change significantly during repair. Furthermore, a schedule is called *flexible* if it can be easily repaired, i.e. changed into a new high quality schedule. If the flexibility of a schedule is low, high effort is needed for the repair.

3.3 Project Controlling

The goal of project controlling is to align the actual project performance with the project plan regarding costs, time and (the quality of) the results. It is necessary because of risks and uncertainties. At project runtime, the following unexpected events may occur [PR05].

Change of goals Changes to the project goals or to customer requirements,

Disturbances Resource unavailability, technical problems, delivery delays, etc.,

Deviations from the plan Wrong estimates or low efficiency.

Project controlling involves *monitoring* and *steering* a project. However, the usage of the term project controlling is not consistent in the literature and in practice. Sometimes controlling is used synonymously for monitoring a project. On the other hand, the term control is often used synonymously with steering.

The German standard DIN 69901-5 subsumes under the term project controlling the measurement of the actual project status, the comparison with the plan, the analysis and evaluation of deviations resulting in recommendations for corrective measures, the planning of corrective measures, and finally the steering of their implementation [DIN09].

Similarly, the Project Management Body of Knowledge (PMBOK) [PMI04] lists as the core functions of project controlling the comparison of actual and planned performance, the analysis of deviations, the evaluation of trends and possible alternatives, and the recommendation of suitable corrective measures. The PMBOK Guide defines processes for controlling the scope, schedule, costs and risks in a project as well as for change and quality control.

The *project management control cycle* depicted in Figure 3.6 describes the connection between project planning, execution and controlling [PR98, Lic06]. Project planning is a prerequisite for project controlling. It provides the initial project base plan. Project execution takes place based on this plan. During execution, the actual status of the project is measured and compared with the plan. Deviations from the plan are analyzed. Based on this analysis, either corrective control measures are taken

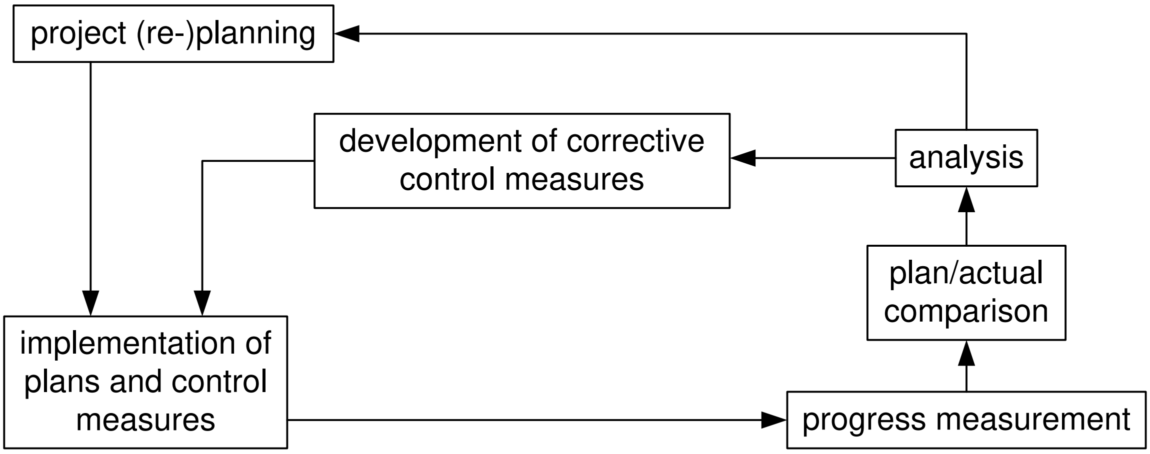


Abbildung 3.6: The project management control cycle, adapted from [PR98].

or the plan is adapted. The execution of the project proceeds either with corrective control measures or an adapted project plan. If corrective control measures are not effective, this is detected in another iteration of the cycle and further management decisions have to be made. In the following, the different steps of project controlling are described in more detail.

3.3.1 Determining the Actual Project Status

The first step is to determine the actual project status. The collected data refers to the parameters performance, time, resources, costs and process quality [PR05]. The actual performance is measured in terms of quality and quantity.

The quality of the (intermediate) products in a project has to be ensured by means of regular reviews and quality tests, and the documentation of their results. This is usually part of quality management in a project.

The quantitative performance is measured for the work packages in the project by computing their *degree of completion* (DOC). The DOC indicates, how much of the work with respect to the defined goal of the work package has already been done. This measure is independent of the actually spent time for the work package. For example, if the goal of a work package is to do the layout of the pipes in a chemical plant, then one can argue that the work is half done when half of the pipes have been laid out, no matter whether 20%, 50% or 80% of the scheduled time for the work package has already passed.

It is fairly easy to determine the DOC of work packages which have not been started yet or which are already completed. The challenging task is to accurately measure the DOC of work packages which have been started but are not completed yet. There are several common methods for calculating the DOC of tasks in a project. Some methods can be used in every project while others can only be used in projects of a certain type. In the following, the methods presented in [PR05] are coarsely described.

A comparably simple technique is the *start-end method*. It assigns a DOC of 0% to a task which has not yet been started and 100% to a finished task. A running task is assigned a DOC of 50% or any other value k with $0\% \leq k < 100\%$. For the individual task, this may be a comparably inaccurate measure. However, when the DOCs of several tasks are aggregated at the common parent task, this results in a fairly accurate DOC for the parent task.

It is common in project planning to define *milestones* in the project schedule. These milestones define specific intermediate results of the project. Estimates for the DOC of the parent task are assigned to milestones. When a milestone is completed, the parent task is assumed to have reached the defined DOC.

It is always possible to *directly estimate* the DOC of a task based on experience and information about the work performed in the task. Since this direct estimation is error-prone, an alternative method is often used in which the *remaining workload* for a task is *estimated*. This results in a forecast for the expected total workload of the task which may deviate from the originally planned total workload. The ratio of actually performed work to the expected total workload yields the DOC of the task in question.

If there is an indicator for the work to be done in a task which increases *proportionally* to the degree of completion of the task, the former can be used to compute the latter. In construction projects for example, the square meters of concreted ground or the amount of used concrete or steel in tons can be used to measure the DOC of a task. In plant design projects, the number of completed device specifications can be used. If a task is subdivided into a larger number of comparable subtasks, it is possible to use the number of finished subtasks as the indicator. The DOC is then calculated as the ratio of the actually used or completed amount to the total planned amount, e.g. the number of finished tasks divided by the number of planned tasks. Two important aspects have to be considered for this measure. The completed work has to fulfill the required quality standards, and the planned total amount of work must not change during the execution of the task.

Besides the degree of completion of tasks, the effort which has actually been spent on the tasks has to be measured as well. This can be realized by means of a time-tracking system in which the performers of the tasks book their actually spent working hours. From the actual workload and the cost rates of the resources, the actual resource related costs of tasks and the whole project can be derived. In development projects, the costs of human resources account for the major part of the overall project costs. In construction projects, costs for machines, devices and material have to be considered as well. Costs for the required technological infrastructure, energy, facilities and also trainings and travel expenses are usually booked as base costs of a project.

3.3.2 Target-Performance Comparison and Analysis

When the actual performance of the tasks, the effort spent on the tasks, and the actual costs have been determined, the values have to be compared with the planned performance, workload and costs.

Besides the actual degree of completion of a task, the *planned degree of completion* can be computed as well. One possibility is to compute the quotient of the elapsed time of a task and the planned total time of a task. However, in this case a linear increase of the planned progress over time is assumed. A more exact approach is to compute the quotient of the planned workload to date and the planned total workload of the task. This results in non-linear functions for the planned progress when the planned workload is not equally distributed over the calendar days for which the task has been scheduled. The actual DOC of a task can be directly compared to the planned DOC. A deviation indicates that the task is either behind schedule or ahead of schedule. However, from this direct comparison of the actual and planned DOC it is not possible to make any predictions about the expected actual duration of the task. Furthermore, nothing can be said about the expected actual costs of the task in comparison to the budgeted costs.

For the quantitative comparison of planned and actual progress and costs, the *earned value analysis* (EVA) can be applied [Anb03, PR05] which is also referred to as *earned value method*. From the degree of completion of a task, the *earned value* (EV) can be computed. It is the monetary value of the accomplished work and is computed as

$$EV = DOC \times BAC \tag{3.1}$$

where BAC is the planned *budget at completion* of the task. The earned value is also called the *budgeted cost of work performed* (BCWP). The *planned value* (PV) is the *budgeted cost of work scheduled* (BCWS), i.e. the money which should have been spent on the task to date according to the plan. The planned value can be derived from the project schedule and the cost rates of the resources. The *actual value* (AV) is the *actual cost of work performed* (ACWP), i.e. the personnel costs of the actual effort spent on the task to date. It can be determined by the results of time tracking and the cost rates of the resources.

Figure 3.7 shows an example of the development of the planned, actual, and earned value over time. From these values, two indices can be computed. The *schedule performance index* (SPI) indicates whether the task is on schedule, behind schedule or ahead of schedule. It is computed as follows.

$$SPI = \frac{EV}{PV} \tag{3.2}$$

The task is exactly on schedule if $SPI = 1$. If $SPI < 1$ then the task is behind schedule since the earned value is smaller than the planned value. If $SPI > 1$, the task is ahead of schedule. Corrective measures for a delayed task are only required if the SPI value falls below a certain threshold. The thresholds are usually project- or company-specific. For example, the following thresholds can be used together with

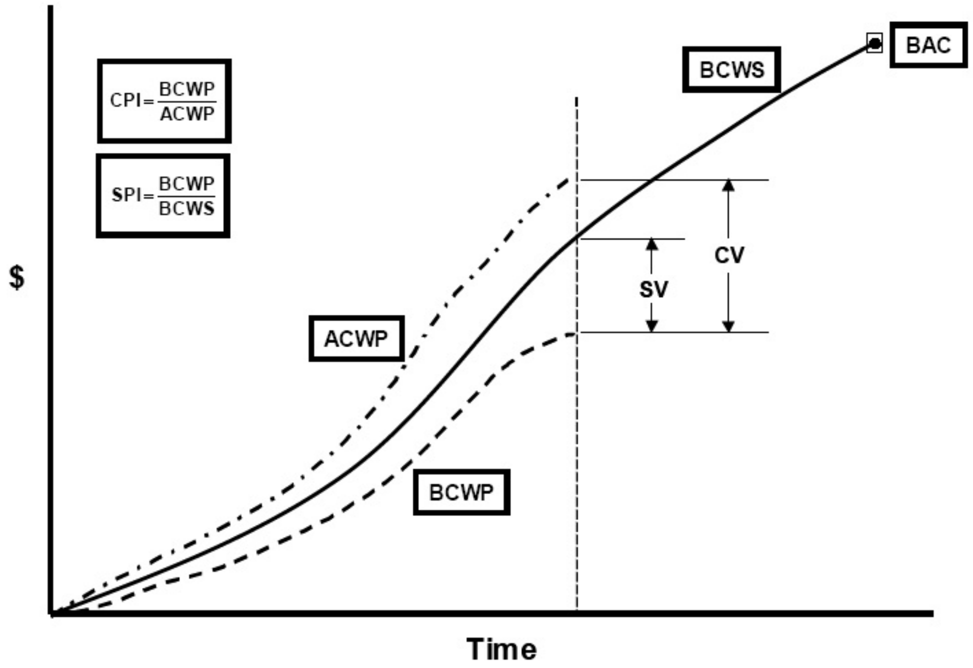


Abbildung 3.7: Key figures of earned value analysis [Lip03].

the implied consequences for project controlling.

$0.9 \leq SPI$	The task is on schedule or only a little bit behind. No corrective measure is required.
$0.7 \leq SPI < 0.9$	Corrective measures may be necessary and should be discussed.
$SPI < 0.7$	The task is considerably behind schedule and corrective measures are required.

The calculation of the schedule performance index has a known issue which is described in [Lip03]. If the planned deadline of a task is exceeded, the SPI converges from below to the value 1 until the task is finished, i.e. the task reaches an optimal performance in the end. This is the case because the planned costs remain constant after the planned deadline, but the task is still executed and the earned value increases. This problem is illustrated in Figure 3.8. Similarly, if the task finishes early, the SPI converges from above to the value 1 until the planned deadline is reached. In [Lip03], the *earned schedule* approach is presented which addresses the described problem. The presented performance index is called SPI_t. It is calculated by determining the date on which the current earned value was or will be the planned value. This date is compared with the current date. If it is before the current date, the task is behind schedule because the current earned value should have been reached earlier. Consequently, the SPI_t is less than one. If it is after the current date the task is ahead of schedule and the SPI_t is greater than one. Empirical studies have shown that the values of the SPI_t do not differ from the values of the SPI until the final stage of a task, but that the SPI_t does not exhibit the described anomalies in the final stage of the task.

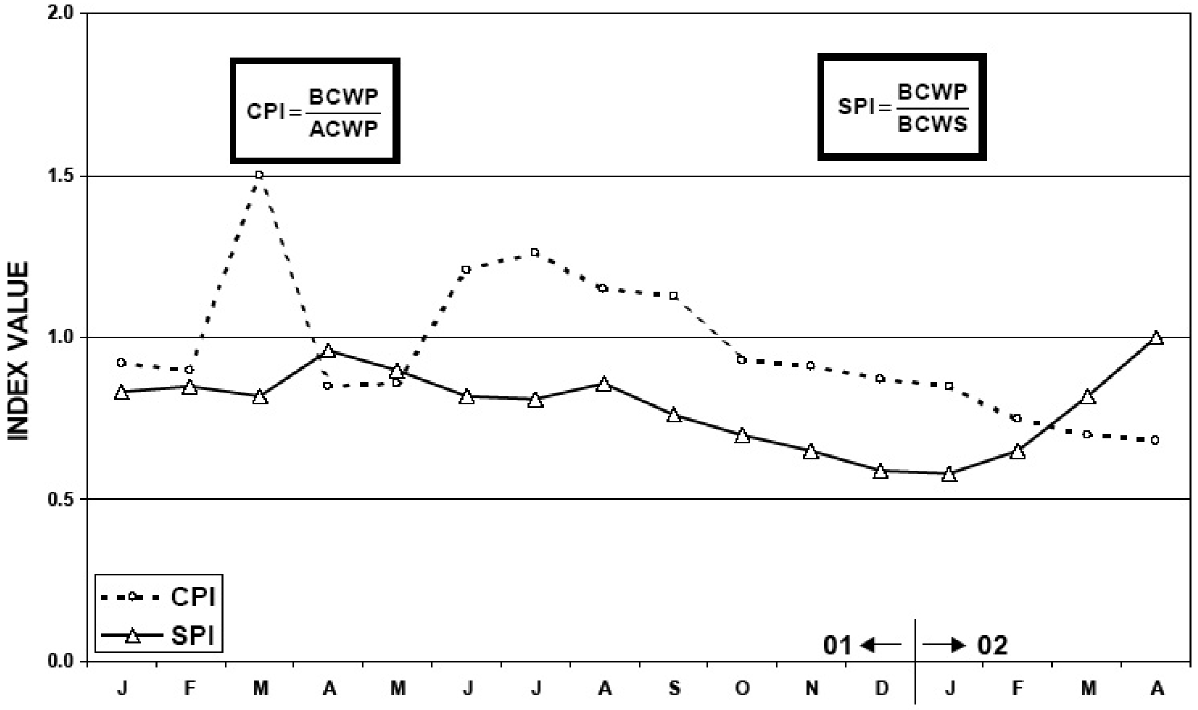


Abbildung 3.8: Development of the SPI and CPI in a delayed project [Lip03].

The *cost performance index* (CPI) indicates whether the budget for the task will be met, overrun or underrun. It is computed as follows.

$$CPI = \frac{EV}{AV} \tag{3.3}$$

If $CPI = 1$, the task is exactly in budget. If $CPI < 1$, the task is over budget since the value of the performed work is less than the spent money. If $CPI > 1$, the task is under budget because the earned value is greater than the actual spent money.

The computation of the performance indices SPI and CPI is a way to analyze the current status of a task. It goes beyond the mere comparison of the actual and planned degrees of completion, since the indices are quantitative measures for how much a task is behind schedule or above budget respectively. In practice, the performance indices SPI and CPI are usually not calculated for every task in a project but only for the top-level tasks in the work breakdown structure and for the whole project.

Further analysis of the current project status is concerned with *forecasting* the expected end time and final budget of tasks. This can be done based on the SPI and CPI respectively [Anb03, Lic06]. The *forecasted duration* (FD) of a task can be computed as

$$FD = \frac{PD}{SPI} \tag{3.4}$$

where PD denotes the planned duration of the task. This method for forecasting assumes that the past performance of the task is a good predictor of the future performance, i.e. efficiencies or inefficiencies observed to date will prevail to completion. This assumption is generally associated with earned value analysis. In contrast, the critical path method assumes that problems and opportunities that affected the project's performance in the past will not occur in the future and that past performance is not a good predictor of future performance. For example, if a critical activity in a project network is delayed by a certain amount of time, it is assumed that the whole project will be delayed by exactly this amount of time, i.e. future performance will parallel the original plan. Based on this assumption an alternative method for forecasting the duration of a task is presented in [Anb03]

$$FD = PD - TV \tag{3.5}$$

where TV denotes the schedule variance in time units, e.g. if the task is 4 time units behind schedule then $TV = -4$. This method is aligned with the critical path method. However, equation (3.4) is generally used in conjunction with earned value analysis.

Using the forecasted durations of tasks, it should be continuously checked in a project whether internal deadlines are met to detect delays early. This is particularly required, if the success of the project strongly depends on meeting the final deadline because a contractual penalty would apply in case of a deadline violation. This is usually the case for plant construction projects [PR05].

The earned value analysis has not been widely used to estimate the expected duration of a task at completion. It is more common to forecast the *estimated cost at completion*, also called estimate at completion (EAC). With the assumption that past performance is a good predictor of future performance, the EAC is computed as:

$$EAC = \frac{BAC}{CPI} \tag{3.6}$$

where BAC denotes the planned *budget at completion*. Analogously to the forecast of the task duration, there is a formula which assumes that future performance will parallel the original plan [Anb03]:

$$EAC = AV + (BAC - EV) \tag{3.7}$$

The planned costs for the remaining work are added to the actual costs to date. As for the duration forecast, equation (3.6) is generally used in conjunction with earned value analysis instead of equation (3.7).

The earned value method is a useful means for quantifying deviations from the project plan and it helps the project management to focus on work packages that need the most attention. In particular, the quantification of deviations allows to distinguish between minor and critical delays and budget overruns.

3.3.3 Steering a Project

Delays, cost overruns, the failure to reach quality standards, and other disruptions require the intervention of the project management personnel. Steering a project is necessary to ensure that the project stays on track, i.e. that it will still be completed within time and budget limits and will yield the aimed-at results with the required quality. Steering as part of project controlling involves the implementation of corrective measures or plan changes at project runtime [Bur00].

With respect to the project management triangle depicted in Figure 3.1, plan changes refer to the aspects time and cost/resources while corrective measures may affect the scope of the project and the quality of the final product. However, corrective measures may also increase the performance of the project team members which is not represented in the project management triangle. Increased performance may yield the same results in terms of scope and quality while using less time and/or less resources.

Corrective measures Corrective measures include all actions which aim at increasing the performance of the project team members or changing the focus of the work performed without changing the project plan. Increasing the performance of the project team members can for example be achieved by personal interviews and feedback, trainings, improved equipment, and additional tools. Sometimes it may be required to change the scope of the project. For example, to meet the project deadline, it may be necessary to drop certain requirements for the final product. However, this is only seldomly possible since the requirements are usually fixed in the contract for the project.

Plan changes Changes to the project plan include the addition and deletion of tasks and changes to the scheduled dates of tasks. Furthermore, the priority of tasks can be changed, i.e. the work on certain tasks is scheduled earlier for the assigned resources. Plan changes may even be required if the project has been executed as planned in the past but the scope of the project has changed. In particular, new customer requirements may lead to significant changes to the design of the product under development and consequently to the project plan. New tasks may have to be created and existing tasks may have to be removed from the plan. With respect to the project management triangle, a change has been made to the scope of the project, but the actual plan changes still refer only to time and costs. Changes to the project plan also include changes to resource assignments. A resource may be reassigned from another task which is ahead of schedule to accelerate the performance of a task which is behind schedule. If additional resources are used or more working hours are planned for resources of the project team, then this involves increased labor costs.

3.4 Workflow Management

In recent years, *workflow management* has become a popular approach in the broader field of process management [Jab95, JB96, Law97, JBS99, vdAvH02]. In particular in the area of business process management the workflow paradigm has been widely adopted. For example, in the insurance domain, *workflow management systems* are used to guide clerks during their diverse cases [Wör10, Law97]. Conventional workflow management systems have proven to be suitable for predefined, rather static business processes. However, they are less suitable for the management of development processes.

The specific characteristics of development processes discriminate them from business processes. Business processes can be instantiated as individual and independent cases. For example, the cases performed by different clerks in an insurance company are usually independent, and so are the workflow instances which are enacted to support the work. In contrast, the subprocesses of a development process are always enacted in the context of an overall development process. The different subprocesses usually depend on each other which makes the management of the overall development process a challenging task.

A process model definition for a business process usually defines a standard procedure which has to be followed, and deviations from this procedure are handled as exceptional cases. Disruptions are seldom but have to be handled nevertheless by the process management system. This problem has been addressed in [WEH08, Wör10]. In contrast to this, disruptions are considered the normal case in development processes. The degree of uncertainty is much higher than in common business cases, and dynamic changes to a process model instance occur frequently [NW94].

Finally, a process model instance of a development process cannot be completely defined in advance, i.e. before runtime of the process. Many tasks cannot be planned until certain intermediate results of the development process are available, e.g. the design of a software architecture or a flow sheet which defines the main components of a chemical plant. These key artifacts are required for the refinement of the process model instance [NW94]. Depending on a released revision of a key artifact, new tasks are defined and existing tasks may have to be aborted. As a consequence, it is not possible to determine all tasks of a process in the process model definition. In contrast, a business process is usually not modified structurally due to the documents which are produced in a certain case.

Despite the limitations of the workflow approach, subprocesses of a development process may be adequately supported by a workflow management system [Hel08a, HHM$^+$06]. Therefore, workflow management functionality has been integrated into PROCEED as described in Section 6.3 and [HBW09]. Furthermore, workflows are used in PROCEED to support management processes as described in Section 9.1. This section introduces the necessary terminology and concepts from the area of workflow management. Different formalisms and a general reference model for *workflow management systems* are presented as well as a class library for workflow management which has been used as a basis for the implementation of the workflow management functionality of the PROCEED prototype.

3.4.1 Definitions and Views

A workflow is usually regarded as "the computerized facilitation of a business process, in whole or part"[Wor95]. Hence, a distinct characteristic of a workflow as a specific type of process is the software support, let it be client tools for the human actors or the partial automation of the process by means of software services.

Furthermore, workflows are classified in [DvdAtH05] as *person-to-application* processes, i.e. human actors interact with software tools to perform the defined processes. In contrast to that, *person-to-person* processes occur in *computer-supported collaborative work* (CSCW) where several human actors collaborate and software tools are only used for the necessary coordination. Finally, *business-to-business* processes define the interaction of software systems where no human actor is involved.

All required information for the execution of a workflow is defined in a *workflow definition* sometimes also called workflow model or schema. This definition can be instantiated several times in a workflow management system leading to several *workflow instances* of the same workflow type. Accordingly, one distinguishes between *build time* and *runtime* of a workflow. During build time, the workflow definition is created and elaborated. The tasks and control flow are defined, as well as the resource requirements of the tasks, the data, required applications, exception handlers and the like. During runtime of the workflow, the execution states of the tasks are changed by the workflow management system according to the workflow definition and the state of the workflow instance. Resources are automatically assigned to tasks who execute the tasks by using the available applications.

To cope with the complexity of workflow models, it is useful to look at them from different perspectives [JB96, vdAvH02].

Tasks and Control Flow This view describes the tasks to be performed within a workflow as well as their relationships which concern the routing of tasks. Workflow definitions usually contain control structures like alternative branches and loops comparable to those in imperative programming languages.

Resources and Organizational Structure Resources include all kinds of objects that are necessary to perform a workflow or task. Human resources are members of organizational units and have competencies and responsibilities.

Data and Data Flow Data are divided into control and production data. Local variables of workflows are examples of control data while documents produced in the process are considered as production data. Parameters can be defined for workflows and tasks. Data flow generally happens between parameters and local variables of workflows.

Temporal Aspects This view includes information about deadlines and durations of activities, temporal distances between activities, availability times for resources and other temporal restrictions. The consistency of temporal constraints has to be guaranteed and schedules can be derived from the constraints.

Applications The applications view focuses on the application programs that are used in order to perform certain tasks. In the business domain, this might be an accounting system, a text editor or a form for structured input.

Exception Handling A workflow definition usually describes the regular process execution. A workflow management system should also provide means for systematic recovery in case of exceptional and faulty situations. However, even these exception handling mechanisms have to be incorporated into the workflow definition before execution.

Additional views include Business Rules, Interorganizational Cooperation and others [DvdAtH05]. While most of the views apply for processes in general, control data and exception handling are particularly required for workflows since they enable process automation.

3.4.2 Modeling Languages

Since workflow management aims at a (partial) automation of the defined processes, the control flow definition usually comprises control structures like alternative branches and loops. The different modeling languages and formalisms for the definition of workflows therefore require modeling elements for these control structures. Other process modeling languages which are merely used for the analysis of processes or for manual enactment, do not necessarily require these control structures. Different formalisms have been used for workflow modeling, and many different modeling languages have been developed.

A common approach is to use Petri nets for workflow definitions [PW08, Aal98, Aal96, DvdAtH05]. Tasks are represented by transitions in a Petri net and the places in the Petri net represent the different possible states of the workflow. The firing of a transition represents the execution of the corresponding task. In [Aal96], van der Aalst gives three good reasons for using a Petri-net-based workflow management system. These are the formal semantics, the state-based model and the available analysis techniques. A disadvantage of Petri nets however is the difficulty to realize dynamic structural changes to the workflow definition at runtime.

An *event-driven process chain* (EPC) is a type of flowchart for business process modeling [KNS92, JBS99, DvdAtH05]. An EPCs is a bipartite graph which incorporates event nodes and function nodes. Event nodes represent states of the workflow while functions are the equivalent to tasks. In EPCs the control flow is defined by logical relationships for branch/merge and fork/join which connect events. EPCs are used in ARIS Workflow and SAP Business Workflow for modeling business processes.

The Web Service Business Process Execution Language (WS-BPEL) [WAM$^+$07] is a textual language for process definitions which can be executed by a process management systems like the WebSphere Process Server [WAM$^+$07]. Process models defined by means of the graphical Business Process Modeling Notation (BPMN) can be transformed to workflow definitions in BPEL [DDO07, Whi05, WDGW08, ODtHvdA07].

UML activity diagrams can be used to specify workflow definitions [DtH01, DvdAtH05]. However, the fact that their syntax and semantics are not fully defined in the OMG standard’s documentation impedes the realization of a WfMS using workflow definitions modeled as UML activity diagrams.

Several research prototypes for workflow management systems use their own modeling languages for workflow definitions. One example is the formal workflow model ADEPT (Application Development Based on Encapsulated Premodeled Process Templates) for the WfMS of the same name [RD98, Rei00]. Another prominent example is the workflow language YAWL (Yet Another Workflow Language) for the WfMS of the same name which is based on Petri nets [vdAtH05].

The Windows Workflow Foundation, which will be described in Section 3.4.4, provides a programming model comprising classes and interfaces for building workflow definitions either programmatically or declaratively. It does not provide a well-defined modeling language.

3.4.3 Workflow Management Systems

A *workflow management system* (WfMS) provides the actual software support for the execution of workflows. It is commonly understood as ä system that completely defines, manages and executes ’workflows’ through the execution of software whose order of execution is driven by a computer representation of the workflow logic"[Wor95].

The prerequisite for workflow execution in a WfMS is a machine-readable workflow definition. From this definition, workflow instances can be instantiated and executed. This is done by a *workflow engine* which is the central component of a WfMS. The workflow reference model of the Workflow Management Coalition defines the components and interfaces of a WfMS which are depicted in Figure 3.9.

Process definition tools are used to create workflow definitions in a machine-readable form and store them in the repository of the WfMS. The *workflow enactment service* comprises one or multiple workflow engines each of which maintains several workflow instances. The human resources assigned to tasks in a workflow instance interact with the WfMS by means of *workflow client applications*. This can be for example a work list handler which presents the assigned tasks to the user and allows for state changes to the assigned tasks. For the execution of tasks, resources may require certain applications like a text editor, an email client or an input form, which are invoked via the workflow management system. A WfMS may communicate with other workflow enactment services to realize *business-to-business integration*. In this case, information is exchanged between different organizations and the control flow of the process is managed by communicating workflow management systems.

A distinguishing feature of a workflow management system is whether it allows for dynamic structural changes to a workflow definition at runtime. Dynamic changes include the addition and removal of tasks and control structures. The realization of such functionality involves consistency checks which guarantee that the modified workflow definition is still executable and in particular that running workflow instan-

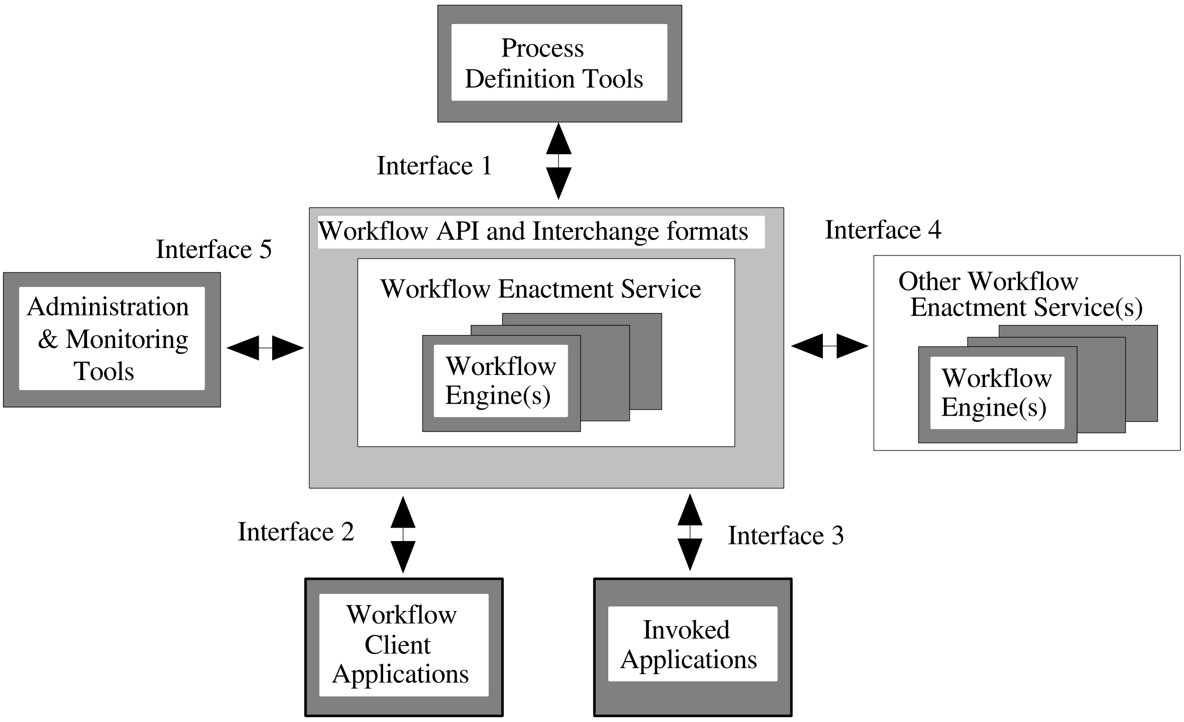


Abbildung 3.9: Workflow Reference Model - Components & Interfaces [Wor95].

ces can continue execution according to the modified definition [Wör10, RD98]. If a copy of the workflow definition is stored for every workflow instance in the WfMS, the definition of an individual instance can be modified without affecting the other instances. If, however, there is only one workflow definition and only information about the execution state is stored for the instances in the WfMS, then a change to the shared definition affects all instances, which therefore have to be migrated to the new definition [Rin04]. Migration may also be necessary for domain specific reasons when open cases have to be continued according to new regulations.

Finally, dynamic changes of running workflow instances must also adhere to compliance constraints, i.e. domain-specific business rules which define on a higher level of abstraction whether a workflow is correct with respect to company regulations, applicable laws, and the like. This compliance is ensured by a workflow management system which has been developed in the transfer project T6 in cooperation with Generali Deutschland Informatik Services [HNWH08, WEH08, WKH08b, WKH08a, Wör10].

3.4.4 The Windows Workflow Foundation

In this thesis, a workflow management system has been implemented as part of the prototype PROCEED. This WfMS is based on a framework for the development of workflow applications which is called the *Windows Workflow Foundation* (WF) [Mic10b, SS06, Buk08].

The WF provides a new programming model for developing workflow-oriented applications. As such, it is not a WfMS and it is not necessarily used to develop workflow management systems. The intended and most common use of the WF is the development of business applications in which the workflow for using the respective application is explicitly represented in its source code. The WF aims at simplifying the development of these applications by providing types, classes and interfaces for the definition of workflows, as well as runtime services which provide functionality like persistence and tracking for the applications. Furthermore, the integrated development environment for .NET applications, Visual Studio, has been extended by a graphical designer for workflow definitions.

In the WF, two different types of workflow definitions are distinguished: the *sequential workflow* and the *state machine workflow*. A sequential workflow definition describes the control flow between activities and is therefore comparable to the common approaches to workflow modeling described in Section 3.4.2. A state machine workflow is essentially a finite state machine as the name indicates. In the context of this thesis, sequential workflows have been used for process automation.

A sequential workflow in WF is a hierarchical composition of .NET classes. Two general types of activities are distinguished: *Composite activities* and *atomic activities*. Composite activities define the control flow of the workflow. Atomic activities can invoke application specific code. The composition hierarchy is a tree structure which implicates that WF workflow definitions are always block structured. The WF provides a class library which contains numerous predefined activity classes. This includes the composite activities `IfElseActivity` for alternative branching, `WhileActivity` for iteration, as well as `SequenceActivity`, `ParallelActivity` and `ReplicatorActivity`. The latter creates and executes multiple instances of a single child activity at runtime. Examples for available atomic activities are the `DelayActivity`, and the `CodeActivity` to execute application specific code.

It is furthermore possible to write custom activity classes in a .NET programming language. This includes composite activities for the realization of special control flow relationships as well as custom atomic activities which can provide special services for an application. As a consequence of this flexibility, there is no fixed, well-defined modeling language for workflow definitions in the WF. In this thesis, the available activity types for workflow modeling have been restricted to a fixed subset of the available activities to enable the integration of WF workflows with dynamic task nets. Only the predefined composite activities `IfElseActivity`, `WhileActivity`, `SequenceActivity`, and `ParallelActivity` may be used to model the control flow of a workflow. A custom atomic activity has been implemented to synchronize with human tasks in PROCEED (cf. Section 6.3). In this thesis, the term activity is reserved for activities in WF workflow definitions, although it is often used synonymously with the term task. In general, the chunks of work specified by a process model are called tasks.

If an activity in a running workflow instance is closed due to an error, it may be necessary to undo previous results of the workflow. In this case, compensating activities can be started. Therefore, so-called compensation handlers can be defined

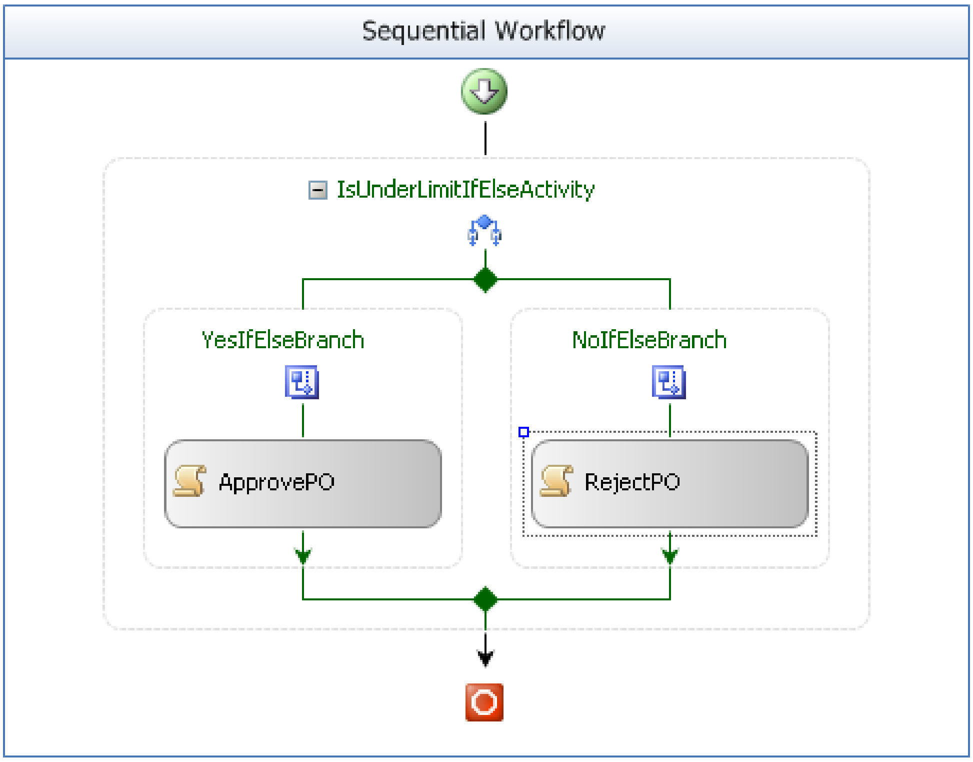


Abbildung 3.10: Graphical representation of a WF workflow [Mic10b].

for all activities in a workflow definition. *Compensation* is a commonly adopted concept in workflow management systems, since the defined flow of work often has the characteristics of a database transaction. Either the workflow is successfully terminated and the results are committed or no changes are made to the database and all intermediate results are undone. This is a reasonable approach for person-to-application and business-to-business processes. However, it is not suitable if workflows are applied to support parts of a complex development process in which intermediate results may be corrected but are never discarded.

A WF workflow definition can either be created programmatically by means of a .NET programming language or declaratively in a WF-specific XML dialect. In the latter case, the composition of activity classes is defined in XML while custom activity classes are still defined in a .NET programming language. A workflow definition is represented graphically in the Visual Studio development environment as depicted in Figure 3.10.

Figure 3.11 shows the component categories of the Windows Workflow Foundation. The WF Class Libraries include the WF programming model and the predefined activity types. The WF Runtime Engine provides an execution environment for workflow instances. It is not a self-contained application. Instead, an instance of the `WorkflowRuntime` class must be hosted by the developed application.

The WF Runtime Services provide additional functionality which may be required for the developed workflow-oriented application. There are core services for which a standard implementation exists like the persistence and the tracking service. It is

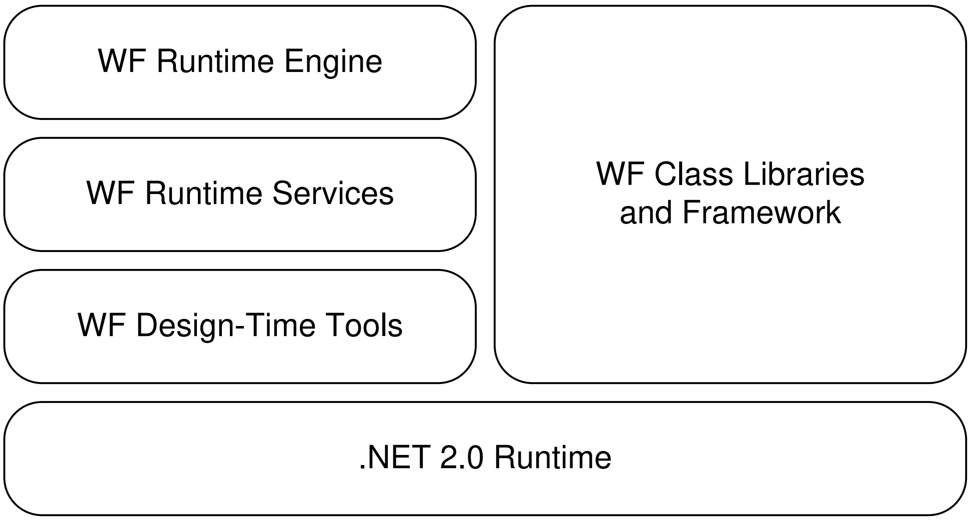


Abbildung 3.11: WF component categories [Buk08, p. 32].

also possible to provide custom implementations for these services. Furthermore, local services can be developed depending on the requirements of the application.

The *WF Design-Time Tools* are used to graphically edit workflow definitions. Figure 3.10 shows the graphical representation of a WF workflow in the design view which is integrated into the Visual Studio development environment. The design time tools can be hosted in any other .NET application and have been used to realize the workflow designer and monitor of the PROCEED prototype.

Dynamic changes to running workflow instances As it has been pointed out in Section 3.4.3, a distinguishing feature of a workflow application is whether it allows for dynamic changes to a workflow definition at runtime. The WF allows for such changes via API method calls. Change operations can be applied to the definition of a workflow instance programmatically. Every workflow instance carries a copy of the workflow definition so that the changes affect only the one instance. All other instances of the same workflow type—whether they are currently executing or instantiated in the future—use their own copies of the original workflow definition. The available change operations include adding and removing an activity, replacing an activity by another one, and changing conditions for alternative branching and loops.

For technical reasons, several restrictions apply for changing a workflow at runtime [SS06]. An executing activity cannot be removed from a workflow instance, and its properties cannot be modified. Put another way, only the following dynamic changes are possible.

- Changing property values of not yet started activities,
- Removing a not yet started activity,

- Inserting a new activity into a not yet terminated activity.

The last two changes implicate that it is possible to move a not yet started activity from a not yet terminated activity to a not yet terminated activity.

Kapitel 4

Previous Achievements

The management of complex development processes nowadays needs adequate software tool support. A software tool can enable the process participants to keep track of the various interdependencies between the task, resources and products in a development process. Furthermore, it can guarantee that the management data is in a consistent state at any time and that all process regulations are satisfied. Examples are the correct sequencing of tasks, the unique assignment of product revisions to tasks, the mandatory release of a document before the completion of a task, and the correct assignment of a task to a resource in accordance with role restrictions.

The management system *AHEAD* (*A*daptable and *H*uman-Centered *E*nvironment for the M*A*nagement of *D*esign Processes) has been developed at the Department of Computer Science 3 at RWTH Aachen University with the goal to support dynamic development processes [JSW00, Jäg00, NWS03, FLW03, SW03, WSJH03, Wes01, Wes99c, Wes99b, JKN^{+}99, KW00, HKW97, HKNW99, WHH07, HJK^{+}08]. AHEAD is based on a formal meta-model defined in the executable specification language PROGRES (PROgrammed Graph REwriting System) [SWZ99]. The formal specification removes any potential for ambiguities and misinterpretation. Furthermore, the usage of an executable specification language allowed to generate the application logic of the management system from the specification. The user interface of AHEAD was realized by means of the UPGRADE framework [BJSW02a] which is commonly used in combination with the PROGRES environment.

The *meta-model* which defines the internal data structures to represent development processes on instance level in the AHEAD system comprises three integrated submodels. The DYNAMITE (DYNAMIc Task nEts) meta-model enables the modeling and execution of so-called *dynamic task nets* [Kra98, NWS03, HJK^{+}08]. It is complemented by ResMod (Resource Modeling) [KW00] and CoMa (Configuration Management) [Wes95, Wes96, Wes99c], which are meta-models for *resource management* and *product management*, respectively. The three partial meta-models are tightly integrated: Tasks in a dynamic task net are assigned to resources, and product revisions are associated with tasks. Thereby, the tasks in a dynamic task net establish a connection between the entities defined in ResMod and CoMa. Figure 4.1 shows an example management configuration in the AHEAD system. The common visualization of a management configuration displays the resources in the left column, activities in the upper part of the right column and products below. The

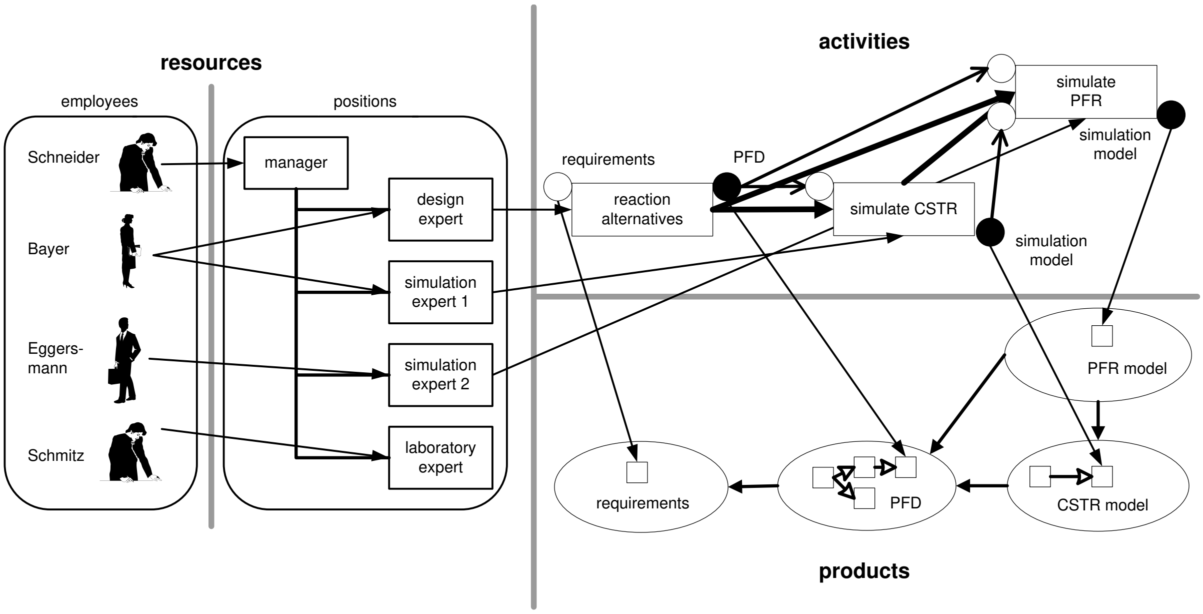


Abbildung 4.1: Example for a management configuration in AHEAD [HJK+08].

arrows visualize the mutual relationships between tasks, resources, and products. In the following three sections, the three partial meta-models are reviewed. Afterwards, advanced capabilities of the AHEAD system for the management of development processes are reviewed and the AHEAD prototype is described.

4.1 RESMOD

In the AHEAD system both, *human resources* and *non-human resources* can be modeled. Human resources are the people in an organization or a project team. Non-human resources are, e.g., computer workstations, software tools, or machines. The same concepts are applied for modeling human and non-human resources which are subsumed under the general term *resource*. Two types of resources are distinguished: *planned resources* and *actual resources* [KW00]. Actual resources can be allocated for planned resources which define positions. Furthermore, a distinction is made between *base resources* in an organization and *project resources*. In the case of human resources, planned project resources are the positions in a project team while actual project resources are the actual team members who fill the positions. Actual base resources are the people belonging to an organization, e.g. employees of a company. Planned base resources are positions in the company to which the employees have been appointed. The human planned base resources are usually structured in organizational units like departments or company branches. In Figure 4.1, only actual base resources and planned project resources are depicted.

There are similarities between the concepts of a planned resource and a *role*. In the example of Figure 4.1, the resource Schmitz plays the role laboratory expert in the project team. Like in role-based models for process management [Rup97, zM99], an actual resource is only indirectly assigned to a task. A planned resource is assigned to a task which can later be executed by the actual resource which fills the position. The difference between roles and planned resources is that the latter define positions in the project team. This becomes obvious by the two positions simulation expert 1 and simulation expert 2 which both require a resource to be a simulation expert to fill the position. The mapping of actual to planned resources is always a 1:1 mapping. In the case of roles, there would be only one role of simulation expert with two resources holding the role. If the role was assigned to a task, it would not be clearly defined which actual resource shall execute the task. In RESMOD, a *class of planned resources* is the equivalent of a role. It defines the type of planned resources and abstracts from individual positions.

Beyond the simple case presented in Figure 4.1, actual and planned resources can be complex, i.e. built up from several parts. A complex actual resource can fill the position of a complex planned resource, if its structure and its parts match the configuration of the planned resource. Complex resources are primarily used to model non-human resources like computer workstations consisting of several parts. However, complex resources have also been used to model project teams of human resources. A detailed description and examples for complex resources can be found in [KW00].

4.2 COMA

The partial meta-model CoMa for product management allows to model *products*, their dependencies, the different *variants* and *revisions* of products, as well as product *configurations* combining specific *versions* of different products into a coherent whole [Wes95, Wes96, Wes99c]. The term product does not merely refer to the final product of a development process and its parts. Instead, every artifact which is produced in the course of the development process is called a product, which includes documents, engineering data in a database, and source code fragments in case of a software development process.

In Figure 4.1, products are represented by ellipses containing their version trees. Product versions are represented by small boxes which are connected with the preceding and succeeding versions. A directed edge between two product ellipses indicates that the source of the edge depends on the target. The products are associated with input and output parameters of tasks in the dynamic task net which represents the development process. Product revisions are associated with the output parameters of those tasks in which they have been created. In this way, the management system AHEAD keeps track of which products are required for the tasks, which task creates or modifies a certain product, and which product version has been produced in the course of which task.

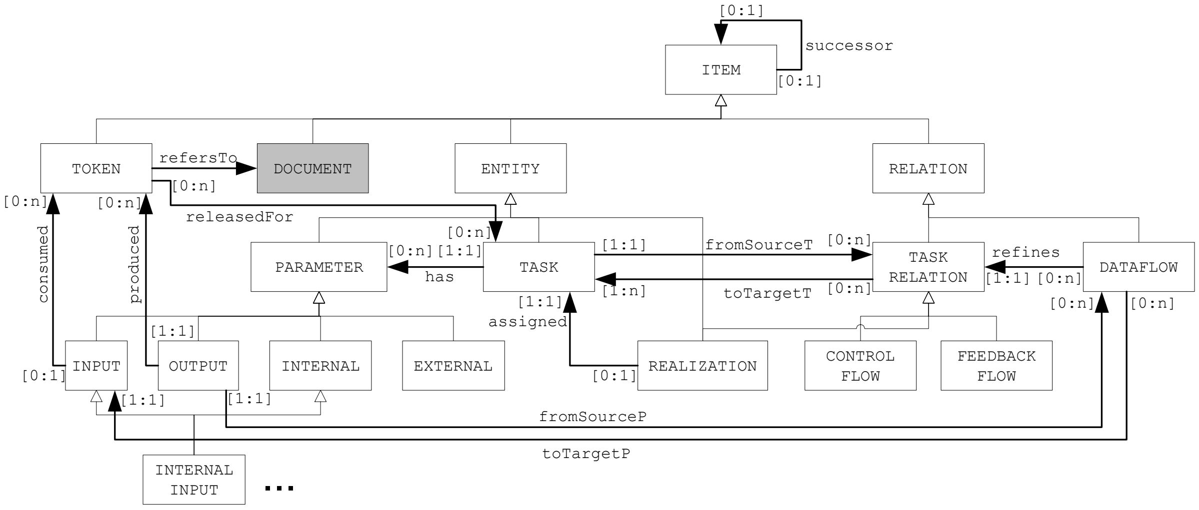


Abbildung 4.2: Graph schema for dynamic task nets [Sch02].

4.3 DYNAMITE

The DYNAMITE meta-model for dynamic task nets connects the partial models for resource and product management. A *dynamic task net* represents an enacted process model instance of a development process. Dynamic task nets can be compared to network diagrams which are commonly used for project planning. Just like in project plans, tasks are connected by control flows with different semantics, and human resources are assigned to tasks. However, in contrast to network diagrams in project management, the data flow between tasks is covered by a dynamic task net as well. Furthermore, the tasks in a dynamic task net have execution states, and the produced revisions of documents are represented as tokens in the task net. Therefore, a dynamic task net does not merely represent a project plan but also the enactment state of a development process. The specification of the DYNAMITE meta-model is divided into a *structural part* and a *behavioral part* [Sch02] which will be described in the following two sections.

4.3.1 Structural Model

In the structural part, the *entities* and *relationships* for modeling dynamic task nets are defined in a graph schema. Furthermore, *structural invariants* are defined which impose semantical constraints on a dynamic task net. Finally, *structural manipulation operations* are specified which can be applied to modify a dynamic task net.

Figure 4.2 shows the graph schema for dynamic task nets. It defines the node types of a graph which represents a dynamic task net. The `successor` relation of the most general node type `ITEM` is used for the versioning of all elements in a dynamic task net. By means of this relation, the history of changes to a task net can be traced. The most important entity is the `TASK`. Two tasks can be connected by a task relation which can be a `CONTROLFLOW` or a `FEEDBACKFLOW` relation. A task can have several input and output parameters, whereby internal and external parameters are distinguished. Input and output parameters of different tasks can be connected by `DATAFLOW` relationships. A document revision is represented in DYNAMITE by a `TOKEN` which refers to the corresponding document and the output parameter of the task in which it has been produced. The `REALIZATION` of a task contains its subtasks and their dependencies. It defines, how the task should be executed.

The structural invariants defined in the DYNAMITE meta-model impose additional semantical constraints on dynamic task nets which cannot be defined in the graph schema. The following list contains the informal description of the structural invariants.

- The name of a task has to be unique in the dynamic task net.
- The hierarchy of tasks within a task net builds a tree structure.
- The control flow relationships between tasks must not form a cycle.
- If two tasks from different subnets (realizations) are connected by a control flow, their respective parent tasks must be connected by a control flow with the same orientation.
- There must exist a control flow path from a feedback flow’s target to its source.
- A data flow may only be created if it refines a task relationship, i.e. the tasks of the connected parameters have to be connected by a control or feedback flow with the same orientation.
- A token may only be released to a task that owns an input parameter which is connected by a data flow to an output parameter of the producing task.
- A token can only be read via an input parameter, if it has been released for the task owning that parameter.

These invariants have been formally defined in [Sch02]. For example, the invariant which prohibits control flow cycles is formally defined as follows.

$$\forall \mathtt{t}_1, \mathtt{t}_2 \in \mathtt{Tasks}(\mathtt{t}_1 \in \{\mathtt{t2.Source}\} \rightarrow \mathtt{t}_2 \notin \{\mathtt{t}_1\mathtt{.Source}\})$$

The property $\mathtt{t.Source} \subset \mathtt{Tasks}$ returns the set of all transitive predecessors of a task $\mathtt{t} \in \mathtt{Tasks}$ with respect to the control flow relationship.

In Figure 4.3 an example of a DYNAMITE task net is depicted. This example has been taken from a larger scenario from the domain of chemical engineering, which is described in [NM08]. Tasks are represented by rectangular boxes. The displayed task net is *hierarchically structured*. The hierarchy is indicated by means of the layout. The subtasks of Design Reaction are arranged below the task itself. Input and output parameters are represented as white and black circles respectively. Not all entities defined in the graph schema of Figure 4.2 are represented by nodes

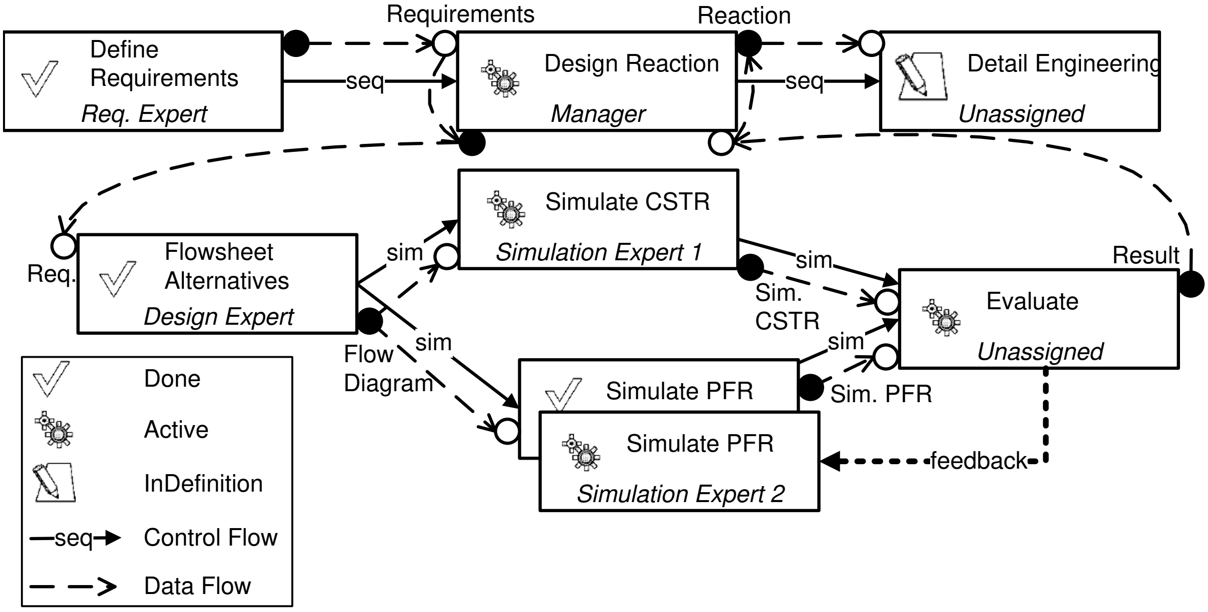


Abbildung 4.3: Example for a dynamic task net.

in the graphical representation of a dynamic task net. The realization of a task is not displayed at all while task relations and data flows are represented by edges between tasks and parameters respectively. The distinction between internal and external parameters allows to model data flow from a parent task to a subtask and vice versa. An internal parameter of a task can be connected with external parameters of its subtasks. If a document which is produced in a subtask shall be available at the parent task, then an external output parameter has to be defined for the subtask which is connected with the internal input parameter of the parent task for this document. This is the case for the document Result in Figure 4.3. Vice versa, the input parameter for the document Requirements of the subtask Flowsheet Alternatives is connected to an internal output parameter of the task Design Reaction, so that it can be used in one of its subtasks.

The structural manipulation operations for dynamic task nets have been specified in the graph rewriting language PROGRES. A representative collection is presented in [Sch02]. Structural manipulation operations may not violate the semantical constraints imposed on dynamic task nets. Consequently, if a task net fulfills all structural invariants then the invariants are still fulfilled after a manipulation operation has been applied to the task net.

4.3.2 Behavioral Model

The behavioral part of the DYNAMITE meta-model is concerned with the execution states of tasks in a dynamic task net. The life cycle of a task is defined by the state transition diagram depicted in Figure 4.4. In the state InDefinition, the task is

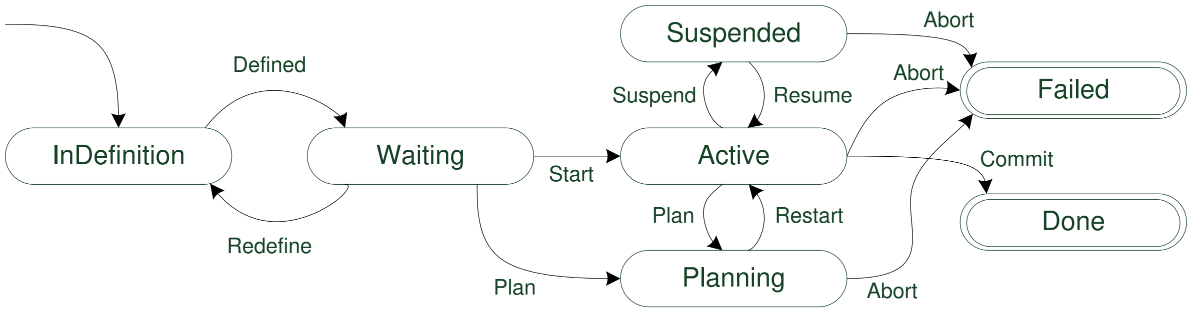


Abbildung 4.4: State transition diagram [Kra98].

prepared for execution. As soon as all relevant properties are set, the task's execution state can be changed to Waiting, i.e. it can now be started by the assigned resource. The state Active indicates, that the assigned resource is currently performing the work defined by the task. A task can be temporarily suspended. If changes to the task or its realization have to be made, the state of the task has to be changed to Planning first. If a task is successfully finished it reaches the final state Done. If it has to be aborted, the final state Failed is reached.

The definition of a single tasks's life cycle is not sufficient to define the execution semantics of a complete task net. The possible execution state transitions of a task also depend on the execution states of all tasks, which are connected to the task by control or feedback flows. In DYNAMITE, a *control flow* connects exactly two tasks and it can have one out of three different semantics.

- A *standard* control flow defines that the target task of the control flow must not be terminated before the source. This is the minimal requirement for a control flow. However, the source and target of a standard control flow can be executed in parallel, and the target can even be started before the source.
- A *simultaneous* control flow defines the additional restriction, that the source task must be started before the target. The motivation for this control flow semantics is that the resources of the source and target tasks cooperate, and that a first intermediate result of the source task is required to start working on the target task. At the same time the constraint with respect to the termination of the tasks guarantees that the last version of the source's output is consumed by the target task before it is terminated.
- The most restrictive control flow is that of type *sequential*. In this case, the target task may only become active after the source task has been terminated.

In DYNAMITE, the concept of a *feedback flow* addresses specific dynamic situations in development processes. Although the model of a development process usually defines dependent tasks and hence an *order of execution*, the actual execution of these tasks in a project is *carried out iteratively*. In case of a change request or a detected error in a product, it may be necessary to *redo previous process steps*. To

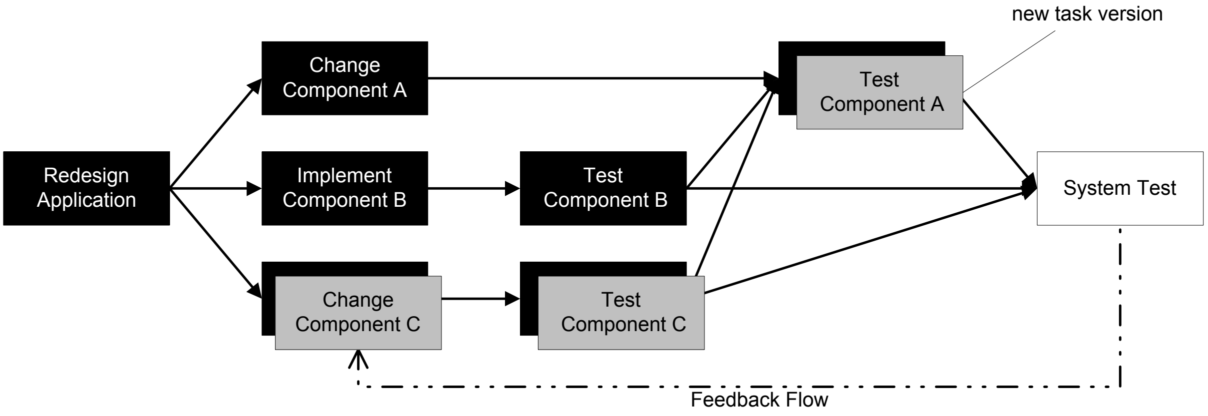


Abbildung 4.5: Feedback flows and task versioning in dynamic task nets [Sch02].

trace cases, in which feedback from a succeeding task is given to a previous task, a feedback flow is introduced. A feedback flow is initially active and is deactivated when the requested changes have been performed. As long as a task has an incoming active feedback flow, it cannot be successfully terminated. If a task has already been terminated and therefore has to be restarted to handle feedback from succeeding tasks, a *new task version* is created for this and succeeding terminated tasks. This case is depicted in Figure 4.3 where a feedback flow is defined from the task Evaluate to Simulate PFR for which a new version has been created. Figure 4.5 shows an example from the domain of software engineering where a feedback flow requires the versioning of several subsequent tasks. The use of feedback flows and versioned tasks is a possibility to redo already terminated process parts during enactment while ensuring *traceability* at the same time.

Besides control flows and feedback flows, also the position of a task in the hierarchy of a dynamic task net constrains it possible execution states. A task may not be started before its parent task, and it can only be committed after all of its subtasks have been terminated. If a task is suspended or aborted, all subtasks have to be suspended or aborted as well, respectively.

Altogether, the allowed state transitions of a task are restricted by the state transition diagram defining the life cycle of a task, by the execution states of all tasks which are connected with the task by control flows or active feedback flows, and by the execution states of its parent task and its subtasks. These constraints have been defined in the form of behavioral invariants in [Sch02] which are informally presented in the following list.

- Tasks that have never been activated cannot have a terminated parent task and the subtasks of a preparing task must also be preparing.
- A parent task must be activated before its subtasks.
- If a task is suspended, its parent task must be either active or suspended.

- Subtasks of failed tasks may not be running.
- The subtasks and predecessors of successfully terminated tasks have to be terminated as well. A task may not be successfully terminated if it is the source of an active feedback flow.

The formal definition of the first invariant taken from [Sch02] is provided here as an example where the property `State` returns the execution state of the given task, the property `Parent` the parent task, and the property `Children` all subtasks. The set $\texttt{Preparing}\{\texttt{InDefinition}, \texttt{Waiting}\}$ is a subset of all possible execution states of a task.

$$\begin{aligned}\texttt{Task.State} \in \texttt{Preparing} \rightarrow (\neg(\texttt{Task.Parent.State} = \texttt{Done}) \wedge \\ (\forall \texttt{t} \in \texttt{Task.Children}(\texttt{t.State} \in \texttt{Preparing})))\end{aligned}$$

State change operations may not lead to a violation of behavioral invariants. Hence, the defined behavioral invariants have several implications for state change operations in a dynamic task net. For example, the fourth constraint implicitly demands that upon the abortion of a task all active subtasks have to be aborted as well. The AHEAD system performs these required adaptations of a dynamic task net automatically when certain state change operations are performed. In this way, the behavioral consistency of dynamic task nets is ensured at any time.

4.3.3 Comparison With Other Paradigms

Dynamic task nets combine the advantages of different paradigms for process instance modeling and are tailored for the management of development processes in large and complex development projects.

Comparison with project plans As mentioned earlier, a dynamic task net can be considered as an enhanced project network diagram. Networks diagrams are commonly used for project planning. Tasks and their dependencies are defined, critical path analysis is performed, resources are assigned to tasks, and the tasks are scheduled. A project plan may be adapted at project runtime to reflect the actual performance of the project. However, a project plan is not executable, i.e. the current status of the planned tasks are not reflected in the network diagram. This makes progress measurement difficult. In contrast, a dynamic task net is executable. A project is modeled as a process model instance. At any point in time, the task net reflects the current plan and at the same time the current enactment state of the process.

The Precedence Diagramming Method (PDM) which has been introduced in Section 3.2 defines four different types of precedence constraints which can be defined between two tasks in a network diagram: *start-end*, *start-start*, *end-end* and *end-start*. The control flow semantics defined in DYNAMITE can be mapped to the precedence constraints of the PDM as shown in Table 4.1. The task dependencies

DYNAMITE control flow	PDM precedence constraint
standard	end-end
simultaneous	start-start + end-end
sequential	end-start
	start-start
	start-end

Tabelle 4.1: DYNAMITE control flows versus PDM precedence constraints.

start-start and start-end have no counterpart in DYNAMITE. During the development of the DYNAMITE meta-model it was found, that they are not suitable for the management of development processes because a control flow should at least have the semantics of an end-end task dependency. The control flow type simultaneous combines the start-start and end-end precedence constraints. The introduction of a distinct control flow type for this combination is in line with [Haj97].

Since a dynamic task net is hierarchically structured and at the same time defines control flows, it combines the main characteristics of a work breakdown structure and an activity network (cf. Section 3.1). It is common practice in project planning to structure network diagrams hierarchically. When a unique identification number is assigned to every task in a hierarchical network diagram, then it is possible to derive the work breakdown structure from the network diagram automatically, and a separate work breakdown structure is dispensable [Bur00, p.145]. This is the case for dynamic task nets.

Project plans usually do not incorporate a notion of data flow. No parameters and data flow relationships can be specified which determine how documents shall be transferred between the tasks. Since a project plan cannot be enacted, the actual data flow [Joe97], i.e. the created and released revisions of documents, is also not represented in a project plan. In DYNAMITE, the data flow relationships between the tasks of a dynamic task net are explicitly modeled and so are the different document versions which are produced, released and consumed by the corresponding tasks. This is an important aspect for the management of development processes which are strongly data-driven. Process model definitions including workflow definitions commonly define data flows between tasks. However, workflow management systems are often limited with respect to modeling the actual data flow in workflow instances because different versions of documents cannot be explicitly represented.

Comparison with workflows Workflow management is a popular approach for process management (cf. Section 3.4). A workflow definition is instantiated several times and the instances are executed according to the definition. Dynamic task nets represent process model instances and therefore have to be compared with workflow instances but not with workflow definitions. For this reason, control structures for alternative courses of action and loops which are common for workflow definition languages are not defined in DYNAMITE.

A major drawback of workflow management systems with respect to their applicability for development processes is that the semantics of control flows is limited to the end-start precedence constraint. As argued before, modeling standard and simultaneous control flows is essential for the adequate modeling of cooperation scenarios in development processes where dependent tasks may be executed in parallel. The restriction to end-start precedence constraints is due to the fact, that the other precedence constraints start-start, start-end and end-end would render process automation impossible. In case of the end-start precedence constraint, the succeeding task can be automatically started when the end-event of the predecessor occurs. This is done by virtually every workflow management system. The constraint does not actually demand that the two events coincide but only that the start of the successor happens after the end of the predecessor. However, the additional assumption is made to enable automation. For the start-start precedence constraint the assumption would not be equally reasonable. For the start-end and end-end precedence constraints it would even be impossible to determine a start time for the automatic start of the successor. As a consequence, a dynamic task net with standard and simultaneous control flows cannot be automatically enacted.

A common form of dynamics in development processes is the unanticipated redo of already finished process parts due to errors found in the design of the product or due to changed requirements. These cases are supported in dynamic task nets by feedback flow relationships and new task versions as described earlier. In workflow management systems, the repetition of process parts is realized by loop control structures in the workflow definition. However, this only covers iterations which can be anticipated before workflow runtime. Repeating finished process parts at workflow runtime for which no loop construct has been defined requires dynamic changes to the workflow definition.

A dynamic task net can be arbitrarily modified at process runtime as long as the defined invariants hold after the modification. Tasks and control flows can be added, changed, or removed. Even if a process model definition restricts the allowed changes to a task net, there are significant degrees of freedom for modifications, and it is even possible to deviate from the process model definition (cf. Section 4.4). It is considerably more difficult to modify a workflow instance which is executed according to a workflow definition. This type of problem gave rise to a whole field of research on flexible process aware information systems [Rei00, RD98, WEH08, Wör10, MS03, Wes99a, VW98, Cas98, CCPP99]. The common approach is to dynamically change the workflow definition at runtime and to continue the enactment of the workflow instance according to the modified definition. This is not required for dynamic task nets where a process model instance can be modified independently of the corresponding process model definition. Finally, most commercial workflow management systems to date do not allow any deviations of a workflow instance from the defined workflow definition at all.

The allowed state changes of tasks in a workflow instance are restricted by the control structures defined in the corresponding workflow definition. A WfMS ensures the consistency of the workflow instance enactment state with the workflow definition. If a WfMS allows dynamic structural changes to the definition of workflow instances at runtime, the correctness of these changes has to be ensured. This can be achieved by defining process knowledge on a more abstract level and ensuring the compliance of workflow definitions to this process knowledge [Wör10]. Research on compliance of workflow definition has been carried out at the Department of Computer Science 3 of RWTH Aachen University [WKH08a, WKH08b, Wör10, WH11] and by other research groups [DAV05, MA07, BGS07, VA00, VBA01, Ver04]. In case of DYNAMITE, the behavioral and structural invariants can be regarded as general process knowledge to which every process model instance has to comply.

Applicability for agile processes For agile processes, which have recently become increasingly popular in the software engineering domain, the usage of a process management system like AHEAD would constitute a significant management overhead. The explicit modeling of tasks, control flows, data flows, and even feedback cases is incompatible to the agile approach where project planning at the level of individual tasks is avoided. Therefore, the AHEAD system is not suitable for projects which are performed according to the agile process management approach. However, several assumptions underly the agile software development approach which imply that it is not applicable for all development processes [TFR05]: all developers are able to have frequent, intensive personal communication with each other, are all equally skilled and informed about the product, and have immediate access to the customer. As a consequence, the agile approach is not applicable for distributed development, subcontracting, large project teams, and the development of large, complex systems [TFR05]. In these cases, software tool support as provided by the AHEAD system is required to keep track of all dependencies between tasks, resources, and the various artifacts which are created in a development project. Plant design projects are a prominent example of design projects with large project teams, and a highly complex product.

Conclusion Concluding, it can be stated that the DYNAMITE meta-model is adequate for modeling process model instances of development processes. It combines the advantages of project plans and workflow instances, namely flexibility and executability. Compared to the workflow approach, a drawback of the DYNAMITE meta-model is that it does not allow the automation of the defined processes. This drawback is addressed in this thesis by an integration of dynamic task nets with workflow instances to enable the automation of subprocesses. Compared to project planning, a drawback of the AHEAD system is that it does not support the scheduling and quantitative progress measurement of tasks. Here lie the main contributions of this thesis to the AHEAD approach.

4.4 Process Model Definitions and Evolution

Dynamic task nets represent process model instances. For effective process management, process model definitions are required which describe the commonalities of

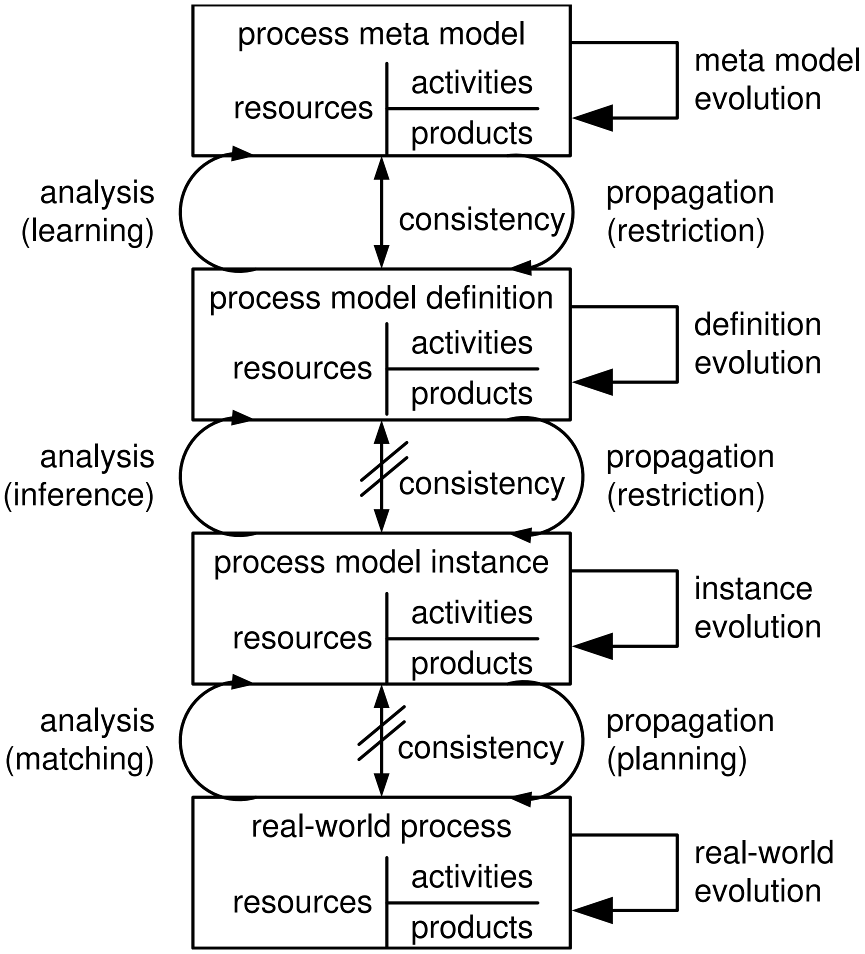


Abbildung 4.6: Conceptual framework for evolutionary process management [Sch02].

all process instances of a certain process type. This enables process improvement over a series of process instances. Knowledge gained during the enactment of a process model instance can be incorporated into the process model definition, and can be subsequently used for the following instances. Furthermore, process model definitions can define domain-specific knowledge about development processes in a certain domain, e.g. chemical engineering, mechanical engineering, or software engineering. This includes knowledge about the typical task types, control flows, roles, products, and data flow relationships.

In [Sch02], a conceptual framework for process management is presented which defines the requirements for continuous process improvement by means of a process management system. The concepts of this framework have been realized in the AHEAD prototype [Sch02, HJK$^+$08, HSW04b, HSW04a]. The proposed approach has been adopted by other research groups, e.g. in [GDMR04]. The framework proposes a four-layer view onto process management as it is depicted in Figure 4.6. The first layer (from bottom to top) reflects the real world process being performed by the process participants. On the second layer, a process model instance represents the real-world process which can be used to manage the process by means of a software tool. Reusable process knowledge is kept within the third layer in the form of a process model definition. A process model definition can be instantiated to create new process model instances. The syntax and semantics of process model definitions and instances are defined within the process meta-model layer which provides according modeling languages.

The layers are connected with each other in that analysis of the process models on one layer may lead to changes of the models on the next higher layer, and changes on a higher layer are propagated to the next lower level. Evolution of process models may take place on every layer.

- The real-world process evolves in the sense that the process participants perform their work, produce results, and make decisions. It is important for software tool support, that the process model instance on the second layer matches the real-world process as exactly as possible. Since it is impossible to maintain a perfect match at any time during the runtime of the process, inconsistencies may occur.
- The process model instance may evolve in two ways. Its enactment state is continuously adapted to match the performance of the real-world process. Furthermore, instance evolution may take place due to management decisions which involve structural changes to the process model instance. In this case, the changed plan has to be propagated to the real-world process, i.e. the process participants have to carry out the modified plan.
- Changes to a process model instance may not have been foreseen in the corresponding process model definition and may therefore result in inconsistencies. From several changed process model instances, new process knowledge can be inferred and incorporated into the corresponding process model definition.
- It may be necessary to change a process model definition due to changed regulations. The instances of a changed definition may have to be adapted to comply to the new restrictions.
- Finally, the modeling languages restrict the creation of process model definitions and instances. If different or new modeling elements are required to express certain process knowledge, the process meta-model has to be adapted which constitutes meta model evolution.

In the AHEAD system, dynamic task nets represent process model instances. For process model definitions, UML class diagrams are used. Every class in such a class diagram is marked with a UML stereotype to indicate its DYNAMITE entity type. The classes define domain-specific types of tasks and parameters. Figure 4.7 shows an example process model definition which defines the design process of Figure 4.3 on the type level. Just like in the graphical representation of dynamic task nets, input and output parameters are represented as white and black circles in the process model definition. The representation corresponds to the defined UML stereotypes.

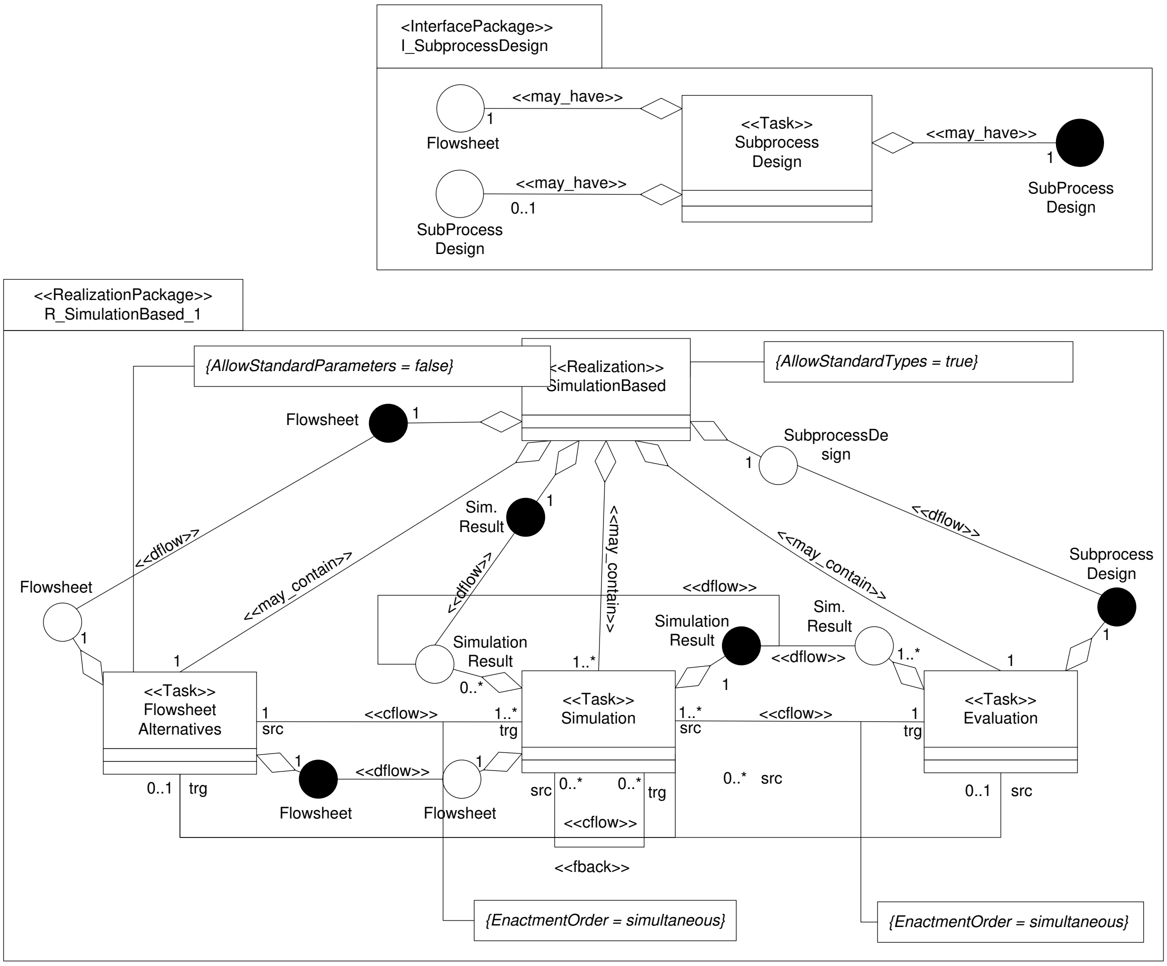


Abbildung 4.7: Class diagram for a design subprocess [HJK+08].

Tasks and realization nodes are marked in textual form with according stereotypes. The hierarchy of tasks and the association of parameters with tasks are modeled as composition associations. Control flows and data flows are modeled as associations between tasks and parameters respectively. Cardinalities are defined for associations to define for example how many instances of a certain task type should be contained in a process model instance or how many successors of a certain type a task may have. Finally, annotations to control flow associations define the required execution semantics. The UML profile which has been defined for process model definitions on type level can be considered as a graphical domain-specific language for process modeling. Design guidelines for domain-specific languages have been presented in [KKP+09].

The approach to process management in AHEAD is considered to be a wide spectrum approach [NM08]. A wide spectrum of processes is supported ranging from ad hoc processes to completely predefined processes. A dynamic task net can be built up from untyped tasks without any process model definition to represent an ad hoc process. If a process model definition is available which defines element and

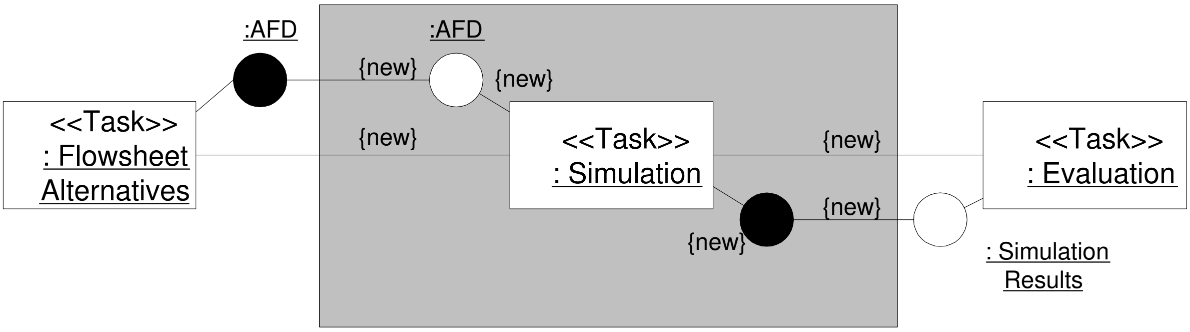


Abbildung 4.8: Instance pattern for partial realization definition [Sch02].

relation types and their cardinalities, a process model instance can be instantiated from this definition. The process may be enacted complying to all constraints imposed by the process model definition. In this case the instance and the definition are called *strongly consistent*. However, if tasks are inserted into the process model instance which were not defined in the definition, this results in inconsistencies. The process models are called *weakly consistent* if the introduced tasks are not typed and the process model definition allows the introduction of untyped tasks. Otherwise they are *inconsistent*. Inconsistency and (weak) consistency can refer to structural and behavioral constraints imposed by the process model definition.

Besides process models on the type level, the approach developed in [Sch02] also provides the concept of *instance patterns*. An instance pattern can be inserted in one step into a dynamic task net. If the instance pattern represents a whole subprocess, it is called an *instance-level process model definition*. For the definition of instance patterns, UML object diagrams are used in AHEAD for two reasons:

- Multiple objects of the same class can be modeled in an object diagram, which is not possible with class diagrams.
- The context in which the pattern may be inserted can be specified as well in an object diagram.

Figure 4.8 shows an example of an instance pattern which can be used to insert a simulation task into a dynamic task net and connect it at the same time with control and data flows to the unique successor and predecessor which have to be present in the task net for the pattern to be applicable. The UML constraint `{new}` indicates which elements are created.

4.5 Interorganizational Cooperation

Development processes are rarely carried out in isolation by one company only. A common scenario is that of a customer-contractor relationship in which the customer provides the requirements for the product to be developed, and the contractor has to fulfill these requirements. Well defined interfaces for the communication between customer and contractor contribute to the success of a development project. Furthermore, it is often the case, that the contractor delegates certain tasks to subcontractors. In this case, subprocesses of the overall development process are enacted by the subcontractors.

To support the various cooperation scenarios between different organizations, the AHEAD approach was extended by concepts for interorganizational cooperation. In a first step, a *delegation-based cooperation* model was developed which enables process modeling and enactment for contractor-subcontractor scenarios [Jäg02]. In a second step, this approach was elaborated and generalized to support a broader range of scenarios including the cooperation of partners with equal rights [Hel08a].

Although interorganizational cooperation is not in the focus of this thesis, the two approaches are shortly introduced here to complete the overview over the AHEAD system. Furthermore, a partial result of the work on the view-based cooperation approach, namely the coupling of AHEAD with workflow management systems, has been the starting point for the development of a similar integration approach which is presented in this thesis.

4.5.1 Delegation-based Cooperation

The implementation of the approach for delegation-based cooperation presented in [Jäg02, BJSW01, HJ04a, HJ04b, HJS$^+$04, HJK$^+$08] lead to a distributed AHEAD system. Figure 4.9 illustrates the concept behind this system. Several instances of AHEAD are coupled with each other in order to support the management of a distributed development process. Every instance provides views for a project manager and the engineers or developers in a development project. Subprocesses of a development process can be delegated to subcontractors, and the enactment of these subprocesses can be monitored by the contractor.

A delegated subprocess may consist of a connected set of subtasks, i.e. it is not confined to a single task. This allows the contractor to define milestones for controlling the work of the subcontractor. The delegated subprocess serves as a contract between contractor and subcontractor. The contractor is obliged to provide the required inputs, based on which the subcontractor has to deliver the outputs fixed in the contract. The subcontractor may refine the delegated subprocess. Such refinements, however, are not visible to the contractor since they are not part of the contract. Finally, a delegation contract may be changed at process runtime according to a change protocol which assures that both parties agree on the change. In this way, dynamic process changes are supported even for interorganizational processes.

4.5.2 View-Based Cooperation

The delegation-based cooperation approach was generalized to support a broader spectrum of cooperation scenarios [Hel08a, HW06b, HW06a, HW07, HJK$^+$08]. This

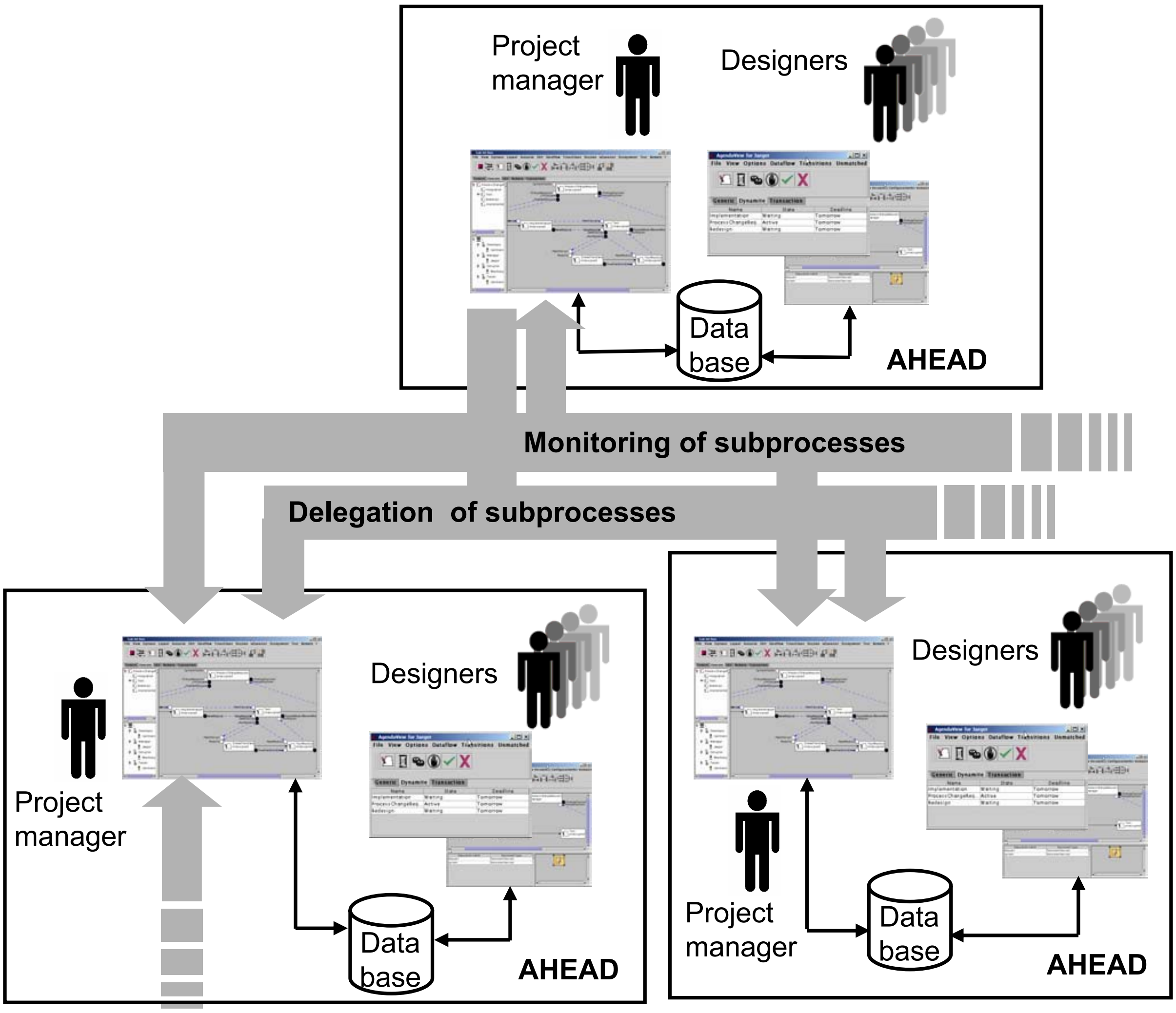


Abbildung 4.9: Distributed AHEAD system [HJK+08]

resulted in the view-based approach for interorganizational cooperation. The main drawbacks of the delegation-based approach which were addressed by the view-based approach are the following:

- It was only possible to expose process parts to another organization by delegating these parts to the organization. This excluded the scenario of monitoring subprocesses without delegation.
- Only delegation from contractor to subcontractor was supported. The cooperation of partner organizations with equal rights was not possible.
- The delegation of several disconnected subprocesses required several delegation contracts even if the subcontractor was the same organization. It was not possible to combine disconnected subprocesses in one delegation.
- The cooperation protocols could not be tailored to specific scenarios with different

levels of trust. In a case where only informal guidelines were required for the cooperation, the delegation still required strict rules.

From these drawbacks, the main requirements for the improved approach were derived as (1) flexible mechanisms for defining the visibility of process information shared with other organizations and (2) support of customizable cooperation relationships.

The fundamental concept which was introduced to meet these requirements was the *dynamic process view*. A dynamic process view is published by one organization and can be subscribed by another organization. It includes a subgraph of a dynamic task net where certain elements can be omitted, a *view product workspace* maintaining all view-related products and product versions, and a *view resource space* containing all view-related resources. Furthermore, a set of *view definition rules* defines which elements of the underlying private process are part of the process view.

Several *cooperation phases* have been defined which structure the whole process of interorganizational cooperation, from the private task net planning to the completion of process inter-connection. Finally, a *cooperation model* defines three layers for interoganizational cooperation: the *private process layer* on which the private processes of the organizations are defined, the *process view layer* on which the published process views reside, and the *cooperation layer* on which different kinds of cooperation relationships between organizations can be defined.

Integration of workflow processes in a development process The view-based cooperation model first of all addressed the interorganizational integration of development processes carried out in different organizations. This was complemented by an approach for the *intraorganizational integration of processes* executed within heterogeneous process management systems, namely AHEAD and conventional workflow management systems.

While development processes as a whole cannot be adequately supported by workflow management systems, there are often well understood, repetitive subprocesses which can be modeled as workflows and enacted by means of WfMS. Examples from plant design projects are the design and procurement of a device. In these processes, only few resources are involved and they usually have to follow a pre-defined procedure using certain tools to produce the required result. This type of processes is very suitable for workflow support. For this reason, an approach was developed to couple AHEAD with a workflow management system, and a prototypical implementation was realized in which the WfMS Shark was coupled with AHEAD.

A workflow instance is represented in AHEAD by a subnet of the dynamic task net which represents the whole development process. A *view-based integration* has been realized in the sense that the subnet always reflects the current enactment state of the workflow instance but changes to the workflow cannot be applied via the AHEAD system. Regarding the visibility of workflow elements, a *gray-box approach* was chosen. A workflow instance is neither represented by an atomic task in the task net (black-box approach) nor are all elements of the workflow necessarily visible

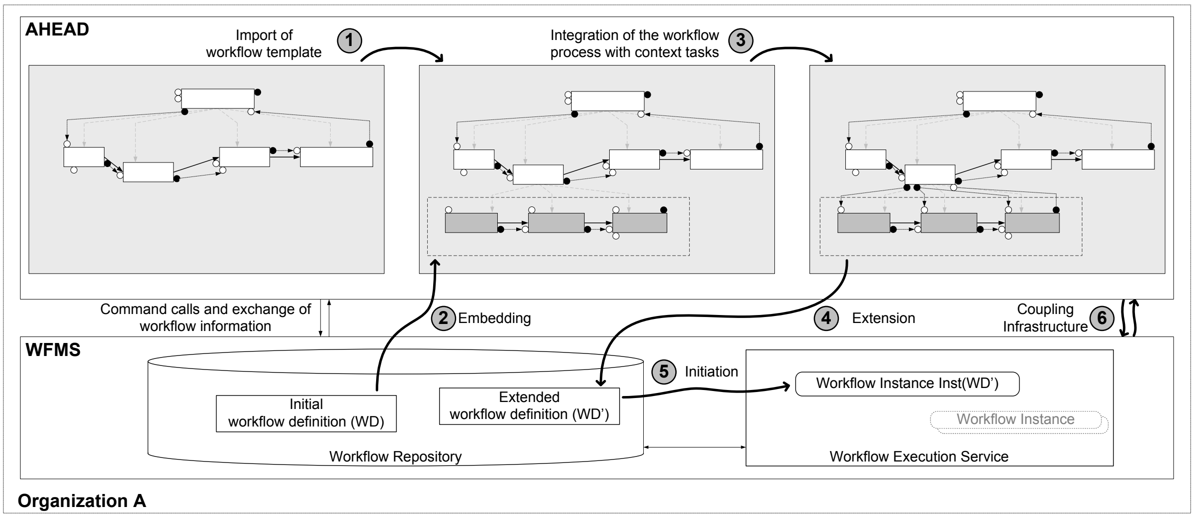


Abbildung 4.10: Integration of workflow processes in AHEAD [HJK+08].

in the task net (white-box approach). For example, control variables or control nodes in the workflow graph are not mapped to elements in the dynamic task net. The processing of workflows in AHEAD takes place in several phases which are illustrated in Figure 4.10.

Workflow Embedding At first, a workflow template is imported, i.e. the corresponding task net representation is embedded into the dynamic task net (1/2). This representation is called a *workflow fragment*.

Workflow Context Definition The workflow fragment is connected with other tasks in the dynamic task net by control and data flows (3). The execution states of the workflow tasks (i.e. the tasks in the workflow fragment) are set to the Waiting state. An extended workflow definition which includes new conditions for the execution of workflow activities is generated (4).

Workflow Instantiation AHEAD sends a request to the WfMS for the creation of an instance of the extended workflow definition. A workflow instance is created and a reference to the instance is returned to AHEAD (5). Afterwards, the workflow instance is started.

Workflow Monitoring The workflow is enacted in the WfMS. Assigned resources start their tasks, modify data and finally commit their tasks. All events regarding task status changes or data modification in the WfMS are sent to AHEAD and the workflow fragment is updated accordingly (6). A manager using the AHEAD system can thus monitor the performance of the workflow.

Process Traceability After the termination of the workflow instance, the workflow fragment remains in the dynamic task net and all documents produced in the course of the workflow remain accessible in AHEAD for reasons of traceability.

With this approach, workflow instances can be embedded in a dynamic task net. The enactment of embedded workflows can be monitored in AHEAD via the corresponding workflow fragments. AHEAD and the WfMS exchange events bidirectionally to inform each other about relevant process changes [Hel08a, HHM$^+$06, Wei06]. The coupling of the two systems supports integrated project and workflow management and enables the integration of previously independent workflow processes in a development project.

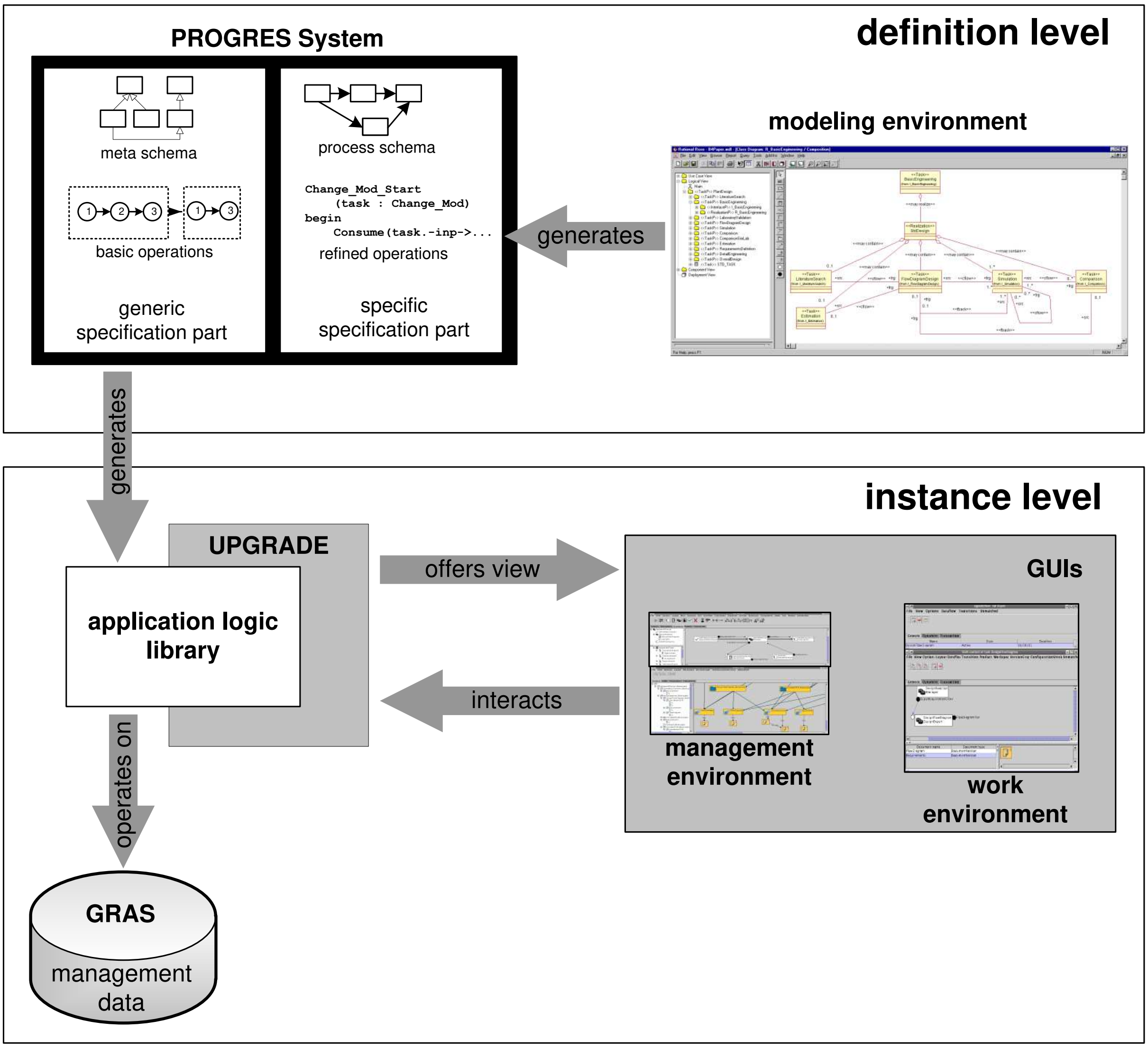
4.6 The AHEAD Prototype

The AHEAD system has been realized as a research prototype using an infrastructure for rapid prototyping of graph-based applications. Figure 4.11 shows the general architecture of the AHEAD system. The application logic of the AHEAD system has been specified in the graph rewriting language PROGRES [SWZ99]. The PROGRES environment allows to generate C-code from such an executable specification. The PROGRES specification of the AHEAD system is two-part. The generic specification part has been manually created by the developers of AHEAD. This part includes the DYNAMITE meta-model for dynamic task nets. Domain specific process knowledge is defined in the second part of the PROGRES specification. This part is generated from the UML class diagrams which are created by means of the modeling environment for process model definitions.

The UML-based process model definitions can be created by means of the commercial CASE tool *Rational Rose*. Process model definitions and their different versions can be managed as UML-packages in the repository of this tool. From the UML models, the domain-specific part of the PROGRES specification of the AHEAD system is generated by a transformer. Afterwards, the process model definitions constitute a part of the application logic of the AHEAD system.

For the implementation of the user interface of the AHEAD system, the UPGRADE framework has been used [BJSW02b, BJSW02a]. UPGRADE is a Java framework for the development of graph-based applications. The source code which is generated from the PROGRES specification is linked with the UPGRADE libraries. The resulting prototype can be adapted and extended by Java classes to customize its user interface. Different tools for the management and enactment of development processes have been realized by means of PROGRES/UPGRADE which will be described in this section. All management data which are created and modified by means of these tools are stored in a GRAS database [KSW95]. This database is optimized for the storage of graph structures.

The realization of the AHEAD prototype by means of PROGRES/UPGRADE has both, advantages and disadvantages. The infrastructure of PROGRES, GRAS and UPGRADE allows for the rapid prototyping of graph-based applications. Due to the available code generation, far less effort is required compared to manual implementation. The application logic is formally specified in PROGRES by means of which the executable specification can be analyzed and interpreted to verify its correctness. The PROGRES language is dedicated to the development of graph-based applications.



```
Change_Mod_Start
    (task : Change_Mod)
begin
    Consume(task.-inp->...
```

Abbildung 4.11: Architecture of the AHEAD system [HJK^{+}08].

Solutions to many common problems like pattern matching in a graph are provided by PROGRES.

One of the drawbacks of the realization approach is that it requires the extensive infrastructure consisting of PROGRES, GRAS and UPGRADE. This infrastructure is not maintained anymore and it is not intended for commercial use. Furthermore, the portability of the infrastructure is limited. While the Java framework UPGRADE can be ported to other platforms, PROGRES is limited to a Unix environment. Finally, the runtime efficiency of a PROGRES prototype is not optimal, since the general purpose code generation cannot incorporate any application-specific optimizations.

AHEAD provides two different environments for the management and enactment of process model instances respectively. The *management environment* is used by the project manager to manage the development process. He can define tasks, control flows and data flows, assign resources to tasks and prepare tasks for execution.

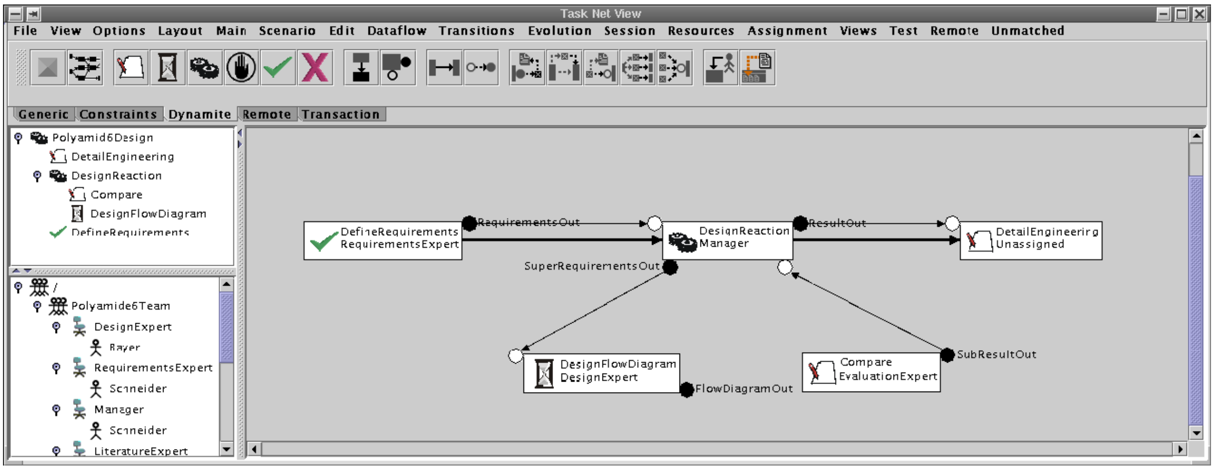


Abbildung 4.12: The task net view of the management environment [HJK+08].

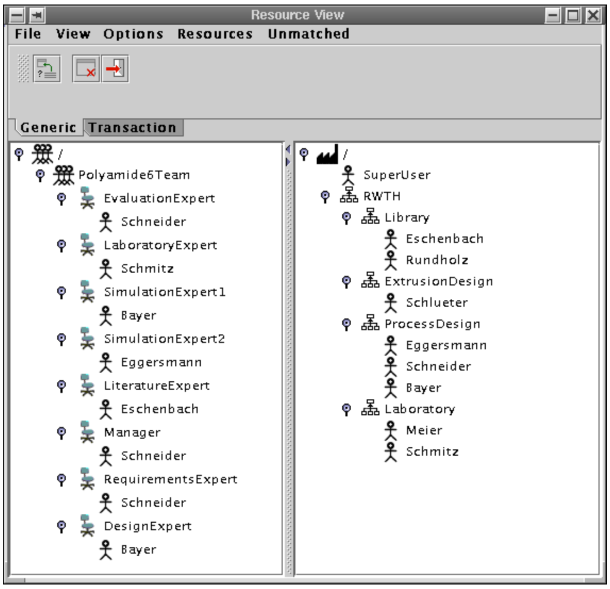
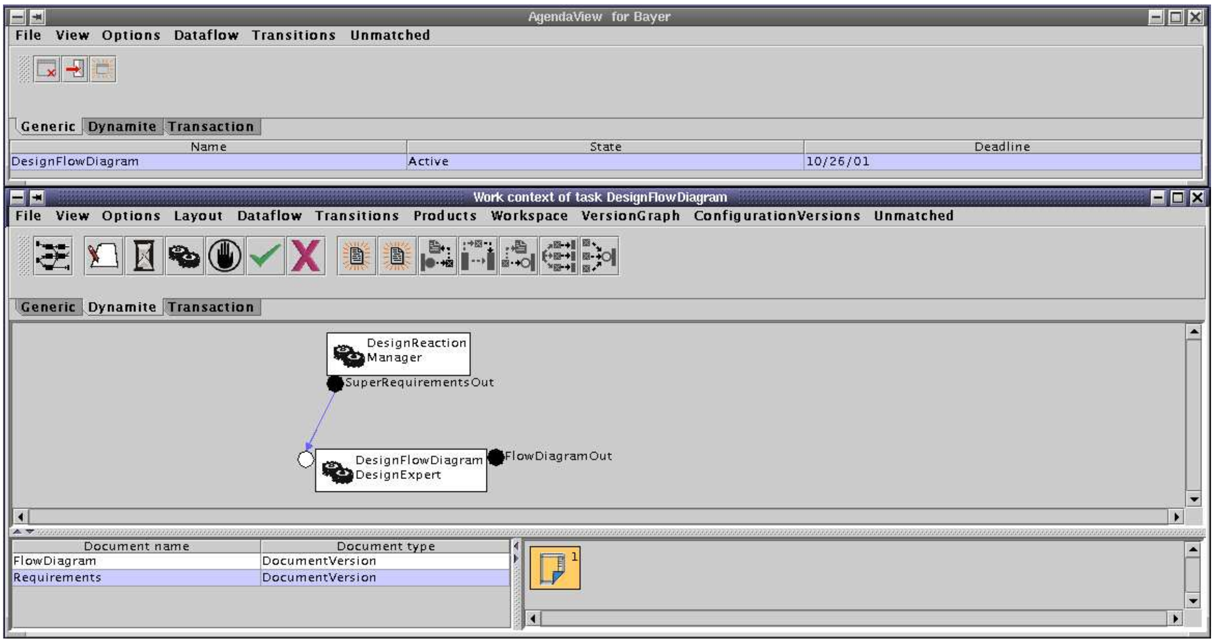


Abbildung 4.13: The resource view of the management environment [HJK+08].

Structural changes to the dynamic task net are exclusively done in the management environment. The development process is enacted by the resources using the *work environment* in which they can change the execution states of their assigned tasks.

Figure 4.12 shows a screenshot of the *task net view* of the management environment. It shows the task hierarchy and the available actual resources on the left side. The main view shows the dynamic task net as a graph in which the elements are represented as in Figure 4.3. The graphical representation is very close to the internal data structures for dynamic task nets. This has been identified as one of the drawbacks of the AHEAD prototype regarding its applicability in practice. One of the contributions of this thesis is the improvement of the graphical representation of dynamic task nets to achieve a higher acceptance by users in industrial projects.



Name	State	Deadline
DesignFlowDiagram	Active	10/26/01

Document name	Document type
FlowDiagram	DocumentVersion
Requirements	DocumentVersion

Abbildung 4.14: The work environment [HJK+08].

In addition to the task net view, the management environment comprises a *resource view*, which is used to assemble the project team. Planned resources can be defined for the project. Actual resources can be selected from the different organizational units in the company and assigned to the planned resources of the project. The resource view is depicted in Figure 4.13 with the project resources in the left tree view and the base resources in the right tree view.

Finally, the management environment comprises the *product view* which gives an overview over all development products and their versions. Versions of different documents can be combined into configurations. Products, versions and configurations are all displayed as nodes in a graph. An example screenshot of the product view can be found in [NM08, p. 317].

The work environment serves for the execution of defined tasks by the assigned resources. A user who has a position in the project team can log on to the system and a list of his assigned tasks is presented to him in the *agenda tool* of the work environment (top of Figure 4.14). For each assigned task, information about its state, deadline etc. is displayed. The user can start, suspend, finish or abort a task using the agenda tool, or he can start the *work context tool* for a selected task. The work context tool (bottom of Figure 4.14) provides access to the documents and tools required for executing a task. The relevant documents are presented in a list and in a graph view at the bottom of the work context tool. Furthermore the work context of the task is presented to the user which contains all tasks in the task net which are relevant for this task, i.e. which are connected via incoming and outgoing data flows. This can be predecessors and successors of a task, its parent task and its subtasks, and tasks connected by feedback flows.

In AHEAD, the use of the management environment is reserved for the project manager, and all other project team members are restrained to the work environment. This way, the access control to the management data is realized. Ordinary project team members can only change their assigned tasks and can only view the work contexts of these tasks. The project manager has full access to the whole management configuration and can modify all parts of the task net, resource model and product configuration.

In plant design projects, there are multiple responsible persons who need to have access to the whole management configuration. This can be achieved in AHEAD by giving these persons the role of project responsible. However, there are also scenarios in practice, where a hierarchical approach is required in which certain project team members are responsible for whole subprocesses and not only for their assigned tasks. Therefore, a more flexible access control model has been developed in this thesis as part of the solution approach for project controlling.

Kapitel 5

Timed Dynamic Task Nets

A process management system has to maintain an internal representation of a process model instance, including the dependencies between the defined tasks, the resources who are assigned to the tasks, and the products which are produced. Tasks, products and resources are managed in an integrated way in PROCEED. Dynamic task nets are used for the representation of process model instances.

The entities, properties, and relationships for dynamic task nets, as well as the constraints for changes to dynamic task nets are defined in a new meta-model which is presented in this chapter. The major extensions compared to the DYNAMITE meta-model are related to timing issues. Therefore, the new meta-model for dynamic task nets is called the TNT meta-model, which stands for *Timed Dynamic Task Nets* or *Timed Nets* in short. The development of the TNT meta-model had three different goals.

- The main concepts of the DYNAMITE meta-model should be transferred to the new meta-model. In particular, dynamic task nets should be used to represent process model instances.
- New requirements derived from the industrial context regarding the usability of the prototype should be addressed. The meta-model should be adapted and simplified where necessary.
- The meta-model had to be extended by new entities and properties for project planning and controlling.

Figure 5.1 shows the five partial models of the TNT meta-model. Every partial model defines on the one hand entities, properties, and relationships, and on the other hand constraints which apply to changes to the management data.

The distinction between a structural and a behavioral model has been adopted from DYNAMITE. The structural and the behavioral model of DYNAMITE have been adapted and extended to incorporate additional modeling entities, additional execution states of tasks, and corresponding invariants. The *structural model* covers the definition of the available entities for task net modeling, their properties and relationships, as well as structural invariants. It is presented in Section 5.1. The *behavioral model* covers the execution states of tasks and according behavioral invariants. Furthermore, it defines consistency constraints for structural change operations which depend on the execution states of tasks. The behavioral model is

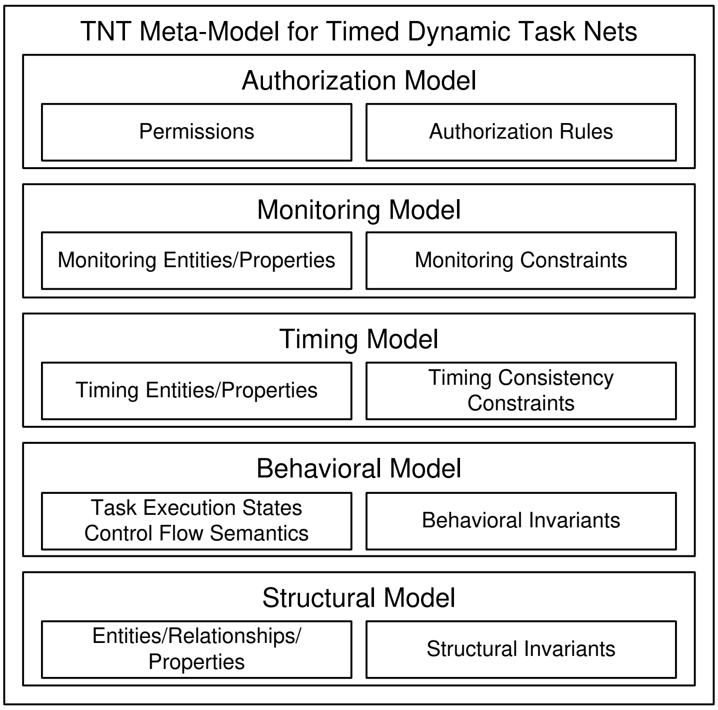


Abbildung 5.1: Partial models of the TNT meta-model for dynamic task nets.

presented in Section 5.2. The invariants defined in the structural and behavioral model may never be violated.

The *timing model* defines additional entities and task properties which are required to schedule the tasks in a dynamic task net. Furthermore, it defines timing consistency constraints which ensure that a scheduled dynamic task net represents a time and resource feasible project schedule. Timing consistency constraints may be temporarily violated The timing model is presented in Section 5.3.

The *monitoring model* introduces additional entities and properties which are required to measure the degree of completion of tasks in a dynamic task net and to evaluate their performance in comparison to the plan. Monitoring constraints define the situations in which the actual performance matches the plan. Their violation indicates a deviation from the plan. Monitoring constraints are non-strict in the sense that they may be permanently violated. The monitoring model is presented in Section 5.4.

When process model definitions are enacted in an engineering project, an authorization model has to be implemented which ensures that only authorized users of the system can perform changes to the management data. In Section 5.5, the authorization model of the TNT meta-model is described. Project specific permissions are assigned to members of a project team, which are evaluated in authorization rules to determine the effective permissions of a user.

Every partial model depends on entities, properties, and relationships defined in the next lower model, i.e. a partial model may define additional properties for entities which are defined in a lower model, and the constraints defined as part of a

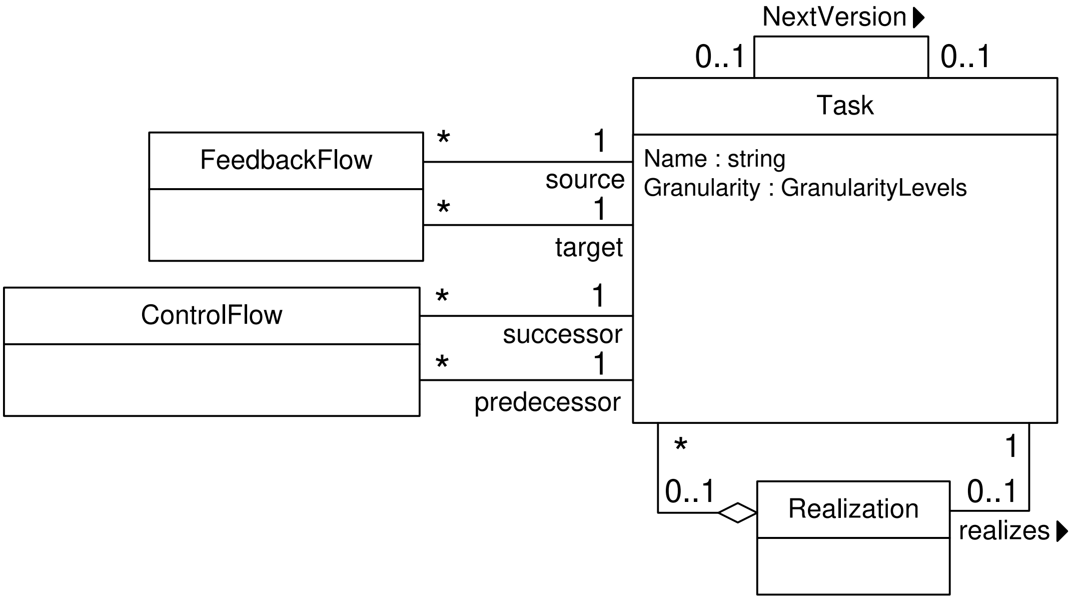


Abbildung 5.2: Classes for tasks and task relationships.

partial model may refer to modeling elements of lower models.

The entities and relationships of the TNT meta-model are introduced by means of UML class diagrams in the following sections. The respective constraints are presented in a formal notation which has been adopted from [Sch02].

5.1 Structural Model

The description of the entities and relationships defined by the structural model is divided into three sections corresponding to the different perspectives on dynamic task nets, namely tasks, products, and resources. Afterwards, the structural constraints are described which apply to all three perspectives.

5.1.1 Tasks and Control Flow

The central entity in the TNT meta-model is the *task*. It integrates the entities related to resource and product management with each other. Figure 5.2 shows the class Task together with related classes for modeling vertical and horizontal task relationships in a dynamic task net.

Vertical relationships refer to the hierarchical structure of dynamic task nets. Except for the root task, all other tasks in a dynamic task net have a unique parent task and may have several subtasks. The root task represents the whole project. A task which has subtasks is called a *complex task*, and a task without subtasks is called an *atomic task*. Like in the DYNAMITE meta-model, a distinction is made between the interface and the *realization* of a task. The interface of a task is its definition in terms of properties, parameters, assigned resources and relationships to other tasks. The realization of a task contains its subtasks and their mutual relationships. Therefore, the subtasks of a task are aggregated in an object of the class Realization which is associated with the parent task. The realization of a task is usually not displayed in the graphical representation of dynamic task nets.

For every task in a dynamic task net, the *granularity level* can be explicitly defined by setting the property value Granularity. This is an extension to the DYNAMITE meta-model. Three granularity levels are distinguished.

- Project structure
- Task
- Work step

The highest level is *project structure*. All tasks in a dynamic task net, which together form the work breakdown structure of the project, have the granularity level *project structure*. The lowest tasks in the hierarchy of a dynamic task net which have the granularity level *project structure* are the work packages. The next lower granularity level is the *task* level. Finally, the lowest level of granularity is the level *work step*. Tasks with this granularity are not taken into account during scheduling. For every task in a dynamic task net, the following two rules have to be fulfilled:

- The task has the same granularity level as its sibling tasks.
- The task has the same or a lower granularity level than its parent task.

The granularity level of a task may be set manually for a task. If no granularity level is explicitly specified for a new task, then the granularity of the new task is automatically set to the granularity of the sibling tasks or, if no siblings exist, to the granularity of the parent task. The granularity levels of the tasks in the example scenario have been defined as depicted in Figure 5.3.

In [HJK^{+}08], three levels of granularity have been distinguished which overlap with the granularity levels introduced in this section. On a coarse-grained level, development processes are divided into phases according to some life cycle model. At a medium-grained level, development processes are decomposed further down to the level of documents or tasks, i.e. units of work distribution. At the fine-grained level, specific details of design subprocesses are considered. The granularity level *work step* can be mapped to the fine-grained level. However, the coarse-grained level does not cover all tasks of the work breakdown structure, i.e. with granularity level *project structure*, but only the first two levels. The granularity levels of [HJK^{+}08] were not explicitly defined for the tasks in a dynamic task net. However, the tasks which were commonly represented in a dynamic task net in AHEAD belonged to the coarse-grained and medium-grained levels. This is still true for dynamic task nets in PROCEED where work steps can be defined but are not scheduled.

With respect to horizontal relationships, tasks can be connected by control and feedback flow relationships. A *control flow* connects exactly two tasks and is directed

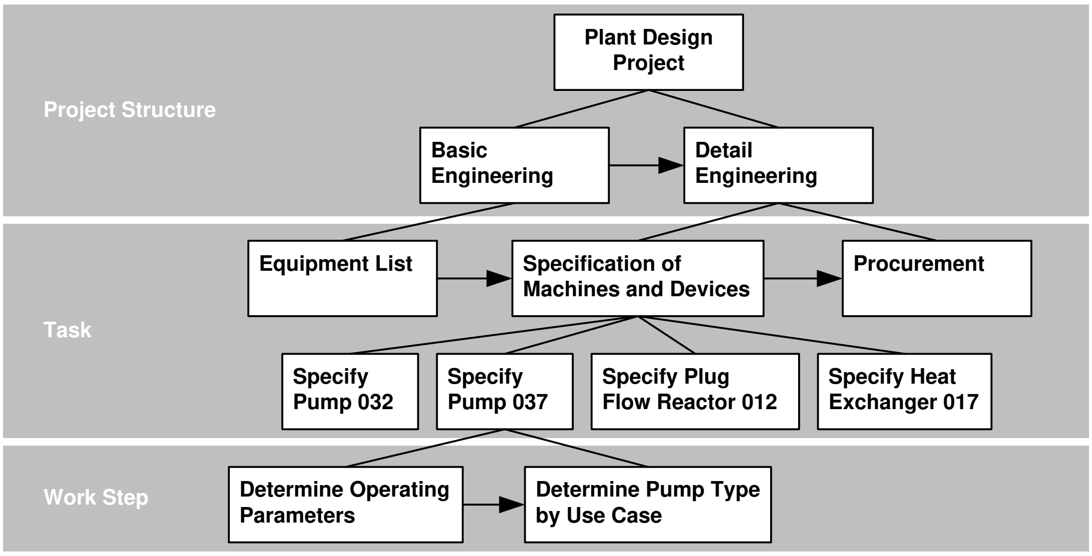


Abbildung 5.3: Task net hierarchy with defined granularity levels.

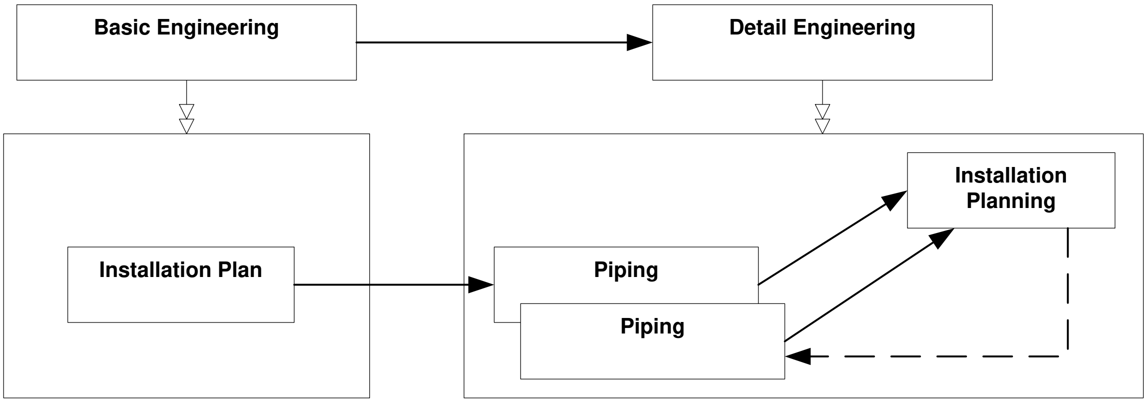


Abbildung 5.4: Example for a hierarchical task net with control and feedback flows.

from the predecessor to the successor A *feedback flow* is defined between two tasks, when the results of the target task have to be changed due to detected errors or new requirements. The information regarding the required changes is provided by the source task. In Figure 5.4, a part of a hierarchically structured dynamic task net with defined control and feedback flows is depicted. If a task has subtasks they are depicted below the task in a box which is connected to the task by a double-headed arrow. Control flows are displayed as solid lines with closed arrowheads. A feedback flow is displayed as a dashed line with a closed arrowhead.

5.1.2 Documents and Data Flow

For modeling documents and data flow in dynamic task nets, the relevant entities and relationships of the DYNAMITE meta-model have been adopted. In Figure 5.5, the relevant classes and associations are depicted. An object of the class Document

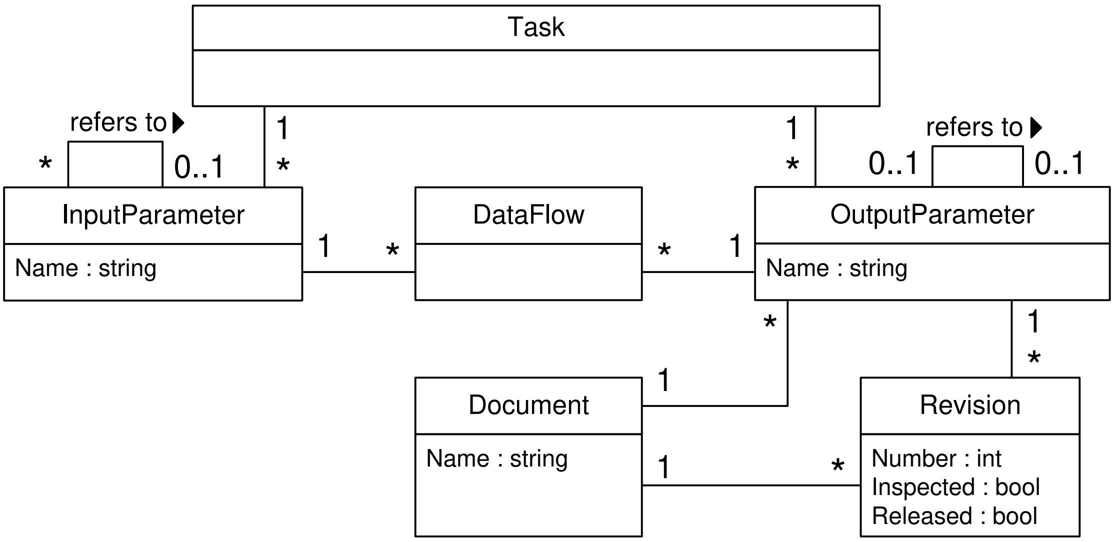


Abbildung 5.5: Classes for documents, revisions, parameters, and data flows.

represents a document in the Comos database, e.g. a flow diagram, a Word or Excel document. Input and output parameters can be defined for a task in PROCEED. An *input parameter* indicates, that a document is required for this task. An *output parameter* indicates, that a document is created or modified in the task. When a document is created in the Comos database, it is associated with the output parameter of the task in which it has been created, and with the output parameters of all tasks in which it will be modified. A *data flow* connects an output and an input parameter of two different tasks. It specifies that the document associated with the output parameter can be used in the task to which the input parameter belongs. An example for a data flow in a dynamic task net is shown in Figure 5.6. The block flow diagram (BFD) is transferred from the task Preliminary Planning to the task Basic Engineering.

In Comos, several *revisions* of a document can be created. In a three-step procedure, a revision is created, inspected and finally approved, and thereby released. Revisions are used to document certain intermediate results during the elaboration of a document. In contrast to common version management systems, it is not possible to retrieve an older version of a document from the Comos database to continue development starting from this version. Nevertheless, my means of document revisions the progress of the work is documented. Therefore, document revisions are represented in the TNT meta-model by objects of the class Revision. A revision of a document is produced in the course of a task. It is connected with the output parameter of this task which is associated with the corresponding document. In the example in Figure 5.6, a first revision of the process flow diagram PFD has been created with the revision number 0, and it has been released for all consuming tasks. In contrast to DYNAMITE, the release of document revisions for individual tasks and the consumption of document revisions by these tasks is not explicitly modeled in

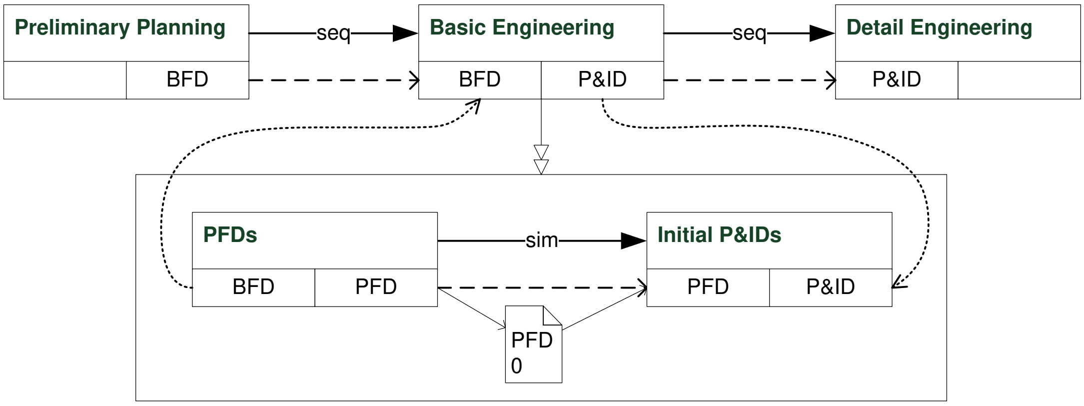


Abbildung 5.6: Example for data flow in a dynamic task net.

PROCEED for reasons of simplification.

In the graphical representation of Figure 5.6, input parameters and output parameters of a task are listed below the name and execution state. Data flows are drawn between input and output parameters as dashed lines with open arrowheads. If a document revision has been produced in a task, it is depicted as a document item labeled with the name of the document, the revision number and the status of the document. The document item is connected with the output parameter. A produced but not yet released revision has no revision number but is labeled with * instead. When a revision is released, it is connected with all consuming input parameters.

In [Sch02], a distinction was made between internal and external parameters to model the data flow between a task and its subtasks (cf. Section 4.3). This concept has been abandoned in PROCEED for reasons of simplification. If the input of a task shall be available at one of its subtasks, then the input parameter of the subtask references the input parameter of the parent task. For this purpose, the association refers to is defined in Figure 5.5. Revisions which are available at the input parameter of the parent task are also available at the connected input parameter of the subtask. If the output of a subtask should be available at the parent task, then an output parameter is defined for the parent task which references the output parameter of the subtask. Revisions which are produced at the output parameter of the subtask are implicitly available at the output parameter of the parent task. Figure 5.6 illustrates the connection between parameters by means of references. The BFD is provided to the task Basic Engineering via a data flow and is also available for the subtask Create PFD because of the reference from its input parameter to the input parameter of Basic Engineering. The concept for modeling data flows in PROCEED is similar to the original data flow concept defined in [Kra98] which also did not distinguish between internal and external parameters. However, in [Kra98] ordinary data flows were defined in the opposite direction compared to the references in PROCEED. Data flows could be defined from an input parameter of a task to the input parameter of a subtask and from the output parameter of a task to an output parameter of the parent task. This approach required to produce and release document revisions which should be transferred from a subtask to its parent task and vice versa. The reference concept implemented in PROCEED simplifies the transfer of data between tasks and their subtasks.

5.1.3 Resource Modeling

In the TNT meta-model, the strict distinction between a resource model and a model for dynamic task nets as it was realized in AHEAD with RESMOD and DYNAMITE has been abandoned. The resource modeling capabilities of the TNT meta-model cover on the one hand the assignments of resources but also the modeling of the organizational structure. Resource management in the organization, allocation of resources for projects, and assignment of resources to tasks are all covered in the TNT meta-model.

All members of a project team in a plant design project use Comos and PROCEED to carry out their work and to manage their personal processes. Project team members are *human resources* for the project and at the same time *users* of the management system. Therefore, the terms human resource and user will be used synonymously in the following because they represent equivalent concepts with respect to the PROCEED system.

Figure 5.7 shows the classes and associations which are defined for resource management in PROCEED. Human resources are represented by objects of the class User. They are organized in departments which are structured hierarchically and are located at certain company locations. This information was already available in Comos before the extension by PROCEED. The classes User, Department and Location integrate the resource model of Comos into the TNT meta-model.

Non-human resources are not modeled in PROCEED for reasons of simplification. Software tools are not modeled explicitly since the main application which is used by the project team members is the Comos environment and the use of external applications is determined by the type of the documents which have to be edited. Required computer hardware, facilities etc. are not explicitly modeled as resources required for a task to simplify project planning. It is assumed that computer workstations, software and all required tools are available to the engineers in a project as needed, so that they do not need to be explicitly modeled and planned, i.e. assigned to tasks. Their costs however are incorporated in the project budget as part of the base costs.

Functional roles Human resources can play different functional roles in an organization and a project. A *role* defines the qualifications of the resources who can play this role. Examples for functional roles in a plant design project are project manager, controller, architect, construction engineer, or process engineer. Roles can be structured in a generalization hierarchy. If a resource can play a specific role, he can also play the more general role. Figure 5.8 shows the generalization hierarchy of roles of the example scenario. A role hierarchy like this may be defined in a plant engineering company. In this example, a mechanical engineer is assumed to have the qualifications of a general engineer like fundamental knowledge in mathematics

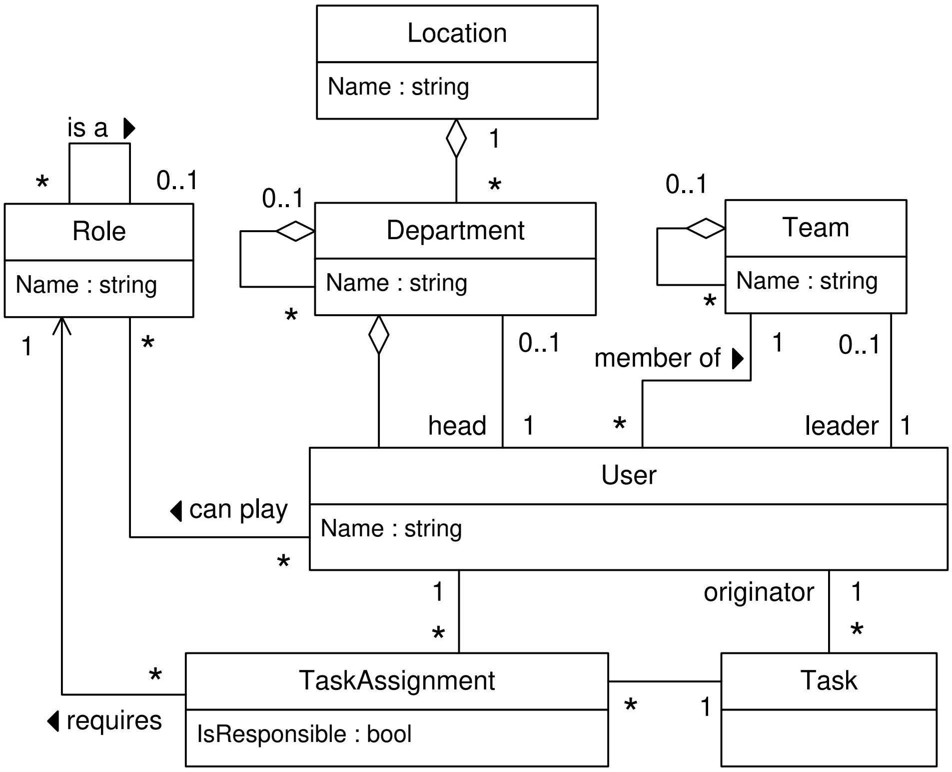


Abbildung 5.7: Classes for resources and task assignments.

and a general understanding of design. In practice, role hierarchies are rather flat, but they are nevertheless useful to organize the different functional roles defined in a company. The functional roles which are available in a concrete project are implicitly defined by the human resources which are members of the project team. A functional role is available in the project if there is a project team member who can play the role in the organization.

Project team A project team is built up from employees of the company who are members of different departments. The department heads decide whether their employees are released to work in a project for a requested time period. The organizational breakdown structure (OBS) defines the structure of the project team as described in Section 3.1. The organizational units of the OBS are subteams of the project team which are modeled in PROCEED as instances of the class Team. A *team* can be subdivided into several subteams. A team has several team members one of which is the team leader. The leader of a team is responsible for his team members and the subteams. He may instruct his team members and the leaders of the subteams.

Figure 5.9 shows a part of the organizational breakdown structure of the example scenario including the team members. The team leaders are represented by resource pictograms with a filled head. The team leader of the team Process Engineering is the resource Dreher. He receives instructions from the project manager and instructs

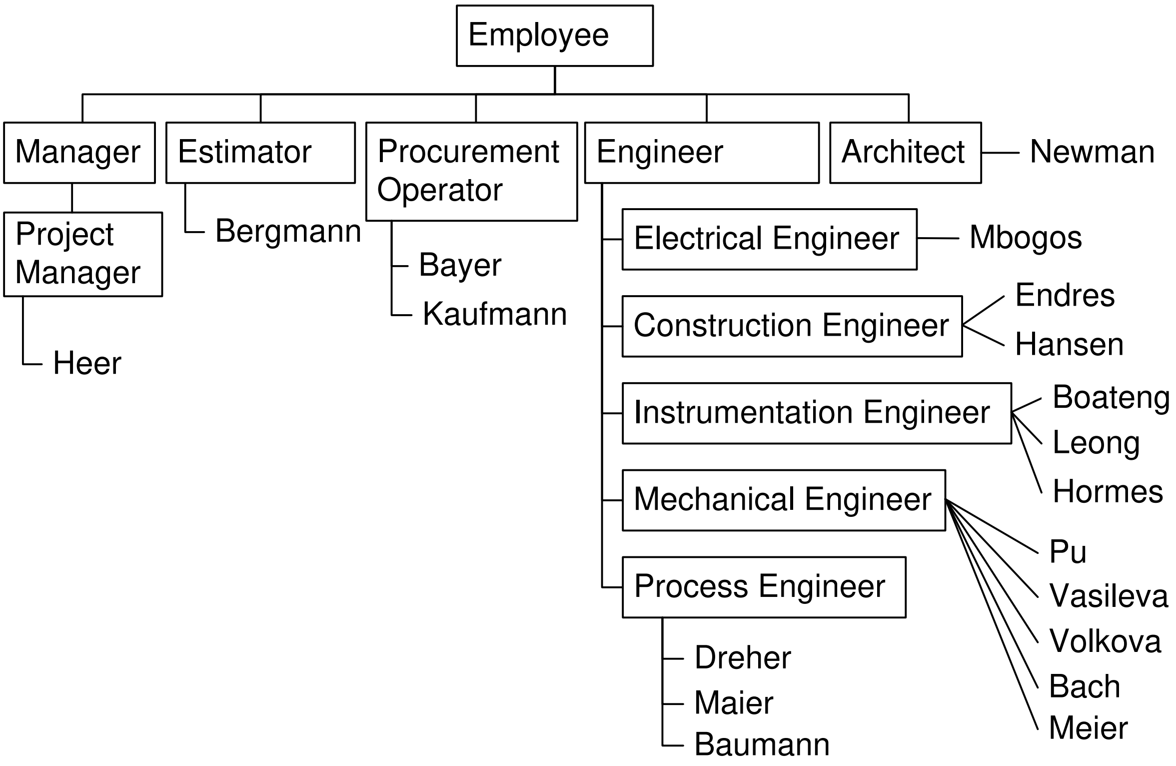


Abbildung 5.8: Role hierarchy of the example scenario.

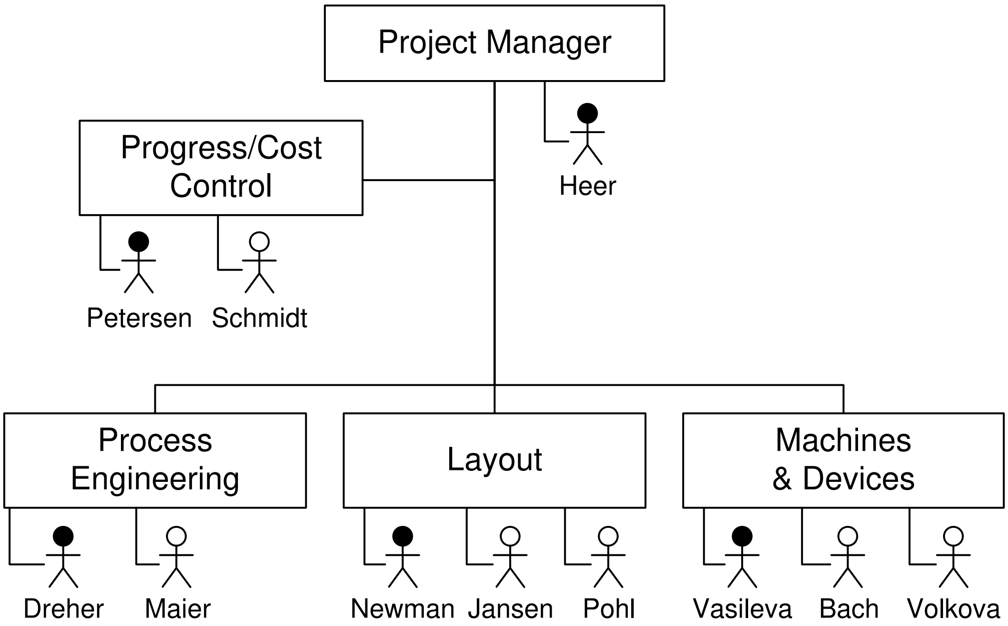


Abbildung 5.9: Organizational breakdown structure of the example scenario.

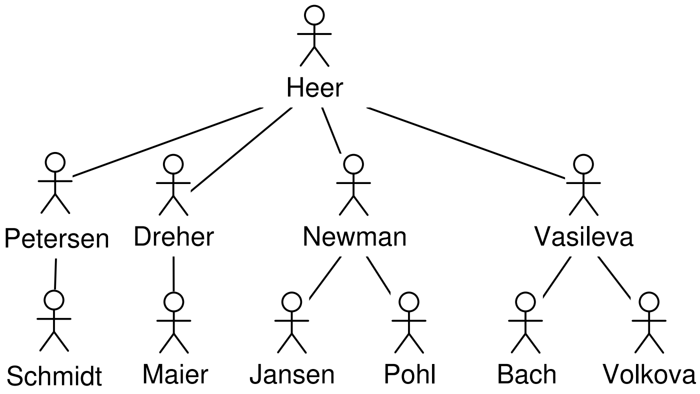


Abbildung 5.10: Responsibility hierarchy derived from the OBS.

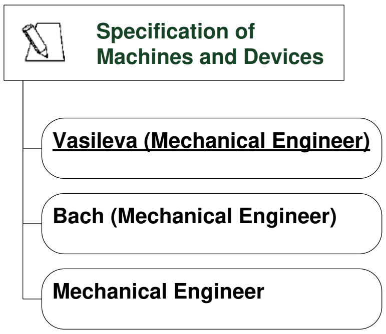


Abbildung 5.11: Example for a task with multiple task assignments.

his team members. The position *project manager* is modeled as a team with only one member. The hierarchy in the example is rather flat. However, the hierarchical structure is generally required to model the distribution of responsibility in the project team. The *responsibility hierarchy* is implicitly defined by the project team structure. For the example of Figure 5.9, the responsibility hierarchy is explicitly shown in Figure 5.10.

Task assignments In PROCEED, the concept of a *task assignment* has been introduced. The assignment of a resource to a task is explicitly modeled by an object of the class TaskAssignment. For a task assignment, a *required role* has to be specified. It is a strict constraint that the user who is assigned to the task via the task assignment can play the required role. Several resources can be assigned to a task. This is modeled by multiple task assignment objects. However, a resource cannot be assigned to two different task assignments of the same task. Exactly one of the assigned resources has to be defined as the person responsible for managing the task. This resource is called the *responsible resource*. The property IsResponsible is set to `true` for the task assignment of the responsible resource.

Figure 5.11 shows an example in which three resources have been assigned to the task Specification of Machines and Devices. Task assignments are depicted as rounded boxes below the task. If no resource is assigned yet, only the role is displayed. Otherwise, the role is displayed in brackets after the resource's name. It is required that both assigned resources can play the role Mechanical Engineer. The resource Vasileva is appointed to be responsible for the task. This is visualized in Figure 5.11 by underlining the required role and the assigned resource. A third task assignment has been defined which also specifies the required role Mechanical Engineer but no resource has been assigned yet.

In development projects, it is a common practice to define so called tickets to keep track of necessary bug fixes or enhancements of the developed product. These tickets are often managed by means of a bug tracking system, e.g. the system *trac* [Sof10]. A ticket can be directly assigned to a resource or it can be put in a task pool to be picked up by an eligible resource at a later point in time. These strategies are called the push- and pull-pattern in the context of workflow management systems [RvdAtHE05]. Both resource assignment patterns are supported by PROCEED. The *push-pattern* is realized by directly assigning tasks to resources. The *pull-pattern* is realized by only defining the required role of a task assignment an preparing the task for execution. An eligible resource who can play the required role in the project can pick up the task.

The explicit modeling of task assignments has several advantages for project planning, scheduling and execution. First, multiple resources can be assigned to a task which is particularly advantageous for rolling-wave planning, when the realization of a task in terms of subtasks is not yet completely defined but the required resources and workload for the task have to be planned. Dynamic task nets allow the creation of additional subtasks even at process runtime and thereby enable rolling-wave planning. Besides the deferred creation of subtasks, the definition of multiple task assignments for a task has the additional advantage that several resources can be assigned to a task which support the responsible resource but are not responsible for any of the subtasks.

Another advantage of explicitly modeled task assignments is that planned and actual workload for a task can be associated with each assigned resource individually. Resources can be assigned to a task with different planned workload, so that one resource has to work 2h per day on the task while another assigned resource has to work 8h per day on the task. Furthermore, the resources can book their actual workload on the task assignments which may deviate from the planned workload.

5.1.4 Structural Constraints

The structural model defines consistency constraints regarding the structure of a dynamic task net. For every structural change operation to a dynamic task net, it has to be ensured that the consistency of the management data is maintained. Therefore, several *structural invariants* have been defined which have to be fulfilled at any time. This implies that no structural change operation on a consistent dynamic task net may lead to a state of the task net in which an invariant is violated. From the structural invariants, *pre- and post-conditions* for structural change operations have been derived and implemented in PROCEED. Several invariants, pre- and post-conditions have been adopted from DYNAMITE while others have been adapted or added. In particular, structural invariants which refer to task assignments were not defined in DYNAMITE.

Definitions

For the formal definition of the invariants and the pre- and post conditions of change operations, several definitions are required. These definitions apply as well for the behavioral model, timing model, and monitoring model, which are based on the structural model.

Several entities have been defined in the structural model, including tasks, resources, and documents. The different classes which have been introduced in this section define entity types. An entity type can be regarded as a set containing all objects of this type, e.g. the set of tasks. The properties which have been defined for the entity types are defined as mappings from an entity type to a certain domain. Let E be a set of entities of the same type and D a domain. A property is formally defined as a mapping

$$\mathtt{P} : \mathtt{E} \to \mathtt{D} \cup \{\perp\}$$

Where $\perp$ indicates that the value of the property is undefined. For reasons of simplification, properties of entities are written in the dot notation which has also been used in [Sch02]. For a concrete entity $\mathtt{e}$ of the entity type $\mathtt{E}$, the property $\mathtt{P}$ of equation Section 5.1.4 is denoted as follows.

$$\mathtt{e.P} \in \mathtt{D} \cup \{\perp\}$$

Likewise, a set-valued property $\mathtt{S}$ is denoted as $\mathtt{e.S} \subset \mathtt{D}$ which is formally defined as a mapping $\mathtt{P} : \mathtt{E} \to \wp(\mathtt{D}) \cup \{\perp\}$ with $\mathtt{e} \in \mathtt{E}$. For several invariants and algorithms presented in this thesis, it is required to check whether the value of a property is defined. For this purpose, the function $\mathtt{undef}$ is defined as follows.

$$\mathtt{undef} : (\mathtt{E} \times (\mathtt{D} \cup \{\perp\})) \times \mathtt{E} \to \{\mathtt{true}, \mathtt{false}\} :$$
$$(\mathtt{P}, \mathtt{e}) \to \begin{cases} \mathtt{true}, & \mathtt{P(e)} = \perp \\ \mathtt{false}, & \mathtt{P(e)} \in \mathtt{D} \end{cases}$$

For a property $\mathtt{P}$ of an entity $\mathtt{e} \in \mathtt{E}$ the application of the function $\mathtt{undef}$ is denoted as $\mathtt{undef(e.P)} \in \{\mathtt{true}, \mathtt{false}\}$. Finally, methods can be defined for entity types in the dot notation $\mathtt{e.M(arg)} \in \mathtt{D}$ for actual parameters $\mathtt{arg} \in \mathtt{A}$ from the argument domain $\mathtt{A}$. These methods would be formally defined as $\mathtt{M} : \mathtt{E} \times \mathtt{A} \to \mathtt{D}$ with $\mathtt{e} \in \mathtt{E}$.

Tasks and control flow The set $\mathtt{Tasks}$ contains all tasks whereby different versions of a task are considered as distinct tasks. The set $\mathtt{ControlFlows} \subset \mathtt{Tasks} \times \mathtt{Tasks}$ contains all control flows. The set $\mathtt{FeedbackFlows} \subset (\mathtt{Tasks} \times \mathtt{Tasks}) \setminus \mathtt{ControlFlows}$ contains all feedback flows. The set $\mathtt{Realizations}$ contains all realizations. For every control flow $\mathtt{c} \in \mathtt{ControlFlows}$ the following attributes are defined.

- $\texttt{c.Pred} \in \texttt{Tasks}$ is the source of the control flow.
- $\texttt{c.Succ} \in \texttt{Tasks}$ is the target of the control flow.

For every feedback flow $\texttt{f} \in \texttt{FeedbackFlows}$ the following attributes are defined.

- $\texttt{f.Source} \in \texttt{Tasks}$ is the source of the feedback flow.
- $\texttt{f.Target} \in \texttt{Tasks}$ is the target of the feedback flow.
- $\texttt{f.Active} \in \{\texttt{True}, \texttt{False}\}$ indicates whether the feedback flow is still active.

For every task $\texttt{Task} \in \texttt{Tasks}$ and $\texttt{Realization} \in \texttt{Realizations}$ the following properties are defined.

- $\texttt{Task.PreviousVersion} \in \texttt{Tasks}$ is the previous version of the task.
- $\texttt{Task.PreviousVersions} \subset \texttt{Tasks}$ is the set of all previous versions of the task, i.e. the non-reflexive transitive closure of the `PreviousVersion` function.
- $\texttt{Task.Realization} \in \texttt{Realizations}$ is the realization of a task.
- $\texttt{Realization.Subtasks} \subset \texttt{Tasks}$ is the set of all tasks in a realization.
- $\texttt{Task.Subtasks} := \texttt{Task.Realization.Subtasks}$ is the set of all subtasks.
- $\texttt{Task.Parent} \in \texttt{Tasks}$ is the task's parent task which may be undefined.
- $\texttt{Task.Ancestors} \subset \texttt{Tasks}$ is the set of all ancestors of a task, i.e. the non-reflexive transitive closure of the `Parent` function.
- $\texttt{Task.ControlFlows} \subset \texttt{ControlFlows}$ denotes the set of outgoing control flows of the task.
- $\texttt{Task.Successors} := \{\texttt{s} | \exists \texttt{c} \in \texttt{Task.ControlFlows} : \texttt{c.Succ} = \texttt{s}\} \subset \texttt{Tasks}$ denotes the set of direct successors of the task.
- $\texttt{Task.Predecessors} := \{\texttt{p} | \exists \texttt{c} \in \texttt{ControlFlows} : \texttt{c.Succ} = \texttt{Task} \wedge \texttt{c.Pred} = \texttt{p}\} \subset \texttt{Tasks}$ denotes the set of direct predecessors of the task.
- $\texttt{Task.TSuccessors} \subset \texttt{Tasks}$ is the set of all transitive successors of the task, i.e. the non-reflexive transitive closure of the `ControlFlows` relation.
- $\texttt{Task.Feedbacks} \subset \texttt{Feedbacks}$ denotes the set of outgoing feedback flows.
- $\texttt{Task.ActiveFeedbacks} := \{\texttt{f} | \texttt{f} \in \texttt{Task.Feedbacks} \wedge \texttt{f.Active} = \texttt{true}\}$ denotes the set of all active outgoing feedback flows.
- $\texttt{Task.FeedbackTargets} := \{\texttt{t} | \exists \texttt{f} \in \texttt{Task.Feedbacks} \wedge \texttt{f.Target} = \texttt{t}\} \subset \texttt{Tasks}$ denotes the set of connected feedback targets.

Documents and data flow The set $\texttt{Documents}$ contains all documents. The set $\texttt{Revisions}$ contains all revisions of documents. The set $\texttt{OutputParameters}$ contains all output parameters and the set $\texttt{InputParameters}$ contains all input parameters. The set $\texttt{DataFlows} \subset \texttt{InputParameters} \times \texttt{OutputParameters}$ contains all data flows. For every task $\texttt{Task} \in \texttt{Tasks}$ the following properties are defined.

- $\texttt{Task.OutputParameters} \subset \texttt{OutputParameters}$
- $\texttt{Task.InputParameters} \subset \texttt{InputParameters}$

For every input parameter $\texttt{InputParameter} \in \texttt{InputParameters}$ the following properties are defined.

- $\texttt{InputParameter.Task} \in \texttt{Tasks}$ the task to which the input parameter belongs.
- $\texttt{InputParameter.DataFlows} \subset \texttt{DataFlows}$ denotes the set of all incoming data flows of the input parameter. There can be more than one incoming data flow for an input parameter when there are several versions of the source task.

For every data flow $\texttt{d} \in \texttt{DataFlows}$ the following attributes are defined.

- $\texttt{d.Source} \in \texttt{OutputParameters}$ denotes the source of the data flow.
- $\texttt{d.Target} \in \texttt{InputParameters}$ denotes the target of the data flow.

For every document $\texttt{Document} \in \texttt{Documents}$ the following properties are defined.

- $\texttt{Document.Revisions} \subset \texttt{Revisions}$ denotes the set of all revisions of the document.

For every output parameter $\texttt{OutputParameter} \in \texttt{OutputParameters}$ the following attributes are defined.

- $\texttt{OutputParameter.DataFlows} \subset \texttt{DataFlows}$ denotes the set of all outgoing data flows of the output parameter.
- $\texttt{OutputParameter.Task} \in \texttt{Tasks}$ is the task to which the output parameter belongs.
- $\texttt{OutputParameter.Document} \in \texttt{Documents}$ is the document which is associated with the output parameter.
- $\texttt{OutputParameter.Revisions} \subset \texttt{OutputParameter.Document.Revisions}$ are the revisions which have been produced in the task of this output parameter.

For every revision $\texttt{Revision} \in \texttt{Revisions}$ the following attributes are defined.

- $\texttt{Revision.IsReleased} \in \{\texttt{True}, \texttt{False}\}$ indicates whether the revision has already been released.

Resource modeling The set $\mathtt{Resources}$ contains all resources in the project team. The set $\mathtt{Roles}$ contains all functional roles defined in the organization. The set $\mathtt{TaskAssignments}$ contains all task assignments. For every task $\mathtt{Task} \in \mathtt{Tasks}$ the following properties are defined.

- $\mathtt{Task.TaskAssignments} \subset \mathtt{TaskAssignments}$ is the set of all task assignments defined for the task.

For every task assignment $\mathtt{a} \in \mathtt{TaskAssignments}$ the following attributes are defined.

- $\mathtt{a.Task} \in \mathtt{Tasks}$ the task for which the task assignment has been defined.
- $\mathtt{a.IsResponsible} \in \{\mathtt{True}, \mathtt{False}\}$ indicates whether the task assignment defines the responsible resource for the task.
- $\mathtt{a.Resource} \in \mathtt{Resources}$ the resource assigned to the task via the task assignment.
- $\mathtt{a.Role} \in \mathtt{Roles}$ the required role specified for the task assignment.

Structural Invariants

Structural invariants are defined for the entities, relationships, and properties of the structural model. They can be divided into invariants concerning tasks and control flow, documents and data flow, and resource modeling. In the following definitions of the structural invariants, it is implicitly assumed that all instances contained in the sets which define the entity types belong to the same project. In that sense, the sets do not exactly represent the entity types but only subsets thereof. The same assumption is made for the definition of behavioral invariants and timing consistency constraints in the following sections.

Tasks and control flow The following structural invariants ensure that the structure of a dynamic task net is consistent with respect to horizontal and vertical task relationships.

Unique naming

$$\forall \mathtt{t}_1, \mathtt{t}_2 \in \mathtt{Tasks}(\mathtt{t}_1.\mathtt{Parent} = \mathtt{t}_2.\mathtt{Parent} \Rightarrow \mathtt{t}_1.\mathtt{Name} \neq \mathtt{t}_2.\mathtt{Name}) \tag{5.1}$$

Two different subtasks of a common parent task must not have the same name.

Task hierarchy

$$\forall \mathtt{t}_1, \mathtt{t}_2 \in \mathtt{Tasks}(\mathtt{t}_1 \in \mathtt{t}_2.\mathtt{Ancestors} \Rightarrow \mathtt{t}_2 \notin \mathtt{t}_1.\mathtt{Ancestors}) \tag{5.2}$$

The hierarchy of tasks within a task net builds a tree structure.

Acyclic control flow

$$\forall \mathtt{t}_1, \mathtt{t}_2 \in \mathtt{Tasks}(\mathtt{t}_1 \in \mathtt{t}_2.\mathtt{TSuccessors} \Rightarrow \mathtt{t}_2 \notin \mathtt{t}_1.\mathtt{TSuccessors}) \tag{5.3}$$

The control flow relationships between tasks must not form a cycle.

Control flow balancing

$$\forall \mathtt{t}_1, \mathtt{t}_2 \in \mathtt{Tasks}(\mathtt{t}_2 \in \mathtt{t}_1.\mathtt{Successors} \wedge \mathtt{t}_1.\mathtt{Parent} \neq \mathtt{t}_2.\mathtt{Parent}) \Rightarrow \mathtt{t}_2.\mathtt{Parent} \in \mathtt{t}_1.\mathtt{Parent}.\mathtt{Successors}) \tag{5.4}$$

If two tasks from different subnets (realizations) are connected by a control flow, their respective parent tasks must be connected by a control flow with the same orientation.

Feedback flow balancing

$$\begin{aligned}\forall \mathtt{t}_1, \mathtt{t}_2 \in \mathtt{Tasks}(\mathtt{t}_1 \in \mathtt{t}_2.\mathtt{FeedbackTargets} \\ \Rightarrow \mathtt{t}_1.\mathtt{Parent} = \mathtt{t}_2.\mathtt{Parent}) \vee \\ \mathtt{t}_1.\mathtt{Parent} \in \mathtt{t}_2.\mathtt{Parent}.\mathtt{FeedbackTargets} \vee \\ \mathtt{t}_1 \in \mathtt{t}_2.\mathtt{Parent}.\mathtt{FeedbackTargets} \vee \\ \mathtt{t}_1.\mathtt{Parent} \in \mathtt{t}_2.\mathtt{FeedbackTargets})\end{aligned} \tag{5.5}$$

Two tasks can be connected by a feedback flow, if they are subtasks of the same task, or if their parent tasks are connected by an equally directed feedback flow. Furthermore, a diagonal feedback flow to or from a subtask is allowed if the parent of the target or source task is connected by a feedback flow with the source or target task respectively (cf. [Kra98, p.80]).

Feedback flow orientation

$$\begin{aligned}\forall \mathtt{t}_1, \mathtt{t}_2 \in \mathtt{Tasks}((\mathtt{t}_1 \in \mathtt{t}_2.\mathtt{FeedbackTargets} \wedge (\mathtt{t}_1.\mathtt{Parent} = \mathtt{t}_2.\mathtt{Parent} \vee \\ \mathtt{t}_1.\mathtt{Parent} \in \mathtt{t}_2.\mathtt{Parent}.\mathtt{FeedbackTargets})) \Rightarrow \mathtt{t}_2 \in \mathtt{t}_1.\mathtt{TSuccessors})\end{aligned} \tag{5.6}$$

There must exist a control flow path from a feedback flow's target to its source, except for diagonal feedback flows.

Redundant control flows

$$\begin{aligned}\forall \mathtt{t} \in \mathtt{Tasks}(\forall \mathtt{c}_1, \mathtt{c}_2 \in \mathtt{t}.\mathtt{ControlFlows}(\mathtt{c}_1 \neq \mathtt{c}_2 \\ \Rightarrow (\mathtt{c}_1.\mathtt{Pred} \neq \mathtt{c}_2.\mathtt{Pred} \vee \mathtt{c}_1.\mathtt{Succ} \neq \mathtt{c}_2.\mathtt{Succ}))\end{aligned} \tag{5.7}$$

There must not exist two different control flows between two tasks.

Redundant active feedback flows

$$\begin{aligned}\forall \mathtt{t} \in \mathtt{Tasks}(\forall \mathtt{f}_1, \mathtt{f}_2 \in \mathtt{t}.\mathtt{Feedbacks}(\mathtt{f}_1 \neq \mathtt{f}_2 \wedge \mathtt{f}_1.\mathtt{Target} = \mathtt{f}_2.\mathtt{Target} \wedge \\ \mathtt{f}_1.\mathtt{Source} = \mathtt{f}_2.\mathtt{Target} \Rightarrow \mathtt{f}_1.\mathtt{Active} = \mathtt{false} \vee \mathtt{f}_2.\mathtt{Active} = \mathtt{false}))\end{aligned} \tag{5.8}$$

There must not exist two different active feedback flows between two tasks.

Documents and data flow With respect to the definition of data flows, the following structural invariants are defined.

Data flow between tasks

$$\forall \mathtt{d} \in \mathtt{DataFlows}(\mathtt{d.Source.Task} \neq \mathtt{d.Target.Task}) \tag{5.9}$$

The input and output parameters which are connected by a data flow do not belong to the same task.

Data flow along task relationships

$$\begin{aligned}\forall \mathtt{d} \in \mathtt{DataFlows}(&\mathtt{d.Target.Task} \in \mathtt{d.Source.Task.Successors} \vee \\ &\mathtt{d.Target.Task} \in \mathtt{d.Source.Task.FeedbackTargets})\end{aligned} \tag{5.10}$$

The tasks which are connected by a data flow are connected by a control or feedback flow with the same orientation.

Unique input

$$\begin{aligned}&\forall \mathtt{i} \in \mathtt{InputParameters}(\forall \mathtt{d}_1,\mathtt{d}_2 \in \mathtt{i.DataFlows}(\mathtt{d}_1 \neq \mathtt{d}_2 \\ &\Rightarrow (\mathtt{d}_1\mathtt{.Source.Task} \in \mathtt{d}_2\mathtt{.Source.Task.PreviousVersions} \vee \\ &\quad \mathtt{d}_2\mathtt{.Source.Task} \in \mathtt{d}_1\mathtt{.Source.Task.PreviousVersions})))\end{aligned} \tag{5.11}$$

There may not be two different data flows between two parameters, and an input parameter cannot have data flows from different tasks defined. If an input parameter has two incoming data flows, the source output parameters belong two different versions of the same task.

Documents and output parameters

$$\begin{aligned}\forall \mathtt{d} \in \mathtt{Documents} \exists ! \mathtt{o} \in \mathtt{OutputParameters}(&\mathtt{o.Document} = \mathtt{d} \wedge \\ &\forall \mathtt{i} \in \mathtt{o.Task.InputParameters}(\mathtt{i.Document} \neq \mathtt{d}))\end{aligned} \tag{5.12}$$

For every document, there exists exactly one producing output parameter in the task net, i.e. the task of the output parameter does not have the document as input.

Revisions and output parameters

$$\forall \mathtt{r} \in \mathtt{Revisions} \exists ! \mathtt{o} \in \mathtt{OutputParameters}(\mathtt{r} \in \mathtt{o.Revisions}) \tag{5.13}$$

For every revision of a document, there exists exactly one output parameter in the task net which produced this revision. This implies that an output parameter cannot be deleted once it has produced a revision.

Resource modeling Structural invariants which apply for modeling resource assignments in dynamic task nets are the following.

Clarified responsibility

$$\begin{aligned}&\forall \mathtt{t} \in \mathtt{Tasks}(\forall \mathtt{a} \in \mathtt{t.TaskAssignments}\\&(\exists \mathtt{b} \in \mathtt{t.TaskAssignments}(\mathtt{b.IsResponsible} = \mathtt{true})))\end{aligned} \tag{5.14}$$

When at least one task assignment has been defined for a task, then there is a task assignment for the responsible resource. As a consequence, the first created task assignment of a task becomes the responsible task assignment, and the responsible task assignment cannot be deleted before other task assignments.

Unique responsible resource

$$\begin{aligned}&\forall \mathtt{t} \in \mathtt{Tasks}(\forall \mathtt{a}_1, \mathtt{a}_2 \in \mathtt{t.TaskAssignments}(\mathtt{a}_1 \neq \mathtt{a}_2\\&\Rightarrow \neg(\mathtt{a}_1\mathtt{.IsResponsible} = \mathtt{true} \wedge \mathtt{a}_2\mathtt{.IsResponsible} = \mathtt{true})))\end{aligned} \tag{5.15}$$

There may not be two different task assignments of a task which both define the responsible resource.

Pre- and Post-Conditions

From the defined invariants, several pre- and post-conditions for structural change operations have been derived. These conditions impose constraints on the structure of a dynamic task net which have to be fulfilled in order to invoke the respective operation.

Operation	**Pre-Condition**
	Post-Condition
$\mathtt{CreateSubtask}(\mathtt{p}, \mathtt{out\ s})$	
	$\forall \mathtt{s}_1, \mathtt{s}_2 \in \mathtt{p.Subtasks}(\mathtt{s}_1\mathtt{.Name} \neq \mathtt{s}_2\mathtt{.Name})$
$\mathtt{AddRealization}(\mathtt{t}, \mathtt{r})$	$\mathtt{undef}(\mathtt{t.Realization})$
	$\mathtt{t.Realization} = \mathtt{r}$
$\mathtt{RemoveRealization}(\mathtt{t})$	
	$\mathtt{undef}(\mathtt{t.Realization})$
$\mathtt{CreateControlFlow}(\mathtt{p}, \mathtt{s})$	$(\mathtt{p.Parent} = \mathtt{s.Parent} \vee$ $\mathtt{s.Parent} \in \mathtt{p.Parent.Successors}) \wedge$ $\mathtt{s} \notin \mathtt{p.Successors} \wedge$ $\mathtt{p} \notin \mathtt{s.TSuccessors}$
$\mathtt{DeleteControlFlow}(\mathtt{c})$	$\neg\exists \mathtt{c}_\mathtt{s} \in \mathtt{ControlFlows}$ $(\mathtt{c}_\mathtt{s}\mathtt{.Pred} \in \mathtt{c.Pred.Subtasks} \wedge$ $\mathtt{c}_\mathtt{s}\mathtt{.Succ} \in \mathtt{c.Succ.Subtasks}$
	$\forall \mathtt{t}_1, \mathtt{t}_2 \in \mathtt{Tasks}(\mathtt{t}_1 \in \mathtt{t}_2\mathtt{.FeedbackTargets}$ $\Rightarrow \mathtt{t}_2 \in \mathtt{t}_1\mathtt{.TSuccessors}$

Operation	**Pre-Condition**	**Post-Condition**
$\mathtt{CreateFeedbackFlow}(s, t, \mathtt{out}\ f)$	$s \in t.\mathtt{TSuccessors} \wedge$ $(t.\mathtt{Parent} = s.\mathtt{Parent} \vee$ $t.\mathtt{Parent} \in s.\mathtt{Parent}.\mathtt{FeedbackTargets} \vee$ $t \in s.\mathtt{Parent}.\mathtt{FeedbackTargets} \vee$ $t.\mathtt{Parent} \in s.\mathtt{FeedbackTargets}) \wedge$ $\neg \exists f' \in s.\mathtt{ActiveFeedbacks}(f'.\mathtt{Target} = t)$	$f.\mathtt{IsActive} = \mathtt{true} \wedge f.\mathtt{source} = s \wedge$ $f.\mathtt{target} = t$
$\mathtt{DeleteOutputParameter}(o)$	$o.\mathtt{Revisions} = \emptyset$	
$\mathtt{CreateDataFlow}(o, i)$	$(\exists c \in o.\mathtt{Task}.\mathtt{ControlFlows}$ $(c.\mathtt{Succ} = i.\mathtt{Task}) \vee$ $\exists f \in o.\mathtt{Task}.\mathtt{ActiveFeedbacks}$ $(f.\mathtt{Target} = i.\mathtt{Task})) \wedge$ $\neg \exists d \in \mathtt{DataFlows}(d.\mathtt{Target} = i \wedge$ $d.\mathtt{Source}.\mathtt{Task} \notin o.\mathtt{Task}.\mathtt{PreviousVersions})$	
$\mathtt{CreateTaskAssignment}(t, \mathtt{out}\ a)$		$(\exists a' \in t.\mathtt{TaskAssignments}$ $(a'.\mathtt{IsResponsible} = \mathtt{true})) \wedge$ $(\neg \exists a_1, a_2 \in t.\mathtt{TaskAssignments}(a_1 \neq a_2 \wedge$ $a_1.\mathtt{IsResponsible} = \mathtt{true} \wedge$ $a_2.\mathtt{IsResponsible} = \mathtt{true}))$
$\mathtt{DeleteTaskAssignment}(a)$	$a.\mathtt{IsResponsible} = \mathtt{false} \vee$ $\neg \exists a' \in a.\mathtt{Task}.\mathtt{TaskAssignments}(a' \neq a)$	
$\mathtt{ModifyTaskAssignment}(a)$		$\exists a' \in a.\mathtt{Task}.\mathtt{TaskAssignments}$ $(a'.\mathtt{IsResponsible} = \mathtt{true}) \wedge$ $\neg \exists a_1, a_2 \in a.\mathtt{Task}.\mathtt{TaskAssignments}$ $(a_1 \neq a_2 \wedge a_1.\mathtt{IsResponsible} = \mathtt{true} \wedge$ $a_2.\mathtt{IsResponsible} = \mathtt{true})$
$\mathtt{CreateNewTaskVersion}(t, \mathtt{out}\ t')$	$\neg \exists \hat{t} \in \mathtt{Tasks}(t \in \hat{t}.\mathtt{PreviousVersions})$	$t = t'.\mathtt{PreviousVersion}$

Tabelle 5.1: Pre- and post-conditions for structural change operations.

The defined pre- and post-conditions have been implemented in PROCEED to ensure the structural consistency of a dynamic task net. If pre-conditions of a change operation are not fulfilled or its application would lead to a violation of a post-condition, then the operation is prohibited. As a consequence, a consistent dynamic task net cannot be transformed by a structural change operation into an inconsistent state.

5.2 Behavioral Model

The behavioral model specifies the available execution states of tasks and the allowed state transitions. Furthermore, it defines additional constraints for structural change operations which take the execution states of tasks into account. In particular, the creation of feedback flows depends on the execution states of the connected tasks.

5.2.1 Life Cycle of a Task

Every task in a dynamic task net has an execution state. The finite state machine depicted in Figure 5.12 defines the *life cycle* of a task. It defines the possible *execution states* and the generally allowed *state transitions* independent of the task's context. The boxes represent task states while the edges represent transitions between these states. Every edge is labeled with the name of the transition.

InDefinition is the initial state of a task. In this state, the properties of the task can be set, resources can be assigned and the realization of the task can be elaborated. When a task is completely defined, its state is changed to Waiting. If a responsible resource has been assigned to the task, this resource can start the task so that it becomes active. Otherwise, if only the required role for the responsible resource has been specified, a resource who can play this role can pick up the task and start it.

The state Active indicates that the work defined by the task is currently performed by the assigned resources. A task can be temporarily suspended. When a task is in the state Suspended, no work is performed on the task and no working hours can be booked on the task. In PROCEED, property values of a started task can be changed when the task is in the states Active or Replanning. However, only in the state Replanning, a task can be structurally changed, i.e. the realization of the task can be modified. The execution state Replanning had been introduced in DYNAMITE in [Kra98] but abandoned in [Sch02]. In PROCEED, the Replanning state has been reintroduced because it is particularly useful for rescheduling dynamic task nets.

A task can be aborted from each of the states Active, Suspended or Replanning. The final state Failed indicates that the task has been unsuccessfully terminated. On the other hand, if the work of an active task has been successfully completed, the task is committed whereby its state is changed to Done. The execution state Skipped has been introduced in PROCEED. This extension was required to cover certain cases during the enactment of workflow-managed tasks which will be described in Section 6.3. In DYNAMITE, it was not possible to skip the execution of a task completely without starting it in the first place. When a task is skipped in PROCEED, it is considered as successfully terminated although the work has not been done. A task can only be skipped when it has not been started yet. A running task has to be either committed or aborted.

Three subsets of the set of states are defined which correspond to the three different phases of the life cycle of a task: *Preparing*, *Running* and *Terminated*. A

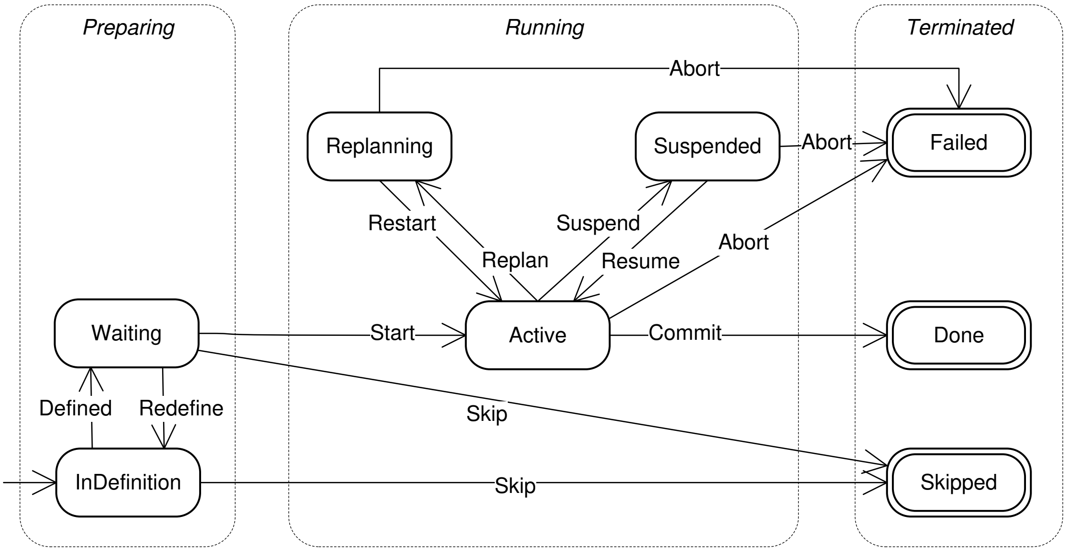


Abbildung 5.12: Finite state machine defining the life cycle of a task.

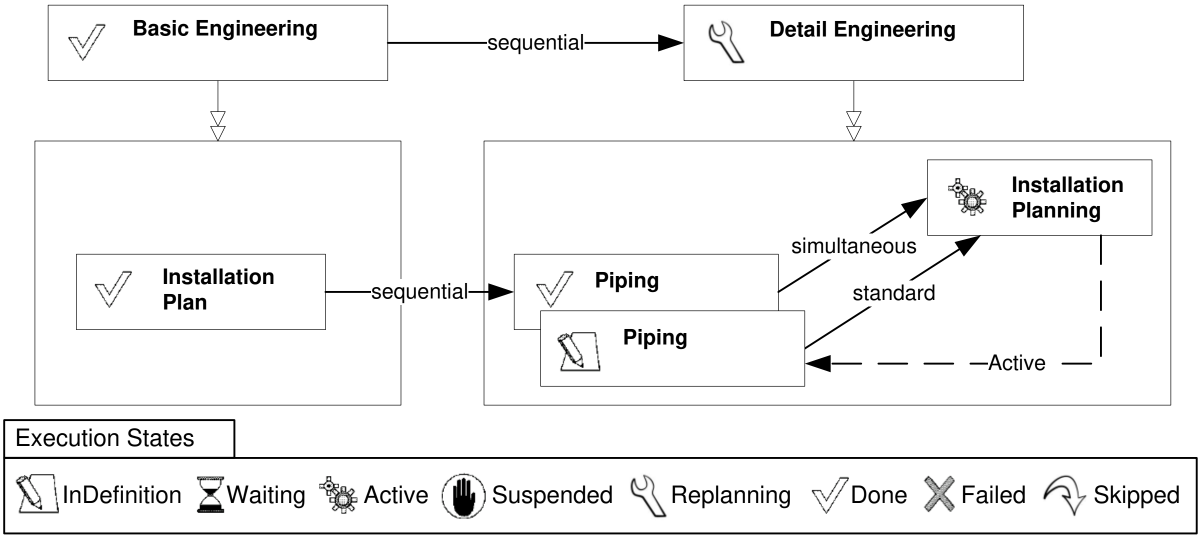


Abbildung 5.13: Example for an enacted dynamic task net.

task is *preparing* if it is either in the state InDefinition or Waiting, i.e. it has not been started yet. If the task is active or temporarily suspended or replanning, it is considered as *running*. Finally, a task can be *terminated*, either successfully if it has been committed or skipped, or unsuccessfully if its execution failed.

The property values of a task may only be changed in the states InDefinition, Active and Replanning. A task which has one of these states is called *editable*. Structural changes to the realization of a task are only allowed if the task is in one of the states InDefinition or Replanning. When a task is in one of these states it is considered as *plannable*.

In Figure 5.13, an example of an enacted dynamic task net is depicted. Several behavioral aspects are visualized in addition to the task net structure. The control flows are labeled with their semantics (cf. Section 4.3.2) and the tasks have execution states. The symbols which are used to represent the execution states of tasks are defined in the legend at the bottom of the figure.

5.2.2 Behavioral Constraints

The finite state automaton defined in Section 5.2.1 already constrains the allowed state transitions of a task. In addition to that, further constraints are imposed by the context of a task which includes the parent task, the subtasks and the predecessors and successors with respect to control and feedback flow relationships.

Definitions

For the definition of behavioral invariants, some additional definitions are required. The set $\mathtt{States} := \{\mathtt{InDefinition}, \mathtt{Waiting}, \mathtt{Active}, \mathtt{Suspended}, \mathtt{Replanning}, \mathtt{Failed}, \mathtt{Skipped}, \mathtt{Done}\}$ contains all available execution states of a task. The following subsets of the set $\mathtt{States}$ are defined.

- $\mathtt{Preparing} := \{\mathtt{InDefinition}, \mathtt{Waiting}\}$
- $\mathtt{Running} := \{\mathtt{Active}, \mathtt{Suspended}\}$
- $\mathtt{Terminated} := \{\mathtt{Failed}, \mathtt{Skipped}, \mathtt{Done}\}$
- $\mathtt{Plannable} := \{\mathtt{InDefinition}, \mathtt{Replanning}\}$
- $\mathtt{Editable} := \{\mathtt{InDefinition}, \mathtt{Replanning}, \mathtt{Active}\}$

For every task $\mathtt{Task} \in \mathtt{Tasks}$ the following property is defined.

- $\mathtt{Task.State} \in \mathtt{States}$ denotes the current execution state of the task.

For every control flow $\mathtt{c} \in \mathtt{ControlFlows}$ the following attribute is defined.

- $\mathtt{c.Semantics} \in \{\mathtt{Standard}, \mathtt{Simultaneous}, \mathtt{Sequential}\}$ is the semantics of the control flow.

For reasons of simplification, the following set-valued properties are defined for a task $\mathtt{Task} \in \mathtt{Tasks}$.

- $\mathtt{Task.StdCFs} := \{\mathtt{c} | \mathtt{c} \in \mathtt{Task.ControlFlows} \land \mathtt{c.Semantics} = \mathtt{Standard}\}$
- $\mathtt{Task.SimCFs} := \{\mathtt{c} | \mathtt{c} \in \mathtt{Task.ControlFlows} \land \mathtt{c.Semantics} = \mathtt{Simultaneous}\}$
- $\mathtt{Task.SeqCFs} := \{\mathtt{c} | \mathtt{c} \in \mathtt{Task.ControlFlows} \land \mathtt{c.Semantics} = \mathtt{Sequential}\}$

Behavioral Invariants

The behavioral invariants which are defined in the following have been in large part adopted from DYNAMITE [Kra98, Sch02]. However, the defined invariants deviate from those defined in [Sch02] in that the execution states *RePlanning* and *Skipped* have been added to the finite state automaton. Additional constraints have been defined which refer to the task assignment for the responsible resource of a task.

Several invariants are defined with respect to vertical task relationships, i.e. the relation of a task to its parent and its subtasks.

- A task that has never been started cannot have a terminated parent task, and the subtasks must also be preparing.

$$\begin{aligned}\texttt{Task.State} \in \texttt{Preparing} \Rightarrow \big(\forall \texttt{t} \in \texttt{Task.Subtasks}(\texttt{t.State} \in \texttt{Preparing}) \wedge \\ \neg(\texttt{Task.Parent} \in \texttt{Terminated})\big)\end{aligned} \tag{5.16}$$

- A task that is running and not suspended has to have a running but not suspended parent task. This constraint demands that the subtasks of a suspended task may not be active or replanning.

$$\begin{aligned}&(\texttt{Task.State} = \texttt{Active} \vee \texttt{Task.State} = \texttt{Replanning}) \Rightarrow \\ &(\texttt{Task.Parent.State} = \texttt{Active} \vee \texttt{Task.Parent.State} = \texttt{Replanning})\end{aligned} \tag{5.17}$$

- The parent task of a suspended task has to be running.

$$\texttt{Task.State} = \texttt{Suspended} \Rightarrow (\texttt{Task.Parent.State} \in \texttt{Running}) \tag{5.18}$$

- If a task is terminated, all subtasks have to be terminated as well and the parent task must not be preparing. This implies that a task may only be skipped when the parent task has been started or skipped.

$$\begin{aligned}\texttt{Task.State} \in \texttt{Terminated} \Rightarrow \big(\forall \texttt{t} \in \texttt{Task.Subtasks}(\texttt{t.State} \in \texttt{Terminated}) \wedge \\ \neg(\texttt{Task.Parent.State} \in \texttt{Preparing})\big)\end{aligned} \tag{5.19}$$

The last invariant does not demand, that a successfully terminated tasks has only successfully terminated subtasks, i.e. a task can be committed when a subtask has failed. The failure of a subtask simply indicates that the assigned resources were not able to achieve the goals defined for the task. However, additional subtasks can be defined in which resources continue the work of the failed task and rework its results. In contrast to that, the failure of a task in a workflow instance which is enacted in a workflow management system usually leads to the failure of the whole workflow. This is due to the fact that workflow management systems are often used to support fully automatic processes. The failure of the whole workflow can only be prevented by the implementation of so called compensation handlers (cf. Section 3.4). In development processes, the failure of a task does not necessarily mean that the whole process is condemned to failure. The definition of additional tasks which make up for the failed task can be regarded as manual compensation.

Besides vertical task relationships, horizontal task relationships have to be considered as well. Two tasks can be connected by a control flow or a feedback flow. The execution state changes of tasks connected by control or feedback flows are constrained by the following invariants.

- The successors of a task may not be terminated before the task. This applies for all control flow semantics. In particular, it is guaranteed that all predecessors of a milestone task are terminated before the milestone terminates.

$$\texttt{Task.State} \notin \texttt{Terminated} \Rightarrow \forall \texttt{t} \in \texttt{Task.Successors}(\texttt{t.State} \notin \texttt{Terminated}) \tag{5.20}$$

- If a task has not been started yet, then its simultaneous successors may not have been started either.

$$\texttt{Task.State} \in \texttt{Preparing} \Rightarrow \forall \texttt{c} \in \texttt{Task.SimCFs}(\texttt{c.Succ.State} \in \texttt{Preparing}) \tag{5.21}$$

- If a task has not been terminated yet, then its sequential successors may not have been started.

$$\texttt{Task.State} \notin \texttt{Terminated} \Rightarrow \forall \texttt{c} \in \texttt{Task.SeqCFs}(\texttt{c.Succ.State} \in \texttt{Preparing}) \tag{5.22}$$

- If two tasks are connected by an active feedback flow, they both may not be committed.

$$\begin{split}\forall \texttt{f} \in \texttt{FeedbackFlows}(\texttt{f.IsActive} = \texttt{true} \Rightarrow \texttt{f.Source.State} = \texttt{Active} \wedge \\ \texttt{f.Target.State} \notin \texttt{Terminated})\end{split} \tag{5.23}$$

The assignment of a responsible resource to a task constrains its allowed execution state transitions as well.

- To complete the definition of a task, a task assignment for the responsible resource has to be defined with a required role.

$$\begin{split}\texttt{Task.State} = \texttt{Waiting} \Rightarrow \exists \texttt{a} \in \texttt{Task.TaskAssignments}(\texttt{a.IsResponsible} = \texttt{true} \\ \wedge \neg \texttt{undef}(\texttt{a.Role}))\end{split} \tag{5.24}$$

- For a task to be started, the responsible resource has to be assigned to the task.

$$\begin{split}\texttt{Task.State} \in \texttt{Running} \Rightarrow \exists \texttt{a} \in \texttt{Task.TaskAssignments}(\texttt{a.IsResponsible} = \texttt{true} \\ \wedge \neg \texttt{undef}(\texttt{a.Resource}))\end{split} \tag{5.25}$$

Document revisions may also constrain the possible state changes of tasks. A task, which still has an open revision, i.e. a revision which has been created and possibly inspected but not yet approved, may not be committed.

$$\begin{split}\texttt{Task.State} = \texttt{Done} \Rightarrow \neg(\exists \texttt{o} \in \texttt{Task.OutputParameters}(\exists \texttt{r} \in \texttt{o.Revisions} \\ (\texttt{o.IsReleased} = \texttt{false})))\end{split} \tag{5.26}$$

Pre- and Post-Conditions

The constraints which are imposed by the defined invariants on state change operations in a dynamic task net have been translated to pre- and post-conditions for state change operations. Table 5.2 shows the conditions for all possible state changes of a task which also cover the constraints imposed by the finite state machine which defines the life cycle of a task. The post-conditions defined in Table 5.2 are fulfilled after a state change of a task because the following automatic adaptations of the context of the task are performed by PROCEED.

- When a task is suspended, all active and replanning subtasks are suspended as well.
- When a task is resumed, all suspended subtasks are resumed as well to the state they had before the suspension (either *Active* or *Replanning*). This adaptation is not required to re-establish consistency but it is convenient for the user.
- When a task is aborted, all running subtasks are aborted and all preparing subtasks are skipped. Furthermore all outgoing feedback flows are deactivated.
- When a task is skipped, all subtasks are skipped as well which are necessarily all preparing.

5.2.3 Execution States and Structural Change Operations

Several structural change operations are constrained by the execution states of the involved tasks. The corresponding invariants could not be formulated in Section 5.1.4 since the available execution states of a task have been introduced in Section 5.2.1. Therefore, this section draws the connection between the structural and the behavioral model. The constraints imposed by task execution states on structural change operations are directly defined as pre- and post-conditions of the corresponding change operations. The underlying invariants are only informally described. The creation of a feedback flow may require complex adaptations to a dynamic task net to maintain its consistency with respect to behavioral constraints. Therefore, these adaptations are described in detail at the end of this section.

Pre- and Post-Conditions

A task is *editable* if it is in one of the states *InDefinition*, *Active* or *Replanning*, i.e. the property values of the task may be changed. Structural changes to the realization of a task are only allowed if it is *plannable*, i.e. it is in one of the states *InDefinition* or *Replanning*. Furthermore, the execution state of a new task has to be initialized correctly. These constraints determine the pre- and post-conditions for structural change operations listed in Table 5.3 which are evaluated by PROCEED in addition to the structural conditions presented earlier.

Operation	**Pre-Condition**
	Post-Condition
$\texttt{Defined}(\texttt{Task})$	$\texttt{Task.State} = \texttt{InDefinition} \land$ $\exists a \in \text{Task.TaskAssignments}(\text{a.IsResponsible} = \text{true} \land$ $\neg\texttt{undef}(\text{a.Resource}))$
	$\texttt{Task.State} = \texttt{Waiting}$
$\texttt{Redefine}(\texttt{Task})$	$\texttt{Task.State} = \texttt{Waiting}$
	$\texttt{Task.State} = \texttt{InDefinition}$
$\texttt{Start}(\texttt{Task})$	$\texttt{Task.State} = \texttt{Waiting} \land$ $\texttt{Task.Parent.State} \in \{\texttt{Active,Replanning}\} \land$ $(\neg\exists c \in \texttt{ControlFlows}(\texttt{Task} = \text{c.Succ} \land \text{c.Semantics}$ $= \texttt{Simultaneous} \land \text{c.Pred.State} \in \text{Preparing})) \land$ $(\neg\exists c \in \texttt{ControlFlows}(\texttt{Task} = \text{c.Succ} \land \text{c.Semantics}$ $= \texttt{Sequential} \land \text{c.Pred.State} \notin \texttt{Terminated})) \land$ $(\exists a \in \text{Task.TaskAssignments}(\text{a.IsResponsible} = \text{true} \land$ $\neg\texttt{undef}(\text{a.Resource})))$
	$\texttt{Task.State} = \texttt{Active}$
$\texttt{Replan}(\texttt{Task})$	$\texttt{Task.State} = \texttt{Active}$
	$\texttt{Task.State} = \texttt{Replanning}$
$\texttt{Restart}(\texttt{Task})$	$\texttt{Task.State} = \texttt{Replanning}$
	$\texttt{Task.State} = \texttt{Active}$
$\texttt{Suspend}(\texttt{Task})$	$\texttt{Task.State} \in \{\texttt{Active}\}$
	$\forall t \in \text{Task.Subtasks}(\text{t.State} \in \text{Preparing} \lor$ $\text{t.State} = \text{Suspended} \lor \text{t.State} \in \text{Terminated})$
$\texttt{Resume}(\texttt{Task})$	$\texttt{Task.State} = \texttt{Suspended} \land$ $\texttt{Task.Parent.State} \in \{\texttt{Active,Replanning}\}$
	$\texttt{Task.State} \in \{\texttt{Active}\}$
$\texttt{Commit}(\texttt{Task})$	$\texttt{Task.State} = \texttt{Active} \land$ $(\forall t \in \text{Task.Subtasks}(\text{t.State} \in \text{Terminated})) \land$ $(\forall t \in \text{Tasks}(\text{Task} \in \text{t.Successors} \Rightarrow (\text{t.State} \in \text{Terminated})))$ $\land(\neg\exists f \in \text{FeedbackFlows}(\text{f.IsActive} = \text{true} \land$ $(\text{Task} = \text{f.Source} \lor \text{Task} = \text{f.Target}))) \land$ $\neg(\exists o \in \texttt{Task.OutputParameters}(\exists r \in \text{o.Revisions}$ $(\text{o.IsReleased} = \text{false})))$
	$\texttt{Task.State} = \texttt{Done}$
$\texttt{Abort}(\texttt{Task})$	$\texttt{Task.State} \in \text{Running}$
	$\texttt{Task.State} = \texttt{Failed} \land$ $(\forall t \in \text{Task.Subtasks}(\text{t.State} \in \text{Terminated})) \land$ $(\forall f \in \text{Task.Feedbacks}(\text{f.IsActive} = \text{false}))$
$\texttt{Skip}(\texttt{Task})$	$\texttt{Task.State} \in \text{Preparing}$
	$\texttt{Task.State} = \texttt{Skipped} \land$ $(\forall t \in \text{Task.Subtasks}(\text{t.State} = \text{Skipped}))$

Tabelle 5.2: Pre- and post-conditions for state change operations.

Operation	**Pre-Condition**
	Post-Condition
CreateSubtask(p, out s)	$\texttt{p.State} \in \texttt{Plannable}$
	$\texttt{s.State} = \texttt{InDefinition}$
DeleteSubtask(p, s)	$\texttt{p.State} \in \texttt{Plannable} \wedge \texttt{s.State} \in \texttt{Preparing}$
AddRealization(t, r)	$\texttt{t.State} = \texttt{InDefinition}$
RemoveRealization(t)	$\texttt{t.State} = \texttt{InDefinition}$
CreateControlFlow(p, s, out c)	$\texttt{p.Parent.State} \in \texttt{Plannable} \wedge$ $\texttt{s.Parent.State} \in \texttt{Plannable} \wedge$ $(\neg(\texttt{s.State} \in \texttt{Terminated}) \vee$ $(\texttt{p.State} \in \texttt{Terminated}))$
	$\texttt{c.Semantics} = \texttt{Standard}$
DeleteControlFlow(c)	$\texttt{c.Pred.Parent.State} \in \texttt{Plannable} \wedge$ $\texttt{c.Succ.Parent.State} \in \texttt{Plannable}$
ModifyControlFlow(c, s)	$\texttt{c.Pred.Parent.State} \in \texttt{Plannable} \wedge$ $\texttt{c.Succ.Parent.State} \in \texttt{Plannable}$
	$\texttt{c.Semantics} = \texttt{s} \wedge$ $(\neg(\texttt{c.Succ.State} \in \texttt{Terminated}) \vee$ $\texttt{c.Pred.State} \in \texttt{Terminated}) \wedge$ $(\texttt{c.Semantics} = \texttt{Simultaneous} \Rightarrow$ $(\texttt{c.Pred.State} \notin \texttt{Preparing} \vee$ $\texttt{c.Succ.State} \in \texttt{Preparing})) \wedge$ $(\texttt{c.Semantics} = \texttt{Sequential} \Rightarrow$ $(\texttt{c.Pred.State} \in \texttt{Terminated} \vee$ $\texttt{c.Succ.State} \in \texttt{Preparing}))$
CreateFeedbackFlow(s, t, out f)	$\texttt{s.Parent.State} \in \texttt{Plannable} \wedge$ $\texttt{t.Parent.State} \in \texttt{Plannable} \wedge$ $\texttt{s.State} = \texttt{Active} \wedge$ $\texttt{t.state} \in \texttt{Running} \cup \texttt{Preparing}$
	$\texttt{f.IsActive} = \texttt{true}$
CreateOutputParameter(t)	$\texttt{t.State} \in \texttt{Editable}$
DeleteOutputParameter(o)	$\texttt{o.Task.State} \in \texttt{Editable}$
CreateDataFlow(o, i)	$\texttt{o.Task.Parent.State} \in \texttt{Plannable} \wedge$ $\texttt{i.TaskParent.State} \in \texttt{Plannable}$
DeleteDataFlow(d)	$\texttt{d.Source.Task.Parent.State} \in \texttt{Plannable} \wedge$ $\texttt{d.Target.Task.Parent.State} \in \texttt{Plannable}$

Operation	**Pre-Condition**
	Post-Condition
$\mathtt{ModifyTask(t)}$	$\mathtt{t.State} \in \mathtt{Editable}$
$\mathtt{CreateTaskAssignment(t, out\ a)}$	$\mathtt{t.State} \in \mathtt{Editable}$
$\mathtt{DeleteTaskAssignment(a)}$	$\mathtt{a.Task.State} \in \mathtt{Editable} \wedge$ $(\mathtt{a.IsResponsible} = \mathtt{false} \vee$ $\mathtt{a.Task.State} = \mathtt{InDefinition})$
$\mathtt{ModifyTaskAssignment(a)}$	$\mathtt{a.Task.State} \in \mathtt{Editable}$
$\mathtt{CreateNewTaskVersion(t, out\ t')}$	$\mathtt{t.Parent.State} \in \mathtt{Plannable} \wedge$ $\mathtt{t.State} \in \mathtt{Terminated}$
	$\mathtt{t'.State} = \mathtt{InDefinition}$
$\mathtt{ProduceRevision(o)}$	$\mathtt{o.Task.State} = \mathtt{Active}$

Tabelle 5.3: Behavioral pre- and post-conditions for structural change operations.

Feedback Handling

The pre-condition for the creation of a feedback flow demands, that the target task is not terminated. In the formal notation used for the conditions, different versions of the same task are regarded as two distinct tasks. To define a feedback flow which targets a terminated task, a new version of the task has to be created first.

Several structural and behavioral constraints apply for the creation of a feedback flow between two tasks. In particular, the target of a feedback flow may not be terminated, which is why a new version of a terminated task is created when a feedback is created which has the terminated task as its target. In the UML class diagram of Figure 5.2, the association NextVersion is defined for the class Task. Two objects of the class which represent subsequent versions of the same task are connected via this association. New task versions have to be created for all terminated successors as well, in order to reach a consistent enactment state of the dynamic task net.

All new versions of tasks are connected by control flows with the same semantics as defined between the old versions of the tasks. However, the semantics of a control flow from a new version of a task to a running successor always has the standard semantics. When the parent task of a versioned task is already terminated as well, a new version has to be created for this task too. The realization of a new task version can be empty. However, if the old version of the task defined a complex subprocess, it is convenient for the user to have the subnet automatically created according to the old version of the task.

Figure 5.14 shows an example which incorporates all aspects of task versioning

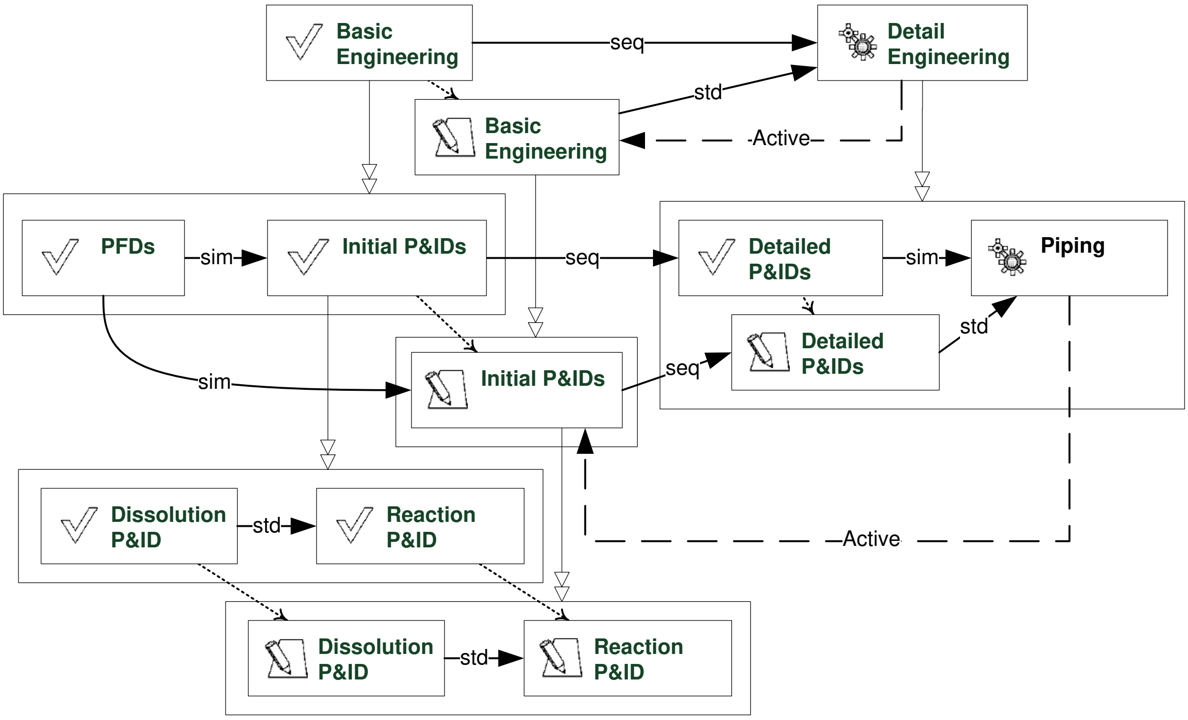


Abbildung 5.14: Versioning of a terminated task.

due to the creation of a feedback flow. Different versions of a task are connected by dotted arrows directed from the older version to the new version. The task Initial P&IDs has been terminated earlier. During the execution of the task Piping, errors are detected in the results of the task Initial P&IDs. This requires the creation of a new task version of the task Initial P&IDs and the definition of a feedback flow from Piping to the new task version. Since the parent task Basic Engineering is already terminated and no feedback has been defined on the level of the parent tasks yet, first, a new version of the task Basic Engineering is created and defined as the target of a feedback flow from Detail Engineering. Not all subtasks of Basic Engineering are versioned but only the task Initial P&IDs. However, the realization of the task Initial P&IDs is completely created anew containing new versions of all subtasks of the previous version. New task versions have to be created for all terminated successor tasks of Initial P&IDs. The semantics of the control flows between new task versions are the same as the semantics of the corresponding control flows between the old versions. Only when the targets of the new control flows have not been versioned and are still running, the semantics is changed to *standard* like for the control flows targeting Detail Engineering and Piping. This adaptation is required for behavioral consistency because the targets of the new control flows are already running. Parameters and data flows are not depicted in Figure 5.14. The new version of a task has by default the same input and output parameters as the previous versions. Naturally, no produced revisions are associated with these parameters yet. Data flows are automatically created according to the data flows defined for the old task versions. New parameters may be introduced for the new task versions, e.g. to accept an error report from a feedback source.

5.3 Timing Model

The DYNAMITE meta-model did not define any properties for tasks, control flows or resources for the purpose of time management in dynamic task nets. The AHEAD prototype did not provide any functionality for scheduling the tasks of a process model instance. Therefore, the TNT meta-model extends several entities which have been adopted from DYNAMITE by the required properties to enable the temporal modeling of processes and the scheduling of tasks. The values of time related properties have to be consistent in a dynamic task net. Therefore, timing consistency constraints are defined which are enforced by the PROCEED system.

5.3.1 Properties for Time Management

For the planning and scheduling of tasks in a dynamic task net, several properties have been defined for the entities task, task assignment, control flow and resource. Figure 5.15 shows the entities which have been extended by new time management properties. Besides the extension of existing entities by new properties, the class `WorkCalendar` has been introduced to define the working days of resources and tasks. Furthermore, the class `WorkloadDistribution` has been introduced to store the daily planned and actual workload of task assignments.

The time management properties can be divided into the following categories.

Planning data refers to all data that is specified in the project planning phase before scheduling, e.g. required workload and budget.

Manually set time constraints are the release and due dates of tasks, but also lag times defined for control flows.

Computed constraint dates result from critical path analysis of a dynamic task net, e.g. earliest possible start times of tasks.

Planned dates are calculated during resource-constrained scheduling and include besides the planned start and end times of tasks also the planned daily workload for task assignments and resources.

Actual and forecasted dates are logged and computed respectively during process enactment. These properties are not part of the timing model but are defined in the monitoring model. In the following, the new entities and properties for time management in dynamic task nets are introduced starting with work calendars and workload distributions.

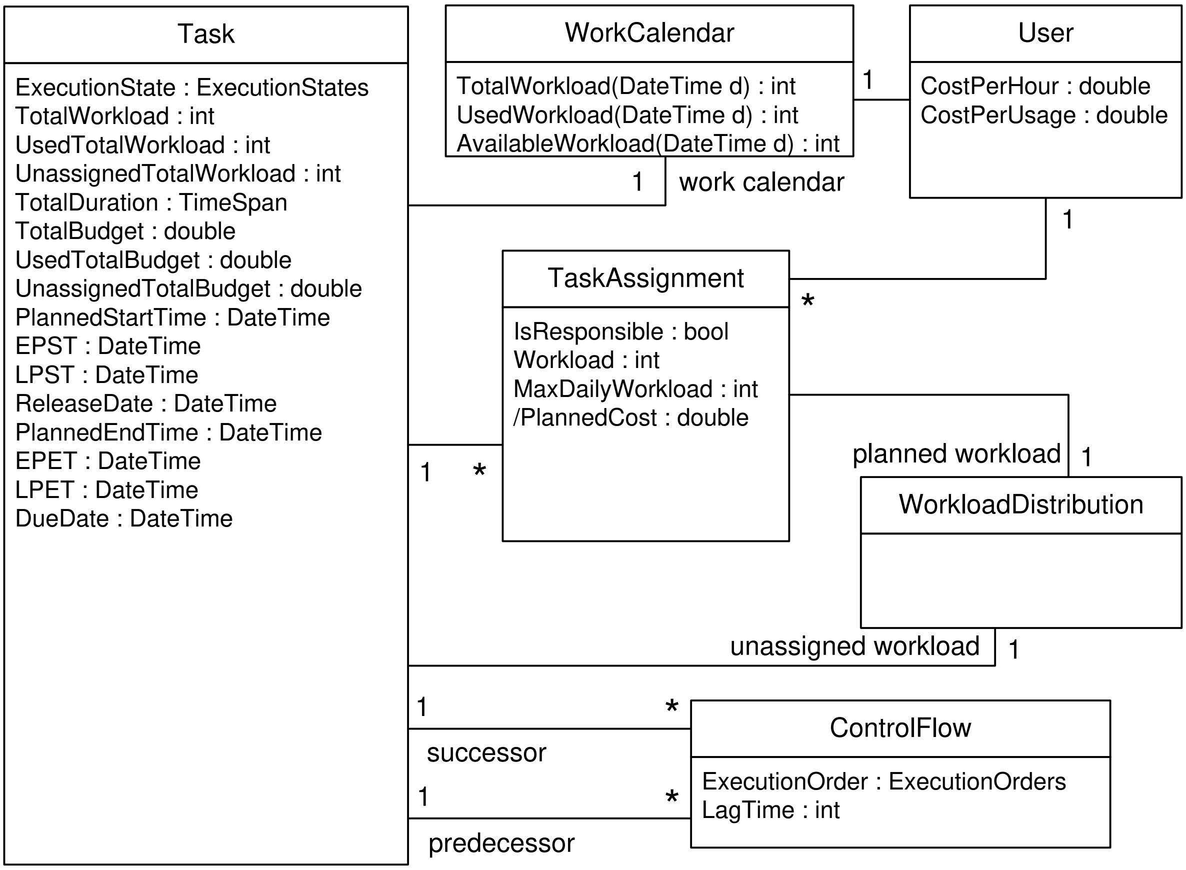


Abbildung 5.15: Entities and properties for time management.

Work Calendars

Because different resources or groups in a project team may have different working times, *work calendars* have been introduced in PROCEED. Work calendars allow to define the working times of tasks and resources individually. For every resource `r` ∈ `Resources`, the property `r.WCal` ∈ `WorkCalendars` is defined where `WorkCalendars` denotes the set of all possible work calendars. Likewise, for every task `t` ∈ `Tasks`, the property `t.WCal` ∈ `WorkCalendars` is defined.

A work calendar defines for every day the number of totally available working hours. This way, weekends, holidays, days of illness and other exceptions can be taken into account during scheduling. Furthermore, a work calendar can store the used working hours per day.

A work calendar defines a regular work week and a set of exceptions from the normal working days. The available work weeks in PROCEED are the 5-day, 6-day and 7-day work week. For design tasks, the common work week is the 5-day week which is used as the default in PROCEED. The alternatives 6-day and 7-day work week are common for construction or transportation tasks. For all dates which deviate from the regular work week of the calendar with respect to the totally available working hours, exceptions are stored in the work calendar. For example, the calendar of a resource may define exceptions for dates which are work days according to the regular work week but where the resource is not available due to vacation, illness, trainings, or work in another project. In the work calendar of a task, general holidays may be stored as exceptions from the regular work week. It is also possible to define exceptions for dates which are no work days according to the regular work week but for which working hours shall be available.

For every work calendar $\mathtt{cal} \in \mathtt{WorkCalendars}$, the following three methods are defined where the set $\mathtt{Dates}$ contains all possible dates.

Total Workload The method $\mathtt{cal.TWL}(\mathtt{d}) \in \mathbb{N}$ returns the number of totally available working hours for the date $\mathtt{d} \in \mathtt{Dates}$.

Used Workload The method $\mathtt{cal.UWL}(\mathtt{d}) \in \mathbb{N}$ returns the number of used working hours for the date $\mathtt{d} \in \mathtt{Dates}$.

Available Workload The method $\mathtt{cal.AWL}(\mathtt{d}) \in \mathbb{N}$ returns the number of available working hours for the date $\mathtt{d} \in \mathtt{Dates}$ which is computed as

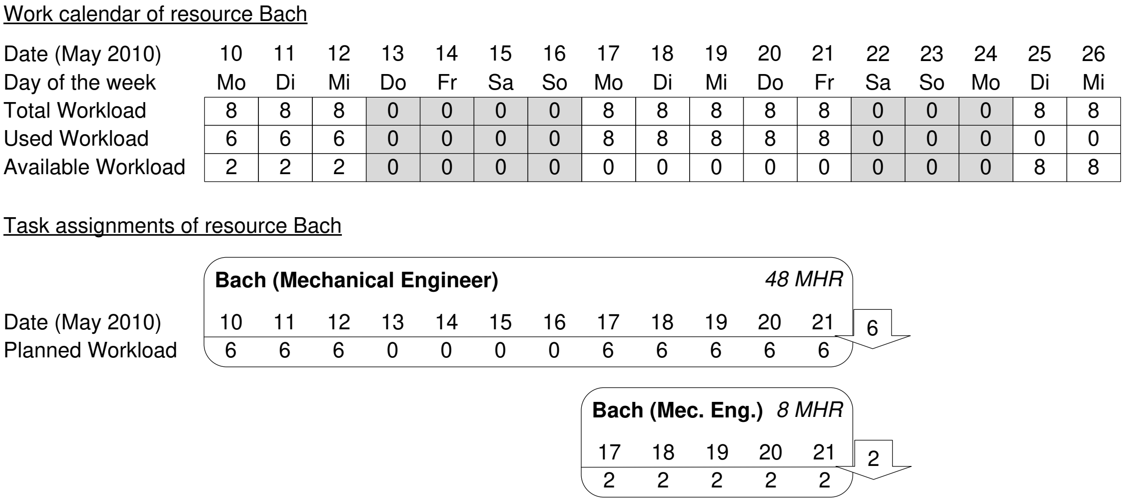
$$\mathtt{cal.AWL}(\mathtt{d}) = \mathtt{cal.TWL}(\mathtt{d}) - \mathtt{cal.UWL}(\mathtt{d})$$

The work calendar of a resource stores for every day the working hours which have been scheduled for the task assignments of the resource for this particular day. The work calendar of a task does not store used working hours. It merely specifies the regular work week and exceptions, e.g. additional holidays.

During resource-constrained scheduling, the workload of task assignments is distributed over several days between the planned start and end times of the corresponding tasks. The property $\mathtt{a.PlannedWorkload} \in \mathtt{WorkloadDistributions}$ is defined for every task assignment $\mathtt{a} \in \mathtt{TaskAssignments}$. It defines the planned working hours for every date.

A workload distribution is basically a mapping of dates to working hours. For every $\mathtt{distr} \in \mathtt{WorkloadDistributions}$ the method $\mathtt{distr.Workload}(\mathtt{d}) \in \mathbb{N}$ is defined which returns the workload for a date $\mathtt{d} \in \mathtt{Dates}$. A workload distribution stores no information about the work calendar which was used to compute the distribution. However, the daily working hours which are planned for a task assignment are added to the used workload of the work calendar of the assigned resource.

Figure 5.16 shows an example for the work calendar of a resource. The depicted cutout of the work calendar of the resource Bach covers several days in May, 2010. The 13th and 24th of May, 2010 are official holidays, and the 14th is a personal holiday of the resource Bach. Therefore, the total workload for these dates is zero while all other days have eight hours total workload each. The resource Bach has two task assignments in the project in the depicted time frame (the corresponding tasks are not depicted). The task assignments have already been scheduled, i.e. the workload has been distributed over several days. The daily planned workload of the task assignments sums up to the used workload in the work calendar of the resource Bach. From this, the available workload is derived which is zero in the work week



Work calendar of resource Bach

Date (May 2010)	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
Day of the week	Mo	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi
Total Workload	8	8	8	0	0	0	0	8	8	8	8	8	0	0	0	8	8
Used Workload	6	6	6	0	0	0	0	8	8	8	8	8	0	0	0	0	0
Available Workload	2	2	2	0	0	0	0	0	0	0	0	0	0	0	0	8	8

Task assignments of resource Bach

Bach (Mechanical Engineer) *48 MHR*

Date (May 2010)	10	11	12	13	14	15	16	17	18	19	20	21
Planned Workload	6	6	6	0	0	0	0	6	6	6	6	6

Bach (Mec. Eng.) *8 MHR*

17	18	19	20	21
2	2	2	2	2

Abbildung 5.16: Example for a work calendar and workload distributions.

from 17th to 21st of May. Therefore, no more tasks can be assigned to the resource Bach in this week.

The usage of work calendars for resources is a way to realize multi-project management. For those days on which a resource is not available in the current project because he is working in another project, an exception is stored in the work calendar specifying less available working hours than the full work day would provide. When the tasks in the current project are scheduled, the unavailability of resources due to their work in other projects is taken into account. This is a practical approach for aligning the schedules of several different projects. It does not lead to a globally optimal schedule over all projects but results in good feasible schedules.

Planning Data

Project planning involves the assignment of resources to tasks. A resource has a limited amount of working hours per day which can be used for different tasks. The cost which arises from the work performed by the resources can be derived from their planned and actual workload and their individual cost rates. Therefore, for every $\texttt{Resource} \in \texttt{Resources}$ the following properties are defined.

Available workload per day The work calendar `Resource.WCal` of the resource returns for every date $\texttt{d} \in \texttt{Dates}$ the total and available working hours via the methods $\texttt{WCal.TWL}(\texttt{d})$ and $\texttt{WCal.AWL}(\texttt{d})$, respectively.

Cost per hour The property $\texttt{Resource.CpH} \in \mathbb{R}^+$ specifies the cost of the resource for one working hour in the currency which is used for budgeting the project.

Cost per usage The property $\texttt{Resource.CpU} \in \mathbb{R}^+$ specifies the cost of the resource for its usage in a task. For the sake of simplicity, the cost per usage is always accrued at the start of the task.

Since task assignments are explicitly modeled in PROCEED, the following properties can be defined for every task assignment $\mathtt{a} \in \mathtt{TaskAssignments}$.

Workload The property $\mathtt{a.Workload} \in \mathbb{N}$ specifies the planned workload for the task assignment in the unit man hours (MHRS).

Planned Cost The property $\mathtt{a.PlannedCost} \in \mathbb{R}^+$ returns the planned costs of the task assignment which are derived from the planned workload and the cost rates of the assigned resource. If no actual resource is assigned yet, the planned cost for a task assignment cannot be determined and is set to zero by default.

In contrast to workload planning, the budget of task assignments cannot be manually defined but is derived from the planned workload and the resource costs. As a consequence of the inherent dynamics in development processes, the assigned resource may change for a task assignment even during the execution of the task. The definition of the planned cost has to take this circumstance into account. Therefore, some additional definitions are required beforehand. For a task assignment $\mathtt{a} \in \mathtt{TaskAssignments}$ the following properties and methods are defined.

- $\mathtt{a.PlannedWorkload.Workload(d)} \in \mathbb{N}$ with $\mathtt{d} \in \mathtt{Dates}$ returns the workload which has been planned for the task assignment for a particular date as defined by the workload distribution.
- $\mathtt{a.Resource(d)} \in \mathtt{Resources}$ returns the resource which is assigned to the task assignment at the date $\mathtt{d} \in \mathtt{Dates}$ which may be a time point in the past, the current date, or a future point in time. For all future dates, the currently assigned resource is returned. In particular, the following equation holds.

  $$\mathtt{a.Resource} = \mathtt{a.Resource(Today)}$$

  where $\mathtt{Today} \in \mathtt{Dates}$ represents the current date.
- $\mathtt{a.Resources} := \{\mathtt{r} \in \mathtt{Resources} | \exists \mathtt{d} \in \mathtt{Dates}(\mathtt{a.Resource(d)} = \mathtt{r})\}$ is the set of all resource which have ever been assigned to the task assignment. This accounts for dynamic plan changes in which a task assignment is transferred from one resource to another.

Using these properties and methods, the value of the derived property $\mathtt{a.PlannedCost}$ is calculated as follows after the task has been scheduled.

$$\begin{aligned}\mathtt{a.PlannedCost} := &\sum_{\mathtt{d} \in \mathtt{Dates}} (\mathtt{a.Resource(d).CpH} \cdot \mathtt{a.PlannedWorkload.Workload(d)}) + \\ &\sum_{\mathtt{r} \in \mathtt{a.Resources}} \mathtt{r.CpU}\end{aligned}$$

When the task has not been scheduled yet, but a resource has already been assigned, the value of the derived property $\mathtt{a.PlannedCost}$ is calculated as follows.

$$\mathtt{a.PlannedCost} := \mathtt{a.Resource.CpH} \cdot \mathtt{a.Workload} + \mathtt{a.Resource.CpU}$$

The prerequisites for task scheduling are good estimates for the required workload and the expected duration of the defined tasks. For this purpose, the following properties can be set for every $\texttt{Task} \in \texttt{Tasks}$.

Total workload The property $\texttt{Task.TotalWorkload} \in \mathbb{N}$ is the estimated and planned workload in man hours which is required to complete the task. It includes the workload of all task assignments and subtasks.

Total duration The property $\texttt{Task.TotalDuration} \in \mathbb{N}$ is the estimated and planned duration of the task. It can be manually set during project planning or it can be derived by scheduling the task assignments and subtasks.

Total budget The property $\texttt{Task.TotalBudget} \in \mathbb{R}^+$ is the overall planned cost of the task including the cost for all task assignments and subtasks.

In PROCEED, it is possible to assign several resources to a task. Complex tasks can have resources assigned as well. The total workload of a task includes the sum of the workload of all task assignments and subtasks but does not necessarily equal it. The total workload can be set manually and may exceed the workload of the subtasks and task assignments which results in a workload buffer. Specifying a workload buffer for a task is a possibility to plan workload which cannot (yet) be assigned to a specific subtask. This is useful for top-down planning and is required for rolling-wave planning where tasks of later project phases are not completely elaborated in the planning phase but only in the execution phase of the project. The sum of the workload of all subtasks and task assignments is the *used total workload* while the remainder from subtracting it from the planned total workload is the *unassigned total workload*.

Definition 5.1 (Used total workload) For a task $\texttt{t} \in \texttt{Tasks}$ the *used total workload* is defined as

$$\texttt{t.UsedTotalWorkload} := \sum_{\texttt{s}\in\texttt{t.Subtasks}} \texttt{s.TotalWorkload} + \sum_{\texttt{a}\in\texttt{t.TaskAssignments}} \texttt{a.Workload} \tag{5.27}$$

Definition 5.2 (Unassigned total workload) For a task $\texttt{t} \in \texttt{Tasks}$, the *unassigned total workload* is defined as

$$\texttt{t.UnassignedTotalWorkload} := \texttt{t.TotalWorkload} - \texttt{t.UsedTotalWorkload} \tag{5.28}$$

A task can for example have a planned total workload of 1200 MHRS although it only has two subtasks with 820 MHRS and 216 MHRS total workload respectively. The complex task therefore has a used total workload of 1036 MHRS and an unassigned total workload of 164 MHRS. This situation is depicted in Figure 5.17.

The unassigned total workload of a task is uniformly distributed over all working days of the task according to the task's work calendar. This workload distribution is used for earned value analysis. The algorithm for resource-constrained scheduling distributes the unassigned total workload of a task. Furthermore, the distribution

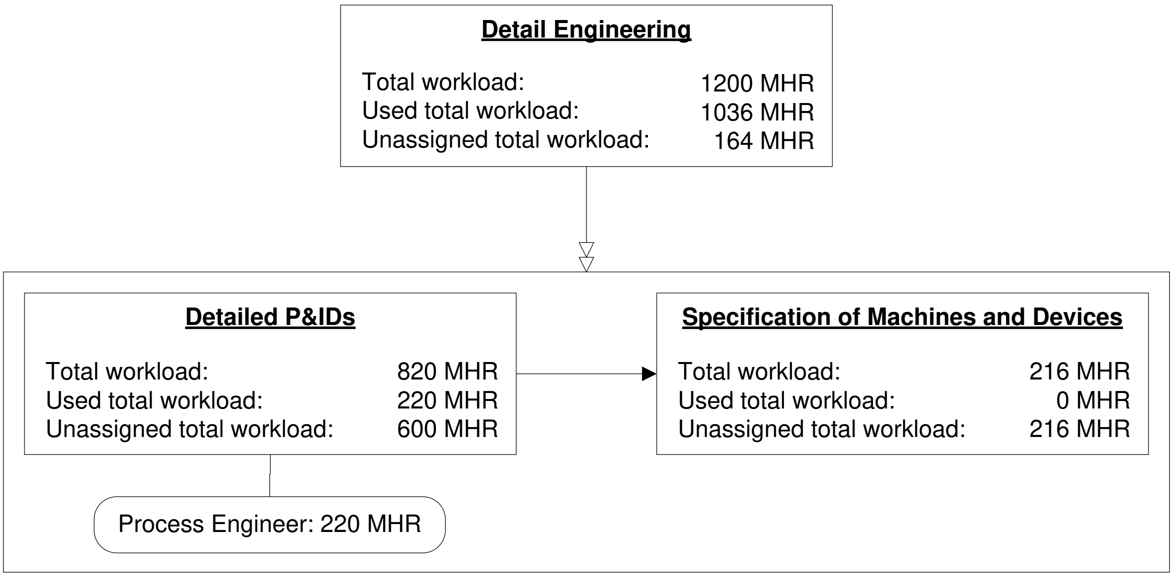


Abbildung 5.17: Planning of total workload for a complex task.

is automatically updated whenever the total duration of the task is changed or the unassigned total workload changes due to modifications to task assignments or subtasks.

In plant engineering projects, experienced estimators estimate the workload of the top-level tasks in the project based on personal experience, market prices and similar projects. The resulting estimates are refined while building the work breakdown structure. At the level of work packages, the duration and resource requirements are estimated in coordination with the lead engineers. The resulting values are aggregated on higher levels and the previously estimated values are adapted. This way, the planning and workload estimation is carried out top-down and bottom-up. PROCEED supports this bidirectional planning process in the following ways. When a subtask is defined for a task and a certain amount of man hours is specified as its workload, these working hours are automatically taken from the total workload of the parent task, i.e. the used total workload is increased and the unassigned total workload is decreased while the total workload stays the same. This way, top-down planning is supported because the workload can be distributed from the respective parent tasks to their subtasks. When the total workload of a subtasks has to be increased due to more accurate estimations, or when an additional subtask has to be created which has not been expected before, then the used total workload may exceed the planned total workload. This inconsistency is resolved by setting the total workload to the new used total workload. This way, bottom-up planning is supported because the workload can be aggregated from the workload of the subtasks.

Besides the planning of workload, a budget has to be planned for every task. It can be set manually by the project manager or another authorized resource. Just like the total workload, the *total budget* of a task may exceed the sum of the budgets of the subtasks. It is common practice in budget planning to define contingency reserves in the budget of a task and not to distribute the whole budget to the subtasks. The total budget of a task includes fixed costs for the task. Since the whole project is also modeled as a task, the base costs of the project can be included in its total budget. The *used total budget* and the *unassigned total budget* are defined analogously to the used and unassigned total workload.

Definition 5.3 (Used total budget) For a task $t \in \mathit{Tasks}$, the *used total budget* is defined as

$$\texttt{t.UsedTotalBudget} := \sum_{\texttt{s}\in\texttt{t.Subtasks}} \texttt{s.TotalBudget} + \sum_{\texttt{a}\in\texttt{t.TaskAssignments}} \texttt{a.PlannedCost} \tag{5.29}$$

Definition 5.4 (Unassigned total budget) For a task $\texttt{t} \in \texttt{Tasks}$, the *unassigned total budget* is defined as

$$\texttt{t.UnassignedTotalBudget} = \texttt{t.TotalBudget} - \texttt{t.UsedTotalBudget} \tag{5.30}$$

The *total duration* of a task is the estimated and planned duration which will presumably be required to complete the task. It is specified as the number of required work days. The duration of the task in terms of calendar days may be much higher due to weekends, holidays, and other resource unavailabilities. The total duration of a task always defines the number of work days, even when it is specified as "n days".

The total duration can be manually set for a task and may be longer than the duration of the scheduled subprocess defined by the realization of the task. In that sense, the total duration is independent of the duration of the subtasks. However, some basic constraints have to be fulfilled, e.g. the constraint that no subtask may have a longer duration than the parent task. If the total duration of a subtask is set to a value that is longer than the total duration of the parent, then the latter has to be adapted, i.e. the total duration of the parent is set to the duration of the prolonged subtask. If no total duration is defined for a task, it is automatically derived during resource-constrained scheduling from the durations of the task assignments and the subprocess. Resource-constrained scheduling may also reveal that the task assignments or subtasks require a longer time frame than defined by the total duration of the parent task when they are scheduled in a time- and resource-feasible way. In this case, the total duration of the parent task is adapted by the scheduling algorithm and the user is informed about the change.

Manually Set Time Constraints

In addition to the estimated workload, budget and duration, the project planner can set *constraint dates* for tasks. For this purpose he can set fixed dates for the following two properties of a task $\texttt{t} \in \texttt{Tasks}$.

Release date The task `t` may not be started before the fixed date specified by the property $\texttt{t.ReleaseDate} \in \texttt{Dates}$.

Due date The task `t` has to be terminated no later than the fixed date specified by the property `t.DueDate` $\in$ `Dates`.

These constraints are taken into account during the scheduling of the tasks in a dynamic task net. If they are violated during the actual execution of the tasks, warnings are shown to the respective resource who is responsible for the subprocess.

For a task assignment `a` $\in$ `TaskAssignments`, the following time constraint can be specified.

Maximal resource usage per day The property `a.MaxDailyWorkload` specifies how many working hours may be scheduled at most per day for the task assignment.

This value is used during resource-constrained scheduling of the task. The workload of the task assignment is distributed over several days in a way that the planned workload for the assigned resource does not exceed the maximal resource usage for those days. This way, tasks can be defined on which a resource needs not work full time but only for some hours per day. As a consequence, several of these tasks can be assigned to and executed by the same resource in parallel for several days. The task assignments which have been presented in Figure 5.16 have a maximal resource usage per day of six and two man hours respectively, which is visualized by the downwards oriented arrows.

Control flows in DYNAMITE were only distinguished by their semantics. For temporal analysis and scheduling it is sometimes necessary to define a *lag time* for a control flow. Therefore, the following property can be set for every control flow `c` $\in$ `ControlFlows`.

Lag time The property `c.LagTime` specifies the minimal time which has to pass between the two events related by the control flow.

Lag times are defined as natural numbers which stand for a number of work days. The lag time of a control flow is always measured with respect to the work calendar of the successor task. This is in line with [Har05].

In case of a sequential control flow, a lag time of x work days requires the target task to start no earlier than x work days after the end time of the source task. In case of a standard control flow, the target task may not end earlier than x work days after the end time of the source task. In case of a simultaneous control flow, the lag time applies to the time span between the respective start times and end times of the connected tasks. Depending on the durations of the tasks this may lead to even larger minimal lag times for the start or end times. In Figure 5.18 on the left side, the duration of the target task Isometries is shorter than the duration of the source task Piping. Hence, the minimal lag time has to be ensured between the end times of the tasks. Scheduling of the tasks leads to an even larger lag time between the planned start times. The opposite case is depicted on the right side of Figure 5.18 where the smaller lag time exits between the start times of the connected tasks because the duration of Instrumentation is shorter than the duration of Procurement.

Minimal lag times are required to adequately model simultaneous engineering scenarios where two tasks are executed in parallel but are connected by a simultaneous

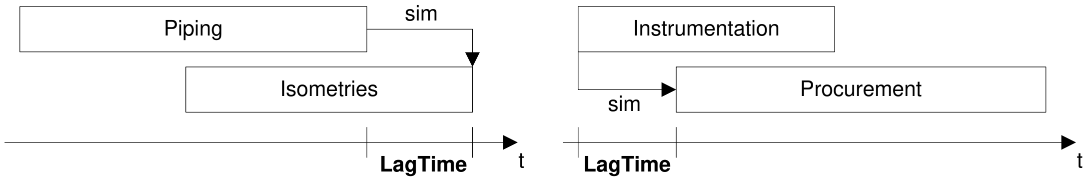


Abbildung 5.18: Lag time for a simultaneous control flow.

or standard control flow. The control flow requires that the target task may not be terminated before the source task. This constraint would also be fulfilled if both tasks would terminate at the same time. However, the intension of the modeled control flow between the tasks is that the final results of the source task are incorporated into the results of the target task which requires a certain amount of time. Therefore, adequate lag times have to be defined for simultaneous and standard control flows.

In PROCEED, only minimal lag times can be defined but no maximal lag times. Maximal lag times have been found to be not required in addition to fixed due dates for modeling temporal constraints on tasks. In contrast to business processes, the demand of maximal time spans between two events in a process are not common in the context of development processes. While in business cases it is often required that a certain task is completed no later than several time units after the completion of a preceding task, the end times of tasks in development projects are usually constrained by explicit deadlines for the delivery of certain artifacts. Maximal lag times could be used for modeling release and due dates of tasks [DH02, p.43], but release and due dates can be defined explicitly for every task in PROCEED, so that control flows with maximal lag times are not required for this purpose. The task relationships of the precedence diagramming method which is commonly applied for project planning can only define minimal lag times between tasks. The generalized resource-constrained project scheduling problem (cf. Section 3.2.2) considers precedence relations of the PDM as well as task release and due dates. Consequently, a subclass of this problem class is addressed by the scheduling algorithm implemented in PROCEED.

Computed Constraint Dates

The automatic scheduling in PROCEED is divided into two phases. In the first phase, critical path analysis is performed on a given task net. The earliest and latest possible start and end times are calculated and saved for every task. For these *computed constraint dates* the following properties are defined for a task $\mathtt{t} \in \mathtt{Tasks}$.

Earliest possible start time The task `t` may not be started before the date specified by the property $\mathtt{t.EPST} \in \mathtt{Dates}$

Latest possible start time The task `t` may not be started later than the date specified by the property $\mathtt{t.LPST} \in \mathtt{Dates}$

Earliest possible end time The task `t` may not be terminated before the date specified by the property $\texttt{t.EPET} \in \texttt{Dates}$

Latest possible end time The task `t` may not be terminated later than the date specified by the property $\texttt{t.LPET} \in \texttt{Dates}$

Planned Dates

After critical path analysis, resource-constrained scheduling is performed, which leads to *planned start and end dates* for tasks and *planned daily workload* for task assignments. For every task $\texttt{t} \in \texttt{Tasks}$, the following properties are defined.

Planned start time The property $\texttt{t.PlannedStartTime} \in \texttt{Dates}$ defines the date on which the task should be started according to the schedule.

Planned end time The property $\texttt{t.PlannedEndTime} \in \texttt{Dates}$ defines the date on which the task should be committed according to the schedule.

The workload of all task assignments of a task is distributed over several work days which lie between the planned start and end times of the task. For every day, the *planned workload per day* is specified individually. For a task assignment $\texttt{a} \in \texttt{TaskAssignments}$, the workload distribution `a.PlannedWorkload` is defined which stores the planned workload per day for the task assignment.

- $\texttt{a.PlannedWorkload.Workload}(\texttt{d}) \in \mathbb{N}$ returns the workload which has been planned for the task assignment for a date $\texttt{d} \in \texttt{Dates}$.

The *duration of a task assignment* is only implicitly defined by the time frame in which workload is scheduled.

Default Values

During initial planning of a dynamic task net as well as during dynamic replanning at project runtime, structural changes and changes to timing properties are made. When a user of PROCEED creates a new task in a dynamic task net or defines a feedback flow resulting in a new version of a task, several property values of this new task (version) have to be set, so that it is correctly embedded into its context. Furthermore, the property values of a control flow have to be set during its creation. For some properties of a new task (version) and a new control flow, default values are defined which are set automatically by PROCEED if the user does not provide custom values.

New task Table 5.4 shows the default values for the properties of a new task. Alternative values can be provided by the user upon the creation of the task, which override the default values. The release date of a new task is set to the release date of the parent task if the latter is defined. Otherwise it remains undefined. The same holds for the due date of a new task. If the user does not specify values for the total

Property	**Default Value**
`Task.ReleaseDate`	`Task.Parent.ReleaseDate`
`Task.DueDate`	`Task.Parent.DueDate`
`Task.TotalWorkload`	0
`Task.TotalBudget`	0
`Task.TotalDuration`	1 day
`Task.EPST`, `Task.EPET` `Task.LPST`, `Task.LPET`	undefined
`Task.PlannedStartTime`	$\mathtt{max}\{\mathtt{Today}, \mathtt{Task.Parent.PlannedStartTime}\}$
`Task.PlannedEndTime`	`Task.PlannedStartTime`

Tabelle 5.4: Default property values for a new subtask.

workload, budget and duration of a new task, they are initialized with zero man hours, zero amount of money, and one day, respectively. The computed constraint dates are not set until the task is scheduled for the first time. The planned start time is set to the maximum of the current date ($\mathtt{Today} \in \mathtt{Dates}$) and the planned start time of the parent task. This way, the task is preliminary scheduled in a consistent way. If the current date is not a working day, the next working day in the work calendar of the task is used instead. Since the total duration is set to one day by default, the planned end time is set to the same date as the planned start time.

Definition of task assignments In PROCEED, the duration of a task and the total workload can be defined independently. The total workload of a task can be distributed to task assignments and subtasks. Supporting tool functionality is provided for the definition of tasks and task assignments. The following functions ease the definition of task assignments.

- When the user has defined the total duration and the total workload of a task, PROCEED computes the number of resources which would be necessary to perform the work in the specified time frame for the project's default maximal resource usage per day. For example, for a workload of 800 MHRS over 10 work days, 10 resources would be required assuming a maximal resource usage of 8 MHRS per day. The computed value can be used for planning the task assignments and subtasks.

- When a user creates a new task assignment for a task, PROCEED computes the maximal workload which a resource can perform over the total duration of the task considering the maximal daily workload specified for the task assignment. This maximal planned workload can be used to create a task assignment which lasts for the whole duration of the task. A larger value for the planned workload of the task assignment would increase the total duration of the task during resource-constrained scheduling.

- Task assignments for additional resources can be used to assign resources to tasks which have no subtasks yet. At a later planning stage, these task assignments may have to be transformed to regular subtasks. PROCEED offers a one-click mechanism for this transformation.

New task version A new version of a task is created when changes to the results of the task are required after its termination. The new task version is inserted into the dynamic task net and is connected to predecessor and successor tasks as described in Section 5.2. The default values for the properties of a new task version are slightly different to the values for a new task. Table 5.5 shows the default values for the properties of a new task version. The release date of the new task version is set to the current date. This setting is required to obtain correct earliest possible start and end times for the new task version during critical path analysis. The due date is set to the due date of the parent task. A new version of a task has by default the same task assignments with the same required roles and assigned resources as the previous version. This ensures that the work which has to be continued or revised is performed by the same resources. The assignments can be manually changed by the responsible resource if previously assigned resources are not available. When a new version of a task is created, the user who performed the operation is asked to provide a percentage estimate for the relative required workload, budget and duration for the new task version compared to the previous version. Since the creation of a new task version means, that the results of the previous version have to be revised and reworked, the required workload is usually less than for the previous version. From the estimated percentage value, the total workload, budget and duration of the new task version are computed as well as the planned workload of all task assignments by multiplying it with the values of the previous task version. Since for every subtask a new version is created as well, the specified percentage value is also used to calculate the respective property values of the new versions of the subtasks. If the user does not provide a percentage value for the planning data, a default of 100% is used. The computed constraint dates are set to the values of the parent task, so that the task will be scheduled to a consistent time frame during rescheduling. The new task version is preliminary scheduled for the current date.

New control flow A new control flow relationship has the standard semantics and a time lag of zero by default. Thereby it imposes the least constraints on the dates of the connected tasks. A new control flow which has been created in the course of task versioning, i.e. which connects a new task version with another task, has by default the same semantics and lag time as the control flow which is defined between the previous versions of the tasks. In the case of task versioning, it is often required to manually adapt the lag time of new control flows.

Task termination When a task is terminated, then its actual end time is set to the current date. Furthermore, several property values of the task are adapted depending on its final execution state. These changes cannot be influenced by the user.

Property	**Default Value**
`Task.ReleaseDate`	`Today`
`Task.DueDate`	`Task.Parent.DueDate`
`Task.TotalWorkload`	User estimated percentage of workload of previous version
`Task.TotalBudget`	User estimated percentage of budget of previous version
`Task.TotalDuration`	User estimated percentage of duration of previous version
`Task.EPST`, `Task.EPET` `Task.LPST`, `Task.LPET`	according values of parent task
`Task.PlannedStartTime`	`Today`
`Task.PlannedEndTime`	`Task.PlannedStartTime` + `Task.TotalDuration` w.r.t. the work calendar of the task

Tabelle 5.5: Default property values for a new task version.

- Committed and aborted task
  - The planned end time is set to the actual end time, and the total duration is adapted accordingly.
  - Planned workload which has been distributed for dates after the actual end time is deleted and subtracted from the total workload. This applies to the planned workload of all task assignments and the unassigned total workload of the task.
  - Accordingly, the planned costs for these dates are subtracted from the total budget.
- Skipped task
  - The planned start and end times are set to the current date.
  - The total duration is set to zero work days.
  - Planned workload which has been distributed for task assignments and the task is deleted and the total workload is set to zero work hours.
  - Actual resources are removed from the task assignments.
  - The total budget is set to zero.

The planned end time and total duration of a committed or aborted task are automatically adapted to the actual values. This constitutes an automatic adaptation of the plan to the actual performance. In contrast to running tasks, the user does not have the choice whether he wants to align the plan to the actual performance or not. The scheduling algorithm which will be described in Chapter 7 uses the planned dates of terminated tasks to schedule their successors. If the planned end time of a committed or failed task would not be set to the actual end time, then the successors would possibly be scheduled to late.

In contrast to the planned end time and the total duration, the distributed planned workload of a committed or aborted task is not aligned to the actual values. The planned workload which has been scheduled for dates later than the actual end time of the task is deleted because the work will not be performed anymore. However, the planned daily workload which has been distributed over the actual duration of the task remains unchanged and may therefore deviate from the actual daily workload.

A task which is skipped is not removed from the dynamic task net for reasons of traceability. However, its planning data is deleted so that it does not contribute to the used total workload and budget of the parent task anymore. Since a skipped task is terminated, it will not be (re)scheduled. The planned start and end times are set to the current date, so that successors may be started or terminated depending on the defined control flows.

The automatic changes to the the planned end time, the total workload, and the total budget of a terminated task override the originally planned values. In these cases, the traceability of plan changes is not provided by the dynamic task net. Therefore, a project data warehouse is used to store successive plan states over the duration of a project. This project data warehouse will be introduced in Chapter 8.

5.3.2 Timing Consistency Constraints

Besides structural and behavioral constraints which were already defined in DYNAMITE, dynamic task nets in PROCEED must adhere to *timing consistency constraints* as well. There are different types of constraints which are related to the timing properties of tasks. An example of a timing consistency constraint is that the earliest possible start time of a task always has to be less or equal to the latest possible start time. An example of a different constraint is that the actual end time of a task must not exceed the planned end time. The first constraint is enforced by PROCEED but not the latter since it is common that tasks are delayed in a project. The strict enforcement of the second constraint would render PROCEED useless for practice. A violation of the second constraint merely indicates a deviation of the actual performance of a task from the plan. These deviations are covered by monitoring constraints which will be described in Section 5.4.

Timing consistency constraints must not be violated, since this would lead to clearly inconsistent management data. Therefore, timing consistency constraints are also called *strict constraints* to be distinguished from the monitoring constraints which may be violated and are therefore also called *non-strict constraints*. All timing consistency constraints have in common that they can only be violated due to user actions, in contrast to monitoring constraints which may also be violated just because time proceeds. Violations of timing consistency constraints refer to:

- Inconsistent planning data
- Inconsistent time constraints and computed constraint dates
- Inconsistent planned dates

- Inconsistencies between planned dates and time constraints or computed constraint dates

The first two types of strict constraints may never be violated. When a user action would lead to an inconsistent state of the dynamic task net, PROCEED proposes an alternative or additional change which resolves the inconsistency. The user may accept this change or cancel his action. The latter two types of timing consistency constraints may be violated temporarily, but only during the (re)planning of the corresponding subprocess, i.e. the planned dates of tasks may only be inconsistent with each other, with time constraints, or with computed constraint dates as long as the respective parent task is in the state `InDefinition` or `Replanning`.

In the following, the timing consistency constraints are formally defined. Two dates $\mathtt{d}_1, \mathtt{d}_2 \in \mathtt{Dates}$ can be compared using the operators $<, \leq, >, \geq$ which determine if a date is earlier or later than the other respectively. Time spans like durations and lag times are defined as natural numbers which stand for the number of work days. If a time span is subtracted from a date, the resulting date is obtained by going back in time for the specified number of work days with respect to the work calendar of the task or resource for which the date is defined. As described earlier, the lag time of a control flow is always measured with respect to the work calendar of the successor task. Several properties are compared in the following constraints which may have undefined values, e.g. the due date of a task may be undefined. In this case, no constraint violation can be determined. Therefore, the corresponding formulas evaluate to `true` if one of the compared dates is undefined. A formula $(\mathtt{e}_1.\mathtt{p}_1 \ \mathtt{op} \ \mathtt{e}_2.\mathtt{p}_2)$ for two properties $\mathtt{p}_1, \mathtt{p}_2$ of two entities $\mathtt{e}_1$ and $\mathtt{e}_2$ and an operator $\mathtt{op} \in \{<, \leq, >, \geq\}$ is actually evaluated as $(\mathtt{undef}(\mathtt{e}_1.\mathtt{p}_1) \vee \mathtt{undef}(\mathtt{e}_2.\mathtt{p}_2) \vee \mathtt{e}_1.\mathtt{p}_1 \ \mathtt{op} \ \mathtt{e}_2.\mathtt{p}_2)$. For reasons of simplification, the additional subformulas for checking whether the properties are defined, are not shown in the following constraints.

The planning data for tasks has to be consistent. The workload, budget and duration of a task may be planned independently of the respective properties of its realization. However, the following timing consistency constraints have to be fulfilled at any time, even during the (re-)planning of a task.

$$\mathtt{Task.TotalWorkload} \geq \mathtt{Task.UsedTotalWorkload} \tag{5.31}$$

$$\mathtt{Task.TotalBudget} \geq \mathtt{Task.UsedTotalBudget} \tag{5.32}$$

$$\mathtt{Task.TotalDuration} \geq \mathtt{max}\{\mathtt{s.TotalDuration} | \mathtt{s} \in \mathtt{Task.Subtasks}\} \tag{5.33}$$

If a user tries to change the total workload, budget, or duration of a task in a way that would violate one of the constraints (5.31) to (5.33), then alternative or compensating changes are proposed by PROCEED. The user may accept these changes or his operation is discarded.

Several time constraints may be set manually by the user: The release and due dates of a task, and the lag times of control flows. These time constraints have to be consistent with each other.

$$\texttt{Task.ReleaseDate} \leq \texttt{Task.DueDate} \tag{5.34}$$
$$\texttt{Task.ReleaseDate} \geq \texttt{Task.Parent.ReleaseDate} \tag{5.35}$$
$$\texttt{Task.DueDate} \leq \texttt{Task.Parent.DueDate} \tag{5.36}$$
$$\forall \texttt{c} \in \texttt{Task.StdCFs}(\texttt{Task.DueDate} \leq \texttt{c.Succ.DueDate} - \texttt{c.LagTime}) \tag{5.37}$$
$$\forall \texttt{c} \in \texttt{Task.SimCFs}(\texttt{Task.DueDate} \leq \texttt{c.Succ.DueDate} - \texttt{c.LagTime}) \tag{5.38}$$
$$\forall \texttt{c} \in \texttt{Task.SimCFs}(\texttt{Task.ReleaseDate} \leq \texttt{c.Succ.ReleaseDate} - \texttt{c.LagTime}) \tag{5.39}$$
$$\forall \texttt{f} \in \texttt{Task.ActiveFeedbacks}(\texttt{Task.DueDate} \geq \texttt{f.Target.DueDate}) \tag{5.40}$$

During scheduling, critical path analysis is performed and the earliest and latest start and end times of all tasks are computed. These computed constraint dates have to be consistent with each other and with the manually set constraint dates. A scheduling pass (re-)establishes this consistency. First of all, the computed constraint dates have to be consistent with each other. These constraints cannot be violated by user actions since they only refer to computed values.

$$\texttt{Task.EPST} \leq \texttt{Task.EPET} - \texttt{Task.TotalDuration} \tag{5.41}$$
$$\texttt{Task.EPET} \leq \texttt{Task.LPET} \tag{5.42}$$
$$\texttt{Task.EPST} \leq \texttt{Task.LPST} \tag{5.43}$$
$$\texttt{Task.LPST} \leq \texttt{Task.LPET} - \texttt{Task.TotalDuration} \tag{5.44}$$
$$\texttt{Task.EPST} \geq \texttt{Task.Parent.EPST} \tag{5.45}$$
$$\texttt{Task.LPST} \geq \texttt{Task.Parent.LPST} \tag{5.46}$$
$$\texttt{Task.EPET} \leq \texttt{Task.Parent.EPET} \tag{5.47}$$
$$\texttt{Task.LPET} \leq \texttt{Task.Parent.LPET} \tag{5.48}$$

Furthermore, the computed constraint dates have to be consistent with the manually set constraint dates.

$$\texttt{Task.ReleaseDate} \leq \texttt{Task.EPST} \tag{5.49}$$
$$\texttt{Task.DueDate} \geq \texttt{Task.LPET} \tag{5.50}$$

Finally, the defined control and feedback flows impose constraints on the computed constraint dates.

$$\forall \mathtt{c} \in \mathtt{Task.StdCFs}(\mathtt{Task.EPET} \leq \mathtt{c.Succ.EPET} - \mathtt{c.LagTime}) \tag{5.51}$$

$$\forall \mathtt{c} \in \mathtt{Task.StdCFs}(\mathtt{Task.LPET} \leq \mathtt{c.Succ.LPET} - \mathtt{c.LagTime}) \tag{5.52}$$

$$\forall \mathtt{c} \in \mathtt{Task.SimCFs}(\mathtt{Task.EPET} \leq \mathtt{c.Succ.EPET} - \mathtt{c.LagTime}) \tag{5.53}$$

$$\forall \mathtt{c} \in \mathtt{Task.SimCFs}(\mathtt{Task.LPET} \leq \mathtt{c.Succ.LPET} - \mathtt{c.LagTime}) \tag{5.54}$$

$$\forall \mathtt{c} \in \mathtt{Task.SimCFs}(\mathtt{Task.EPST} \leq \mathtt{c.Succ.EPST} - \mathtt{c.LagTime}) \tag{5.55}$$

$$\forall \mathtt{c} \in \mathtt{Task.SimCFs}(\mathtt{Task.LPST} \leq \mathtt{c.Succ.LPST} - \mathtt{c.LagTime}) \tag{5.56}$$

$$\forall \mathtt{c} \in \mathtt{Task.SeqCFs}(\mathtt{Task.EPET} \leq \mathtt{c.Succ.EPST} - \mathtt{c.LagTime}) \tag{5.57}$$

$$\forall \mathtt{c} \in \mathtt{Task.SeqCFs}(\mathtt{Task.LPET} \leq \mathtt{c.Succ.LPST} - \mathtt{c.LagTime}) \tag{5.58}$$

$$\forall \mathtt{f} \in \mathtt{Task.ActiveFeedbacks}(\mathtt{Task.EPET} \geq \mathtt{f.Target.EPET}) \tag{5.59}$$

$$\forall \mathtt{f} \in \mathtt{Task.ActiveFeedbacks}(\mathtt{Task.LPET} \geq \mathtt{f.Target.LPET}) \tag{5.60}$$

After a successful scheduling pass, the user may change the manually set constraint dates as well as the lag times and the semantics of control flows. If a constraint is violated by this change, the corresponding subprocess has to be rescheduled to re-establish consistency.

The automatic resource-constrained scheduling which is implemented in PROCEED generates a consistent schedule and sets the values of the planned dates of tasks. The following timing consistency constraints have to be fulfilled for the planned dates to be consistent. Manual changes to the planned dates may lead to inconsistencies which may remain unresolved until scheduling is performed again.

$$\mathtt{Task.PlannedStartTime} = \mathtt{Task.PlannedEndTime} - \mathtt{Task.TotalDuration} \tag{5.61}$$

$$\mathtt{Task.PlannedStartTime} \leq \mathtt{Task.PlannedEndTime} \tag{5.62}$$

$$\mathtt{Task.PlannedStartTime} \geq \mathtt{Task.Parent.PlannedStartTime} \tag{5.63}$$

$$\mathtt{Task.PlannedEndTime} \leq \mathtt{Task.Parent.PlannedEndTime} \tag{5.64}$$

$$\forall \mathtt{c} \in \mathtt{Task.StdCFs}(\mathtt{Task.PlannedEndTime} \leq \mathtt{c.Succ.PlannedEndTime} - \mathtt{c.LagTime}) \tag{5.65}$$

$$\forall \mathtt{c} \in \mathtt{Task.SimCFs}(\mathtt{Task.PlannedEndTime} \leq \mathtt{c.Succ.PlannedEndTime} - \mathtt{c.LagTime}) \tag{5.66}$$

$$\forall \mathtt{c} \in \mathtt{Task.SimCFs}(\mathtt{Task.PlannedStartTime} \leq \mathtt{c.Succ.PlannedStartTime} - \mathtt{c.LagTime}) \tag{5.67}$$

$$\forall \mathtt{c} \in \mathtt{Task.SeqCFs}(\mathtt{Task.PlannedEndTime} \leq \mathtt{c.Succ.PlannedStartTime} - \mathtt{c.LagTime}) \tag{5.68}$$

$$\forall \mathtt{f} \in \mathtt{Task.ActiveFeedbacks}(\mathtt{Task.PlannedEndTime} \geq \mathtt{f.Target.PlannedEndTime}) \tag{5.69}$$

The scheduled workload for a task assignment has to be consistent with its planned workload.

$$\texttt{TaskAssignment.Workload} = \sum_{\texttt{d}\in\texttt{Dates}} \texttt{TaskAssignment.PlannedWorkload.Workload}(\texttt{d}) \tag{5.70}$$

The scheduled workload for a task assignment has to be consistent with the planned dates of the corresponding task.

$$\begin{aligned}\forall \texttt{d} \in \texttt{Dates}((&\texttt{d} < \texttt{TaskAssignment.Task.PlannedStartTime} \vee \\ &\texttt{d} > \texttt{TaskAssignment.Task.PlannedEndTime}) \\ &\Rightarrow \texttt{TaskAssignment.PlannedWorkload.Workload}(\texttt{d}) = 0)\end{aligned} \tag{5.71}$$

Furthermore, the scheduled workload for a task assignment has to be consistent with the time constraint for the maximal daily workload.

$$\begin{aligned}\forall \texttt{d} \in \texttt{Dates}(&\texttt{TaskAssignment.PlannedWorkload.Workload}(\texttt{d}) \\ &\leq \texttt{TaskAssignment.MaxDailyWorkload})\end{aligned} \tag{5.72}$$

Finally, the resource usage must never exceed the totally available workload of the resource.

$$\forall \texttt{r} \in \texttt{Resources}(\forall \texttt{d} \in \texttt{Dates}(\texttt{r.WCal.UWL}(\texttt{d}) \leq \texttt{r.WCal.TWL}(\texttt{d}))) \tag{5.73}$$

The planned dates of a task have to be consistent with the manually set constraint dates.

$$\texttt{Task.ReleaseDate} \leq \texttt{Task.PlannedStartTime} \tag{5.74}$$

$$\texttt{Task.PlannedEndTime} \leq \texttt{Task.DueDate} \tag{5.75}$$

Furthermore, the planned dates have to be consistent with the computed constraint dates.

$$\texttt{Task.EPST} \leq \texttt{Task.PlannedStartTime} \tag{5.76}$$

$$\texttt{Task.PlannedStartTime} \leq \texttt{Task.LPST} \tag{5.77}$$

$$\texttt{Task.EPET} \leq \texttt{Task.PlannedEndTime} \tag{5.78}$$

$$\texttt{Task.PlannedEndTime} \leq \texttt{Task.LPET} \tag{5.79}$$

Post-conditions for structural change operations Structural change operations to a dynamic task net and changes to the properties of tasks and control flows may cause violations of timing consistency constraints. Therefore, post-conditions are defined for change operations, which ensure that the timing data is in a consistent state after the respective operation. These post conditions are derived from the timing consistency constraints. The post conditions are evaluated before the management data is actually modified, i.e. the PROCEED system checks whether the structural change or the new property value would lead to an inconsistent state. If so, then the operation may be prohibited, compensating changes may be performed, or the operation is temporarily accepted. The different actions which can be taken to avoid the inconsistency of the management data with respect to timing properties will be discussed in Chapter 9. The timing consistency constraints are translated to formulas which check for an individual task, task assignment, control flow, or feedback flow whether their timing properties are consistent with the property values of the tasks and task relationships in its context.

The following formula combines all timing consistency constraints which refer to the properties of a single task. The formula is evaluated whenever the value of a timing property of a task is changed.

$$\begin{aligned}
&\texttt{TaskPropertiesConsistent(t)} \equiv \\
&\quad \texttt{t.ReleaseDate} \leq \texttt{t.DueDate} \land \\
&\quad \texttt{t.EPST} \leq \texttt{t.EPET} \land \texttt{t.EPET} \leq \texttt{t.LPET} \land \\
&\quad \texttt{t.EPST} \leq \texttt{t.LPST} \land \texttt{t.LPST} \leq \texttt{t.LPET} \land \\
&\quad \texttt{t.ReleaseDate} \leq \texttt{t.EPST} \land \texttt{t.DueDate} \geq \texttt{t.LPET} \land \\
&\quad \texttt{t.PlannedStartTime} = \texttt{t.PlannedEndTime} - \texttt{t.TotalDuration} \land \\
&\quad \texttt{t.PlannedStartTime} \leq \texttt{t.PlannedEndTime} \land \\
&\quad (\forall \texttt{a} \in \texttt{t.TaskAssignments}(\forall \texttt{d} \in \texttt{Dates}((\texttt{d} < \texttt{a.Task.PlannedStartTime} \\
&\qquad \lor \texttt{d} > \texttt{a.Task.PlannedEndTime}) \Rightarrow \texttt{a.PlannedWorkload.Workload(d)} = 0))) \land \\
&\quad \texttt{t.ReleaseDate} \leq \texttt{t.PlannedStartTime} \land \texttt{t.PlannedEndTime} \leq \texttt{t.DueDate} \land \\
&\quad \texttt{t.EPST} \leq \texttt{t.PlannedStartTime} \land \texttt{t.PlannedStartTime} \leq \texttt{t.LPST} \land \\
&\quad \texttt{t.EPET} \leq \texttt{t.PlannedEndTime} \land \texttt{t.PlannedEndTime} \leq \texttt{t.LPET}
\end{aligned}$$

The following formula checks for an individual task assignment whether the planned workload equals the distributed workload and whether its share of the used workload of the corresponding task is not to large. This formula is evaluated when the planned workload of a task assignment shall be changed or a new task assignment is created.

$$\begin{aligned}
&\texttt{TaskAssignmentConsistent(a)} \equiv \\
&\quad \texttt{a.Task.TotalWorkload} \geq \texttt{a.Task.UsedTotalWorkload} \land \\
&\quad \texttt{a.Task.TotalBudget} \geq \texttt{a.Task.UsedTotalBudget} \land \\
&\quad \texttt{TaskAssignment.Workload} = \\
&\qquad \sum_{\texttt{d} \in \texttt{Dates}} \texttt{TaskAssignment.PlannedWorkload.Workload(d)}
\end{aligned}$$

The following formula combines all timing consistency constraints which refer to the relation of a subtask to its parent task. This formula is evaluated for a new subtask before it is created or when an existing task shall be modified. Furthermore, it is evaluated for all subtasks of a task when its timing related properties or its realization shall be changed.

$$\begin{aligned}
\texttt{SubtaskConsistent}(\texttt{t}) &\equiv \\
\texttt{t.Parent.TotalWorkload} &\geq \texttt{t.Parent.UsedTotalWorkload}\wedge \\
\texttt{t.Parent.TotalBudget} &\geq \texttt{t.Parent.UsedTotalBudget}\wedge \\
\texttt{t.Parent.TotalDuration} &\geq \texttt{t.TotalDuration}\wedge \\
\texttt{t.ReleaseDate} &\geq \texttt{t.Parent.ReleaseDate}\wedge \\
\texttt{t.DueDate} &\leq \texttt{t.Parent.DueDate}\wedge \\
\texttt{t.EPST} \geq \texttt{t.Parent.EPST} \wedge \texttt{t.LPST} &\geq \texttt{t.Parent.LPST}\wedge \\
\texttt{t.EPET} \leq \texttt{t.Parent.EPET} \wedge \texttt{t.LPET} &\leq \texttt{t.Parent.LPET}\wedge \\
\texttt{t.PlannedStartTime} &\geq \texttt{t.Parent.PlannedStartTime}\wedge \\
\texttt{t.PlannedEndTime} &\leq \texttt{t.Parent.PlannedEndTime}
\end{aligned}$$

The formula $\texttt{ControlFlowConsistent}(\texttt{c})$ checks for a control flow $\texttt{c} \in \texttt{ControlFlows}$ if the timing properties of the connected tasks are consistent with respect to the semantics and the lag time of the control flow. This formula is evaluated after the creation of a new control flow or the modification of an existing control flow, and it is evaluated for all control flows which are connected to a modified task.

$$\begin{aligned}
\texttt{ControlFlowConsistent}(\texttt{c}) &\equiv \\
\texttt{c.Pred.DueDate} &\leq \texttt{c.Succ.DueDate} - \texttt{c.LagTime}\wedge \\
\texttt{c.Pred.EPET} &\leq \texttt{c.Succ.EPET} - \texttt{c.LagTime}\wedge \\
\texttt{c.Pred.LPET} &\leq \texttt{c.Succ.LPET} - \texttt{c.LagTime}\wedge \\
\texttt{c.Pred.PlannedEndTime} &\leq \texttt{c.Succ.PlannedEndTime} - \texttt{c.LagTime}\wedge \\
(\texttt{c.Semantics} = \texttt{Simultaneous} &\Rightarrow \\
(\texttt{c.Pred.ReleaseDate} &\leq \texttt{c.Succ.ReleaseDate} - \texttt{c.LagTime}\wedge \\
\texttt{c.Pred.EPST} &\leq \texttt{c.Succ.EPST} - \texttt{c.LagTime}\wedge \\
\texttt{c.Pred.LPST} &\leq \texttt{c.Succ.LPST} - \texttt{c.LagTime}\wedge \\
\texttt{c.Pred.PlannedStartTime} &\leq \texttt{c.Succ.PlannedStartTime} - \texttt{c.LagTime}))\wedge \\
(\texttt{c.Semantics} = \texttt{Sequential} &\Rightarrow \\
(\texttt{c.Pred.EPET} &\leq \texttt{c.Succ.EPST} - \texttt{c.LagTime}\wedge \\
\texttt{c.Pred.LPET} &\leq \texttt{c.Succ.LPST} - \texttt{c.LagTime}\wedge \\
\texttt{c.Pred.PlannedEndTime} &\leq \texttt{c.Succ.PlannedStartTime} - \texttt{c.LagTime}))
\end{aligned}$$

Likewise, the formula $\texttt{FeedbackFlowConsistent}(\texttt{f})$ checks whether the timing properties of the feedback flow's source and target are consistent.

$$\begin{aligned}
\texttt{FeedbackFlowConsistent}(\texttt{f}) &\equiv \\
\texttt{f.Source.DueDate} &\geq \texttt{f.Target.DueDate}\wedge \\
\texttt{f.Source.EPET} &\geq \texttt{f.Target.EPET}\wedge \\
\texttt{f.Source.LPET} &\geq \texttt{f.Target.LPET}\wedge \\
\texttt{f.Source.PlannedEndTime} &\geq \texttt{f.Target.PlannedEndTime}
\end{aligned}$$

Operation	**Post-Condition**
$\texttt{CreateSubtask}(\texttt{p}, \texttt{out s})$	$\texttt{TaskConsistent}(\texttt{s})$
$\texttt{AddRealization}(\texttt{t}, \texttt{r})$	$\texttt{TaskConsistent}(\texttt{t})$
$\texttt{CreateControlFlow}(\texttt{p}, \texttt{s}, \texttt{out c})$	$\texttt{ControlFlowConsistent}(\texttt{c})$
$\texttt{ModifyControlFlow}(\texttt{c}, \texttt{s})$	$\texttt{ControlFlowConsistent}(\texttt{c})$
$\texttt{CreateFeedbackFlow}(\texttt{s}, \texttt{t}, \texttt{out f})$	$\texttt{FeedbackFlowConsistent}(\texttt{f})$
$\texttt{ModifyTask}(\texttt{t})$	$\texttt{TaskConsistent}(\texttt{t})$
$\texttt{CreateTaskAssignment}(\texttt{t}, \texttt{out a})$	$\texttt{TaskAssignmentConsistent}(\texttt{a})$
$\texttt{ModifyTaskAssignment}(\texttt{a})$	$\texttt{TaskAssignmentConsistent}(\texttt{a})$
$\texttt{CreateNewTaskVersion}(\texttt{t}, \texttt{out t}')$	$\texttt{TaskConsistent}(\texttt{t}')$

Tabelle 5.6: Timing post-conditions for structural change operations.

Finally, the formula $\texttt{TaskConsistent}(\texttt{t})$ combines the previous formulas to check for an individual task $\texttt{t} \in \texttt{Tasks}$ whether its timing properties are consistent with each other, with the timing properties of the parent task, the subtasks, the connected tasks, and the task assignments.

$$\begin{aligned}
&\texttt{TaskConsistent}(\texttt{t}) \equiv \\
&\quad \texttt{TaskPropertiesConsistent}(\texttt{t}) \wedge \\
&\quad \forall \texttt{a} \in \texttt{t.TaskAssignments}(\texttt{TaskAssignmentConsistent}(\texttt{a})) \wedge \\
&\quad \texttt{SubtaskConsistent}(\texttt{t}) \wedge \\
&\quad \forall \texttt{s} \in \texttt{t.Subtasks}(\texttt{SubtaskConsistent}(\texttt{s})) \wedge \\
&\quad \forall \texttt{c} \in \texttt{t.ControlFlows} \cup \texttt{t.IncomingCFs}(\texttt{ControlFlowConsistent}(\texttt{c})) \wedge \\
&\quad \forall \texttt{f} \in \texttt{FeedbackFlows}((\texttt{f.Source} = \texttt{t} \vee \texttt{f.Target} = \texttt{t}) \\
&\qquad \Rightarrow \texttt{FeedbackFlowConsistent}(\texttt{f}))
\end{aligned}$$

Table 5.6 lists all timing related post-conditions for structural change operations and property change operations. the previously defined formulas are used to check the consistency of the management data in case the respective operation would be performed.

5.4 Monitoring Model

The monitoring model defines entities and properties which are required to determine the current status of an enacted development process, and to evaluate its performance in comparison to the plan. Furthermore, monitoring constraints are defined which are evaluated to detect deviations of the actual performance from the plan.

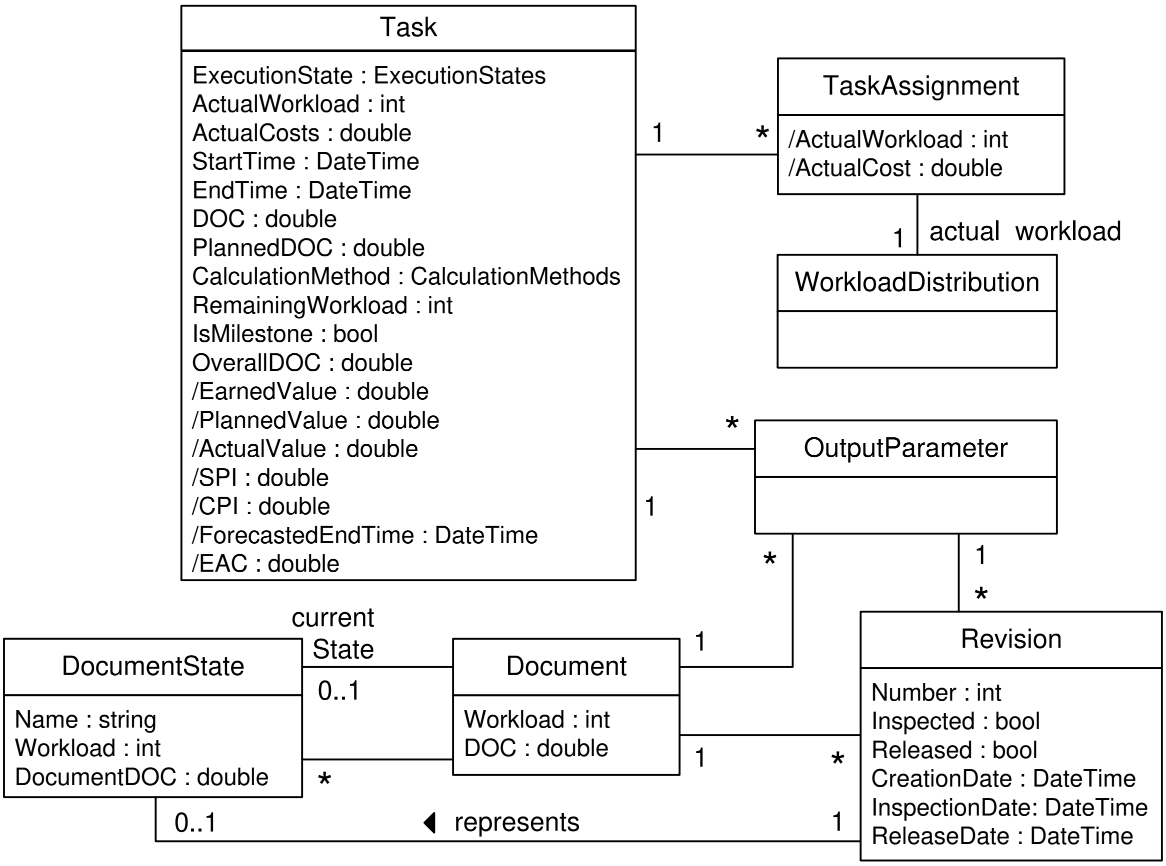


Abbildung 5.19: Entities and properties for monitoring.

5.4.1 Properties for Monitoring

Several properties have been defined for the entities task, task assignment, document and revision for determining the current process status and its performance. Figure 5.19 shows the entities which have been extended or introduced. The additional entities and properties can be categorized into the following three categories.

Actual dates The start and end times of tasks and the actual workload spent on task assignments.

Performance indicators The degree of completion of a task as well as performance indices computed by means of earned value analysis.

Forecasted values The forecasted duration, end time, and budget at completion of a task.

These properties will be described in detail in the following.

Actual Dates

The execution states of tasks in a dynamic task net represent the current status of the enacted process. It can be analyzed which parts of a process model instance have already been started and which have already been terminated. However, this information is not precise enough for project monitoring. It is required to know since when a task is active, and how much effort has already been spent on the task. Therefore, the monitoring model adds properties for actual dates, workload and costs to the entities task and task assignment.

When a process instance is enacted, tasks are started and eventually committed. The PROCEED system automatically logs the actual start and end times of all tasks in a dynamic task net. For this purpose, the following properties are defined for every task $\mathtt{t} \in \mathtt{Tasks}$.

Start time The property $\mathtt{t.StartTime} \in \mathtt{Dates}$ is set to the current date when a task is started.

End time The property $\mathtt{t.EndTime} \in \mathtt{Dates}$ is set to the current date when a task is committed, aborted, or skipped.

The start time of a task which is skipped is set to the same date as the end time. The actual start and end times of a task may deviate from its planned start and end times. This may indicate delays during the enactment of a process model instance.

When resources execute their assigned tasks, they are obliged to register their actual working hours for their task assignments. The following properties are defined for every task assignment $\mathtt{a} \in \mathtt{TaskAssignments}$.

Actual Workload The workload distribution `a.ActualWorkloadDistribution` stores the registered actual working hours per day. The property `a.ActualWorkload` returns the actual workload for the task assignment in the unit MHRS which is the sum of all working hours which the assigned resource has spent on the task assignment down to the present day. It is derived from the registered working hours of the task assignment.

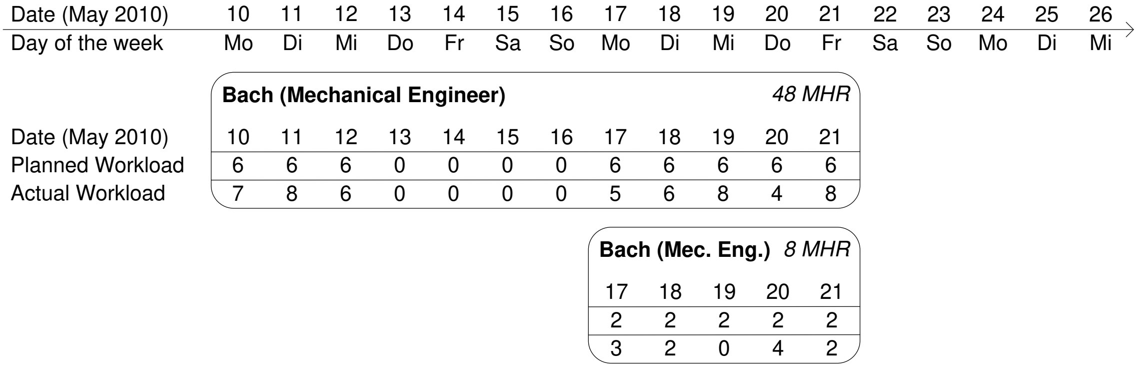
$$\mathtt{a.ActualWorkload} := \sum_{\mathtt{d} \in \mathtt{Dates}} \mathtt{a.ActualWorkloadDistribution.Workload(d)}$$

Actual Cost The property `a.ActualCost` is the actual cost of the task assignment which is derived from the actual workload and the cost rates of the assigned resource(s).

$$\begin{aligned}\mathtt{a.ActualCost} := \sum_{\mathtt{d} \in \mathtt{Dates}} & (\mathtt{a.Resource(d).CpH} \cdot \\ & \mathtt{a.ActualWorkloadDistribution.Workload(d)}) + \\ & \sum_{\mathtt{r} \in \mathtt{a.Resources}} \mathtt{r.CpU}\end{aligned}$$

If no actual resource is assigned yet, no actual working hours can be registered and hence no actual costs are accrued.

Figure 5.20 shows an example of two task assignments with workload distributions for the planned and actual workload. The registered values may deviate from the



Date (May 2010)	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
Day of the week	Mo	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi

Bach (Mechanical Engineer) *48 MHR*

Date (May 2010)	10	11	12	13	14	15	16	17	18	19	20	21
Planned Workload	6	6	6	0	0	0	0	6	6	6	6	6
Actual Workload	7	8	6	0	0	0	0	5	6	8	4	8

Bach (Mec. Eng.) *8 MHR*

17	18	19	20	21
2	2	2	2	2
3	2	0	4	2

Abbildung 5.20: Workload distributions for actual workload of task assignments.

planned values. In contrast to the planned workload, the actual workload spent on the task assignments is only stored at the task assignments but not in the work calendars of the respective resources.

The actual workload of the task assignments and subtasks together amount to the *actual workload of a task* which is defined analogously to the used total workload of a task. Likewise, the actual costs of the task assignments and subtasks amount to the *actual costs of the task* which is defined analogously to the used total budget of a task. In Chapter 8, it will be described that earned value analysis can be performed in PROCEED based on either workload or costs. Depending on which modus is chosen, the *actual value* of a task is the actual workload or the actual costs.

Performance Indicators

The goal of progress measurement is to determine the actual performance of the enacted process and to compare it to the plan. The actual execution states of tasks in a timed dynamic task net can be compared to the planned execution states which are derived from the planned dates of the subtasks, e.g. a task has the planned execution state *Active* between its planned start and end times. This comparison can be very effective when a large number of small tasks is considered, e.g. a task is behind schedule when 70 out of 100 subtasks should be committed but only 50 are already committed. However, for long-running tasks whose progress cannot be determined in terms of committed subtasks, the execution state does not provide sufficiently precise information about the actual performance of the task compared to the plan. When the task is active as planned, the execution state does not show how much work has already been completed.

For this purpose, the *degree of completion* (DOC) of a task has to be determined, which is a quantitative measure for the progress of a task. The DOC does not represent the mere progress in time, e.g. 6 of the planned 10 work days have passed, but it represents the progress of a task in terms of accomplished objectives. In the formal notation, the degree of completion is defined as the property `t.DOC` for a task `t` $\in$ `Tasks`.

The degree of completion of a new task (version) is always 0%. The degree of completion of a task is automatically set to 100% upon termination, even when the task is aborted or skipped. This is required to ensure that the aggregated degree of completion of the parent task can reach 100%. According to constraint (5.19) of the behavioral model, a complex task in a dynamic task net can be committed when all of its subtasks are terminated, whether successfully or unsuccessfully. Setting the degree of completion of failed and skipped tasks to 100% is in line with this aspect of the behavioral semantics of dynamic task nets.

Several different *progress measures* can be used in PROCEED to compute the DOC of a task, which will be introduced in Chapter 8. Different progress measures do not provide different key figures by means of which the progress of a task can be evaluated, but they are different *calculation methods* for the DOC of a task which utilize different sources of information for the computation. The property CalculationMethod (cf. Figure 5.19) specifies for every task in a dynamic task net individually, which progress measure shall be used to compute the DOC of the task. A new version of a task has by default the same calculation method for the DOC as the previous version. One of the available progress measures requires the estimation of the remaining workload of a task. The property RemainingWorkload can be set to the estimated value. The remaining workload of a new task (version) is set by default to the total workload of the task (version).

A task can be a *milestone* in a dynamic task net. A task is defined as a milestone by setting the value of the property IsMilestone to true. In contrast to conventional project plans, a milestone in PROCEED can have a duration of several days, assigned resources, and planned workload and costs. The classical concept of a milestone corresponds to the end event or planned end time of a milestone task. Milestones can be used for the computation of the DOC of their respective parent tasks. The property OverallDOC defines the overall degree of completion which is set for the parent task when a milestone is committed. The property is only used for a milestone task.

The degree of completion of tasks can also be measured based on the states of their output documents. For this purpose, the class DocumentState has been introduced. A document can have several *document states* which are associated with a degree of completion of the document and the proportional workload required to reach the state. The current state of a document can be changed during the execution of a task by releasing a new document revision which represents the next state.

While the DOC of a task represents the actual progress of the task in terms of accomplished objectives, the *planned degree of completion* (PlannedDOC) indicates how far a task should have progressed at the current date. It is computed as the quotient of the planned working hours to the current date divided by the total planned workload of the task. In the formal notation, the planned degree of completion is defined as the property `t.PlannedDOC` for a task $\mathtt{t} \in \mathtt{Tasks}$. If the planned working hours are non-uniformly distributed over the duration of the task, then this results in a non-linear increase of the planned degree of completion over time. The planned

DOC can be directly compared to the actual DOC of a task to check whether the task is delayed. However, this direct comparison does not allow any prediction about how much the task will be delayed when it is finally terminated.

Earned value analysis (cf. Section 3.3.2) enables a more precise evaluation of the performance of a task. It involves the computation of the EarnedValue, PlannedValue and ActualValue of a task, which can be determined based on either workload or costs. From these values, the schedule performance index (SPI) and the cost performance index (CPI) are computed. The SPI indicates whether the task is on schedule while the CPI indicates whether the task is expected to stay in budget limits.

Forecasted Values

Progress measurement and earned value analysis allow to forecast the duration, end time, and budget of a task. Based on the SPI, the expected end time of a task is forecasted. The calculated value is assigned to the property ForecastedEndTime. The property is formally defined for a task $\mathtt{t} \in \mathtt{Tasks}$ as $\mathtt{t.ForecastedEndTime} \in \mathtt{Dates}$. Depending on the performance of the task, the forecasted end time may be earlier or later than the planned end time. This information can be used to detect probable deadline violations and to adapt the plan when required.

The CPI allows to forecast the expected total budget at the end of the task. The property EAC (Earned at Completion) returns the forecasted value. It is formally defined for a task $\mathtt{t} \in \mathtt{Tasks}$ as $\mathtt{t.EAC} \in \mathbb{R}^+$. The EAC value indicates whether a task is likely to exceed its planned total budget.

Graphical Representation

The graphical representation of dynamic task nets which is used in this thesis has been gradually introduced in Section 2.3 and in the previous sections. At this point the graphical representation of all entities and properties defined in the structural, behavioral, timing, and monitoring models shall be presented.

Figure 5.21 shows an abstract example of a dynamic task net. The release date and the due date of a task are displayed in arrows on top of the task box (RDATE and DDATE). Dates are generally displayed in the British date format dd/mm/yyyy with the day of the month (dd), the month (mm) and the year (yyyy), or in the short version dd/mm. All other properties of a task are arranged below the task name: The total workload (TW), total duration (TDUR), total budget (TB), the earliest and latest possible start and end times as well as the total float (TFLOAT) which is calculated during critical path analysis. Furthermore, the planned start and end times (PSTART and PEND), the actual start and end times (START and END), and the degree of completion of a task (DOC) are depicted. The performance indices SPI and CPI are displayed below the actual dates and the degree of completion. The input and output parameters of a task are listed below the performance indices. They can be connected with parameters of other tasks via data flows which connect the respective boxes in the graphical representation. Depending on the examples, not all properties of a task will be displayed in the figures presented in this thesis which

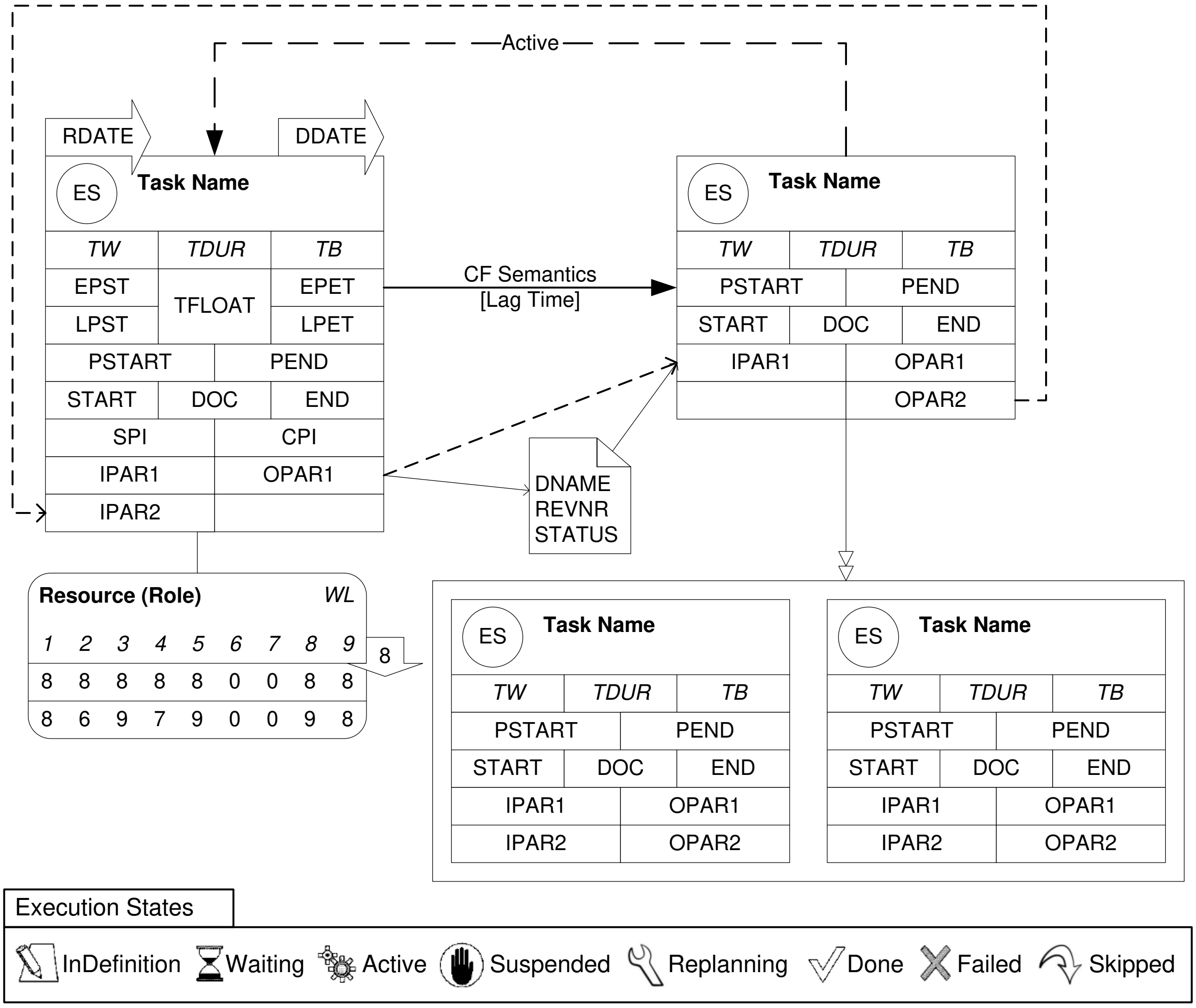


Abbildung 5.21: Graphical representation of dynamic task nets.

show dynamic task nets. From the context, the data formats, and the position of the property boxes, it will be obvious which properties are displayed.

The planned workload for a task assignment is depicted in the top right corner of the rounded box representing the task assignment (WL). The first line of numbers specifies the days of the month which depend on the planned dates of the task. The second line shows the planned daily workload and the third line the actual daily workload for that task assignment. A downwards oriented arrow at the right side of the task assignment shows the maximal daily resource usage which is allowed for the task assignment.

A control flow is labeled with its semantics and the lag time. A feedback flow is labeled with Active if it is still active.

Document revisions are visualized by document icons containing the document name (DNAME), the revision number (REVNR), and the status of the document (STATUS) which has been reached with the depicted revision.

5.4.2 Monitoring Constraints

Violations of monitoring constraints are not prohibited by PROCEED. They merely indicate a bad performance of the enacted process which deviates from the plan. In that sense, monitoring constraints are non-strict constraints in contrast to structural, behavioral, and timing consistency constraints, which may not be violated.

While violations of strict constraints can only occur due to manual modifications of a dynamic task net, monitoring constraints may also be violated simply because time proceeds. For example, a monitoring constraint demands that a task should not be committed later than its planned end time. If a task has not been committed yet, but the planned end time has not been reached yet, the constraint is satisfied. However, if time proceeds beyond the planned end time, and the task still has not been committed, then the monitoring constraint is violated. This deviation from the plan can even be anticipated before the planned end time of the task has actually passed. PROCEED allows to measure the progress of individual tasks and to forecast their expected end times. This way, potential constraint violations can be detected before they occur. In general, violations of monitoring constraints refer to one of the following two cases.

- Inconsistencies between actual performance and plan,
- Expected inconsistencies between actual performance and plan,
  i.e. inconsistencies between forecasted and planned values.

PROCEED identifies violations of monitoring constraints and informs the responsible user, who can then take different actions to resolve the constraint violations. The actions which the person responsible can take to resolve the non-strict inconsistencies can be divided into corrective measures or plan changes. Corrective measures may be to increase the motivation or qualification of the process participants, or to eliminate existing conflicts [Bur00]. These actions do not cause a change in the management data but may influence the performance of the process participants. Plan changes on the other hand affect the management data and may lead to violations of strict constraints. PROCEED supports the user during replanning to arrive at a consistent plan, which may require the rescheduling of the affected subprocesses.

In the following, the defined monitoring constraints are presented. The actual performance of a task should comply to the manually set constraint dates. The actual start time is compared to the release date to check whether the task has been started too early. To detect a delayed termination of a task, its execution state has to be checked after the due date has passed. Comparing the end time of the task to the due date is not sufficient because the former is undefined as long as the task has not been terminated yet.

$$\texttt{Task.StartTime} \geq \texttt{Task.ReleaseDate} \tag{5.80}$$

$$\texttt{Today} > \texttt{Task.DueDate} \Rightarrow \texttt{Task.State} \in \texttt{Terminated} \tag{5.81}$$

The actual performance may comply to the manually set constraint dates but still may not meet the plan. If a task has been scheduled early in the time window defined by the release and due date but is started and/or finished later than planned, the corresponding responsible resource should be informed. Therefore, the following two monitoring constraints are defined.

$$\texttt{Today} > \texttt{Task.PlannedStartTime} \Rightarrow \texttt{Task.State} \notin \texttt{Preparing} \tag{5.82}$$

$$\texttt{Today} > \texttt{Task.PlannedEndTime} \Rightarrow \texttt{Task.State} \in \texttt{Terminated} \tag{5.83}$$

If the second constraint is fulfilled for a task, then the constraint (5.81) for the due date is fulfilled as well because the planned end time may not be set to a later date than the due date according to the timing consistency constraint (5.75). However, the monitoring constraint (5.81) is not redundant since a task for which a due date has been defined may not be scheduled, so that the planned date is undefined. In that case, the constraint for the planned date is not violated, but the constraint for the due date may be violated.

The following monitoring constraints are refinements of the behavioral constraints (5.20) to (5.22). They take the lag times of control flows into account.

$$\forall \texttt{c} \in \texttt{Task.StdCFs}(\texttt{Task.EndTime} \leq \texttt{c.Succ.EndTime} - \texttt{c.LagTime}) \tag{5.84}$$

$$\forall \texttt{c} \in \texttt{Task.SimCFs}(\texttt{Task.EndTime} \leq \texttt{c.Succ.EndTime} - \texttt{c.LagTime}) \tag{5.85}$$

$$\forall \texttt{c} \in \texttt{Task.SimCFs}(\texttt{Task.StartTime} \leq \texttt{c.Succ.StartTime} - \texttt{c.LagTime}) \tag{5.86}$$

$$\forall \texttt{c} \in \texttt{Task.SeqCFs}(\texttt{Task.EndTime} \leq \texttt{c.Succ.StartTime} - \texttt{c.LagTime}) \tag{5.87}$$

If a control flow defines a lag time which is greater than zero, the defined time span has to pass between the two events which are related by the control flow. For example, it is not sufficient to require that the successor of a given task which is connected by a standard control flow is not terminated earlier than the task when the control flow defines a minimal lag time. In this case the specified minimal lag time has to pass until the successor can be committed. When the source task of a control flow has not been started or terminated yet, then the behavioral constraints (5.20) to (5.22) apply for the execution states of the successors, i.e. the successor may not be started or terminated in contradiction to the control flow semantics. When the source task has been terminated, then the constraints (5.84) to (5.87) are evaluated when the successor is started or terminated to check for consistency with respect to the defined lag times.

While the corresponding behavioral constraints are strict constraints which are enforced by PROCEED, the monitoring constraints (5.84) to (5.87) are not strictly enforced. It is possible to start or commit a task too early with respect to the minimal lag time defined for an incoming control flow. This non-strict enforcement of monitoring constraints is required for the applicability of the management approach in practice. In particular, it is not possible to enforce the timely termination of tasks. Furthermore, the monitoring constraints (5.84) to (5.87) for control flows are handled in the same way as the monitoring constraints (5.80) and (5.81) for manually set constraint dates. In the latter case, it is also possible to start a task before its release date and to commit it after its due date. These deviations from the plan merely lead to warnings presented to the corresponding responsible resources.

The direct comparison of the degree of completion of a task with its planned degree of completion is formally defined by the following constraint.

$$\texttt{Task.DOC} \geq \texttt{Task.PlannedDOC} \tag{5.88}$$

The evaluation of this constraint does not lead to a warning when it is violated for the following reasons. First, the DOC of a task may deviate from the planned DOC for a longer period of time, so that many warnings would be generated. Second, not every minor deviation from the planned DOC requires the intervention of the project manager or another responsible resource. Third, the schedule performance index computed by means of earned value analysis is a better indicator for the delay of a task compared to the direct comparison of actual and planned degree of completion.

Expected violations of the constraints (5.81) and (5.83) are detected as violations of the following two constraints which compare the forecasted end time of a task with its due date and planned end time.

$$\texttt{Task.ForecastedEndTime} \leq \texttt{Task.DueDate} \tag{5.89}$$

$$\texttt{Task.ForecastedEndTime} \leq \texttt{Task.PlannedEndTime} \tag{5.90}$$

Finally, the actual costs of a task should not exceed its total budget. Furthermore, the budget at completion can be forecasted and compared to the total budget in order to detect budget overruns early. The following two monitoring constraints check whether the actual costs of a task have exceeded or will exceed the total budget, respectively. While a violation of the former constraint leads to a warning when the task is terminated, a violation of the latter constraint does not lead to a warning for the same reasons which have been given for the comparison of actual and planned DOC.

$$\texttt{Task.ActualCosts} \leq \texttt{Task.TotalBudget} \tag{5.91}$$

$$\texttt{Task.EAC} \leq \texttt{Task.TotalBudget} \tag{5.92}$$

The monitoring constraints (5.80) and (5.84) to (5.87) are evaluated upon the corresponding behavioral change operations, i.e. starting and committing a task, to check whether the respective operation has been performed too early. However, no post-conditions for the behavioral change operations are defined because the monitoring constraints are not strictly enforced. The monitoring constraints (5.81) to (5.83) are checked upon every date change to check whether a required behavioral change operation has been performed. The monitoring constraints (5.88) to (5.92) are not checked at all by PROCEED. They merely specify constraints which the user can check manually by inspecting the corresponding property values. The performance indicators of the earned value analysis are used to visualize deviations from the plan in the management views of PROCEED in order to call the user's attention.

5.5 Authorization Model

The issues of *access control* and *authorization* are of high relevance for all kinds of information systems which are simultaneously used by several users. Authorization and access control are terms often mistakenly interchanged. Authorization is the act of checking to see if a user has the proper permission to access a particular resource or perform a particular operation, assuming that the user has successfully authenticated himself. Access control is the general term for the ways of controlling access to system resources which includes authorization. In development processes, several process participants with different roles and responsibilities collaborate. For a process management system which is used by several users, an *authorization model* has to be defined.

In the AHEAD prototype (cf. Section 4.6), the access control issue was resolved by having two separate environments: The management environment and the work environment. Process participants used only one of the environments depending on their role in the project. Structural changes to the process model like inserting new tasks or control flows were only possible in the management environment. In the work environment, only the tasks assigned to the logged-in user were visible in a work list, and the user had only limited modification possibilities, e.g. changing the state of the tasks.

In PROCEED, the strict distinction between management environment and work environment has been abandoned. In order to enable process participants to manage their personal tasks and the subprocesses for which they are responsible, the graphical task net representation is available for all users. However, it is not desirable that every project team member can view and change all parts of the dynamic task net. Therefore, a control mechanism has been implemented which restricts the rights of certain users to view and modify the management data.

In static workflow management systems, the workflow definition is predefined and does not change structurally during enactment. Therefore, access control and authorization are usually concerned with the question whether a resource may execute a defined task and may access the related data. For example, in a claim assessment task of an insurance process, only the claims manager can approve the claim and view all the corresponding documents of the particular claim. In contrast, the authorization model of a dynamic process management systems also has to cover dynamic changes to running process model instances including task creation, deletion, and assignment.

The entities and relationships for resource management in PROCEED have been introduced in Section 5.1.3. Resources can have different functional roles in a project. The project team is structured hierarchically in sub-teams. Every team has a team leader who is authorized to direct the other team members. From the project team structure, a responsibility hierarchy can be derived. For every task in a project, there is exactly one task assignment for the *responsible resource*. There may be several additional task assignments for the *additional resources*.

The assignment of tasks to resources can be done in two different ways. In the

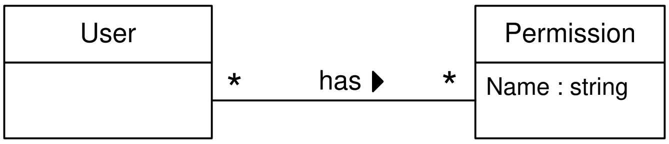


Abbildung 5.22: Related classes for users and permissions.

pull mechanism only the required role is specified for a task assignment. Users who can play the role in the project are notified about the new task assignment and can pick up the task. In the *push mechanism* a task is directly assigned to a specific resource. In this case the authorization model comes into play, because not every resource can be allowed to directly assign tasks to other resources. The *resources* in a project team are at the same time *users* of the PROCEED system. Users can make changes to a dynamic task net. They can add or delete tasks and control flows, assign resources to tasks, and execute tasks. The authorization model regulates which operations can be performed by the users.

The access control approach in PROCEED is based on *permissions* and *authorization rules*. Permissions are assigned to the members of a project team individually. The authorization rules are defined for the PROCEED system and cannot be changed by a user. The rules are evaluated by PROCEED to decide whether a user is authorized to perform a certain operation in a given situation. The result of the evaluation depends on the user's permissions, his task assignments, and his position in the project team. The constraints for the activation of permissions take the task net structure of the running process model instance into account but they are independent of any specific process model definition. The authorization model combines the *active* and *passive access control approaches* [TS97]. While some permissions are only activated in certain contexts defined in the authorization rules, other permissions are always activated. The access control policy of the PROCEED system is configurable for each individual project to support diverse collaboration scenarios. The details of the access control mechanism in PROCEED have been described in [Leo08].

5.5.1 Permissions

The authorization model defines *permissions* which can be assigned to users of the system. Figure 5.22 shows the extension of the meta-model by the class Permission which is associated with the class User. Users can be resources in a project team, and permissions are granted for individual projects. Therefore, a user can have a certain permission in one project while in another project he does not have the permission.

For a given project, permissions are assigned to every member of the project team individually. Permissions are not associated with the functional roles available in the project. Roles solely define professional competencies and qualifications. The permissions are assigned to users directly. Therefore, users with the same functional role can have different permissions. This flexibility is essential since in development projects there are often resources with the same qualifications, e.g. mechanical engineer, but with different responsibilities. The involved administrative overhead is lowered by default settings which are applied when a user is added to a project team.

The set of available permissions is fixed for the PROCEED system and cannot be extended. For every project, the set of predefined permissions is instantiated. There are two types of permissions, *basic permissions* and *super permissions*.

Basic Permissions implement the active access control approach and are activated depending on the evaluation of certain constraints. Constraints refer to the relation of a user to a task, e.g. whether the user is the responsible resource of the task. The following basic permissions are defined in PROCEED.

Manage Tasks This basic permission defines that a user may change the properties and the realization of tasks.

Assign Tasks This basic permission defines that a user may assign a task to a resource in the project team.

Execute Tasks This basic permission defines that a user can be assigned to tasks and can execute the assigned tasks.

Management of a task refers to changes to the plan, e.g. structural changes to the realization of a task, and to the time management data of a task. Management also includes the creation of a new task assignment or the change of an existing task assignment, except for setting the assigned resource. Assignment of a task means that a user may initially set or change the assigned resource of an existing task assignment. Task execution refers to all operations which are necessary to reflect the actual performance of a task in the PROCEED system, i.e. execution state changes but also the production of document revisions and the registration of actual working hours.

Super permissions implement the passive access control approach and are always activated independent of the given context. The following super permissions are defined in PROCEED.

Manage All Tasks A user who has this super permission may manage every task in the project.

Assign Any Task to Any Resource A user who has this super permission may directly assign a task to any resource in the project team.

Execute Any Task A user who has this super permission may execute any task in the project.

View All Tasks A user who has this super permission may view all tasks in the project.

Resource Management A user who has this super permission may assemble a project team, and he may assign permissions to users and revoke permissions from users in the project team.

While there is a super permission to view all tasks in a project, there is no corresponding basic permission, because it would not make sense to deprive a user of the permission to view the tasks he is assigned to.

Permissions in the PROCEED system are always *positive permissions*. A user is authorized to execute a certain operation if he has the required basic permission and the corresponding constraints are fulfilled. The absence of a basic permission prohibits a user from performing certain operations.

The access control mechanism implemented in PROCEED is project based. Every project in the system has its own instance of the permission set. Thus, a user who is a member in several project teams can have different permissions in the different projects. For example, a user can be the project manager for a small project in which he has super permissions, while in another project he has only basic permissions.

Because permissions are assigned to every member of a project team individually, *defaults* are applied to alleviate the involved management overhead. When a new user is added to a project team, all basic permissions are automatically assigned to the user. This setting can be changed afterwards by the project manager or any other user who has the permission for resource management in the project.

Permissions can be *assigned and revoked* from a user *at project runtime*. These changes can only be performed by a user who has the super permission for resource management. The consistency of the management data has to be guaranteed in this case. If the basic permission for task management or assignment or a super permission is revoked from a user, then this user cannot perform certain operations anymore. However, these changes cannot lead to any inconsistencies of the management data. On the other hand, the basic permission to execute a task cannot be revoked from a user as long as he is still assigned to a running task, because otherwise it might happen that there is no user left who is authorized to execute and commit the task.

5.5.2 Authorization Rules

The *authorization rules* define which operations a user of the PROCEED system may perform in a certain situation. Every authorization rule is specified in terms of permissions and constraints. The *contextual information* which constrains the activation of a basic permission defines the relation of the resource to the entity he wants to modify, e.g. whether he is the responsible resource of a task. The constraints depend on the operation for which the authorization rule is defined. In general, an authorization rule is defined as follows.

$$\begin{aligned} &\texttt{user is authorized to perform operation} \equiv \\ &\quad \big(\texttt{user has basic permission} \land \texttt{constraints are fulfilled}\big) \\ &\quad \lor \texttt{user has super permission} \end{aligned}$$

A user is allowed to perform the *operation* if he has the required *basic permission* and all *constraints* are satisfied, or if he has the required *super permission*.

The sub formulas are defined using the entities and properties introduced in the previous sections for the definition of structural, behavioral, and timing consistency constraints. Furthermore, additional properties, predicates, and formulas are defined for reasons of clarity. The predicate $\mathtt{Superior}(\mathtt{u},\mathtt{u}')$ defines for two users $\mathtt{u},\mathtt{u}' \in \mathtt{Resources}$ that user $\mathtt{u}$ is superior to user $\mathtt{u}'$ in the responsibility hierarchy derived from the project team structure (cf. Section 5.1.3). For a task $\mathtt{t} \in \mathtt{Tasks}$ the property $\mathtt{t.ResponsibleResource}$ returns the responsible resource of the task, so that the following formula is fulfilled.

$$\begin{aligned} &\forall \mathtt{t} \in \mathtt{Tasks} \forall \mathtt{r} \in \mathtt{Resources}(\mathtt{r} = \mathtt{t.ResponsibleResource} \Leftrightarrow \\ &\qquad \exists \mathtt{a} \in \mathtt{t.TaskAssignments}(\mathtt{a.IsResponsible} = \mathtt{True} \wedge \mathtt{a.Resource} = \mathtt{r}) \end{aligned}$$

To determine the authorization of a user, it is often required to check whether the user is responsible for one of the ancestor tasks of a given task. For this purpose, the following formula is defined for a task $\mathtt{t} \in \mathtt{Tasks}$ and a user $\mathtt{u} \in \mathtt{Resources}$.

$$\mathtt{ResponsibleForAncestor}(\mathtt{u},\mathtt{t}) \equiv \exists \mathtt{t}' \in \mathtt{t.Ancestors}(\mathtt{t}'.\mathtt{ResponsibleResource} = \mathtt{u})$$

The following predicates define for a user $\mathtt{u} \in \mathtt{Resources}$ that he has the respective permission in the project.

$$\begin{array}{ll} \mathtt{ManageTasks}(\mathtt{u}) & \mathtt{ManageAllTasks}(\mathtt{u}) \\ \mathtt{AssignTasks}(\mathtt{u}) & \mathtt{AssignAnyTask}(\mathtt{u}) \\ \mathtt{ExecuteTasks}(\mathtt{u}) & \mathtt{ExecuteAnyTask}(\mathtt{u}) \\ & \mathtt{ViewAllTasks}(\mathtt{u}) \\ & \mathtt{ResourceManagement}(\mathtt{u}) \end{array}$$

The authorization rules are defined as formulas $\mathtt{AuthorizedToOPNAME}(\overline{\mathtt{x}},\mathtt{u})$ which specify under which circumstances a user $\mathtt{u} \in \mathtt{Resources}$ is authorized to perform the operation $\mathtt{OPNAME}$ for the entities specified by the variables $\overline{\mathtt{x}}$. The formulas are implicitly all-quantified over all free variables. If not explicitly stated, the variables $\mathtt{t},\mathtt{a},\mathtt{c},\mathtt{i},\mathtt{o},\mathtt{d},\mathtt{r}$ stand for a task, task assignment, control flow, input parameter, output parameter, data flow, and resource, respectively.

The authorization rules will be explained by means of the example task net depicted in Figure 5.23 which is the same task net cutout as depicted in Figure 2.8. Thereby, it is assumed that all involved resources have only basic permissions.

A user $\mathtt{u} \in \mathtt{Resources}$ is authorized to manage a task $\mathtt{t} \in \mathtt{Tasks}$, if he has the basic permission for task management and he is responsible for the task or one of the ancestor tasks. Alternatively, he may manage the task, if he has the super permission

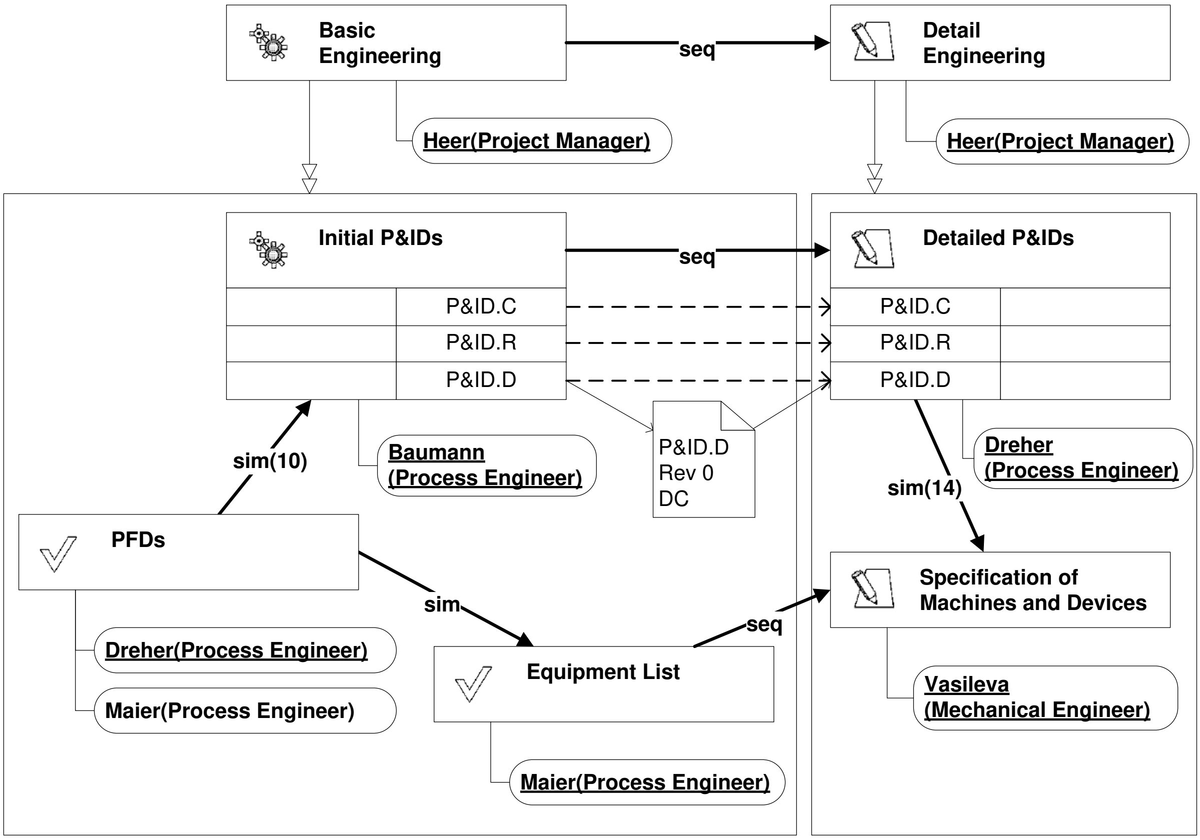


Abbildung 5.23: Example task net for the application of authorization rules.

to manage all tasks in the project.

$$\begin{aligned}
\texttt{AuthorizedToCreateSubtask}(\texttt{t},\texttt{u}) &\equiv \\
\texttt{AuthorizedToDeleteSubtask}(\texttt{t},\texttt{u}) &\equiv \\
\texttt{AuthorizedToAddRealization}(\texttt{t},\texttt{u}) &\equiv \\
\texttt{AuthorizedToRemoveRealization}(\texttt{t},\texttt{u}) &\equiv \\
\texttt{AuthorizedToCreateInputParameter}(\texttt{t},\texttt{u}) &\equiv \\
\texttt{AuthorizedToCreateOutputParameter}(\texttt{t},\texttt{u}) &\equiv \\
\texttt{AuthorizedToDeleteInputParameter}(\texttt{t},\texttt{u}) &\equiv \\
\texttt{AuthorizedToDeleteOutputParameter}(\texttt{t},\texttt{u}) &\equiv \\
\texttt{AuthorizedToCreateTaskAssignment}(\texttt{t},\texttt{u}) &\equiv (\texttt{ManageTasks}(\texttt{u})\wedge \\
&(\texttt{t.ResponsibleResource} = \texttt{u}\vee \\
&\texttt{ResponsibleForAncestor}(\texttt{u},\texttt{t})))\vee \\
&\texttt{ManageAllTasks}(\texttt{u}) \qquad (5.93)
\end{aligned}$$

In the example of Figure 5.23, only resource Heer is authorized to manage the task Basic Engineering.

A user $\texttt{u} \in \texttt{Resources}$ is authorized to view a task $\texttt{t} \in \texttt{Tasks}$, if he is assigned to the task or if he is the responsible resource of one of the ancestor tasks. The user can also view all tasks in the work context of one of his tasks, i.e. the immediate predecessors and successors and the parent task. If he has the super permission to view all tasks in the project, no restrictions apply for the visibility of tasks. All tasks which a user may not view are not displayed in the management views of PROCEED.

$$\begin{aligned}
&\mathtt{AuthorizedToViewTask}(\mathtt{t},\mathtt{u}) \equiv \\
&\quad (\exists \mathtt{a} \in \mathtt{t.TaskAssignments}(\mathtt{a.Resource} = \mathtt{u}) \vee \\
&\quad \mathtt{ResponsibleForAncestor}(\mathtt{u},\mathtt{t}) \vee \\
&\quad \exists \mathtt{t}' \in \mathtt{t.Successors} \cup \mathtt{t.Predecessors}(\mathtt{t}'.\mathtt{ResponsibleResource} = \mathtt{u}) \vee \\
&\quad \exists \mathtt{s} \in \mathtt{t.Subtasks}(\mathtt{s.ResponsibleResource} = \mathtt{u})) \vee \\
&\quad \mathtt{ViewAllTasks}(\mathtt{u})
\end{aligned} \tag{5.94}$$

In the example of Figure 5.23, resource Heer can view all depicted tasks while resource Maier can only view the tasks PFDs, Equipment List, Specification of Machines and Devices, and Basic Engineering. However, resource Dreher can also view all depicted tasks because they form the work context of the tasks for which he is responsible.

A user $\mathtt{u} \in \mathtt{Resources}$ is authorized to modify or delete a task assignment $\mathtt{a} \in \mathtt{TaskAssignments}$, if he has the basic permission for task management and he is responsible for the parent task of the assignment or one of the ancestor tasks. Alternatively, he may change or delete the task assignment, if he has the super permission to manage all tasks in the project.

$$\begin{aligned}
\mathtt{AuthorizedToDeleteTaskAssignment}(\mathtt{a},\mathtt{u}) &\equiv \\
\mathtt{AuthorizedToModifyTaskAssignment}(\mathtt{a},\mathtt{u}) &\equiv (\mathtt{ManageTasks}(\mathtt{u}) \wedge \\
&\quad (\mathtt{a.Task.ResponsibleResource} = \mathtt{u} \vee \\
&\quad \mathtt{ResponsibleForAncestor}(\mathtt{u},\mathtt{a.Task}))) \vee \\
&\quad \mathtt{ManageAllTasks}(\mathtt{u})
\end{aligned} \tag{5.95}$$

A user $\mathtt{u} \in \mathtt{Resources}$ is authorized to create a control flow from the preceding task $\mathtt{p} \in \mathtt{Tasks}$ to the succeeding task $\mathtt{s} \in \mathtt{Tasks}$, if he has the basic permission for task management and he is responsible for the successor $\mathtt{s}$ or for the common ancestor of both tasks. The former allows a responsible resource of a task to define incoming control flows for his task and subsequently according data flows for required documents. A user may also create a control flow if he has the super permission to manage all tasks in the project.

$$\begin{aligned}
\mathtt{AuthorizedToCreateControlFlow}(\mathtt{p},\mathtt{s},\mathtt{u}) &\equiv (\mathtt{ManageTasks}(\mathtt{u}) \wedge \\
&\quad (\mathtt{s.ResponsibleResource} = \mathtt{u} \vee \\
&\quad \exists \mathtt{t}' \in \mathtt{p.Ancestors} \cap \mathtt{s.Ancestors} \\
&\quad (\mathtt{t}'.\mathtt{ResponsibleResource} = \mathtt{u}))) \vee \\
&\quad \mathtt{ManageAllTasks}(\mathtt{u})
\end{aligned} \tag{5.96}$$

In the example of Figure 5.23, the control flow from task PFDs to task Initial P&IDs could have been created by resources Baumann and Heer.

The deletion and modification of a control flow $\mathtt{c} \in \mathtt{ControlFlows}$ is reserved for the responsible resource of the common ancestor of the predecessor and the successor tasks $\mathtt{c.Pred}, \mathtt{c.Succ} \in \mathtt{Tasks}$.

$$\begin{aligned}&\mathtt{AuthorizedToDeleteControlFlow}(\mathtt{c},\mathtt{u}) \equiv\\&\mathtt{AuthorizedToModifyControlFlow}(\mathtt{c},\mathtt{u}) \equiv (\mathtt{ManageTasks}(\mathtt{u})\wedge\\&\quad(\exists \mathtt{t}' \in \mathtt{c.Pred.Ancestors} \cap \mathtt{c.Succ.Ancestors}\\&\quad(\mathtt{t}'.\mathtt{ResponsibleResource} = \mathtt{u})))\vee\\&\quad\mathtt{ManageAllTasks}(\mathtt{u})\end{aligned} \tag{5.97}$$

In the example of Figure 5.23, only resource Heer is authorized to delete the control flow between the tasks PFDs and Initial P&IDs.

A user $\mathtt{u} \in \mathtt{Resources}$ is authorized to create a feedback flow from the source task $\mathtt{s} \in \mathtt{Tasks}$ to the target task $\mathtt{t} \in \mathtt{Tasks}$, if he has the basic permission for task management and he is responsible for the source $\mathtt{s}$ or for the common ancestor of both tasks. If the target task has already been terminated, then the user has to be authorized to create a new version of this task.

$$\begin{aligned}&\mathtt{AuthorizedToCreateFeedbackFlow}(\mathtt{s},\mathtt{t},\mathtt{u}) \equiv (\mathtt{ManageTasks}(\mathtt{u})\wedge\\&\quad((\mathtt{s.ResponsibleResource} = \mathtt{u}\vee\\&\quad\exists \mathtt{t}' \in \mathtt{s.Ancestors} \cap \mathtt{t.Ancestors}(\mathtt{t}'.\mathtt{ResponsibleResource} = \mathtt{u}))\wedge\\&\quad(\mathtt{t.State} \in \mathtt{Terminated} \Rightarrow \mathtt{AuthorizedToCreateNewTaskVersion}(\mathtt{t},\mathtt{u}))))\vee\\&\quad\mathtt{ManageAllTasks}(\mathtt{u})\end{aligned} \tag{5.98}$$

A user $\mathtt{u} \in \mathtt{Resources}$ is authorized to create a new version of a task $\mathtt{t} \in \mathtt{Tasks}$ if he has the basic permission for task management and he is responsible for the ancestor of the task and all succeeding terminated tasks. The latter is required because new versions of the terminated successors will be automatically created as well. Alternatively, he may create the task version, if he has the super permission to manage all tasks in the project.

$$\begin{aligned}&\mathtt{AuthorizedToCreateNewTaskVersion}(\mathtt{t},\mathtt{u}) \equiv (\mathtt{ManageTasks}(\mathtt{u})\wedge\\&\quad(\mathtt{ResponsibleForAncestor}(\mathtt{u},\mathtt{t})\wedge\\&\quad\forall \mathtt{s} \in \mathtt{t.TSuccessors}(\mathtt{s.State} \notin \mathtt{Terminated} \vee \mathtt{ResponsibleForAncestor}(\mathtt{u},\mathtt{s}))))\vee\\&\quad\mathtt{ManageAllTasks}(\mathtt{u})\end{aligned} \tag{5.99}$$

In the example of Figure 5.23, resource Baumann is not authorized to define a feedback flow from Initial P&IDs to PFDs because the latter task is already terminated and resource Baumann may not create a new version of the task. Resource Heer can define the feedback flow which involves the creation of new versions of the tasks PFDs and Equipment List.

Analogously to control flows, a user $\mathtt{u} \in \mathtt{Resources}$ is authorized to create a data flow from an output parameter $\mathtt{o} \in \mathtt{OutputParameters}$ to an input parameter $\mathtt{i} \in \mathtt{InputParameters}$, if he has the basic permission for task management and is responsible for the task of the input parameter or for the common ancestor of both involved tasks. In the first case, a responsible resource of a task can define incoming data flows for documents which are required to perform the task.

$$\begin{aligned}&\mathtt{AuthorizedToCreateDataFlow}(\mathtt{o},\mathtt{i},\mathtt{u}) \equiv (\mathtt{ManageTasks}(\mathtt{u})\wedge\\&\quad(\mathtt{i.Task.ResponsibleResource} = \mathtt{u}\vee\\&\quad\exists \mathtt{t}' \in \mathtt{o.Task.Ancestors}\cap\\&\quad\mathtt{i.Task.Ancestors}(\mathtt{t}'\mathtt{.ResponsibleResource} = \mathtt{u})))\vee\\&\quad\mathtt{ManageAllTasks}(\mathtt{u}) \qquad (5.100)\end{aligned}$$

The control flow and the data flows between Initial P&IDs and Detailed P&IDs could have been created by resources Dreher and Heer.

The deletion of a data flow $\mathtt{c} \in \mathtt{ControlFlows}$ is reserved for the responsible resource of the common ancestor of the involved tasks.

$$\begin{aligned}&\mathtt{AuthorizedToDeleteDataFlow}(\mathtt{d},\mathtt{u}) \equiv (\mathtt{ManageTasks}(\mathtt{u})\wedge\\&\quad(\exists \mathtt{t}' \in \mathtt{d.Source.Task.Ancestors}\cap\\&\quad\mathtt{d.Target.Task.Ancestors}(\mathtt{t}'\mathtt{.ResponsibleResource} = \mathtt{u})))\vee\\&\quad\mathtt{ManageAllTasks}(\mathtt{u}) \qquad (5.101)\end{aligned}$$

A responsible resource of a task may assign additional resources to his task, if he has the basic permission for task assignment. The responsible resource of any ancestor task may assign resources to the task as well. This includes the assignment of the responsible resource. In both cases, the assigned resource has to have the permission to execute tasks, and the assigning user has to be superior to the assigned resource in the project team. Otherwise, a user is only authorized to assign tasks, if he has the corresponding super permission.

$$\begin{aligned}\mathtt{AuthorizedToAssignRessource}(\mathtt{a},\mathtt{r},\mathtt{u}) \equiv\ &(\mathtt{AssignTasks}(\mathtt{u})\wedge\\&(\mathtt{ExecuteTasks}(\mathtt{r})\wedge\\&(\mathtt{a.Task.ResponsibleResource} = \mathtt{u}\vee\\&\mathtt{ResponsibleForAncestor}(\mathtt{u},\mathtt{a.Task}))\wedge\\&\mathtt{Superior}(\mathtt{u},\mathtt{r})))\vee\\&\mathtt{AssignAnyTask}(\mathtt{u}) \qquad (5.102)\end{aligned}$$

In the example of Figure 5.23, resource Dreher could assign resource Maier to the task Detailed P&IDs because the former is the team leader of the latter (cf. Figures 5.9 and 5.10). Resource Heer could assign any resource of the project team to any task in Figure 5.23, because he is superior to all other resources in the project team.

There are several operations which are related to the enactment of a process model instance. First of all, changes to the execution state of a task can be performed by the responsible resource or a responsible for one of the ancestor tasks depending on the state transition. The preparation of a task for execution is the responsibility of the responsible resource of an ancestor task, because it involves the assignment of the responsible resource. A task in the execution state *InDefinition* may not have a responsible resource assigned yet.

$$\begin{aligned}\texttt{AuthorizedToDefineTask}(\texttt{t},\texttt{u}) &\equiv \\ \texttt{AuthorizedToRedefineTask}(\texttt{t},\texttt{u}) &\equiv \\ \texttt{AuthorizedToSkipTask}(\texttt{t},\texttt{u}) &\equiv (\texttt{ExecuteTasks}(\texttt{u})\wedge \\ &\quad \texttt{ResponsibleForAncestor}(\texttt{u},\texttt{t}))\vee \\ &\quad \texttt{ExecuteAnyTask}(\texttt{u}) \qquad (5.103)\end{aligned}$$

A task may only be started by the assigned responsible resource. Likewise, it is reserved to the responsible resource to commit the task. Only the responsible resource of a task can decide when the work on the task is actually started, and when all required results have been produced.

$$\begin{aligned}\texttt{AuthorizedToStartTask}(\texttt{t},\texttt{u}) &\equiv \\ \texttt{AuthorizedToCommitTask}(\texttt{t},\texttt{u}) &\equiv (\texttt{ExecuteTasks}(\texttt{u})\wedge \\ &\quad \texttt{t.ResponsibleResource} = \texttt{u})\vee \\ &\quad \texttt{ExecuteAnyTask}(\texttt{u}) \qquad (5.104)\end{aligned}$$

Finally, suspension, replanning and abortion can be performed by the responsible resource of the task as well as by the responsible resources of the ancestor tasks. The latter is required in cases when the whole subprocess containing the task shall be suspended, replanned, or aborted.

$$\begin{aligned}\texttt{AuthorizedToReplanTask}(\texttt{t},\texttt{u}) &\equiv \\ \texttt{AuthorizedToRestartTask}(\texttt{t},\texttt{u}) &\equiv \\ \texttt{AuthorizedToSuspendTask}(\texttt{t},\texttt{u}) &\equiv \\ \texttt{AuthorizedToResumeTask}(\texttt{t},\texttt{u}) &\equiv \\ \texttt{AuthorizedToAbortTask}(\texttt{t},\texttt{u}) &\equiv (\texttt{ExecuteTasks}(\texttt{u})\wedge \\ &\quad (\texttt{t.ResponsibleResource} = \texttt{u}\vee \\ &\quad \texttt{ResponsibleForAncestor}(\texttt{u},\texttt{t})))\vee \\ &\quad \texttt{ExecuteAnyTask}(\texttt{u}) \qquad (5.105)\end{aligned}$$

In the example of Figure 5.23,the active task Initial P&IDs can only be committed by resource Baumann. However, resource Heer could suspend, replan or abort the task.

The production of document revisions is actually an operation which refers to the enactment of a task, although it has been defined as part of the structural model. Every assigned resource can produce document revisions for the output parameter of a task.

$$\begin{aligned}\texttt{AuthorizedToProduceRevision}(\texttt{o},\texttt{u}) \equiv (&\texttt{ExecuteTasks}(\texttt{u})\wedge \\ &\exists \texttt{a} \in \texttt{o.Task.TaskAssignments}(\texttt{a.Resource} = \texttt{u}))\vee \\ &\texttt{ExecuteAnyTask}(\texttt{u}) \qquad (5.106)\end{aligned}$$

In the example of Figure 5.23, the depicted document revision has been produced by resource Baumann.

Besides the production of document revisions, registering working hours is the only operation which can be performed by assigned resources who are not responsible for a task. The two operations are required to reflect the actually performed work in the system but are not associated with any responsibilities.

$$\begin{aligned}\mathtt{AuthorizedToRegisterWorkingHours(a,u)} \equiv {} & (\mathtt{ExecuteTasks(u)} \wedge \\ & \mathtt{a.Resource} = \mathtt{u}) \vee \\ & \mathtt{ExecuteAnyTask(u)}\end{aligned} \tag{5.107}$$

When a task is executed, its degree of completion can be determined inter alia by direct estimation or by an estimation of the remaining workload required for the task (cf. Chapter 8). These values may only be specified by the responsible resource of the task. In particular, additionally assigned resources may not provide estimates. Since progress measurement is related to the enactment of defined processes, the basic and super permissions to execute tasks are relevant in this context.

$$\begin{aligned}\mathtt{AuthorizedToEstimateProgress(t,u)} \equiv {} & \\ \mathtt{AuthorizedToEstimateRemainingWorkload(t,u)} \equiv {} & (\mathtt{ExecuteTasks(u)} \wedge \\ & \mathtt{t.ResponsibleResource} = \mathtt{u}) \vee \\ & \mathtt{ExecuteAnyTask(u)}\end{aligned} \tag{5.108}$$

With respect to changes to time management properties, different cases have to be distinguished. The planning data, constraint dates and planned dates of a task may not be changed by the assigned responsible resource but only by the responsible resources of the ancestor tasks. Otherwise, a task responsible could arbitrarily increase the total workload, budget, and duration of his task which is beyond his area of responsibility. Instead, he can only request the increase of these values from an authorized resource.

$$\begin{aligned}\mathtt{AuthorizedToChangePlanningData(t,u)} \equiv {} & \\ \mathtt{AuthorizedToChangeConstraintDate(t,u)} \equiv {} & \\ \mathtt{AuthorizedToChangePlannedDate(t,u)} \equiv {} & (\mathtt{ManageTasks(u)} \wedge \\ & \mathtt{ResponsibleForAncestor(u,t)}) \vee \\ & \mathtt{ManageAllTasks(u)}\end{aligned} \tag{5.109}$$

In the example of Figure 5.23, resource Dreher was not allowed to increase the total duration of the task PFDs while it was still active. He had to request this duration increase from resource Heer.

In case of a task assignment, the planned workload can be changed by the responsible resource of the corresponding task. If an increase does exceed the unassigned total workload of the task, the authorization rule (5.109) is evaluated to determine whether the resource may increase the total workload and budget of his task. The responsible resources of a task and its ancestor tasks are authorized to change the time constraint for the maximal daily workload of a task assignment of the task.

$$\begin{aligned}&\texttt{AuthorizedToChangePlannedWorkload}(\texttt{a,u}) \equiv\\&\texttt{AuthorizedToChangeMaxDailyWorkload}(\texttt{a,u}) \equiv (\texttt{ManageTasks}(\texttt{u}) \wedge\\&\quad(\texttt{a.Task.ResponsibleResource} = \texttt{u} \vee \texttt{ResponsibleForAncestor}(\texttt{u,a.Task}))) \vee\\&\quad\texttt{ManageAllTasks}(\texttt{u})\end{aligned} \tag{5.110}$$

The lag time of a control flow may only be changed by the responsible resource of the common ancestor task.

$$\begin{aligned}&\texttt{AuthorizedToChangeLagTime}(\texttt{c,u}) \equiv (\texttt{ManageTasks}(\texttt{u}) \wedge\\&\quad(\exists \texttt{t}' \in \texttt{c.Pred.Ancestors} \cap \texttt{c.Succ.Ancestors}(\texttt{t}'\texttt{.ResponsibleResource} = \texttt{u}))) \vee\\&\quad\texttt{ManageAllTasks}(\texttt{u})\end{aligned} \tag{5.111}$$

The responsible resource of a task and the responsible resources of the ancestor tasks may initiate a scheduling pass for a task in one of the execution states *InDefinition* or *Replanning*. The responsible resource who is not at the same time responsible for an ancestor task may initiate scheduling, but the changes are only applied if the planning data and planned dates of the task itself are not affected.

$$\begin{aligned}\texttt{AuthorizedToScheduleTask}(\texttt{t,u}) \equiv\ &(\texttt{ManageTasks}(\texttt{u}) \wedge\\&(\texttt{t.ResponsibleResource} = \texttt{u} \vee\\&\texttt{ResponsibleForAncestor}(\texttt{u,t}))) \vee\\&\texttt{ManageAllTasks}(\texttt{u})\end{aligned} \tag{5.112}$$

Finally, there are operations related to resource management. Changes to the project team structure and changes to the permission settings can only be performed by a user who has the permission for resource management in the project. The cost rates and work calendars of resources cannot be changed by a user who has the resource management permission for the project. Changing the cost rates and working times of resource is not in the area of responsibility of project managers.

$$\begin{aligned}&\texttt{AuthorizedToChangeProjectTeam}(\texttt{u}) \equiv\\&\texttt{AuthorizedToChangePermission}(\texttt{r,u}) \equiv \texttt{ResourceManagement}(\texttt{u})\end{aligned} \tag{5.113}$$

At this point, the approach should be elaborated to distinguish between resource allocation for a project and restructuring the project team. In practice, a project manager cannot simply add new resources to his project team but has to request the resources from line managers or department heads. This distinction has not been made in PROCEED for reasons of simplification.

The authorization rules defined in this section are evaluated whenever a user tries to invoke the respective operation. If he is not authorized to perform the operation, a warning is shown an the operation is not invoked. Furthermore, the functionality provided by the graphical user interface of PROCEED is adapted according to the evaluation of authorization rules. Several functions are deactivated if the user is not authorized to perform the associated operations on the dynamic task net.

5.5.3 Project-Specific Tailoring of Access Control Policy

With the combination of passive and active access control approaches, the authorization model implemented in PROCEED can be tailored to support the management of dynamic task nets in *various ways of collaboration*. The permission settings can be configured for an individual project to support the *delegation* of tasks from superiors to subordinates, the *collaboration* of engineers with equal rights, and even the *observation* by progress chasers or consultants.

Usually, a combination of several management styles is required in a plant design project. The project manager is given the required super permissions to assign coarse grained tasks directly to the lead-engineers of the different maintenance groups. The lead-engineers can assign tasks to their team members but not to other resources. This is realized by the authorization rule for direct task assignment, which takes the project team structure of the project into account. A lead-engineer is only given the basic permissions, but since he is the leader of his team, he may assign tasks directly to his team members. In addition to the direct task assignment, a task assignment can also be defined by only specifying the required role. Then a user, who can play this role, can pick up the task. This mode of task assignment reflects a common practice in development projects.

In plant design projects there are usually so-called progress chasers who are responsible for gathering the progress estimates from the engineers. From this data, the project status report is compiled. Furthermore external consultants may be hired to evaluate the performance of the development process. Progress chasers and consultants should be able to view the tasks in the project, but they should not be allowed to make changes to the management data. To achieve this, their user accounts are given only the super permission to view all tasks in the project but none of the basic permissions. Without the permissions to manage and execute tasks, they cannot be assigned to any task and therefore are prevented from making any changes.

The permission settings of a project can also be configured to support the purely hierarchical delegation mode. In PROCEED, this behavior can be simulated by assigning all super permissions to the project manager, and only the basic permission to execute tasks to the engineers. As a result, the managers can view and modify all tasks in the project. The engineers can only view their assigned tasks, perform the allowed execution state changes, produce document revisions, and register working hours. The AHEAD-prototype only supported this management mode, because changes to the management data were only possible with the dedicated environment for project managers. The engineers could not make structural changes in their work environments. Likewise, in conventional static workflow management systems, process participants are usually restricted to executing their assigned tasks while the system manages the process.

In recent times, *agile process models* have become increasingly popular [TFR05]. In these process models there is no strict hierarchy, and the rather small development team consists of developers with equal rights. The PROCEED system can even support these agile processes by assigning only the basic permissions to all project team members, but no super permissions to anyone (except for the person who initiates the project and assembles the project team). As a result, the constraints defined in the authorization rules are always evaluated, and permissions are only activated in the defined contexts.

5.6 Related Work

Research that is related to the contributions described in this section can be distinguished into work on resource modeling and user authorization. Related work that deals with time management in process models and the monitoring of development processes will be reviewed in Section 7.6 and Section 8.4, respectively.

5.6.1 Resource Modeling

The approach to resource modeling in AHEAD and PROCEED has undergone a long development. First concepts were presented in [NW94] which were substantially elaborated in [Krü96, KW00, KKSW00] and resulted in the resource management meta-model RESMOD which has been described in Section 4.1. The approach to resource modeling defined in RESMOD has been adapted and simplified in PROCEED to account for the specific requirements arising from the technology transfer to the Comos system. Thereby, some of the concepts defined in RESMOD have been abandoned.

In Table 5.7, the three different approaches to resource modeling are compared to each other by mapping equivalent concepts. In [NW94] and in RESMOD, technical and human resources could be modeled. The research on RESMOD focused even more on technical resources than on human resources and in particular on modeling complex technical resources. In PROCEED, only human resources are modeled to simplify project planning. For the following comparison only human resources are considered.

The concept of an actual resource in [NW94] corresponds to the actual base resource in RESMOD and the resource in PROCEED. Employees of the company which are available for engineering projects are represented by objects in the database.

In RESMOD, actual project resources were introduced to model the members of a project team distinct from the actual base resources in the organization. An actual project resource could correspond to an actual base resource or to an external resource who was not a member of the organization. In [NW94], this distinction was not made. In PROCEED it has been abandoned because resources are defined on the organizational level in Comos and are only referenced in individual projects. External resources have to be represented as members of the organization.

The concept of a planned base resource was introduced in RESMOD to be able to model positions in an organization independent of actual resources who fill these positions. Again, this concept has been abandoned in PROCEED because the simple resource model of Comos does not allow to model positions separate from users.

Nagl and Westfechtel	**RESMOD**	**PROCEED**
Technical and human resources	Technical and human resources	Only human resources
Actual resource	Actual base resource	Resource
	Actual project resources	
	Planned base resource	
	Planned project resource	
Abstract resource	Planned resource class	Role
	Needed resource	
	Resource properties	
Resource assignment mechanism	Resource assignment	Resource assignment
	Resource allocation	Project team definition
Complex resource	Complex resource	
	Complex actual project resource with human resources as atomic parts	Team
		Task assignment

Tabelle 5.7: Equivalent concepts in [NW94], RESMOD, and the TNT meta-model.

The concept of a planned project resource in RESMOD was an advancement from the concept of an abstract resource in [NW94]. It did not only define the required role of a resource but at the same time a position in the project team. A planned project resource is therefore not equivalent to a role as argued in Section 4.1. Instead, the concept of a planned resource class is equivalent to a role as well as the abstract resource in [NW94]. It abstracts from individual positions in the project team. In PROCEED, roles are used to define resource requirements for tasks since the concept of a role is more common in practice. Furthermore, the explicit modeling of task assignments allows to define several required resources with the same role for a task. The project team is modeled by means of teams. Therefore, the position aspect of a planned resource is not needed in PROCEED.

The concept of a needed resource is used in RESMOD to define resource requirements of actual resources. This is particularly useful for technical resources which often depend on other technical resources. In contrast to that, it is uncommon to define dependency relationships between human resources. Since in PROCEED only human resources are modeled, the concept of a needed resource has been abandoned as well.

In RESMOD, properties of resources can be defined for actual and planned resources. Properties of planned resources define requirements which have to be matched by the property values of the assigned actual resources. In PROCEED, only human resources are modeled and roles are used instead of planned resources. A role defines implicitly the properties a resource who can play this role has. There has not been the need to explicitly model individual properties of resources and according requirements for tasks or task assignments. Thereby, the associated modeling overhead is avoided and resource modeling is simplified, which increases the applicability of the approach in practice.

Complex resources do not exist in PROCEED except for teams which can be regarded as complex human resources. In RESMOD, a team would be modeled as a complex actual project resource with human resources as atomic parts. However, it would not be possible to define a team leader in a straightforward way in RESMOD.

Finally, in RESMOD resources can be shared with respect to the composition relationship, i.e. a human resource could be a member of two different teams. This has not been adopted in PROCEED since project team structure should unambiguously define the responsibility hierarchy in a project and therefore should be a tree structure in which every resource belongs to exactly one team.

In [NW94], actual resources were defined for the organization and abstract resources for individual projects. When an actual resource was assigned to an abstract resource this resource assignment mechanism included two steps: First, the allocation of the resource for the project, and second the assignment of the actual resource to the planned resource. In RESMOD, these two steps were explicitly separated. The resource allocation created an actual project resource which referred to the corresponding actual base resource. The definition of actual project resources was the equivalence to the assembly of a project team. The resource assignment was then established as a link between actual and planned project resources. In PROCEED, there are no actual project resources, but the two steps can be executed separately as well. The resource allocation is performed by defining the project organization in the form of a hierarchical team structure, and by assigning resources to the defined teams. Resource assignment in PROCEED and RESMOD are not exactly comparable. However, the assignment of an actual resource to a planned resource in RESMOD can at best be compared to the assignment of a resource to an explicit task assignment for which a required role is defined. In this case, the resource has to be able to play the required role which is defined on the organizational level.

5.6.2 User Authorization

The management of access rights is an issue in many types of information systems. In this section we first discuss general approaches before we consider research work concerning to access control in process management systems.

There are basically two approaches to access control: the passive approach and the active approach [TS97]. In the *passive approach*, also known as subject-object view, permissions are maintained as assignments. Once a permission is assigned to a user, it is always assumed to be activated independent of the given context. On the other hand, the *active approach* distinguishes context-based permission activation from permission assignment. Permissions are constantly monitored and can be activated and deactivated according to the respective context. The access control approach presented in this thesis combines the active and the passive approach. Basic permissions are only activated in certain contexts while super permissions are always activated.

Role-Based Access Control (RBAC) [SCFY96] is a passive security model. The permissions are assigned to roles rather than to individual users. Roles are created based on job functions in the organization and users are assigned to roles based on their qualification and responsibilities. Thus, a user assigned to a role thereby acquires the permissions of that role. This grouping of users in roles for access control eases the management of permissions. One of the drawbacks of RBAC is its inability to support users with the same role but with different permissions. This is an important criterion in a dynamic process management system where users with the same functional role can have different permissions. Therefore, the role-based approach has not been implemented in PROCEED. Instead, permissions are assigned to users of the system individually.

In several research projects, the RBAC model has been adapted and applied to workflow management systems [ASKP00, KPF01]. The implementation of RBAC on web-based workflow systems is presented in [ASKP00], while RBAC is extended to support inter-organizational workflow in [KPF01]. In the context of workflow management, the role-based access control model is more suitable than in the context of development processes. When it comes to the enactment of business processes, functional roles usually coincide with permissions, e.g. for the accounting clerks and the accounts manager in an accounting office.

In [AH96], Atluri and Huang present an active security model where authorization is synchronized with the current state of tasks in workflows. In other words, a resource can gain access to required objects, e.g. documents, only during the execution of a task. To achieve this, an authorization template is used to specify the parameters of the authorization which is attached to a task. When the task is actually started, the authorization template is used to derive the actual authorizations. In this approach, authorization constraints are associated with specific tasks or workflow definitions, and there are no authorization rules regulating the dynamic creation of subtasks or other dynamic structural changes to a workflow instance.

In the Team-based Access Control model presented in [AC04], permissions can be specified for a group of collaborating users, acting in various roles. This means that a team member not only has permissions from the role he is playing, he also has additional permissions from the team he is in. This model only defines if team members can access certain objects within the context of the team. There is no explicit connection to any process model instance and task assignments. Furthermore, specific permissions are also not associated with teams in the TNT meta-model.

In [WBK03], the W-RBAC framework for workflow systems is presented, which is based on RBAC. Authorization constraints are specified using a logic-based language. A permission service in the framework is used to compile a list of users who can execute a task based on the specified constraints. W-RBAC only focuses on the assignment and the execution of tasks. The defined constraints are specific for certain types of tasks whereas in PROCEED general constraints for all tasks in a process model instance are required.

5.7 Conclusion

In this chapter, the TNT meta-model for timed dynamic task nets has been presented. Several adaptations and extensions have been applied to the DYNAMITE meta-model to obtain the TNT meta-model. The structural and behavioral model have been adapted to meet the requirements which arose from the approach to project planning and process enactment realized in PROCEED. With respect to resource management, subteams of the project team with respective team leaders can be defined, and task assignments are explicitly modeled as separate objects, so that several resources can be assigned to a task. With respect to the definition and execution of tasks, the major extensions are the explicit modeling of the granularity of a task, and a the extension of a task’s life cycle by the possibility to skip tasks without starting them in the first place.

The major extensions of the DYNAMITE meta-model refer to time management and progress monitoring. The structural model and the behavioral model have been complemented by the timing model and the monitoring model. These partial models define the required entities and properties for task scheduling and progress measurement, respectively. The actual performance of a process is explicitly modeled, separately from the planned performance. This allows a comparison of actual and planned performance. The alignment of the plan to the actual performance is an explicit management decision.

The authorization model of the TNT meta-model identifies human resources with users of the system and introduces the concept of user permissions. Authorization rules constrain the possible change operations of individual users depending on their permissions and task assignments. This extension of the DYNAMITE meta-model has been required due to the switch from separate environments for different roles to one environment for all users.

Altogether, this chapter introduced the modeling elements for process model instances in PROCEED. The next chapter describes how process model definitions can be created in PROCEED, and how they are applied and enacted in a concrete project. The algorithms for project scheduling and monitoring, which build on the modeling elements introduced in this chapter, will be described in Chapter 7 and Chapter 8, respectively.

Kapitel 6

Process Modeling and Enactment

A dynamic task net as it has been defined in Chapter 5 represents a process model instance. The combination of a project plan with data flow dependencies and execution states of tasks in one integrated model is advantageous for the management of development projects. However, dynamic task nets do not allow to define process knowledge which can be reused in several similar projects, and which can be improved and extended from project to project. For this purpose, it is required to maintain process model definitions containing the process knowledge which is independent of actual process model instances.

The first step is to define domain specific task types which may be instantiated several times in a process model instance. For example, the creation of a process flow diagram is a task type which has several instances in a plant design project, one for each individual process flow diagram. Task types are used in process model definitions to distinguish the different tasks in the process. Knowledge about the document input and output parameters, the resource requirements in terms of roles and workload, and the planning data of a task type can be stored in the process knowledge base. In general a type defines common properties of a class of objects and possibly initial property values. In the TNT meta-model, the properties of a task are fixed and so are the properties of a task type. Only the different property values distinguish different task types. When a task type is used in a project, an instance of the type is inserted into the dynamic task net as a new task whose property values are set to the values defined by the type. The usage of task types for process modeling in PROCEED is described in Section 6.1.

Task types contain process knowledge about individual tasks, but they do not define the possible relationships between different tasks and the allowed or required number of task instances in a process model instance. For this purpose, process model definitions are required.

There are different approaches for modeling process model definitions. On the one hand there are procedural approaches. Procedural process model definitions explicitly specify in which order the defined tasks have to be executed, when the process is enacted. As a consequence, procedural process model definitions can be executed automatically if required. On the other hand, there are declarative process model definitions which merely constrain the allowed manual changes to a process model instance. The process model definitions on type level which have been described in Section 4.4 fall into this category.

A third type of process model definitions, which does not clearly fall into one of the two categories, are instance patterns or process model definitions on the instance level (cf. Section 4.4). These process model definitions are basically *process templates* which can be reused for the manual modeling of a process model instance by copying and inserting them into an existing dynamic task net. Process templates are used in PROCEED for the definition of process knowledge. Section 6.2 describes how process templates can be defined and used in PROCEED.

Process templates can be augmented by a workflow definition which enables the automatic enactment of the respective processes in a dynamic task net. Workflows are modeled using control structures like alternative branching constructs and loops (cf. Section 3.4). Therefore, they are classified as procedural process model definitions. The integration of workflow instances into dynamic task nets is described in Section 6.3.

6.1 Task Types

A task in a dynamic task net can be derived from a *task type*. A task type specifies default values for the properties of all instances of this type. When a task is instantiated from a task type, then the default values are set for the respective properties of the task. For properties of the task type which are undefined, the general default values are used for the properties of the task instance, which have been described in Section 5.3.1.

The class diagram depicted in Figure 6.1 shows on the left the class Task with all relevant properties for time management and progress measurement and on the right the class TaskType. Only those properties are defined for the class TaskType whose values can be reused in different projects. A specific task type does not define additional properties but only specific property values. The planning data of a task constitutes valuable process knowledge. Task instances of the same type will probably require a similar amount of workload, duration, and budget if they are instantiated in comparable projects. The respective property values defined for the task type are good starting points for project planning.

With respect to progress measurement, the calculation method for the degree of completion of a task can be specified for a task type. It is very probable that the progress of all instances of a common task type is measured by means of the same calculation method. Tasks which are defined as milestones in a dynamic task net can be used for measuring the progress of the parent task. For this purpose, a milestone defines a percentage value for the property OverallDOC (overall degree of completion). When a type is defined for a milestone, then the value for the overall degree of completion can be specified for the type and reused for the milestones which are instantiated from the type.

All instances of a task type are connected by the instance of association with the respective type as depicted in Figure 6.1. The PROCEED system keeps track of which tasks have been instantiated from which type. This information is required to derive reference values for the planning data of the task type from the task instances.

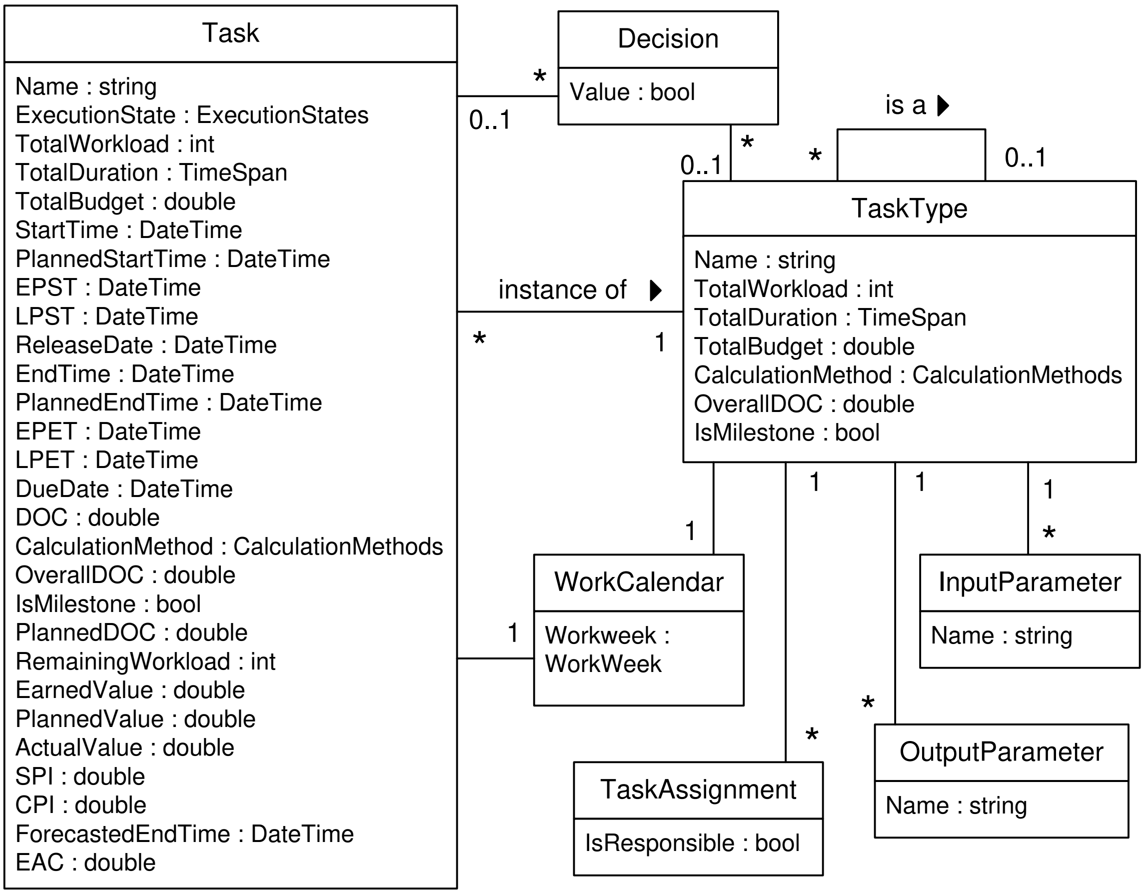


Abbildung 6.1: Classes and associations for task types.

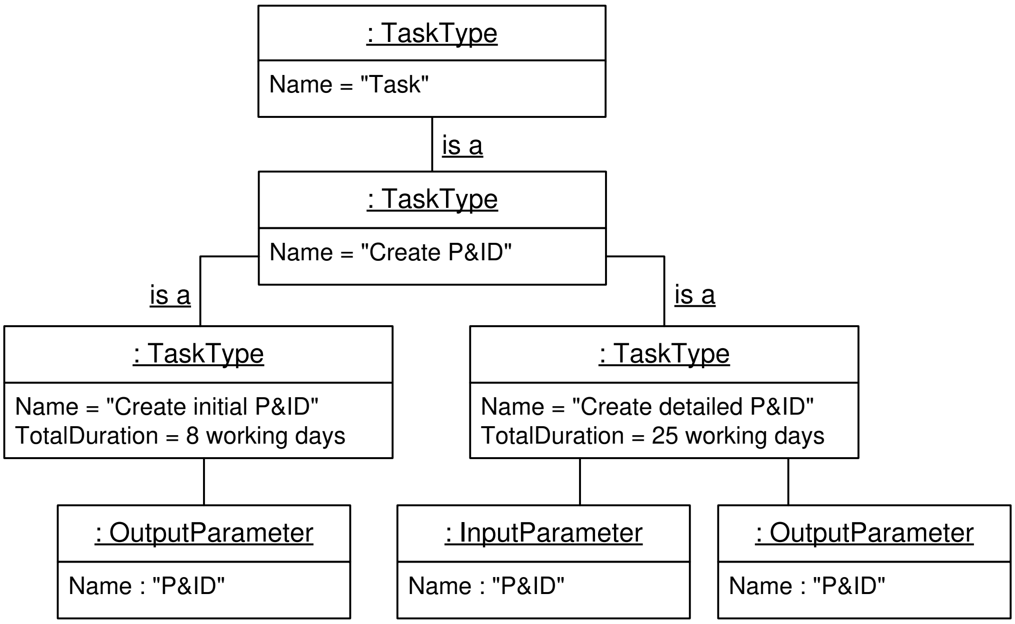


Abbildung 6.2: Example for the specialization of task types.

Task types can be structured in a generalization hierarchy. The root of this hierarchy is the most general and domain-independent task type which cannot be changed by the users of PROCEED. By means of specialization, domain-specific task types can be derived. A special task type which has been derived from a more general one does not define additional properties but only specific property values. By default, property values are inherited by a specialized task type from the general type, but they can be overridden with specific values. The generalization is defined by the association is a in Figure 6.1 and not as a UML generalization relationship. This way, a generalization hierarchy of objects of the class TaskType can be built at tool runtime which corresponds to the parameterized specification approach described in [KN04].

Figure 6.2 shows an example of a generalization hierarchy of task types. The most general type with the name Task is predefined by PROCEED and cannot be changed by the user. A specific task type has been defined for the creation of piping and instrumentation diagrams. There are two specializations of this type, one for the initial creation of a P&ID and one for the elaboration of a P&ID. The former specifies a shorter duration than the latter. Furthermore, the former type has only one output parameter for the P&ID to be created. In contrast, the task type for the elaboration of a P&ID has an input and output parameter for the modified document.

Document input and output parameters can be defined for a task type. These parameters are cloned and associated with an instance of the type upon instantiation. Likewise, decision variables can be defined for task types which are then available for every instance of the type. A decision variable is an object of the class Decision which is depicted in Figure 6.1. Its boolean value represents the result of a decision made by the responsible resource of the corresponding task. Decision variables are evaluated during the enactment of workflow-managed task nets which will be introduced in Section 6.3.

Task types can be used for the creation of dynamic task nets which represent process model instances. Whenever the user who defines a task net wants to create a new subtask, he can browse the process knowledge base for appropriate types. If he selects a suitable type for a new task, all specified property values of the type are set for the new task. The user can adapt the property values as required, but this will not change the type of the task. The connection between the task instance and the type has to be deleted manually, if the task shall no longer have the type.

The life cycle of a task can be divided into three phases. First, the type is defined. Second, the type is instantiated as a task in a process model instance, and third, the task is executed. Table 6.1 gives an overview with respect to the three life cycle phases over which properties may have values assigned, which property values are allowed, and who is authorized to change the property values. A task type does not have an execution state explicitly defined. However, in terms of the task execution states a type would be in the state *InDefinition* because this is the default value for a new task after its creation. When a task has been instantiated from a type, its execution state may be changed to *Waiting*. Later, the task is started and eventually terminated. With respect to resource management, only required roles of

	Type definition	**Type instantiation**	**Task execution**
Execution state	InDefinition	Preparing	Running, Terminated
Resource properties	Required roles	Assigned resources	Assigned resources
Product properties	Input/Output parameters	Documents	Revisions
Time management properties	Planning data,	Constraint dates, planned start and end times	Actual start and end times, forecasted duration and end time
Authorization	Type creator	Task creator	Responsible resource(s)

Tabelle 6.1: Editable properties of types and tasks.

task assignments can be defined for a task type. Because a task type is independent of any concrete project, no actual resources from a project team can be assigned to a task type. Resources can be assigned after the task has been instantiated. With respect to product management, only input and output parameters can be defined for a task type. The association of parameters with actual documents can only be done for a task instance in the context of a development project. Revisions of the associated documents can only be produced at runtime of the task. With respect to time management properties, the planning data of a task can be specified in a type. Constraint dates and planned dates are project-specific. Actual start and end times and forecasts can only be determined during the execution of the task. Finally, a user must be authorized to make changes to a task type or a task in the different life cycle phases. The person who creates a task type may be a domain expert or a so-called knowledge engineer in a company. He populates a knowledge base with domain-specific task types. However, he is usually not involved in any concrete development project. The user who is authorized to create a new task instance in a development project is also authorized to change the property values of this new task. When a resource is assigned to a task instance and designated to be responsible for the task, he may modify the task according to the authorization rules which have been presented in Section 5.5.

6.2 Process Templates

Task types are already a way to reuse process knowledge for different projects. However, task types alone do not allow to define any task relationships. Therefore, process templates are required.

A process template in PROCEED is a dynamic task net which can be reused by

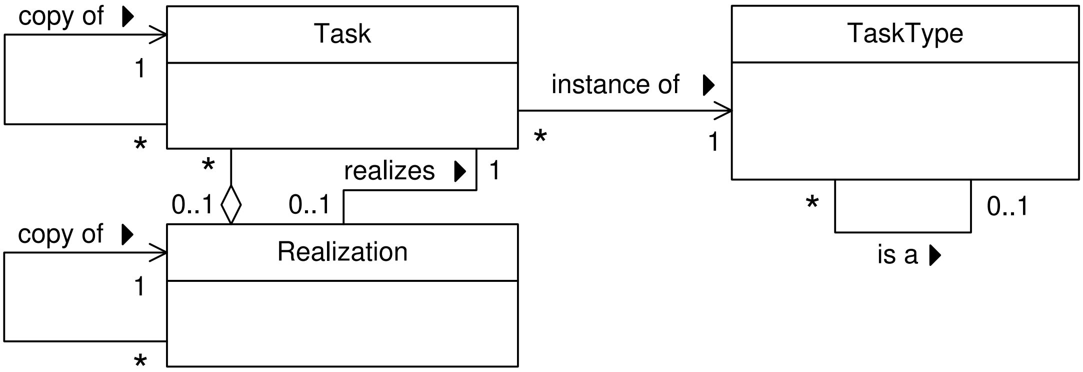


Abbildung 6.3: Additional associations for process templates.

copying it as a subprocess into a process model instance. A task and its realization together amount to a process template. This means that a process template always defines a complete subprocess and not only a building block for a process. The realization of a process template may contain several subtasks which can be connected by control and data flows and which can themselves have subtasks. Consequently, a process template can be a hierarchically structured dynamic task net, but it always has a unique root task.

When a process template is used in a concrete project, the root task and its complete realization are copied and inserted as a new task in the dynamic task net which represents the development process. Afterwards, resources can be assigned to the task and its subtasks, the tasks can be scheduled and finally executed.

Besides the insertion of a process template as a new task in an existing dynamic task net, there is a second way of using a process template. If a task has already been defined in a process model instance, its realization can be created or exchanged by copying the realization of a process template. In this case, the properties of the root task of the process template are not reused but only the subtasks and their relationships. The connection between document parameters of the task in the process model instance and its new realization has to be established manually. Likewise, possible inconsistencies related to time management properties of the task and its new realization have to be resolved in the process model instance. This functionality is useful whenever an existing task is already connected with other tasks in the dynamic task net and project specific property values have been specified, but the realization shall be created according to an existing template.

In [Sch02], process model definitions were divided into process model definitions on type level and on instance level (cf. Section 4.4). In contrast to task types, process templates are located on the instance level. There is no generalization relationship for process templates. A process template is copied into an existing task net but is not instantiated. The tasks in a process template are already instances of task types. As a consequence, the tasks in a process template and a process model instance are instances of the same class Task defined in the TNT meta-model.

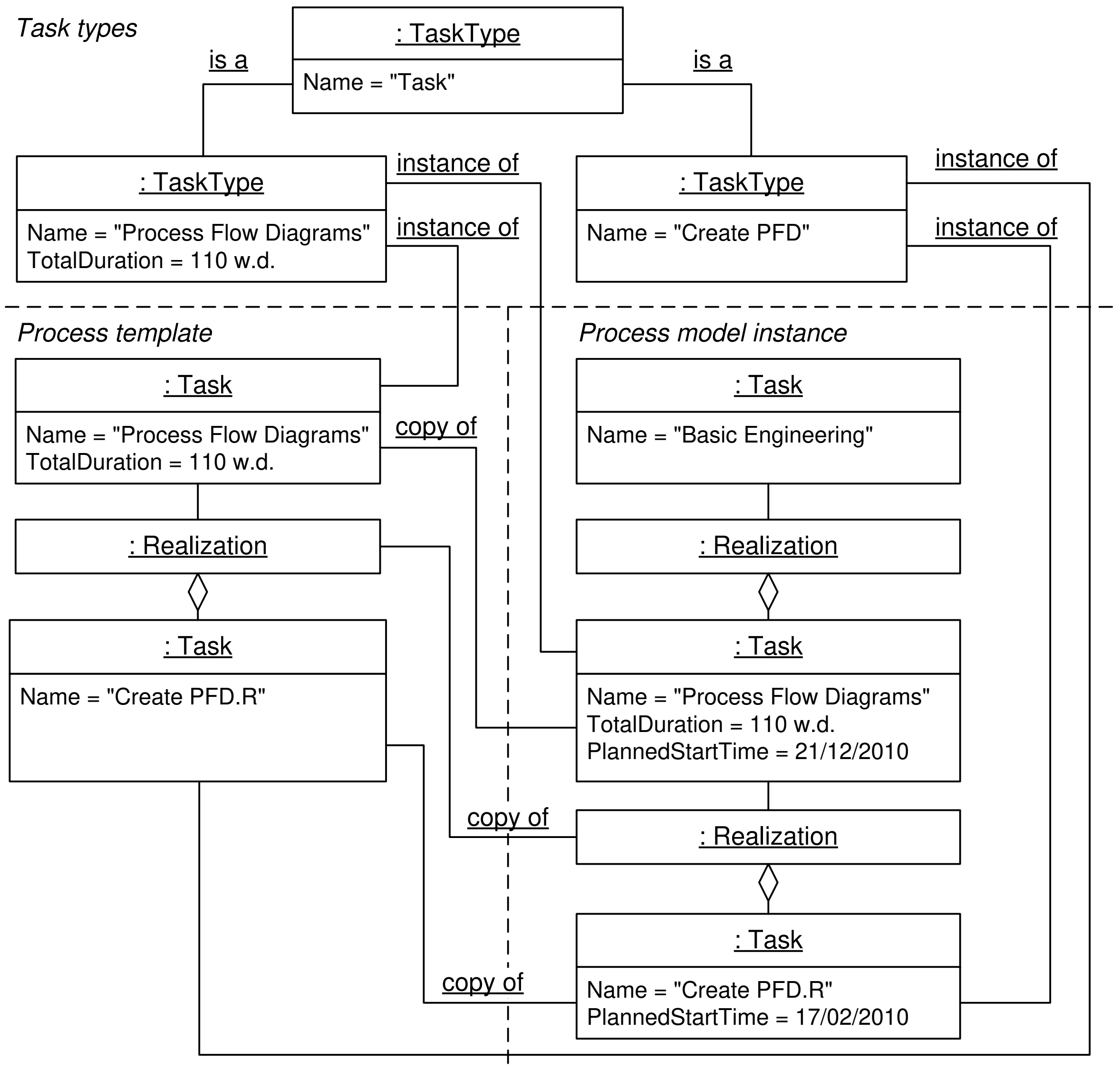


Abbildung 6.4: Example for the usage of a process template.

Figure 6.3 shows the extensions to the meta-model which are required for process templates. Objects of the classes Task and Realization may be contained in a process model instance or a process template. The association instance of between the classes Task and TaskType has already been introduced in Figure 6.1. It connects tasks which have been instantiated from a task type with the respective type.

When a process template is copied into a dynamic task net, then the copied tasks and realization objects are automatically connected with the respective objects in the process template by means of the association copy of. In this way, the PROCEED system keeps track of which subprocess of a development process is a copy of a process template and which subtask is a copy of which subtask in the respective template. This information is required to improve a process template based on dynamic changes to the process model instances.

An example for the usage of a task type is shown in Figure 6.4 in the form of a UML object diagram. On the type level, three task types are defined: the general domain-independent task type which is predefined by PROCEED and two specializations, one for tasks in which several process flow diagrams are created, and one for tasks in which one individual process flow diagram is created. For the task type Process Flow Diagrams, an expected total duration of 110 working days has been defined. This type is instantiated as a task which is the root task of a process template. The task instance has the same property value for the total duration. The process template Process Flow Diagrams contains several instances of the task type Create PFD, one for every process flow diagram that is expected to be created in the defined subprocess, i.e. the process template defines tasks for specific process flow diagram while the task type Create PFD can be generally used for all tasks in which a process flow diagram has to be created. One of the subtasks is depicted in Figure 6.4. In the subtask Create PFD.R, the process flow diagram for the reaction part of the chemical process shall be designed. However, the task does not yet refer to an actual document in a development project. This connection can only be established after the process template has been copied into a process model instance. In the lower right corner of Figure 6.4, a process model instance is depicted in which the process template has been used. The copied tasks and the realization object are linked to the original instances in the process template. Tasks in a process model instance may be scheduled in contrast to tasks in a process template, and their output parameters can be connected with actual documents.

Although process templates are composed of task instances, the properties of the tasks are not editable like the properties of tasks contained in process model instances. For the tasks in a process template, the same restrictions apply as for task types, because the tasks are still part of a template and not part of a process model instance, although they are technically instantiated from task types. According to Table 6.1, a task in a process template can only be edited as defined in the column Type definition, i.e. the execution state can only be *InDefinition*, required roles for task assignments can be defined but no actual resources, input and output parameters for documents can be defined but not associated with actual documents, and only the planning data can be defined for the task but no constraint dates or planned dates. A process template may also define task relationships. In this regard, the only constraint for editing process templates is that no feedback flows can be defined. A feedback flow can only be defined originating from active tasks but a process template is not enacted. With respect to the authorization of users for editing operations, the same restrictions apply as for task types (cf. row five in Table 6.1). The template creator may modify the template. The user who used a process template to create a new subtask may edit the copied task net structure. Finally, when responsible resources are assigned to the root task of the subprocess and the subtasks, then these resources may edit their assigned tasks according to the authorization rules which have been presented in Section 5.5.

A process template can only be used as a new task in an existing task net or as a new realization of an existing task. In this regard, process templates differ from instance patterns which were introduced in [Sch02]. An instance pattern did not necessarily represent a whole realization of a task but could consist of connected tasks which could be inserted into an existing realization (cf. Section 4.4). Furthermore, an instance pattern defined the required context for inserting the pattern and how the tasks of the pattern should be connected with the context tasks. These modeling possibilities have not been adopted in PROCEED. Process templates have to have a unique root task. This is an essential requirement for workflow templates which will be introduced in the next section. No context for the insertion of a process template into a task net can be defined. The abandonment of context definitions for process templates has the advantage, that process modelers do not have to learn and use additional modeling constructs or even a new modeling language like UML object diagrams which were used in [Sch02] for this purpose. They can model process templates just like a process manager would define a dynamic task net in a concrete project, whereby the previously described limitations apply.

Declarative process model definitions on type level which were introduced in [Sch02] were also not adopted in PROCEED. Although this concept for declarative process modeling on type level might in general be a valuable extension to a process management system like PROCEED, it has not been considered suitable in the industrial context of the research project. The usage of UML class diagrams for the declarative modeling of processes has been considered disadvantageous with respect to the acceptability of the process management functionality in Comos. Domain experts in the chemical industries are usually not familiar with the UML notation. The concepts of type and instantiation are not well-known among engineers. On the other hand, Comos users are very familiar with the concept of a template. Furthermore, the declarative approach to process modeling is rather uncommon. Instead, operational process definitions have been demanded by the industry partner. Therefore, workflow templates are used in PROCEED which will be described in the next section. Workflow templates comprise additional information about alternative courses of action and the planned iteration of tasks. Finally, task, process and workflow templates are sufficient for elementary process modeling as it is required for project controlling. Reference data regarding the required workload, duration, budget, task assignments, functional roles, and document parameters of tasks can be maintained using task types and process templates.

6.3 Workflow Management

Workflow management has recently gained in importance in the different engineering domains. Workflow management systems [JB96] are used to support well-defined individual and collaborative processes. The workflow approach allows for a partial automation of processes, and the available technologies enable interoperability with other systems and applications in service oriented architectures. The usefulness of workflow support for development processes has been identified in academia [Bau04, MDB^{+}00, CC02, Bus98] and industry.

However, workflow management systems are not suitable for the management of whole development processes. They can only support the enactment of structured subprocesses which may be predefined in advance. Unlike project management systems, project planning goes beyond the scope of workflow management systems. Furthermore, it is not feasible to model a complete development process as one workflow due to the inherent uncertainties and dynamics. As a consequence, enacted workflow instances which represent subprocesses in a development project would be disconnected in a WfMS and would not be embedded into the context of the development process.

For these reasons, several research groups have investigated the opportunities of integrating project management systems (PMS) with workflow management systems (cf. related work presented in Section 6.4.1). In all of these approaches, project plans are used for global planning, and structured subprocesses are enacted by means of workflow management systems. Integration components couple these systems at runtime, so that workflow instances may be represented in project plans. However, the different integration approaches all face the problem that no information about the process enactment state is maintained in the respective project management system. Thus, the enactment state of workflow instances cannot be adequately mapped to the project plan which limits the possibilities for project monitoring. Furthermore, since products and data flows are not represented in project plans, product management is still beyond the scope of the integrated systems. Data flows between workflow instances have to be managed manually. These restrictions are overcome by integrating workflow management functionality into the process management system PROCEED.

In Section 4.3.3, dynamic task nets have been compared to project plans and workflow instances. It has been highlighted that dynamic task nets are a suitable representation of development process instances. However, process automation as supported by workflow management systems is not possible for a dynamic task net without an operational process model definition. Declarative process model definitions and process templates do not enable the automation of process model instances. The former provide too many degrees of freedom for a process engine to decide which operation to perform upon a certain event. The latter are merely templates to be copied and do not provide any additional information for the automatic enactment of the processes. The functionality provided by workflow management systems is complementary to the process management functionality provided by the PROCEED core system. Integrating workflow management functionality into PROCEED extends the process support, so that it also covers partially automated processes.

The workflow paradigm is particularly suitable to guide individual engineers in their everyday tasks. This has been the main motivation for the integration of workflow support into PROCEED. The procedures which have to be followed by engineers when they work with the Comos system can be modeled as workflow definitions and can be automatically enacted in a development project. Intermediate steps in these procedures, like sending an email or uploading files to a remote server, can be performed automatically by the workflow engine. The progress of running workflow instances can be determined automatically, so that the engineer who performs the defined tasks does not have to provide additional information. Workflows can also be applied to support the coordination of several engineers or technical crews, if the respective processes can be predefined before project runtime. In particular, quality and change management processes can be modeled as workflows.

In this section, it is described how a workflow engine has been integrated into PROCEED. The workflow engine enacts workflow instances which are represented by subnets of the dynamic task net representing the whole development process. The enactment state of workflow instances can be monitored in the dynamic task net because workflow operations are mapped to state transitions of tasks and structural changes of the task net. Workflow instances are not only represented in a dynamic task net but the workflow engine automatically enacts and manages the part of the dynamic task net which represents the workflow instance, i.e. it changes the execution states of tasks and performs dynamic structural changes to the task net. Thereby, the workflow engine relieves a human resource from managing the subprocess. The workflow engine can be viewed as an automatic manager of the subprocess.

The realized integration builds on previous results which have been developed in the AHEAD project [HHM+06, Hel08a, HHWW10] and which have been described in Section 4.5. The integration approach has been presented in a workshop on process management for highly dynamic and pervasive scenarios [HBW09]. It is furthermore shortly described in [HHWW10]. The details of the approach are described in [Bri08].

6.3.1 Workflow Instances in Dynamic Task Nets

In the context of the research cooperation with Siemens Industry Software, it has been decided to use the Windows Workflow Foundation (WF) to realize the workflow management functionality in PROCEED. The WF has been introduced in Section 3.4.4. It is a framework for the development of workflow based applications but it is not a full-fledged WfMS. Most workflow management solutions come with their own specific modeling language for workflow definitions. Likewise, the WF provides its own modeling language for workflow definitions which has been introduced in Section 3.4.4. Workflows are composed of nested building blocks, the *workflow activities*. There are *atomic activities* and *complex activities*. While the former define automatically or manually performed tasks, the latter define the control flow of a workflow.

Figure 6.5 shows a schematic representation of a workflow definition. The example is taken from the domain of chemical plant design. The workflow defines the procedure which has to be followed to specify a pump of a chemical plant. In the first step, the operating parameters of the pump have to be defined including the flow rate and the medium data. For the second step, the determination of the pump type, there are two alternatives. The pump type can either be determined based on the use case at hand or based on the process data defined in the process flow

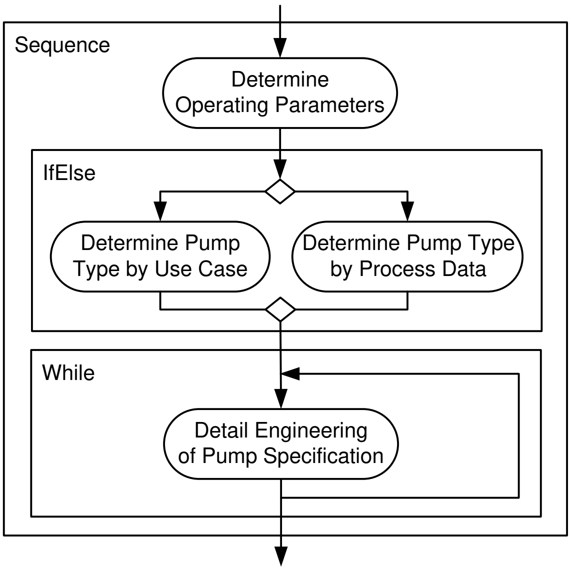


Abbildung 6.5: Example workflow definition for the design of a pump.

diagram. Finally, the detailed technical specification of the pump is elaborated. This step may have to be repeated several times.

A workflow instance is represented by a task in the dynamic task net which represents the development process. Such a task is called a *workflow-managed task* because structural changes to the realization of the task and certain state changes of its subtasks are automatically performed by the workflow engine once the workflow has been started. In fact, these changes can only be performed by the workflow engine and manual changes are prohibited. The realization of a workflow-managed task is called a *workflow-managed task net*. Finally, the subtasks of a workflow-managed task are called *workflow tasks* because they are related to activities in the workflow instance.

Workflow tasks themselves can again be workflow-managed or not. In the first case, a workflow instance is started upon the start of a workflow task. In the second case, the enactment and management of the workflow task and its subtasks is performed manually by the responsible resource. Consequently, an arbitrary nesting of workflow-managed and manually managed tasks can be realized in a dynamic task net.

A workflow-managed task is always derived from a *workflow template*. A workflow template is a process template for which a workflow definition has been defined. The workflow definition is associated with the realization of the root task of the workflow template. The realization of the root task contains a subtask for every workflow activity in the workflow definition, i.e. the workflow template defines a dynamic task net which matches the workflow definition associated with the root task of the template. Figure 6.6 shows the dynamic task net for the workflow definition of Figure 6.5. Only the control flow perspective is depicted in Figure 6.6. Document

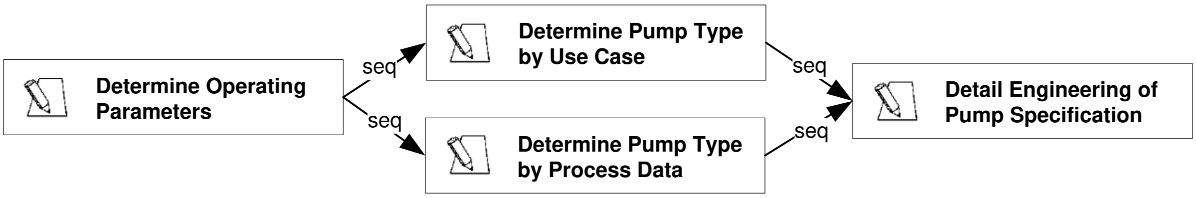


Abbildung 6.6: Tasks and control flows of example workflow template.

parameters, data flows, and task assignments are not shown but generally belong to a workflow template as well. The tasks are all in the execution state *InDefinition*. The iterated activity Detail Engineering of Pump Specification is represented by only one task in the dynamic task net.

A workflow template is an operational process model definition on instance level. Like ordinary process templates, a workflow template is copied into an existing dynamic task net. The template defines the initial state of the process. The associated workflow definition contains additional information about the correct enactment of the process. When the workflow template is finally used and a workflow-managed task net is enacted, the task net is automatically restructured and the execution states of subtasks are automatically changed depending on the conditions defined in the workflow definition which are evaluated by the workflow engine.

6.3.2 Mapping of Meta-Model Elements

The representation of WF workflow instances as dynamic task nets requires the mapping of two different meta-models. The language elements available for the creation of WF workflow definitions have to be mapped to the entities and relationships used to model dynamic task nets.

The WF allows to implement custom activity types (cf. Section 3.4.4). A special atomic activity type has been implemented for workflow tasks. Activities of this type are represented by tasks in the workflow-managed dynamic task net. Atomic activities of other types which are executed automatically are not represented as tasks in the task net. Figure 6.7 shows an abstract example of a workflow definition and the corresponding dynamic task net which together amount for a workflow template. Every activity labeled with a single letter is an instance of the activity type for workflow tasks. The activity labeled automatic does not have a counterpart in the dynamic task net.

Complex activities in the workflow definition determine the control flow relationships between the workflow tasks in the dynamic task net. The workflow tasks are connected by sequential control flows according to the relation of the corresponding activities in the workflow definition. Only sequential control flows with zero lag times can occur in a workflow-managed task net. Simultaneous and standard control flows and lag times cannot be modeled in WF workflow definitions and are therefore not allowed in a workflow-managed task net. They can also not be created manually

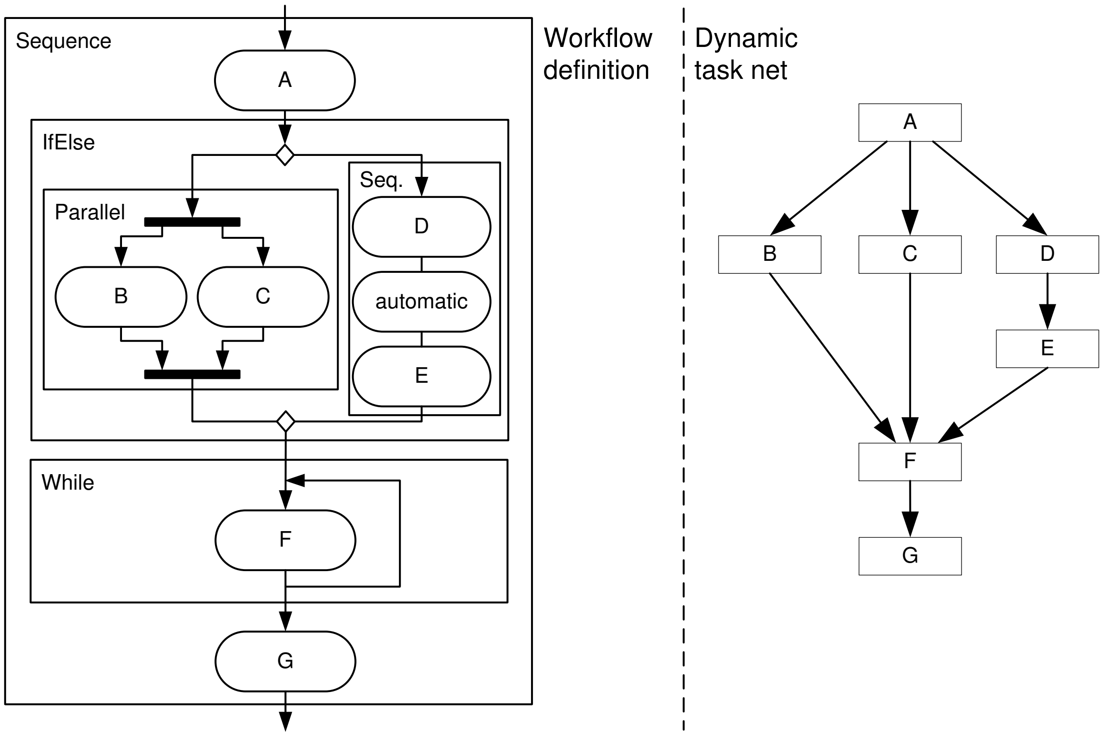


Abbildung 6.7: Mapping workflow block structure to task net control flow structure.

because manual structural changes are not allowed for workflow-managed task nets.

For the connection of the workflow tasks by control flows, it is required to determine the first and last atomic activities in a complex activity which are represented by workflow tasks. These activities are determined by recursively descending into the hierarchical structure of complex activities and collecting all atomic activities which have no predecessor or successor as well as all first and last activities from complex child activities. For example, in Figure 6.7 the first activities of the IfElse activity are the first activities of the Parallel activity together with the first activity of the inner Sequence activity, i.e. activities B, C, and D.

In a `SequenceActivity`, several atomic or complex activities are arranged in sequence. The workflow tasks corresponding to the activities are connected by sequential control flows in the dynamic task net according to the order defined by the `SequenceActivity`. In case of complex activities, the first and last atomic activities are determined and their corresponding workflow tasks are connected with the tasks representing the preceding and succeeding activities, respectively. In Figure 6.7, activity D is one of the first activities in the IfElse activity. It is connected to activity A via a control flow because activity A precedes the IfElse activity. Workflow tasks representing the first and last activities of the different branches of a `Parallel` activity are connected with the workflow tasks corresponding to the predecessors and successors of the whole `Parallel` activity, respectively. Likewise, all workflow tasks associated with the first and last activities of different alternative branches of an `IfElse` activity are connected by sequential control flows with the predecessors

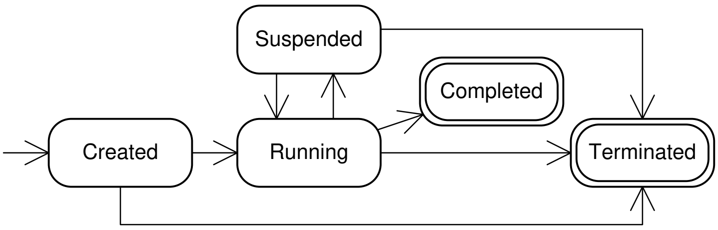


Abbildung 6.8: Finite state machine defining the life cycle of WF workflow instances.

and successors, which results in a parallel construct. The decision on which tasks are actually executed and which tasks are skipped is made at workflow runtime and is not reflected in the control flow structure. Finally, for every activity directly or indirectly contained in a `While` loop, exactly one workflow task is inserted in the dynamic task net, although the loop may be skipped completely or iterated several times. The actual number of iterations at runtime is reflected in the dynamic task net by automatic structural changes and execution state changes performed by the workflow engine. In all these cases, atomic activities which are not represented by workflow tasks are neglected for the control flow structure, i.e. the workflow tasks corresponding to their predecessors are directly connected with the workflow tasks corresponding to the successors.

6.3.3 Mapping of Execution States and State Transitions

The integration of WF workflow instances into dynamic task nets is eased by the circumstance that tasks in dynamic task nets have execution states, in contrast to tasks in project plans. However, the execution states and allowed state transitions defined for WF workflow instances and activities do not exactly match the finite state machine defined for tasks in a dynamic task net which has been described in Section 5.2.1.

The finite state machine for WF workflow instances is depicted in Figure 6.8. The initial execution state of a workflow instance is the state Created. A workflow instance has to be explicitly started to take on the execution state *Running*. A running workflow instance can be temporarily suspended which is indicated by the execution state Suspended. If the execution of a workflow fails due to technical problems or premature cancellation, the execution state of the workflow is changed to the final state Terminated. Only a running workflow can be successfully completed whereby its execution state is changed to Completed.

The synchronization of the enactment of workflow instances with their respective workflow-managed tasks has been formally specified by the product automaton depicted in Figure 6.9. Every state of the combined automaton is labeled with the execution state of the workflow-managed task on top and the execution state of the workflow instance below. A workflow instance is created when the execution state of the workflow-managed task is changed from InDefinition to Waiting for the first

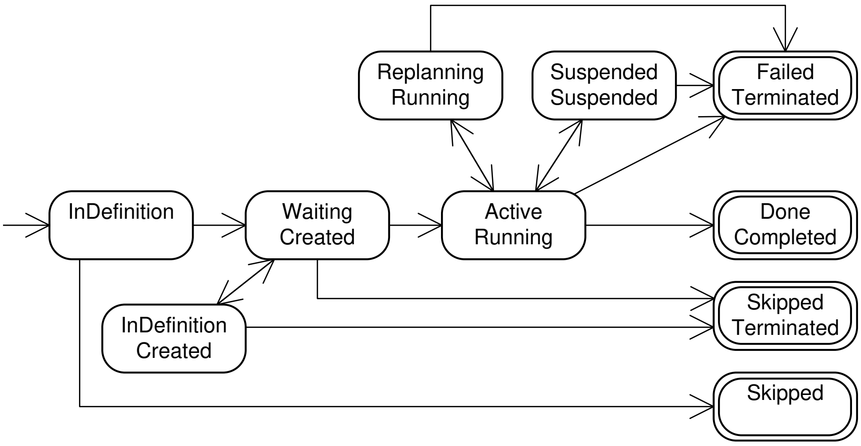


Abbildung 6.9: Synchronizing automaton for workflow-managed tasks.

time. If the task shall be redefined afterwards, the existing workflow instance has to be modified. The workflow instance is started when the execution state of the workflow-managed task is changed to Active. Suspension of the workflow-managed task results in the suspension of the workflow instance. The workflow instance is not suspended to replan the workflow-managed task which involves dynamic changes to the workflow definition. The execution state Completed indicates a successful completion of a workflow and is therefore mapped to the execution state Done. The workflow instance is terminated when the workflow-managed task is aborted or skipped.

The finite state machine for WF workflow activities is depicted in Figure 6.10. For a WF workflow activity, the initial execution state is Initialized. After an activity has been started it is in the execution state Executing. If the activity is successfully completed, than its execution state is changed from Executing to Closed directly. There are two intermediate execution states in which exceptional events are handled before the activity is closed. The state Canceling indicates that the command to cancel the activity has been invoked and is currently processed. The state Faulting is reached if technical problems have occurred. Both intermediate states lead to an unsuccessful closure of the activity. The execution state is changed to Closed just like for successful termination. The final state does not reflect whether the activity has been terminated successfully or unsuccessfully. For this purpose, a separate return value is transmitted to the workflow engine. If an activity is closed due to an error, it may be necessary to undo previous results of the workflow. In this case compensating activities can be started. The failed activity is changed to the execution state Compensating while the compensation takes place. Afterwards, the execution state is changed back to Closed and can never be changed again. Therefore, Closed is actually the final state of the state automaton but only if the state Compensating has been reached before.

The synchronization of the execution of workflow activities with their correspon-

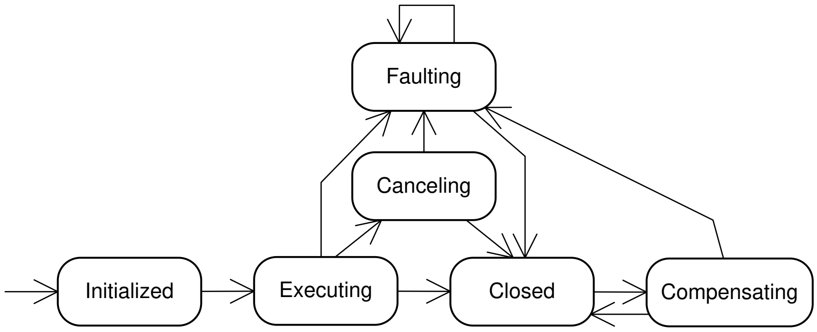


Abbildung 6.10: Finite state machine defining the life cycle of WF workflow activities.

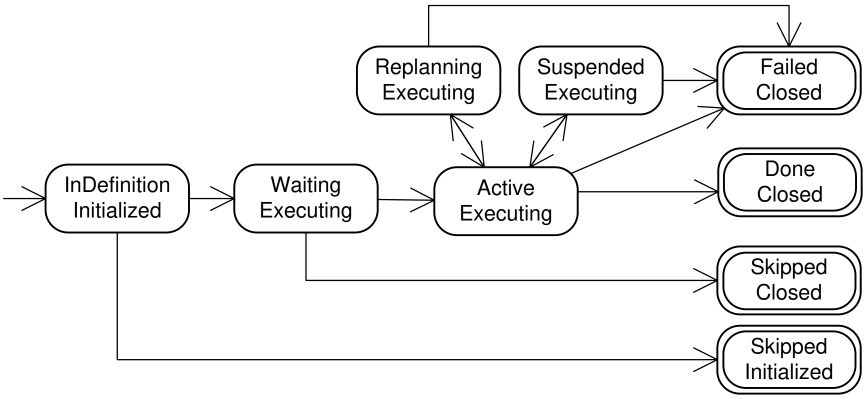


Abbildung 6.11: Synchronizing automaton for workflow tasks.

ding workflow tasks is defined by the product automaton depicted in Figure 6.11. Every state of the combined automaton is labeled with the execution state of the workflow task on top and the execution state of the workflow activity below. The intermediate execution states Canceling and Faulting have been left out for reasons of clarity. Likewise, the execution state Compensating has been left out in Figure 6.11 because PROCEED does not make use of the compensation functionality of WF workflows. The full product automaton which takes all activity execution states into account has 21 states and 38 transitions.

In Figure 6.11, it can be seen that the start of a workflow activity is mapped to the transition from InDefinition to Waiting of the corresponding workflow task. The alternative solution to map the start of an activity to the start of the corresponding task has not been implemented for the following reasons. The workflow engine manages the workflow-managed task net, i.e. it creates new task version when necessary and prepares tasks for execution. The enactment of the workflow tasks is left to the assigned resources. The responsible resource of a workflow task changes the execution state to *Active* when the assigned resources actually start working on the task. The workflow engine cannot decide when a task can actually be started. It can only prepare the task for execution by changing its execution state to *Waiting*. For a workflow task, this transition can only be performed by the workflow engine. As a consequence, the responsible resource of a workflow task can only start the task when it has been prepared for execution by the workflow engine. Tasks, which correspond to activities in the workflow definition which have not been reached by the workflow instance yet, cannot be started by the respective responsible resources. A workflow task cannot be redefined because this would require a manual state change from *InDefinition* to *Waiting* afterwards. Therefore, the state InDefinition/Executing and the corresponding state transitions are not contained in the synchronizing automaton.

The responsible resource of a workflow task starts and terminates the task. The successful termination of a workflow task is signaled to the workflow engine which closes the corresponding workflow activity and proceeds with the enactment of the workflow instance. If a workflow task is aborted by the responsible resource to indicate that the objective of the task could not be achieved, the corresponding workflow activity is nevertheless successfully closed and the workflow proceeds as planned. This behavior is in line with the semantics of task failure in dynamic task nets (cf. Section 5.2). A complex task does not fail only because one of its subtasks has failed. The complex parent task can be successfully terminated nevertheless. The objectives of the failed subtask can possibly still be achieved by performing additional tasks.

The UML sequence diagram depicted in Figure 6.12 gives an overview over the interaction between the four entities workflow-managed task, workflow instance, workflow activity, and workflow task. This figure merely illustrates the depending state changes. The interaction actually takes place between the PROCEED process engine, which manages dynamic task nets, and the workflow engine, which enacts workflow instances. When a workflow-managed task is defined for the first time, a new workflow instance is created. With this workflow instance, an activity instance is created for every activity defined in the workflow definition. When the workflow-managed task is started, the workflow instance is started as well. When a workflow activity is executed, the corresponding workflow task is defined, i.e. its execution state is set to *Waiting*. Afterwards the task can be started and eventually committed by the responsible resource, whereupon the workflow activity is closed. When the workflow instance is finally completed, the workflow-managed task is committed.

6.3.4 Execution of Control Flow Activities

The state machine mapping for execution states of workflow activities and workflow tasks does not cover complex activities, which define the control flow in a workflow instance. During the execution of `IfElse` and `While` activities, conditions are evaluated which determine how the workflow instance proceeds. These decisions lead to behavioral and structural changes of the workflow-managed task net.

All workflow tasks which correspond to activities inside an `IfElse` activity remain

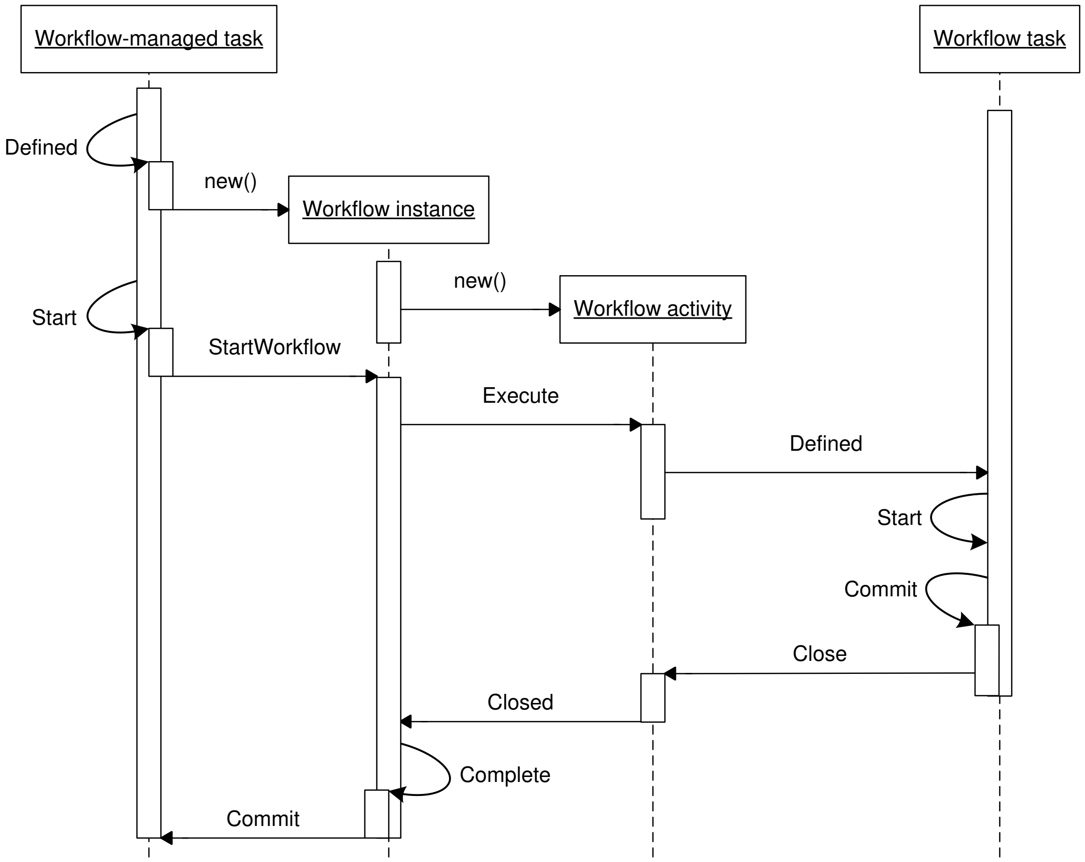


Abbildung 6.12: Sequence diagram for workflow integration.

in the state *InDefinition* until the decision for one of the alternative branches has been made. When an `IfElse` activity is reached by the workflow instance and the decision is made, the workflow tasks corresponding to the first activities in the selected branch are set to the execution state *Waiting* while all workflow tasks corresponding to activities in the neglected branches are skipped. The former can be started by the respective responsible resources while the latter cannot be started anymore.

A workflow-managed task net initially contains a workflow task for every activity contained in a `While` loop. If a `While` activity is not executed at all in a workflow instance due to the evaluation of the condition for entering the loop, then all corresponding workflow tasks are skipped in the dynamic task net. If the workflow instance enters the while loop, the workflow tasks corresponding to the first activities in the loop are prepared for execution by changing their execution states to *Waiting*. If a `While` loop is iterated once more, then a new task version is created for every workflow task which corresponds to an activity contained in the `While` loop. The previous versions of these tasks are necessarily all terminated, i.e. in one of the execution states *Done*, *Failed*, or *Skipped*.

Figure 6.13 shows the a workflow-managed task net which has been created as

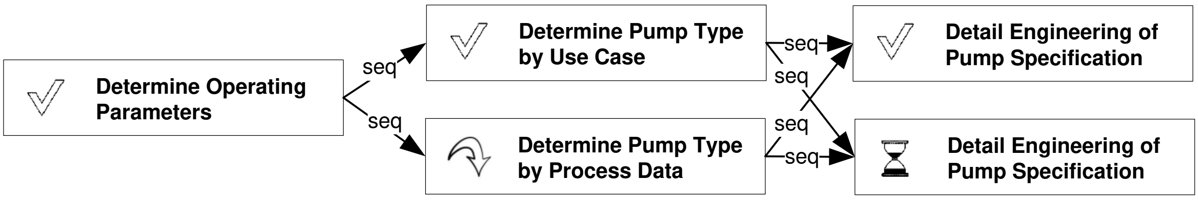


Abbildung 6.13: Enacted workflow-managed task net.

a copy of the workflow template of Figure 6.6. The workflow instance has already proceeded to the second iteration of the `While` loop (cf. Figure 6.5). In the `IfElse` activity, the alternative branch with the activity Determine Pump Type by Use Case has been selected whereupon the workflow task Determine Pump Type by Process Data has been skipped. The first iteration of the `While` loop has already been completed and the first version of the workflow task Detail Engineering of Pump Specification has been committed. A new version of the iterated workflow task has been automatically created and prepared for execution by the workflow engine.

At workflow runtime, decisions are made regarding which alternative branch to chose or whether a loop shall be iterated once more. These decisions are made based on conditions specified in the workflow definition. In PROCEED, the conditions evaluate *decision variables* which are defined for the workflow-managed task or the workflow tasks. Custom decision variables can be defined for task types (cf. Section 6.1). The conditions which are specified in a workflow definition refer to the decision variables of the tasks in the dynamic task net. If several versions of a task exist, the values of the last version are used. The values of the decision variables can be changed by the responsible resources of the respective tasks. In this way, the enactment of a workflow-managed task net can be influenced by authorized users. In the example of Figure 6.13, the workflow-managed task which represents the whole workflow instance has a decision variable for the selection of one of the alternative ways to determine the pump type. The responsible resource of the workflow-managed task can decide which alternative is chosen at runtime. The workflow task Detail Engineering of Pump Specification has a decision variable which is set to `true` if another iteration is required. The responsible resource of the workflow task may decide whether another iteration is performed by setting the value of the decision variable. The value of the variable is initially set to `true`.

6.3.5 Data Flow in Workflow-Managed Task Nets

In general, a workflow definition covers different views on a workflow (cf. Section 3.4). In PROCEED, only the control flow view of WF workflows is used for workflow-managed task nets. The data flow between workflow tasks is defined in the dynamic task net of a workflow template. Input and output parameters can be defined for workflow tasks and can be connected by data flows. According to the structural constraint (5.10), a data flow can only be defined between two parameters when the respective tasks are connected by an equally directed control flow or feedback flow. Feedback flows can be manually created in workflow-managed task nets only when the target task is active. Automatic structural change operations performed by the workflow engine at workflow runtime are restricted to the creation of new task versions. The creation of new task versions takes the data flow into account, i.e. the new task versions have the same parameters as the old versions, and the parameters are connected by data flows accordingly (cf. Section 5.2.3). At workflow runtime, document revisions have to be produced and released manually by the assigned resources of the workflow tasks. For the same reason why the workflow engine cannot start and commit workflow tasks automatically, it cannot automatically produce document revisions.

6.3.6 Dynamic Changes to Workflow-Managed Tasks

The enactment of workflow-managed task nets involves automatically performed structural changes in case of the iteration of loop structures. However, these structural changes are predefined in the associated workflow definitions. In some situations it may be required to apply structural changes to a workflow-managed dynamic task net which have not been predefined in the workflow definition, e.g. it may be required to add an additional task. In this case, it is necessary to change the workflow definition of the running workflow instance as well. Therefore, it is not possible to directly change a workflow-managed task net in PROCEED, but manual structural changes to a workflow-managed task net can only be performed by changing the workflow instance which in turn entails the automatic adaptation of the associated task net.

Every WF workflow instance contains a copy of the workflow definition from which it has been instantiated. The windows workflow foundation allows to dynamically change the definition of a running workflow instance, so that the workflow is enacted according to the changed definition afterwards. The changes affect only the one workflow instance but not other instances which have been instantiated from the same workflow definition. In Section 3.4.4, the constraints have been listed which apply for dynamic changes to running WF workflow instances. An activity can only be deleted if it has not been started yet. A new activity can only be inserted into a complex activity which has not been terminated yet. It is possible to move an activity from one complex activity to another as long as the two constraints are satisfied.

Dynamic changes to a workflow instance are reflected in the corresponding workflow-managed dynamic task net. The deletion of an activity results in the deletion of the associated workflow task. The creation of a new activity involves the creation of a new workflow task. In both cases, the control flows in the workflow-managed task net are adapted, so that they reflect the new structure of the workflow definition. Thereby, the same transformation rules apply as for a workflow template, which have been described earlier and have been illustrated in Figure 6.7. If an activity is moved from one complex activity to another, then the existing workflow task is retained but its control flow dependencies are adapted to reflect the new position of the activity in the workflow definition.

The changes which are automatically applied to a workflow-managed task net upon the change of the associated workflow instance have to fulfill all structural and behavioral constraints for dynamic task nets (cf. Sections 5.1 and 5.2). Otherwise, the changes are prohibited. This means among other things, that the workflow-managed task has to be in the execution state *Replanning*. Possible violations of timing consistency constraints are handled according to the change management procedure which will be introduced in Chapter 9.

When a workflow-managed task has been terminated, it may be necessary to create a new version of the task, e.g. if a feedback flow is defined which targets the terminated workflow-managed task. The completed workflow instance associated with the previous version of the workflow-managed task cannot be reused. Therefore, a new workflow instance is created based on the—possibly modified—workflow definition of the previous version of the workflow-managed task. The new task version is again workflow-managed and is enacted accordingly.

6.3.7 Time Management Data in Workflow Templates

The primary functions of process management are to use previously gained process knowledge in development projects and to improve this knowledge from project to project. Time management data represents process knowledge. For task types, the planning data can be specified and reused for all instances of the respective type. In process templates, the lag times of control flows can be specified. Workflow templates enable another form of collecting and reusing time management data for workflow-managed tasks.

Reference values are determined for the durations of all activities in a workflow definition—atomic activities as well as complex activities. The reference value for the duration of an activity is calculated as the mean duration of all occurrences of the activity in the terminated instances of the workflow template. The reference values for the activity durations are stored at the workflow template. This information is used for scheduling and progress measurement of workflow-managed tasks derived from the template. In Sections 7.4 and 8.1, it will be described in detail which data is collected for a workflow template and how it is used for scheduling and monitoring, respectively.

The quality of the reference values for activity durations depends on the number of workflow instances which are taken into account and the standard deviation of the measured durations. Only those workflow instances are considered, which have been executed in a comparable project and under the same circumstances.

Dynamically changed workflow instances are not taken into account for the computation of reference data for the original workflow template. They are handled as new variants of the workflow definition and separate reference values are calculated for these variants if sufficiently many instances with the same dynamic changes exist. If several versions of a workflow-managed task have been created in a project, only the activity durations of the first version are taken into account for the original workflow template.

The tracking service of the workflow engine determines the total durations of workflow activities including work time and idle time. Only the effective work time is used to compute the reference values. For this reason, running workflow instances have to be suspended and resumed by the responsible resources before and after working on the tasks, respectively. The suspension time is subtracted from the total activity durations. Consequently, the activity durations reflect, when the assigned resources were actually working on the respective workflow tasks.

For activities directly or indirectly contained in a `While` activity, the mean duration is determined for every iteration of the surrounding loop construct separately. Furthermore, for every iteration, the number of workflow instances which actually completed the respective iteration is determined. For an `IfElse` activity, the average durations of the alternative branches are stored as well as the average duration of the `IfElse` activity itself. Furthermore, for every alternative branch the number of workflow instances which actually chose the branch is stored.

6.3.8 Conclusion

The integration of a workflow engine into PROCEED allows for the modeling and enactment of partially automated processes. Automatically executing activities can be defined in a workflow definition, which are not represented in the project plan. Alternative courses of action can be defined, and the workflow engine decides automatically at runtime which alternative is executed. The iteration of process parts can be predefined in a workflow definition, so that the required structural changes to the task net can be automatically performed by the workflow engine. Thereby, the responsible resource of a workflow-managed task is released of managing the subprocess. The enactment of multiple instances of a workflow definition allows the automatic calculation of reference values for the activity durations, which can be used for workflow scheduling and monitoring.

In PROCEED, workflows are used alternatively to declarative process model definitions on type level which were introduced in [Sch02]. However, both approaches could be used together and actually complement each other. Declarative process model definitions are more suitable to model whole development processes or subprocesses on higher levels of a hierarchically structured dynamic task net. Workflows are more suitable to model and enact subprocesses on lower levels of a dynamic task net. However, in the industrial context of the research project, only the workflow approach was required and has been implemented. The declarative approach for process modeling has not been applied for the reasons discussed in Section 6.2.

The implemented integration of workflow instances into dynamic task nets has one limitation. It is not possible to define and enact workflows which span across several subprocesses, i.e. the workflow tasks which correspond to the activities of a workflow definition may not be contained in the realizations of different parent tasks. To realize this, it would be required to detach workflow definitions and instances from the realizations of individual tasks in a dynamic task net. This problem has not been addressed in this thesis because it would require a fundamentally different integration approach.

6.4 Related Work

In this section, related work is reviewed regarding the workflow management functionality in PROCEED.

6.4.1 Integration of Project and Workflow Management

The approach for integrating workflow management functionality into PROCEED is closely related to the problem of integrating a WfMS with a project management system (PMS). In both cases, different meta-models for modeling workflows on the one hand and project plans or task nets on the other hand have to be mapped. A runtime mechanism has to be provided which translates the events and actions in one system to according operations in the other system to maintain the consistency of the different models.

Chan and Chung In [CC02], the IPPM system is described which integrates project and workflow management functionality. Custom workflow modeling and enactment tools have been integrated with MS Project. Control flow structures in workflows are mapped to task net structures in a project plan. For alternative branching constructs, the prudential branching method is applied, i.e. tasks corresponding to unselected branches are removed from the project plan as soon as the decision for one of the branches is made by the workflow management system. This method differs from the method implemented in PROCEED where neglected branches are not removed from the workflow-managed task net but are marked as skipped. Retaining the skipped tasks in the task net ensures the traceability of workflow execution in the task net. With respect to loop constructs, different possibilities are distinguished: loops with only compulsory tasks and loops with compulsory and optional tasks. The case that a loop can be skipped completely is neglected. Loops are unfolded in project plans at workflow runtime, i.e. the subsequent iterations of tasks are represented as individual tasks in the project plan and are sequentially connected by control flows. In contrast, new versions of iterated tasks are created in PROCEED. In the IPPM system, only the compulsory tasks of a loop are inserted in the project plan before workflow runtime. The optional tasks are replaced by a so-called prudent-estimated task which represents all further iterations of the loop. Just like in PROCEED, workflow definitions in the IPPM system have to be well-structured—or in other words block-structured—for the described mappings to be applicable. All aspects regarding the scheduling of workflow instances which are covered by the IPPM system will be reviewed in Section 7.6.1.

Bussler In [Bus98], Bussler discusses issues regarding the integration of WfMS and PMS in general. He distinguishes two parts: schema integration and behavior integration. Schema integration is also called semantic integration and refers to the mapping of the conceptual objects of the systems, e.g. workflow activities and tasks. Behavior integration means that the state changes taking place in one system have to be mapped to operations in the other system. Regarding schema integration, the main conflicts are identified, e.g. the absence of alternative branching and loops in project plans, and the restriction to sequential control flows in workflows. Different possible mappings for control structures are discussed: static mapping and continuous mapping. Static mapping means that all possible routes through a workflow are completely mapped to the project plan before workflow runtime, which does not work for loops. At workflow runtime, tasks of neglected alternative branches are removed from the project plan. Continuous mapping means that only the portion of a workflow is mapped to the project plan which will actually be executed. For the mapping of control structures in PROCEED we used the continuous mapping approach for loops and the static mapping approach for alternative branching with the exception that no tasks are deleted after a decision has been made. Regarding behavior integration, the interaction of the integrated systems is specified by means of sequence diagrams which cover the enactment of workflow instances as well as dynamic changes to workflow instances. Bussler describes an ideal integration of a WfMS and a PMS independent of any specific system. Similarly, the interaction of the workflow and task net entities in PROCEED has been specified in Figure 6.12. However, the challenging problems arise from the peculiarities of the systems to be integrated.

Bauer Bauer also addresses in [Bau04] the issue of integrating existing workflow and project management systems. He describes the respective strengths and weaknesses of PMS and WfMS and motivates their integration amongst others by the need to schedule workflows in a project. He distinguishes two approaches: loose coupling, where several workflow instances can be mapped to a single project task, and close coupling, where there is a one-to-one mapping of workflow activities and tasks in the project plan. The close coupling approach is not applicable, when the project plan and the workflows are on different abstraction levels. Hence Bauer presents a generic integration architecture for loose coupling based on an integration layer between the WfMS and the PMS. The propagation and aggregation of runtime data via the integration layer is specified by means of event-condition-action (ECA) rules. In PROCEED, a close coupling of the process engine and the workflow engine has been realized. There is a one-to-one mapping between workflow activities and tasks in the task net.

Maurer et al. The MILOS system [MDB$^+$00] is an integrated solution for the management of software development processes. It combines project and workflow management functionality by coupling a custom process engine with MS Project. The workflow management part supports ad-hoc workflows, which can be elaborated during enactment. MS Project is used as the planning interface and is extended by functionality to define information flow between different tasks. However, the actual performance of tasks in terms of execution states is not reflected in the project plan. The runtime coupling between the workflow management component and MS Project is realized by means of ECA-rules. The problem of different abstraction levels of the project plan and the workflows is not addressed, i.e. a close coupling is realized with respect to the terminology introduced in [Bau04]. In the MILOS system, there is a strict distinction between the user interfaces for the project planner and the process performers. The former uses MS Project while the latter work only with to-do lists. The process context of a task is not visible to the assigned resource. The scheduling and project monitoring functionality of the MILOS system is limited by the functionality provided by MS Project.

Bahrami In [Bah05], an architecture for an integrated project and workflow management system is proposed. Technical issues are addressed like the exchange format for workflow definitions which is XPDL in this case. The conceptual problems of structural and behavioral mapping are not addressed in the paper. The paper claims that workflow instances are scheduled by the project management system but no details are revealed to substantiate this statement. In general, the paper does not verify that the aimed-at functionality of the integrated system has been implemented and is demonstrably working. It should rather be considered as a statement of intent.

The reviewed integration approaches all face the problem that no information about the process enactment state is maintained in the respective project management system. The plan is adapted according to the enactment of workflow instances, but the ability to monitor the status of process instances in the PMS is limited. Furthermore, since products and data flows are generally not represented in project plans, product management is still beyond the scope of the integrated systems, except for the MILOS system. In most cases, data flows between workflow instances have to be managed manually. These deficiencies are overcome by integrating workflow instances into dynamic task nets.

Heller The solution which is most related to the workflow integration approach presented in this thesis is the integration of the AHEAD system with the workflow management system Shark as described in [Hel08a, HHM$^+$06, Wei06] and Section 4.5. The goal of the integration of Shark and AHEAD was to monitor workflow processes in AHEAD which were enacted in a WfMS. In contrast, a workflow engine has been integrated into PROCEED to automatically manage parts of a dynamic task net. The motivation for this integration has been the availability of additional modeling capabilities for alternative branching and loops. The different motivations lead to differences in detail between the solutions. In the integrated solution of AHEAD and Shark, it was not possible to refine workflow tasks by manually managed task nets. The control over the enactment of a workflow fragment remained at the WfMS. A workflow task could only be refined by another workflow instance. In contrast, workflow-managed tasks and manually managed tasks can be arbitrarily nested in PROCEED. The resources assigned to tasks in a workflow fragment used the client application of the Shark WfMS to execute their tasks. The workflow fragment in AHEAD which represented the workflow instance could only be monitored. In contrast to that, the assigned resources use PROCEED to start and commit their workflow tasks. No separate client application is required. The start of a workflow activity in the Shark system was mapped to the start of the corresponding task in the workflow fragment in AHEAD. This mapping was correct for the integration of AHEAD and Shark because the workflow fragment should reflect the status of the workflow instance. When an assigned resource had started his activity in the WfMS, this had to be reflected in AHEAD. In PROCEED however, the start of a workflow activity is mapped to the preparation of the corresponding workflow task for execution because the assigned resources start the tasks manually via PROCEED. Further differences between the two approaches stem from the different modeling capabilities and available functionalities of the integrated systems. The set of available execution states of a task has been extended by the state *Skipped* in PROCEED. This has enabled a more seamless integration of the execution states of workflow tasks and workflow activities. In the AHEAD solution, a neglected task of an alternative branch remained in the execution state *InDefinition* until the workflow was terminated. Finally, dynamic changes to running workflow instances were not possible for the Shark WfMS. Therefore, the integrated solution did not cover the adaptation of a workflow fragment in case of dynamic changes to the corresponding workflow instance. In PROCEED, the workflow-managed task net is automatically adapted to the changed workflow definition.

6.4.2 Direct Process Support in Engineering Design Projects

In the Collaborative Research Center (CRC) 476 IMPROVE [NM08], another research project was conducted which was concerned with process management. In the project *Experience-based Development Processes* [MJW08], new concepts and tools were developed for direct process support in engineering design projects. The PRIME framework [PWD$^+$99] was applied to realize *process-integrated software tools* which are used by the engineers in plant design projects to create design artifacts like flow diagrams. Individual and cooperative processes are supported by dynamically adapting the available tool functionality and by providing advices and hints to the user to guide him.

In contrast to process management on the medium- and coarse-grained level as provided by the AHEAD and the PROCEED system, direct process support is provided for fine-grained technical processes in a design project. The process support is built into the technical software tools which are used by engineers and other process participants instead of providing separated guidance tools. From the user perspective, process support is provided implicitly by adapted tool functionality and not by explicitly defined tasks and work packages presented on a to-do list. The general process support provided by PROCEED falls into the latter category. The PRIME approach is nevertheless related to the workflow support provided by PROCEED in that similar processes are targeted. Workflow management in PROCEED is best suitable for fine-grained processes in a design project.

In [MJW08], the authors argue that the early phases of plant design processes are highly creative and dynamic and cannot be completely predefined in advance. In particular on the fine-grained level, a design process cannot be predetermined before runtime. However, individual process chunks may be identified which appear often in the same way in an engineering project and even in different projects. These parts of the overall process represent best practices followed by the majority of the engineers. These process chunks are generally well-understood and can be formally defined. In the PRIME approach, so-called *method fragments* are created for this purpose.

The approach for direct process support builds on three basic ideas. First, method guidance is provided by the process-integrated software tools. Thereby, no strict conformity with process prescriptions is enforced. Instead, advice is given to the user. When a valid situation with eligible method fragments for enactment eventuates, then the user gets notified about the available options and can selectively request their enactment. Second, the process support is provided by means of process-integrated tools instead of separated guidance tools. These tools detect situations which correspond to method fragments and provide the advice. Furthermore, the user interface is automatically adapted, i.e. the available options are presented to the user and certain tool functionalities which are not required in the situation are hidden from the user to focus his attention on the relevant functions. Finally, process and product traces are reused to generate method fragments. While engineers are working with their process-integrated tools, their actions are logged and stored in a database. This feedback information is organized according to a traceability meta-model which is adjusted to the project-specific needs. The captured process traces provide evidence for the dependencies between the design products (e.g. flow sheets), the supplementary products (goals, decisions), and the process observation data (process steps). From the process traces, the method fragments are derived. Compared to prescriptive process definitions which may be biased by perceptions of the process modelers, process traces are objective in the sense that they reflect the actually performed processes.

The realization of process-integrated tools is supported by the integration framework PRIME [PWD$^+$99, Poh99, PWD$^+$98]. The framework includes the situation-based process meta-model NATURE [Poh96] which defines the required entities and relationships for the explicit definition of method fragments. By means of the NATURE meta-model, situations and intentions can be explicitly represented. An intention reflects the goal that the human actor has in mind. The process knowledge about how to reach an intention in a given situation is defined by a so-called context. The method definitions based on NATURE are integrated with tool models in the so-called environment model. Furthermore, PRIME enables the capturing of traces during the usage of process-integrated tools for product design. Finally, PRIME comprises an object-oriented implementation framework for the interpretation of environment model definitions by the process-integrated tools and the dynamic adaptation of their behavior. Using the PRIME framework, a process-integrated flow sheet editor has been realized which is based on the drawing tool Visio. This prototype has been used to evaluate the approach for direct process support in the CRC 476 IMPROVE.

The direct process support provided by the flow sheet editor and the workflow support integrated into PROCEED can be applied to similar processes in plant design projects. The procedures which have to be followed by individual engineers can be supported by both solutions. In both approaches, certain process steps can be automatically executed while others require human intervention. Alternative courses of action can be defined in the process models and decisions can be made by the users at process runtime.

However, there are significant differences between the two approaches. The workflow support in PROCEED still relies on the concept of explicitly defined tasks which are assigned to human resources. In this regard, the workflow support for fine-grained processes does not differ from the process support on the medium- and coarse-grained level. In contrast, the process support provided by process-integrated tools based on PRIME is implicit. The user is not aware that a formally specified process model controls the behavior of his tool. However, direct process support only is not sufficient for the management of complex development processes. Furthermore, in the PRIME approach process traces are captured to derive the process model from these traces, i.e. the structure of the process regarding the available actions, situations, and intentions, as well as their mutual dependencies are determined. In contrast, the structure of a workflow is prescribed in PROCEED, i.e. it is manually specified by a domain expert. At workflow runtime, timing data and structural dynamic changes are captured and used afterwards to improve the corresponding workflow template. The process-integrated flow sheet editor which has been realized using PRIME is based on the general purpose drawing tool Visio. PROCEED is an extension to the life cycle asset information system Comos which is widely used in the plant engineering industries, in particular for the creation of flow diagrams.

Altogether, the PRIME approach and the workflow support in PROCEED offer complementary support for fine-grained processes of individual engineers and for the coordination of technical crews. Direct process support would constitute a valuable extension to the PROCEED system and would complement the process support functionalities integrated in Comos.

6.5 Conclusion

This chapter introduced the modeling concepts for process model definitions and the functionality for their enactment in PROCEED. Task types are used to store and reuse knowledge about the time, effort, and costs involved with performing a task instance of the respective type. This is already very valuable information for project planning. Process templates additionally comprise information about the relationships and dependencies between tasks including required lag times. Finally, workflow templates comprise additional information about the enactment of the process. This allows for the automatic enactment of subprocesses of a development process and thereby releases the responsible resources from managing these processes. Workflow-managed task nets have to be scheduled in a particular way as it will be described in Chapter 7, and the knowledge contained in workflow templates enables specific progress measurement as it will be described in Chapter 8.

Kapitel 7

Scheduling of Dynamic Task Nets

The process of project planning as described in Section 3.1.2 and depicted in Figure 3.3 comprises several steps. The steps Activity Definition, Activity Sequencing and the estimations of workload, budget, durations and resource requirements can be performed manually. The task net which is defined in the course of these planning steps can be built up from scratch or by using pre-defined task types and process templates. Afterwards, the defined tasks are scheduled. The development of a good, time- and resource-feasible schedule needs adequate tool support. A project manager cannot manually generate a feasible schedule, which respects all defined control flow and resource dependencies. In particular in mid-size to large projects with over 50 resources and hundreds of tasks, it is impossible to take all dependencies into account. Furthermore, secondary objectives like the minimization of the project makespan and the leveling of resource usage cannot be achieved manually with reasonable effort. Therefore, algorithms have been implemented in PROCEED for automatic schedule generation.

This chapter describes, how the tasks in a dynamic task net are scheduled in PROCEED to obtain a timed process model instance. The planning data, defined task assignments with required roles, and the manually set time constraints are used to calculate planned start and end times for tasks, to assign resources to task assignments, and to distribute the planned workload over several work days [Dre09].

A dynamic task net is an activity-on-node representation of a project network.In PROCEED, release and due dates can be defined for tasks. Resource availabilities may vary for different time units as defined in the resources’ work calendars. The available semantics of control flows in dynamic task nets cover most of the generalized precedence relations with minimal lag times, except for start-start and start-end relationships. Therefore, the problem of scheduling dynamic task nets can be considered as a restricted version of the GRCPSP (cf. Section 3.2.2).

In contrast to the GRCPSP, it is not the objective of scheduling in PROCEED to produce an optimal schedule with respect to the project makespan for the following reasons. The problem of finding the optimal solution to the GRCPSP is NP-hard in the strong sense [DH02]. Hence, exact methods for solving the GRCPSP have an exponential runtime complexity in the worst case. Heuristic methods may not find the global optimum among the feasible schedules but solutions can be computed more efficiently. In practice, it is more important for project managers to obtain good schedules fast than to wait for optimal schedules for a long time [DH02, p.264].

The generation of an optimal schedule does only make sense, when all tasks in the project are known, i.e. when the work breakdown structure is completely defined. However, this is usually not the case at the start of a development project. Many tasks are defined at runtime and have to be incorporated into the schedule. As a consequence, frequent rescheduling is required to react to these disruptions. According to [ZBY05], computing an optimal schedule at the beginning of a project makes it harder to repair the schedule in case of disruptions at project runtime, i.e. suboptimal schedules are more flexible. Schedule recovery is more likely to be unfeasible for an optimal schedule than for a good schedule which incorporates additional slack time for the scheduled tasks. The duration of a plant design project is usually estimated based on reliable reference data from similar previous projects, and it is fixed in the contract before a detailed scheduling of all tasks in the project is performed. The goal of task net scheduling therefore is to find a good feasible schedule which takes all time constraints and in particular the defined project deadline into account. The computed project end time will usually be earlier than the defined project deadline, because the estimate for the project duration contains a contingency buffer to cover unforeseen delays at project runtime. Instead of finding the optimal schedule for the tasks in a project which leaves, e.g., 30% of the project duration as buffer time, it is reasonable to compute a suboptimal schedule which leaves, e.g., 20% of the project duration as buffer.

Since the GRCPSP is NP-hard and the goal of resource-constrained scheduling in PROCEED is the fast computation of a good but not necessarily optimal schedule, a heuristic approach has been implemented in PROCEED. Since the scheduling algorithm is used for initial project planning, a constructive heuristic is required which generates a good feasible schedule from scratch. The constructive heuristic is also used to repair the schedule in case of dynamic changes at project runtime. Therefore, the heuristic allows partial rescheduling of dynamic task nets.

The implemented heuristic for resource-constrained scheduling in PROCEED is based on the parallel scheduling scheme and uses CPM-based priority rules (cf. Section 3.2.2). The fact that the optimal schedule may not be in the set of non-delay schedules which are produced by a parallel scheduling scheme can be neglected for the above mentioned reasons. The usage of CPM-based priority rules requires a CPM-scheduling pass before the actual resource-constrained scheduling. Therefore, the whole scheduling algorithm is divided into two phases.

1. Critical path analysis

2. Heuristic resource-constrained scheduling

Critical path analysis has been extended in this thesis to cover hierarchical task nets in which simultaneous and standard control flows can be defined. The earliest and latest possible start and end times which result from the critical path analysis are used to compute the priority list of tasks for the second phase. They are furthermore used during resource-constrained scheduling as additional constraints.

The scheduling of a dynamic task net requires certain input parameters. The user specifies the *start date* for scheduling which defines the first date for which preparing tasks can be scheduled. This is by default the next work day after the current date. Scheduling can be performed before the start of the project but also at project runtime. When a task net is rescheduled at runtime, terminated tasks are not rescheduled, running tasks keep their planned start time, and preparing tasks are scheduled later or equal to the specified start date. Rescheduling at project runtime can be performed for a subnet of the whole dynamic task net. For this purpose, the user specifies the *root task* of this subnet. Only the descendants of this root task in the task net hierarchy are rescheduled while all other tasks remain unchanged.

The heuristic approach has been chosen because the GRCPSP is NP-hard. The implemented parallel heuristic may not yield an optimal schedule with respect to the project's makespan which is acceptable for the above mentioned reasons. However, even the problem of deciding whether a feasible schedule for the GRCPSP exists is NP-complete [DH02]. If the algorithm for automatic scheduling fails, it cannot be efficiently decided in the worst case whether a solution exists at all given the defined time and resource constraints. However, the reason for the failure of the scheduling run for the given problem instance can be identified. This information is presented to the user and he or she may decide how the defined time and resource constraints can be adapted to enable a successful scheduling pass. This is a practicable solution because the automatic scheduling fails in most cases due to conflicting constraints which are revealed during resource-constrained scheduling. Consequently, scheduling of a dynamic task net in PROCEED is an interactive process. The user defines planning data and time constraints and invokes the scheduling algorithm. If scheduling fails or does not yield the expected result, the user can add, remove or change time constraints and control flows, adapt the planning data, and invoke the scheduling algorithm again until an acceptable solution is reached.

The dynamic task net which is stored in the Comos database is mirrored in the computer's main memory for scheduling. This *memory representation* of the dynamic task net contains all information required for scheduling. When scheduling has been successfully terminated, the computed timing property values are written to the database all at once. Working on a memory representation instead of the database speeds up the scheduling algorithms. Furthermore, inconsistent states of the management data in the database are avoided by writing the computed values to the database only after a complete and successful scheduling run.

7.1 Partial Scheduling

Existing algorithms for solving the GRCPSP always take all tasks of a given problem instance into account [DH02]. This is required due to the possible resource dependencies between different tasks. It is in general not possible to schedule parts of the overall activity network separately without considering the resource constraints imposed by other tasks in the network. This procedure would in most cases lead to resource-infeasible schedules. Furthermore, feasible schedules would probably not be optimal, and the optimization of the schedule is the main objective of the majority of approaches found in literature.

Even approaches for schedule repair work on the whole task network and try to globally optimize the schedule after local modifications [Wan05]. With respect to scheduling under uncertainty, other optimization objectives than the minimal project makespan are considered, e.g. the minimization of the sum of the weighted absolute differences between the start time of each task in the repaired schedule and the original start time of that task [HL05, SW00]. In these cases, it is desirable to minimize the effects of local changes on other parts of the schedule to achieve a stability of the planned start times.

The objective for resource-constrained scheduling in PROCEED is to compute a good feasible schedule which is consistent with the planned project deadline. The optimization of the project makespan is not the main objective. In case of disruptions at project runtime, the consistency of the schedule has to be re-established. The effects of local changes to a dynamic task net on other parts of the task net should be limited. Therefore, only some parts of the dynamic task net are rescheduled while others remain unchanged.

There are several reasons for excluding tasks from scheduling. The user may have explicitly specified that certain tasks shall not be changed. Tasks can be too short or too small to be scheduled in a meaningful way. Tasks which define project management activities should not be scheduled as part of the project plan. Therefore, the tasks in a dynamic task net are divided into the following three categories with respect to scheduling.

Zero-duration tasks are not (re)scheduled and do not impose constraints on scheduled tasks. These tasks are excluded from scheduling because of their duration, workload, granularity, or because they represent project management activities.

Not scheduled tasks are not (re)scheduled but do impose constraints on scheduled tasks. These tasks are not scheduled because they are not contained in the subnet to be scheduled or because of their execution states.

Partially scheduled tasks are not moved by the scheduling algorithm but may be prolonged or foreshortened, i.e. their planned start time remains unchanged but their planned end time may change. In this category fall all running but not suspended tasks which do not fall into one of the previous categories.

Scheduled tasks are those tasks for which the scheduling algorithm computes planned start and end times.

When the memory representation of the persistent task net is created, the property values of the tasks in memory are set to the respective property values of the tasks in the Comos database. During this import of the task net into main memory, some property values are not imported.

- The computed constraint dates of tasks are not set in the memory representation. They are completely calculated anew during critical path analysis. However, the manually set constraint dates are imported.

- The planned dates and workload distributions of scheduled tasks are not imported. They are computed during resource-constrained scheduling.
- The planned end times of partially scheduled tasks are not imported. They are updated during resource-constrained scheduling. However, the workload distributions of partially scheduled tasks are imported.

In the following, the criteria for the membership of tasks in the different categories are motivated and defined in detail, and it is described how the tasks which are not regularly scheduled are handled.

7.1.1 Zero-Duration Tasks

Tasks which are too short or too small and tasks which represent management activities are excluded from scheduling. The decision whether a task is a zero-duration task is made based on its planning data, granularity level, and its type. When a task is categorized as a zero-duration task, then all of its descendants in the task net hierarchy are zero-duration tasks as well.

Planning data The size of a task in terms of workload and duration may exclude it from scheduling. Scheduling is useful for tasks with a significant amount of required workload which require at least one work day for completion. For short term tasks resource-constrained scheduling involves a disproportionate overhead. Therefore, the following tasks are not taken into account during resource-constrained scheduling.

- Tasks with a total duration of zero work days. A task for which a duration of zero work days has been explicitly defined is not meant to be scheduled. This includes tasks in the execution state *Skipped*.
- Tasks with an undefined total duration and a total workload of 0 MHRS or an undefined total workload, If the duration of a task is undefined and no man hours are specified, the task cannot be scheduled in a meaningful way because the duration cannot be derived from the total workload.
- Tasks with undefined total duration and without task assignments and subtasks. Unassigned total workload of these tasks cannot be distributed in a meaningful way because the duration cannot be determined.
- Tasks whose respective parent task has a total duration of one work day only. A task with a duration of one work day is scheduled. However, the subtasks of this task are not considered for scheduling since all work can be performed in one day which is the smallest time unit for scheduling.

In all other cases, a task can be scheduled. A task with a duration of more than zero work days but with a total workload of zero man hours is scheduled based on the explicitly defined duration. If the duration of a task is undefined but workload of more than zero man hours is specified for at least one task assignment, the duration is determined by distributing the workload over several workdays according to the work calendars of the task and the assigned resources.

Granularity level In Section 5.1.1, the concept of granularity has been introduced. The granularity level of a task can be explicitly defined. The value *work step* indicates that the task represents a small step in a procedure which is commonly executed by only one resource. Therefore, work steps are excluded from scheduling. Only the tasks with granularity levels *project structure* or *task* are considered for the generation of the project schedule. Even when a significant amount of workload or a duration of several work days is specified for a task with granularity level *work step*, this task is not taken into account during scheduling. To include the task into the project plan, the user has to change its granularity level to *task*.

Project management tasks The work breakdown structure of a project usually contains a first level element which subsumes all project management activities in the project [Bur00, Hau01]. In PROCEED, all project management tasks are stored under the task *Project Management* which is defined by default on the first level of the hierarchical dynamic task net below the root task which represents the whole project. Project management tasks are defined to structure the processes which are enacted to manage a project. These processes include reporting, quality management, and change management, and in particular project planning and replanning. The details of how management tasks are handled in PROCEED will be described in Section 9.1. Management tasks are not part of the project plan because their execution affects the project plan. Therefore, the task *Project Management* and all its descendants are excluded from scheduling.

Processing of zero-duration tasks Zero-duration tasks except management tasks may have scheduled tasks as predecessors or successors in the dynamic task net. Although the zero duration tasks are not scheduled, the task dependencies have to be taken into account during scheduling since they connect the predecessors and successors with each other. Therefore, zero-duration tasks are removed from the dynamic task net before scheduling. Control flows which connect zero-duration tasks are replaced by equivalent ones. This way, dependencies between (partially) scheduled tasks via zero-duration tasks are retained while the zero-duration tasks are deleted. These structural modifications are applied to the memory implementation of the dynamic task net but are not written to the Comos database after scheduling.

For every combination of a control flow from a predecessor and a control flow to a successor of a zero-duration task, an equivalent *replacement control* flow is defined between the predecessor and the successor. The lag time of the replacement control flow is the sum of the lag times of the replaced control flows. The semantics of the replacement control flow can be determined by the following rule. When the control flow to the successor has the standard semantics, then the replacement control flow also has the standard semantics. Otherwise, the replacement control flow has the

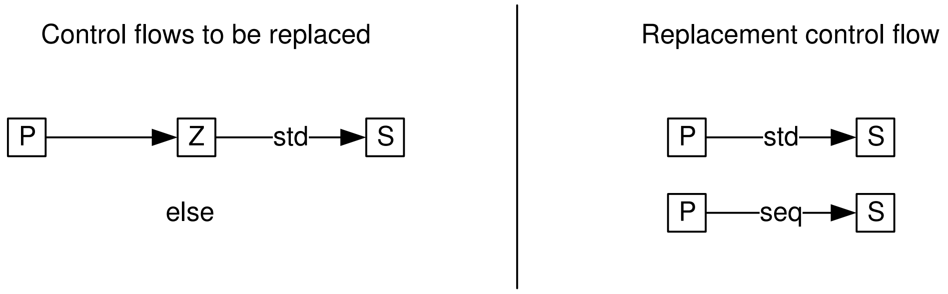


Abbildung 7.1: Replacement rule for control flows of a zero-duration task.

sequential semantics. This rule is illustrated in Figure 7.1 where the task labeled Z is the zero-duration task to be replaced, and the tasks S and P represent a successor and a predecessor task.

The correctness of the rule can be verified by examining all possible combinations of control flows and the resulting constraints imposed on the predecessor and successor tasks. Thereby, the zero-duration task to be deleted is regarded as a task with a duration of zero work days, which means that the start and end events of the task fall on the same date. As a consequence, most combinations transform to a sequential control flow, because a simultaneous control flow requires the same timing as a sequential one. A formal logical proof of the equivalence of the replaced control flows and the replacement control flow is based on this observation. In the following, a sketch of a formal proof is presented for the case of zero lag times. The variable P represents the predecessor task, Z the zero-duration task, and S the successor task in the presented formulas. For every task T the start event is denoted as $T.Start$ and the end event is denoted as $T.End$.

- The replacement control flow has at least standard semantics since both replaced control flows have at least standard semantics.
  $P.End \leq Z.End \wedge Z.End \leq S.End \Rightarrow P.End \leq S.End$

- If the control flow to the predecessor task has standard semantics, the replacement control flow cannot have simultaneous or sequential semantics since the start event of the successor may still occur before the start event of the predecessor.
  $P.End \leq Z.End \wedge Z.End \leq S.End \nRightarrow P.Start \leq S.Start \vee P.End \leq S.Start$
  An according counter example is illustrated in Figure 7.2.

- If the second control flow has simultaneous semantics, the replacement control flow has to have sequential semantics, because the first control flow has at least standard semantics and the start and end event of the 0-duration task fall on the same date.
  $P.End \leq Z.End \wedge Z.Start \leq S.Start \wedge Z.End = Z.Start \Rightarrow P.End \leq S.Start$

- If the second control flow has sequential semantics, the replacement control flow

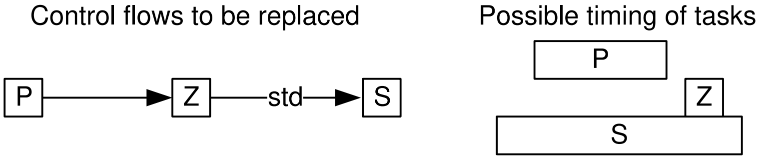


Abbildung 7.2: Possible timing of tasks for the standard successor case.

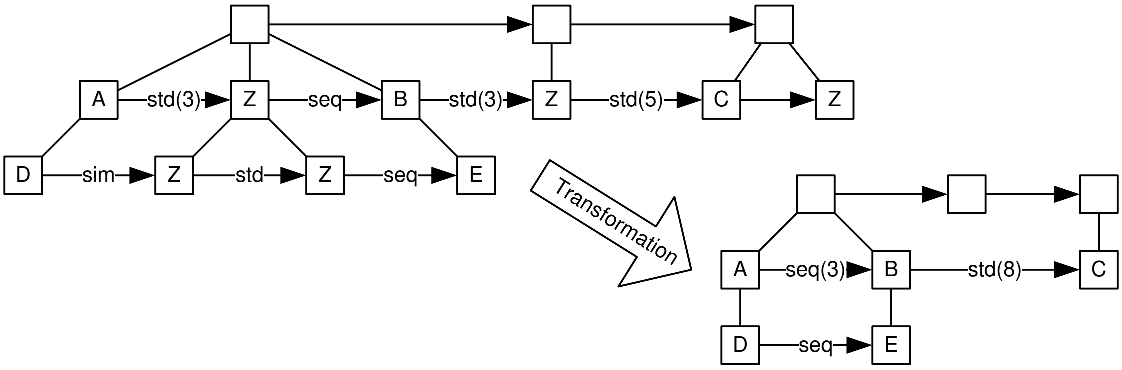


Abbildung 7.3: Example for the elimination of zero-duration tasks from a task net.

has to have sequential semantics as well.
$P.End \leq Z.End \wedge Z.End \leq S.Start \Rightarrow P.End \leq S.Start$

The removal of zero-duration tasks from the memory representation of the dynamic task net and the replacement of control flows is performed as follows. Zero-duration tasks are removed from bottom to top. A zero-duration task may have subtasks which have to be removed first. Before a zero-duration task is removed, the replacement control flows are inserted into the task net. For all combinations of two control flows from a predecessor and to a successor, a replacement control flow is inserted. Afterwards, the zero-duration task and all its control flows are deleted.

As a consequence of the application of the replacement rule, a control flow path between two scheduled tasks is replaced by one control flow when all tasks on the path are zero-duration tasks. If different control flow paths of this kind exist between two scheduled tasks in the original task net, then multiple control flows are introduced between these tasks. In this case, only the most restrictive control flow is kept and the others are discarded.

The removal of zero-duration tasks together with the introduction of replacement control flows may lead to the violation of the structural constraint (5.4) for control flow balancing. However, this violation does not impede scheduling of the transformed task net, and the modified structures are not saved to the Comos database after scheduling.

Figure 7.3 shows an abstract example for the removal of zero duration tasks from a task net. All tasks labeled with Z are zero-duration tasks which are removed by the transformation. The sequential control flow between the tasks A and B results from the combination of the standard and sequential control flows in the original task net. The zero-duration tasks between the tasks D and E are removed before their common parent task. The control flow path between the tasks D and E is replaced by a single control flow because only zero-duration tasks lie on the path. The replacement control flow between tasks B and C violates the control flow balancing constraint. The zero-duration successor of task C is removed without any control flow replacement.

7.1.2 Not Scheduled Tasks and Partially Scheduled Tasks

After zero-duration tasks have been eliminated from the dynamic task net, critical path analysis can be performed for the project. Critical path analysis is always performed on the full task net. Planned start and end times however, are only computed for (partially) scheduled tasks. Tasks which are not descendants of the specified root task are not scheduled. However, not all descendants of the specified root task are necessarily scheduled.

One of the major advantages of dynamic task nets is the incorporation of the execution state of an enacted process instance into a project plan. The information about the execution states of tasks is used for scheduling. Only those tasks in a dynamic task net are scheduled which are in one of the execution states *InDefinition*, *Active*, or *Replanning*. Active and replanning tasks are only *partially scheduled*, i.e. only the planned end time may change. A suspended task has to be resumed to be scheduled. Property values of a waiting task may generally not be changed. Hence a waiting task is not scheduled unless its execution state is changed to *InDefinition* or *Active*, first. Terminated tasks are not changed anymore. Their planned dates are not touched by the scheduling algorithm.

In Section 5.1.3 it has been described that resources can be assigned to tasks by means of the pull- and the push-pattern. In case of the pull-pattern, the tasks which are still in the task pool should not be scheduled and no resource should be automatically assigned by the scheduling algorithm. This is achieved by defining the tasks without assigning actual responsible resources but only the required roles, and changing the execution states to *Waiting*. This way, the tasks in the task pool are not taken into account during scheduling and can be picked up by eligible resources at any time. When a resource has picked up a task and activates it, the actual start time is automatically set to the current date. Since the planned start time is still undefined, it is automatically set to the actual start time. This enables partial rescheduling of the active task at a later point in time.

Besides the execution states, the hierarchical structure of the task net is considered as well to decide which tasks are scheduled. The scheduling algorithm only considers the subtasks of those tasks which are either in the state *InDefinition* or *Replanning*. The planned dates and workload distributions of an active task may change during scheduling, but its subtasks are not scheduled. If a task shall be rescheduled at project runtime, all parent tasks up to the root task specified for

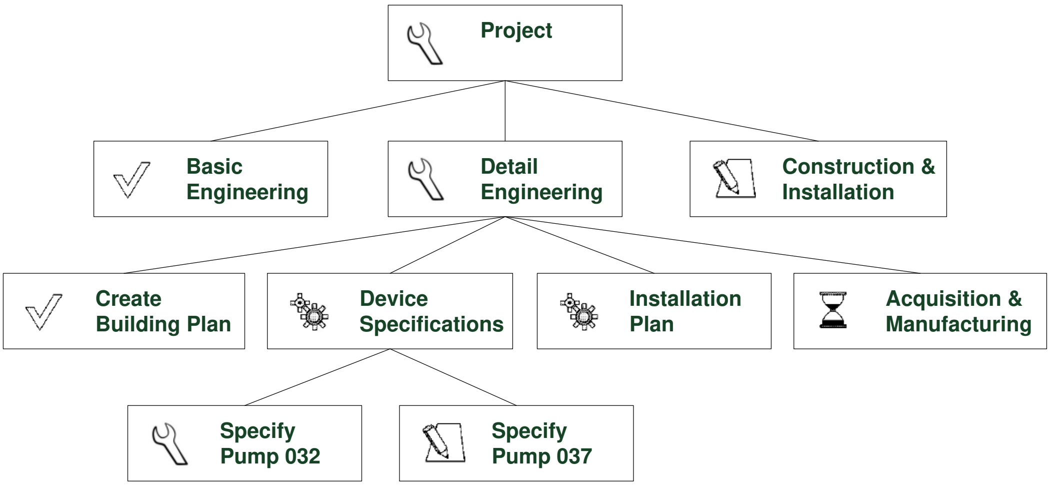


Abbildung 7.4: Influence of execution states on rescheduling.

scheduling have to be in the execution state *Replanning*.

Figure 7.4 shows an example of a task net hierarchy in which tasks have different execution states. Scheduling of the whole task net may lead to changed planned dates of the tasks Project, Detail Engineering, Device Specifications, Installation Plan and Construction & Installation. The tasks Basic Engineering, Create Building Plan and Acquisition & Manufacturing are not scheduled due to their execution states. The tasks Specify Pump 032 and Specify Pump 037 are not scheduled because their parent task is active and not in the state *Replanning*. If the task Detail Engineering is specified as the root task for scheduling, the task Construction & Installation becomes a not scheduled task and may for example not be moved to a later date.

Not scheduled and (partially) scheduled tasks can impose resource and time constraints on (partially) scheduled tasks. The not scheduled tasks may have been scheduled before, e.g. when they had different execution states. In this case, their planned start and end times are set and resources are assigned which use part of their total workload for these tasks. There are many possible interdependencies between not scheduled tasks and scheduled tasks like the following examples show.

- The duration of an active task cannot be reduced because the scheduled subtasks require the currently planned duration and their planned start and end times and durations are not modified.
- A task in the execution state *InDefinition* cannot be scheduled for a later date because a sequential successor task is in the *Waiting* state and cannot be moved.
- A task in the execution state *InDefinition* has to be moved to a later planned start time because the required resource is still assigned to a prolonged active task.

The planned dates of not scheduled tasks are taken into account during scheduling as additional constraints for the planned start and end times of the scheduled tasks.

The used workload of the not scheduled tasks is implicitly taken into account via the work calendars of the assigned resources.

7.2 Critical Path Analysis

The critical path method (CPM) is applied to compute the earliest and latest start and end times and the total float of the tasks in a dynamic task net. The computed latest possible start times of tasks are used for the prioritization of tasks during resource-constrained scheduling (cf. Section 7.3). Furthermore, the computed earliest possible start times are used as constraints for resource-constraint scheduling. If critical path analysis is performed for a fixed project deadline, then the computed latest possible end times can be used as constraints during resource-constraint scheduling as well.

The general approach for CPM is divided into two phases: forward scheduling and backward scheduling (cf. Section 3.2.1). During forward scheduling, the earliest possible start and end times of all tasks in the dynamic task net are calculated. Backward scheduling yields the latest possible start and end times of the tasks.

The critical path analysis requires that the release date of the project has been defined. Furthermore, the due date of the project has to be defined for backward scheduling to obtain meaningful latest possible times and total floats for the tasks in the project. If no project deadline is specified, the earliest possible end time of the project which is computed by forward scheduling is used as the latest possible end time of the project for backward scheduling. However, in this case the computed latest possible end times of tasks cannot be used for consistency checks during resource-constrained scheduling as described in Section 7.3.

Critical path analysis is always performed on the full task net after zero-duration tasks have been eliminated as described in Section 7.1. Earliest and latest possible times are computed for (partially) scheduled tasks and not scheduled tasks, because the latter constrain the scheduling of the former.

Before the CPM algorithm starts, the task net is initialized. Values for computed constraint dates of tasks which are stored in the Comos database are not imported into the memory representation for scheduling. Instead, the computed constraint dates of the tasks are set as follows, or remain undefined otherwise. If the value of a property is not defined, it is not assigned to the respective earliest or latest time, and the latter remains undefined until it is set during CPM scheduling.

- For *all tasks*, the earliest possible start and latest possible end times are set to the release and due dates respectively.
  `Task.EPST` := `Task.ReleaseDate`
  `Task.LPET` := `Task.DueDate`
  The implemented CPM algorithm takes these initial values into account and thereby ensures that the computed constraint dates are at least as restrictive as the manually set constraint dates.

- For all *partially scheduled tasks*, the earliest and latest possible start times are set to the planned start time of the task.
  `Task.EPST` := `Task.PlannedStartTime`
  `Task.LPST` := `Task.PlannedStartTime`
  This ensures, that the planned start times of active and replanned tasks are not moved by the scheduling algorithm.

- For all *not scheduled tasks*, the earliest and latest possible start and end times are set to the planned start and end times respectively.
  `Task.EPST` := `Task.PlannedStartTime`
  `Task.LPST` := `Task.PlannedStartTime`
  `Task.EPET` := `Task.PlannedEndTime`
  `Task.LPET` := `Task.PlannedEndTime`

Several approaches exist to extend the CPM to task relations of the precedence diagramming method which include standard and simultaneous control flows with minimal time lags [EK92, Wie81, DH02]. However, all algorithms which are known to the author of this thesis assume a flattened task net and do not explicitly address the problem of a hierarchical structure. In particular, the intricacies involved with task relations of the precedence diagramming method in combination with complex tasks has not been tackled elsewhere. The neglecting of hierarchical task nets in CPM literature may be due to the fact, that in conventional approaches to project planning, task relations are only defined for activities below the level of work packages. The work breakdown structure is assumed to be complete and its hierarchy is not taken into account. In contrast, scheduling of dynamic task nets should also be possible for an incomplete WBS. Furthermore, the duration of complex tasks in a dynamic task cannot be directly derived from its subtasks, because its duration can be defined independently, and task assignments may lead to an even longer duration. Finally, the duration does not need to be specified explicitly but can be derived from the defined task assignments of a task even during CPM. The critical path analysis is performed in PROCEED on a hierarchical dynamic task net by performing a depth first traversal of the hierarchy.

7.2.1 Hierarchical Critical Path Method

The critical path method for hierarchical task nets requires a preprocessing step to ensure the correct scheduling of all subtasks in a common realization. Afterwards, forward and backward scheduling is performed which traverse the task net along the task-subtasks relationships and the defined control flows.

Preprocessing For every task, virtual start and end nodes are created. These virtual tasks have no duration and no resource requirements. The virtual start and end nodes are introduced to ensure that every subtask is reached and the task net of the realization is traversed in topological order. Furthermore, the consistency

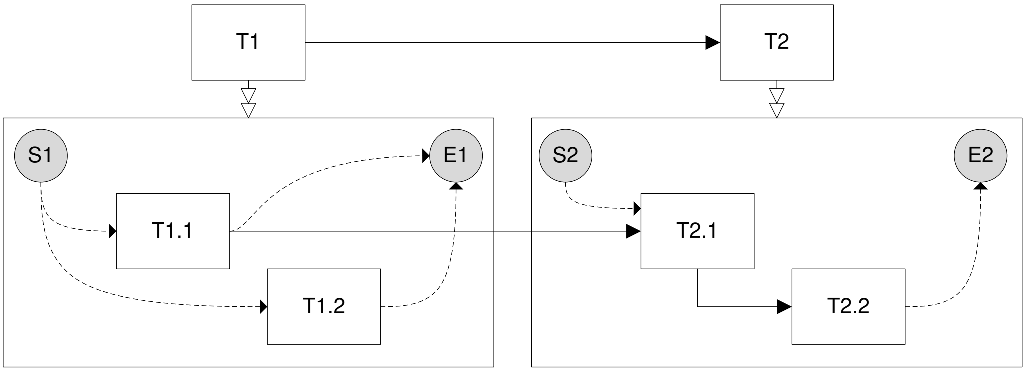


Abbildung 7.5: Virtual start and end nodes in a hierarchical task net.

constraints that no subtask may have an earlier EPST or a later LPET than its parent task is ensured by the introduction of virtual start and end nodes. Therefore, every subtask of the considered task, which does not have a predecessor in the same realization or only predecessors connected by standard control flows, becomes a sequential successor of the virtual start node. Every subtask which does not have a successor in the same realization is connected by a sequential control flow with the virtual end node.

An example for virtual start and end nodes in a hierarchical dynamic task net is depicted in Figure 7.5. The tasks T1.1, T1.2, and T2.1 do not have a predecessor in the same realization. Therefore, they are connected to the virtual start nodes S1 and S2 respectively. The tasks T1.1, T1.2, and T2.2 do not have a successor in the same realization. Therefore, they are connected to the virtual end nodes E1 and E2 respectively. Task T1.1 is connected to E1 and T2.1 is connected to S2 despite the control flow between T1.1 and T2.1 since it connects two tasks from different realizations.

The earliest and latest possible start times of the virtual start nodes are set to the respective values of their parent tasks. Likewise, the earliest and latest possible end times of the virtual end nodes are set to the respective values of their parent tasks. The property values of the parent tasks stem from the initialization described earlier.

Forward scheduling Algorithm 7.1 shows the procedure for forward scheduling. Forward scheduling is started by invoking the method `ScheduleForward(t)` for the root node of the dynamic task net. The earliest possible start time of the task $\mathtt{t} \in \mathtt{Tasks}$ has to be set. First, the subtasks of the task `t` are scheduled to determine its earliest possible end time. Second, the constraints imposed by incoming active feedback flow relationships on the earliest possible end times of the respective source tasks are handled. If the currently processed task is the target of an active feedback flow, then the earliest possible end time of the source must be later than the EPET of the current task. Third, the resulting earliest possible end time is used to compute the earliest start and end times of all successors recursively.

The method `HandleControlFlowForward(c)` determines the earliest possible start

Algorithm 7.1 `ScheduleForward(t)`

```
ScheduleForwardTask(t)
for all f ∈ FeedbackFlows do
  if f.IsActive = true ∧ f.Target = t then
    if undef(f.Source.EPET) ∨ f.Source.EPET < f.Target.EPET then
      f.Source.EPET := f.Target.EPET
    end if
  end if
end for
for all ControlFlow c ∈ t.ControlFlows do
  HandleControlFlowForward(c)
end for
```

or end date of the successor depending on the semantics of the control flow and invokes recursive forward scheduling for the successor if it is contained in the same realization as the predecessor. The pseudo code for this method is given in Algorithm 7.2. The sequential control flow is handled similarly to the classical Critical Path Method except that minimal lag times are taken into account. For a sequential control flow, the earliest possible start time (EPST) of the successor is derived from the earliest possible end time (EPET) of the predecessor by adding the minimal lag time of the control flow. As defined in Section 5.3.2, the lag time in calendar days is generally derived from the lag time in work days by using the work calendar of the successor task of the control flow.

Standard and simultaneous control flows are not considered in the classical Critical Path Method. A standard control flow does not directly impose a constraint on the earliest possible start time of the successor but on its earliest possible end time. The method `HandleControlFlowForward(c)` first checks whether the earliest possible end time of the successor is still undefined or whether the control flow imposes a more restrictive constraint. In these cases the EPET of the successor is set to the EPET of the predecessor plus the minimal lag time of the control flow. The earliest possible start time of the successor of the standard control flow is constrained by the EPST of the parent task, and it can already be defined by forward scheduling of another incoming control flow. If required, the EPST of the successor is set to the EPST of its parent task.

A simultaneous control flow imposes constraints on the EPST and EPET of the successor. As described in Section 5.3.1, the minimal time lag has to elapse between the start and end events of the connected tasks. Therefore, the handling of a simultaneous control flow sets the EPST and the EPET of the successor to consistent dates if required.

Recursive forward scheduling is only performed for the successor task if it is contained in the same realization as the predecessor. Otherwise, the earliest possible start and/or end times of the successor task are set but the method `ScheduleForward` is not invoked. In this case, the successor task will be handled later by the CPM algorithm when its parent task is scheduled. At that point, the constraints imposed

Algorithm 7.2 `HandleControlFlowForward(c)`

```
if $c.Semantics = Sequential$ then
  if $undef(c.Succ.EPST) \lor c.Pred.EPET + c.LagTime > c.Succ.EPST$ then
    $c.Succ.EPST := c.Pred.EPET + c.LagTime$
  end if
else if $c.Semantics = Standard$ then
  if $undef(c.Succ.EPET) \lor c.Pred.EPET + c.LagTime > c.Succ.EPET$ then
    $c.Succ.EPET := c.Pred.EPET + c.LagTime$
    if $c.Succ.Parent = c.Pred.Parent \land undef(c.Succ.EPST)$ then
      $c.Succ.EPST := c.Succ.Parent.EPST$
    end if
  end if
else if $c.Semantics = Simultaneous$ then
  if $undef(c.Succ.EPST) \lor c.Pred.EPST + c.LagTime > c.Succ.EPST$ then
    $c.Succ.EPST := c.Pred.EPST + c.LagTime$
  end if
  if $undef(c.Succ.EPET) \lor c.Pred.EPET + c.LagTime > c.Succ.EPET$ then
    $c.Succ.EPET := c.Pred.EPET + c.LagTime$
  end if
end if
if $c.Succ.Parent = c.Pred.Parent$ then
  ScheduleForward(c.Succ)
end if
```

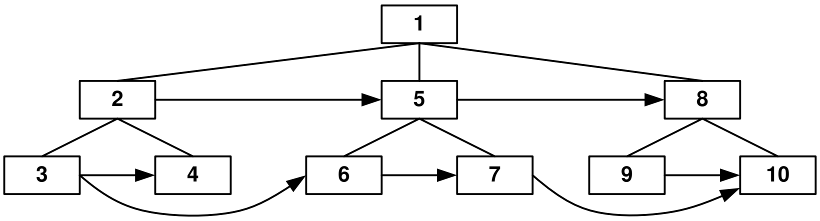


Abbildung 7.6: Traversal order for critical path analysis.

by the predecessors are already incorporated into the earliest possible times of the task and will be taken into account by the algorithm. Altogether, the CPM algorithm performs a depth-first traversal of the task net hierarchy which is illustrated in Figure 7.6 where the numbers of the tasks determine the order in which they are handled by the algorithm.

While the method `HandleControlFlowForward(c)` traverses the dynamic task net in horizontal direction, the method `ScheduleForwardTask(t)`, which is defined by Algorithm 7.3, schedules the workload, task assignments and subtasks of a task and thereby traverses the dynamic task net in vertical direction. The first step is to determine the EPET of the task $t \in \texttt{Tasks}$ based on its total duration if it is defined. Then, the task assignments of task `t` are scheduled. Thereby no actual resources are considered. For the scheduling of a task assignment, the standard work calendar with the most available work days is selected from the work calendars of the resources who can play the required role of the task assignment, e.g. a 6-day calendar if there is an eligible resource with such a calendar. If the scheduling of the task assignments leads to a longer duration and a later EPET, this value is used in the following.

Algorithm 7.3 `ScheduleForwardTask(t)`

```
if $\neg$undef(t.TotalDuration) then
    EPET := DetermineEndDate(t.EPST, t.TotalDuration, t.WCal)
end if
AEPET := ScheduleAssignmentsForward(t)
if AEPET $>$ EPET then
    EPET := AEPET
end if
if $|$t.Subtasks$| > 0$ then
    t.VirtualStart.EPST := t.EPST
    ScheduleForward(t.VirtualStart)
end if
if t.VirtualEnd.EPET $>$ EPET then
    EPET := t.VirtualEnd.EPET
end if
if undef(t.EPET) $\vee$ EPET $>$ t.EPET then
    t.EPET := EPET
end if
```

After scheduling the task assignments, the subtasks of task $\mathtt{t}$ are scheduled, where $\mathtt{t.VirtualStart}, \mathtt{t.VirtualEnd} \in \mathtt{t.Subtasks}$ are the virtual start and end nodes of the realization of the task $\mathtt{t}$. Scheduling the subtasks results in the earliest possible end time of the realization of the task. Thereby, possibly set constraint dates for the subtasks are taken into account. If the earliest possible end time of the realization is later than the EPET derived from the total duration and task assignments, then it is used in the following. The EPET of the task may have been set before when a standard or simultaneous control flow from a predecessor was handled. If it is later than the EPET which has been determined by scheduling task assignments and subtasks, then the value is not changed.

In the classical critical path method which only considers end-start precedence relations, the earliest possible end time of a task is derived from the earliest possible start time during forward scheduling by adding the duration of the task. As a consequence, the formula $\forall \mathtt{t} \in \mathtt{Tasks}(\mathtt{t.EPET} - \mathtt{t.EPST} = \mathtt{t.TotalDuration})$ is valid for a consistent CPM result.

For a task net with standard and simultaneous control flows, this equation does not necessarily hold. Only the formula $\forall \mathtt{t} \in \mathtt{Tasks}(\mathtt{t.EPET} - \mathtt{t.EPST} \geq \mathtt{t.TotalDuration})$ is valid for a consistent CPM result. The time span between the earliest possible start and end times of a task does not necessarily equal the duration of the task but

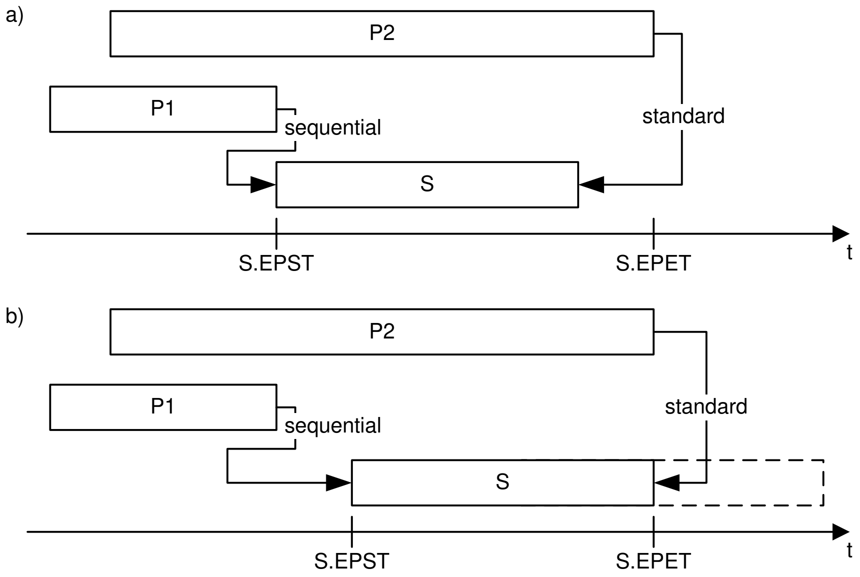


Abbildung 7.7: Example for constrained EPET and discarded solution.

may be longer because standard and simultaneous control flows from predecessors may impose constraints on the earliest possible end time.

This circumstance is illustrated in Figure 7.7 a) where the task S has two predecessors P1 and P2. The sequential control flow from P1 constrains the start event of the task S and leads to the depicted earliest possible start time during forward scheduling. The standard control flow from P2 constrains the earliest possible end time, so that S.EPET-S.EPST>S.TotalDuration. In this example, the earliest possible start time of task S is too optimistic for the computed total duration.

A possible solution to this problem is illustrated in Figure 7.7 b) where the earliest possible start time of the task S is moved to a later point in time, so that the end time of the task obtained by adding its duration equals the earliest possible end time, i.e. S.EPET-S.EPST=S.TotalDuration holds. This solution seems to make sense for critical path analysis. However, it is not useful in the context of resource-constrained scheduling. The CPM results are used during resource-constrained scheduling as additional constraints, so that the task S may not be scheduled earlier than its earliest possible start time S.EPST. However, the durations of tasks are probably longer during resource-constrained scheduling because resource availabilities are taken into account. The task durations computed during CPM are the minimal durations. This is illustrated by the dashed line which extends the task S in Figure 7.7 b). The task S could possibly be scheduled before S.EPST without violating the constraint imposed by the standard control flow. Hence, the moved earliest possible start time would be too strict for resource-constrained scheduling.

For this reason, the described solution has been discarded, and the ïnconsistencyöf earliest times and task durations in the CPM result are accepted. The earliest possible start times which result from the critical path analysis are anyway too optimistic for resource-constrained scheduling since the CPM works with minimal task durations. Furthermore, control flow dependencies have to be checked during resource-constrained scheduling, so that tasks with standard and simultaneous predecessors are not scheduled too early. The same considerations can be made for backward scheduling where simultaneous control flows may constrain the latest possible start times which leads to similar ïnconsistencies".

Backward scheduling Backward scheduling is performed similarly to forward scheduling. If a project deadline has been manually specified by the user, i.e. the due date of the root node of the task net is defined, this date is used as the LPET. Otherwise, the computed EPET of the root node is used as the LPET. In this case, the computed latest possible start and end times are not used as constraints for resource-constrained scheduling.

The algorithm for backward scheduling incorporates several methods which have been defined analogously to the methods for forward scheduling. Algorithm 7.4 shows the method `ScheduleBackward(t)` which takes a task with defined LPET as input. First, the task is scheduled recursively which results in an LPST for the task. Afterwards, the outgoing active feedback flows are handled to ensure that the targets of the feedback flows have an earlier latest possible end time than the currently processed task. Finally, the dynamic task net is traversed along the control flow relationships in opposite direction from successors to predecessors.

Algorithm 7.4 `ScheduleBackward(t)`

```
ScheduleBackwardTask(t)
for all f $\in$ t.ActiveFeedbacks do
  if undef(f.Target.LPET) $\vee$ f.Target.LPET $>$ t.LPET then
    f.Target.LPET := t.LPET
  end if
end for
for all ControlFlow c $\in$ ControlFlows : c.Succ $=$ t do
  HandleControlFlowBackward(c)
end for
```

Control flows are handled similarly to forward scheduling. A standard control flow constrains the LPET of the predecessor just like the parent task of the predecessor. In contrast to that, the EPST and EPET are set during forward scheduling, when a standard control flow is handled. When a simultaneous control flow is handled during backward scheduling, the LPET and the LPST of the predecessor are set as required.

The constraint imposed on the LPST of a simultaneous predecessor does not cause the same problems as the constraint imposed on the EPET of a simultaneous successor which has been described before. Because the LPST and LPET specify the latest dates for the respective events, scheduling a task at the LPST with a duration which is shorter than the time span between LPST and LPET can never lead to inconsistencies with respect to the LPET of the task.

Algorithm 7.5 `HandleControlFlowBackward(c)`

```
if $\texttt{c.Semantics} = \texttt{Sequential}$ then
    if $\texttt{undef}(\texttt{c.Pred.LPET}) \lor \texttt{c.Succ.LPST} - \texttt{c.LagTime} < \texttt{c.Pred.LPET}$ then
        $\texttt{c.Pred.LPET} := \texttt{c.Succ.LPST} - \texttt{c.LagTime}$
    end if
else if $\texttt{c.Semantics} = \texttt{Standard}$ then
    if $\texttt{undef}(\texttt{c.Pred.LPET}) \lor \texttt{c.Succ.LPET} - \texttt{c.LagTime} < \texttt{c.Pred.LPET}$ then
        $\texttt{c.Pred.LPET} := \texttt{c.Succ.LPET} - \texttt{c.LagTime}$
    end if
else if $\texttt{c.Semantics} = \texttt{Simultaneous}$ then
    if $\texttt{undef}(\texttt{c.Pred.LPET}) \lor \texttt{c.Succ.LPET} - \texttt{c.LagTime} < \texttt{c.Pred.LPET}$ then
        $\texttt{c.Pred.LPET} := \texttt{c.Succ.LPET} - \texttt{c.LagTime}$
    end if
    if $\texttt{undef}(\texttt{c.Pred.LPST}) \lor \texttt{c.Succ.LPST} - \texttt{c.LagTime} < \texttt{c.Pred.LPST}$ then
        $\texttt{c.Pred.LPST} := \texttt{c.Succ.LPST} - \texttt{c.LagTime}$
    end if
end if
if $\texttt{c.Pred.Parent} = \texttt{c.Succ.Parent}$ then
    ScheduleBackward(c.Pred)
end if
```

The method `ScheduleBackwardTask(t)` is shown in Algorithm 7.6. First the total duration of the task is used to derive the LPST from the LPET. In the second step, the workload of the task assignments is distributed backwards, starting from the LPET. If the distribution leads to an earlier LPST, this value is used in the following. Afterwards, the subtasks are scheduled and it is checked whether the resulting LPST is even earlier than the previously computed value. Finally, the LPST is set to the computed value if the latter is earlier than a previously set date or if the LPST of the task has not been defined yet.

7.2.2 Criticality and Consistency

The total float of a task is the amount of time for which the task can be delayed without delaying the whole project. For task nets with task relations of the precedence diagramming method, only the earliest and latest possible start times are used for computing task floats [DH02]. The *total float* of a task $\texttt{t} \in \texttt{Tasks}$ is defined as follows.

$$\texttt{t.TotalFloat} := \texttt{t.LPST} - \texttt{t.EPST} \tag{7.1}$$

In the presence of standard and simultaneous control flows, the time span between the earliest and latest possible start times does not necessarily equal the time span between the earliest and latest possible end dates, i.e. the formula

$$\forall \texttt{t} \in \texttt{Tasks}(\texttt{t.LPST} - \texttt{t.EPST} = \texttt{t.LPET} - \texttt{t.EPET})$$

Algorithm 7.6 ScheduleBackwardTask(t)

```
if $\neg$undef(t.TotalDuration) then
  LPST := DetermineStartDate(t.LPET, t.TotalDuration, t.WCal)
end if
ALPST := ScheduleAssignmentsBackward(t, t.LPET)
if ALPST $<$ LPST then
  LPST := ALPST
end if
if $|$t.Subtasks$|$ $<$ 0 then
  t.VirtualEnd.LPET := t.LPET
  ScheduleBackward(t.VirtualEnd)
end if
if t.VirtualStart.LPST $<$ LPST then
  LPST := t.VirtualStart.LPST
end if
if undef(t.LPST) $\vee$ LPST $<$ t.LPST then
  t.LPST := LPST
end if
```

does not necessarily hold for the CPM results. For the classical CPM approach the equality is true for all tasks and defines the total float of a task.

A task which has a total float of zero days is called a *critical task*. A *critical path* is a path in the task net from the virtual start node of the project to the virtual end node which only contains critical tasks. However, due to release and due dates, a task can be critical without lying on a critical path. A complex task can be critical although the subnet defined by its subtasks does not contain a critical path. This is the case, when the total duration of the task is longer than the computed duration of the subnet.

In this thesis, the critical path analysis is also applied to dynamic task nets which represent running process instances. To avoid that already started tasks are moved to a different start date, the earliest and latest possible start times of running tasks are initially set to the planned start time, and for terminated tasks the earliest and latest possible end times are set tot the planned end time as well. As a consequence, the total float of started tasks would always be zero if it was computed according to equation (7.1). Therefore, the total float of running tasks is computed based on the earliest and latest possible end times.

$$\forall \mathtt{t} \in \mathtt{Tasks}(\mathtt{t.State} \in \mathtt{Running} \Rightarrow \mathtt{t.TotalFloat} = \mathtt{t.LPET} - \mathtt{t.EPET}) \tag{7.2}$$

The total float of terminated tasks is always zero.

If the result of critical path analysis is inconsistent, i.e. there is at least one task $\mathtt{t} \in \mathtt{Tasks}$ for which $\mathtt{t.LPST} < \mathtt{t.EPST}$ or $\mathtt{t.LPET} < \mathtt{t.EPET}$, then there is no time-feasible schedule which fulfills all specified time constraints (manually set constraint dates, control flows, and task durations).

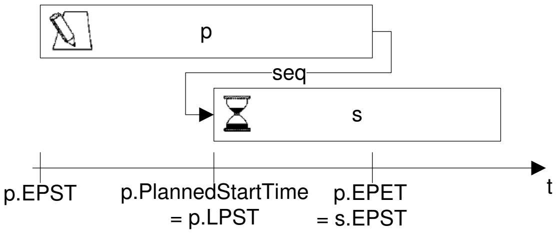


Abbildung 7.8: Example for CPM failure due to a not scheduled task.

In particular, the constraints imposed by partially scheduled and not scheduled tasks may lead to inconsistent computed constraint dates. In this case, the critical path analysis fails. For tasks which are in the execution states *Active* or *Replanning*, the planned start time is fixed. For all other running and terminated tasks the planned start and end times are fixed. Furthermore, the planned dates of waiting tasks may not be changed. This is realized by initializing the earliest and latest possible times with the corresponding planned dates. As a consequence, if critical path analysis computes a later earliest time or an earlier latest time for a task compared to the initially set values, then the constraint dates are inconsistent and the CPM computation fails. The user is informed about the reason for the failure and may adapt the dynamic task net to enable a successful scheduling run.

In the example in Figure 7.8, the task $\mathtt{p} \in \mathtt{Tasks}$ has been created at project runtime and the dynamic task net shall be rescheduled. The task $\mathtt{s} \in \mathtt{Tasks}$ has been defined and scheduled before, and its execution state has been changed to *Waiting*. As a consequence, the task `s` is not rescheduled. Its EPST and LPST are set to the planned start time before critical path analysis is performed. The CPM increases the EPST of task `s` due to its sequential predecessor `p`. The computed constraint dates are inconsistent because $\mathtt{s.EPST} > \mathtt{s.LPST}$. The reason for this inconsistency is that task `p` cannot be scheduled before task `s` without moving `s`. The user is informed about this problem and could for example change the execution state of task `s` to *InDefinition*, so that it can be moved to a later planned start time.

Figure 7.9 shows the cutout of the dynamic task net introduced in Section 2.3 with computed constraint dates and total float values. The earliest possible end times and latest possible start times have been derived from the respective start and end times based on the defined total durations and total workload of the tasks. Thereby, unconstrained resource availabilities are assumed. For scheduling task assignments, the work calendars of the assigned resources or required roles with the most available working days are used. If this still leads to a longer duration for a task than the manually set total duration, then the longer duration is used for the CPM computations but the total duration is not adapted in the database. The total duration of a task is only increased during resource-constrained scheduling if required.

The total float of 58 work days for the tasks Basic Engineering and Detail Engineering results from the circumstance that the project deadline is set to the 17th

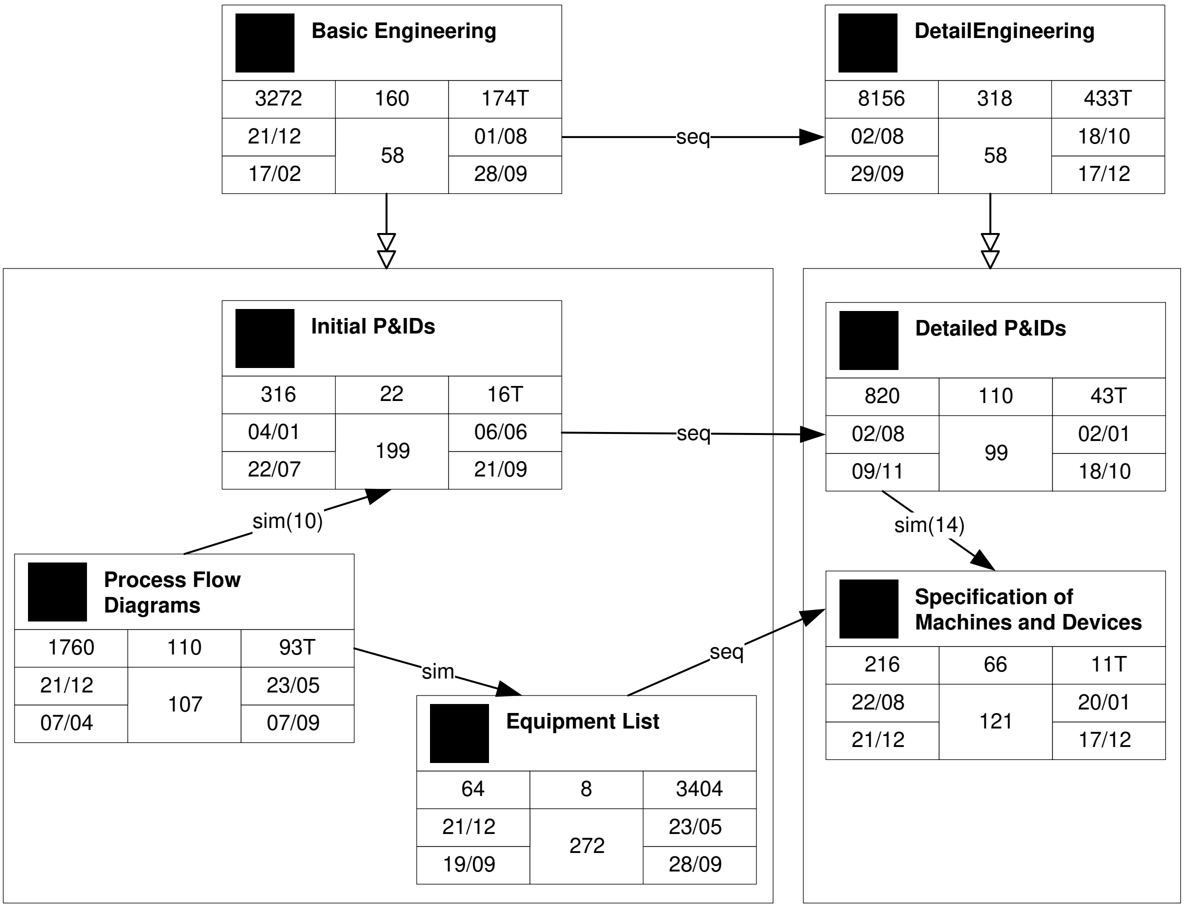


Abbildung 7.9: Computed constraint dates for cutout of example scenario.

December 2012 while the earliest possible end time of Detail Engineering is the 18th October 2012. This leaves a time buffer of 58 work days and avoids any critical tasks in the base schedule. As a consequence, some delays of tasks and the creation of new tasks at project runtime can be compensated. The total float values of the subtasks of the tasks Basic Engineering and Detail Engineering are even larger because the durations of the parent tasks also incorporate a certain time buffer. As a consequence, the subtasks may be delayed for a certain amount of time without increasing the duration of their respective parent tasks, and the total float of the parent tasks is incorporated into the total float of the subtasks as well.

7.2.3 Correctness and Time Complexity

In this section, the correctness of the algorithm for critical path analysis is shown, and an upper bound for the time complexity of the algorithm is determined.

Termination The first step to prove the correctness of the developed algorithm is to show that it always terminates. The hierarchical critical path method performs a depth first traversal of the task net hierarchy and traverses the control flows within the realization of each task. For every complex task, the algorithm descents into its realization exactly once. Every task in a realization is visited only a finite number of times since the control flow relationships between tasks in a dynamic task net do not form a cycle. Therefore, the implemented critical path method always terminates.

Correctness The requirements for a time feasible schedule have been defined in the form of timing consistency constraints in Section 5.3.2. It remains to be shown that the scheduling algorithm yields computed constraint dates which fulfill the timing consistency constraints. This is informally proven in the following by mapping the constraints to those parts of the presented algorithm which ensure their fulfillment.

Table 7.1 shows the timing consistency constraints which are relevant for critical path analysis in the left column. In the right column, the corresponding methods and lines in the presented pseudo code are given. The EPET of a task $\mathtt{t} \in \mathtt{Tasks}$ is determined based on the EPST of the task during forward scheduling by the method $\mathtt{ScheduleForwardTask}(\mathtt{t})$. The implementation guarantees that $\mathtt{t.EPET} \geq \mathtt{t.EPST}$ holds so that constraint (5.41) is satisfied. The durations of the tasks which are used during CPM depend on the earliest possible start date or latest possible end date for which they are computed during forward or backward scheduling, respectively. When for a task $\mathtt{t} \in \mathtt{Tasks}$ the inequation $\mathtt{t.LPET} \geq \mathtt{t.EPET}$ holds, then also $\mathtt{t.LPST} \geq \mathtt{t.EPST}$ holds, because for $\mathtt{t.LPET} = \mathtt{t.EPET}$ the computed duration is equal for forward and backward scheduling because equally many working days are available in both cases. Therefore, the LPST cannot be earlier than the EPST when the LPET is later than the EPET. Since backward scheduling is started for a LPET which is greater or equal to the EPET of the project, constraints (5.42) and (5.43) are fulfilled. The timing consistency constraints (5.45) to (5.48) which relate the computed constraint dates of a task to the constraint dates of its parent task are fulfilled in a scheduled task net because the constraint dates of the subtasks are computed based on the constraint dates of the parent task during the depth first traversal of the task net hierarchy. The earliest possible start times and latest possible end times are initialized with the release and due dates of the respective tasks if they are defined. During forward and backward scheduling, these dates are only updated if they have to be more restrictive. Therefore, the timing consistency constraints (5.49) and (5.50) are fulfilled by the scheduled task net. The timing consistency constraints (5.51) to (5.60) which concern the computed constraint dates of tasks connected by control flows are fulfilled due to the implementation of the methods `HandleControlFlowForward` and `HandleControlFlowBackward` which handle every control flow in the task net to set the constraint dates of the predecessors and successors accordingly. The constraints (5.59) and (5.60) with respect to feedback flows are implicitly fulfilled when a control flow path exists from the feedback flow's target to its source as demanded by constraint (5.6). However, diagonal feedback flows require special treatment in the methods `ScheduleForward` and `ScheduleBackward`.

Constraint	**According part(s) of CPM algorithm**
(5.41)	`ScheduleForwardTask` computes the EPET based on the EPST
(5.42)	Backward scheduling is started with LPET$\geq$EPET for the project and if LPET$\geq$EPET for a task, then LPST$\geq$EPST
(5.43)	
(5.44)	`ScheduleBackwardTask` computes the LPST from the LPET
(5.45)	`ScheduleForwardTask` lines 8-11
(5.46)	`ScheduleBackwardTask` lines 12-14
(5.47)	`ScheduleForwardTask` lines 12-14
(5.48)	`ScheduleBackwardTask` lines 8-11
(5.49)	Initialization and `HandleControlFlowForward` lines 2 and 13
(5.50)	Initialization and `HandleContro0lFlowBackward` lines 2, 6 and 10
(5.51)	`HandleControlFlowForward` lines 6-7
(5.52)	`HandleControlFlowBackward` lines 6-7
(5.53)	`HandleControlFlowForward` lines 16-18
(5.54)	`HandleControlFlowBackward` lines 10-12
(5.55)	`HandleControlFlowForward` lines 13-15
(5.56)	`HandleControlFlowBackward` lines 13-15
(5.57)	`HandleControlFlowForward` lines 2-4
(5.58)	`HandleControlFlowBackward` lines 2-4
(5.59)	`ScheduleForward` lines 2-8
(5.60)	`ScheduleBackward` lines 2-6

Tabelle 7.1: Timing consistency constraints fulfillment by CPM algorithm.

Time complexity The time complexity of the hierarchical CPM implementation does not differ from the time complexity of the classical forward and backward calculations for flat task nets which are of time complexity $O(|\mathtt{T}|^2)$ [DH02] where $\mathtt{T} \subset \mathtt{Tasks}$ is the set of all tasks in the dynamic task net except for zero-duration tasks. Every subnet which is contained in the realization of a complex task is handled like in the classical approach. Every complex task is visited exactly once during the depth-first traversal of the task net hierarchy.

7.3 Resource-Constrained Scheduling

Resource-constrained scheduling sets planned start and end times for tasks and assigns resources to task assignments. Thereby, control flow dependencies, constraint dates and limited resource availabilities are taken into account. Furthermore, the planned workload of tasks and task assignments is distributed over the days of the respective work calendars.

For the description of the algorithm some definitions are required in addition to the definitions from Chapter 5. For a task $\mathtt{t} \in \mathtt{Tasks}$, the following additional properties are defined.

- $\mathtt{Task.IncomingCFs} := \{\mathtt{c} | \mathtt{c} \in \mathtt{ControlFlows} \land \mathtt{c.Succ} = \mathtt{t}\} \subset \mathtt{ControlFlows}$

denotes the set of incoming control flows from predecessors.

- $\texttt{Task.IStdCFs} := \{\texttt{c}|\texttt{c} \in \texttt{Task.IncomingCFs} \land \texttt{c.Semantics} = \texttt{Standard}\}$
- $\texttt{Task.ISimCFs} := \{\texttt{c}|\texttt{c} \in \texttt{Task.IncomingCFs} \land \texttt{c.Semantics} = \texttt{Simultaneous}\}$
- $\texttt{Task.ISeqCFs} := \{\texttt{c}|\texttt{c} \in \texttt{Task.IncomingCFs} \land \texttt{c.Semantics} = \texttt{Sequential}\}$
- $\texttt{t.P}_{\texttt{Std}} := \{\texttt{p}|\exists \texttt{c} \in \texttt{p.StdCFs} : \texttt{c.Succ} = \texttt{t}\} \subset \texttt{Tasks}$ denotes the set of direct predecessors of the task which are connected by a standard control flow.
- $\texttt{t.P}_{\texttt{Sim}} := \{\texttt{p}|\exists \texttt{c} \in \texttt{p.SimCFs} : \texttt{c.Succ} = \texttt{t}\} \subset \texttt{Tasks}$ denotes the set of direct predecessors of the task which are connected by a simultaneous control flow.
- $\texttt{t.P}_{\texttt{Seq}} := \{\texttt{p}|\exists \texttt{c} \in \texttt{p.SeqCFs} : \texttt{c.Succ} = \texttt{t}\} \subset \texttt{Tasks}$ denotes the set of direct predecessors of the task which are connected by a sequential control flow.
- $\texttt{t.EST}, \texttt{t.EET} \in \texttt{Dates}$ are additional constraint dates for a task which are only used during resource-constrained scheduling when backtracking is required due to standard and simultaneous control flows. The date $\texttt{t.EST}$ is the earliest planned start time, which is used to move a task to a later date. The date $\texttt{t.EET}$ is the earliest planned end time, which is used to prolong a running tasks, so that it does not end before the specified date.

The function $\texttt{f}(\texttt{a}, \texttt{d})$ returns for a task assignment $\texttt{a} \in \texttt{TaskAssignments}$ and a date $\texttt{d} \in \texttt{Dates}$ the available workload for this task assignment, which is either the available workload of the assigned resource or the maximal available workload of any resource which can play the required role if no resource is assigned.

$$\texttt{f}(\texttt{a}, \texttt{d}) = \begin{cases} \texttt{r.WCal.AWL}(\texttt{d}), & \texttt{r} = \texttt{a.Resource} \\ \texttt{max}\{\texttt{r.WCal.AWL}(\texttt{d})|\texttt{r} \in \texttt{Resources} \cap \texttt{a.Role}\}, & \texttt{undef}(\texttt{a.Resource}) \end{cases}$$

7.3.1 Initialization

Priority list The parallel scheduling scheme requires that the tasks to be scheduled are ordered in a priority list. When two tasks which compete for resources can be scheduled at the same time, then the task with higher priority is preferred. The results of critical path analysis are used to compute the priority list.

If one of the tasks is running while the other is still preparing, the running task is preferred. If both tasks are running or both tasks are still preparing, the one with the earlier latest possible start time has the higher priority. If the latest possible start times of the tasks are equal, other criteria have to be considered. The third criterion is the total float of the tasks. The task with less total float has the higher priority because it is more critical. If the total floats are equal, the task with higher total workload is preferred. In the case that all five criteria lead to an equal priority of the tasks, a manually specified value for the scheduling priority is evaluated for both tasks. If these values are not defined or equal, the task to be scheduled first is selected randomly. Altogether, the priority of of two tasks is decided based on the following criteria which are evaluated in the given order. If no decision can be made based on one criterion, the next criterion is used.

1. Execution state
2. Latest possible start time
3. Total float
4. Total workload
5. Manually set scheduling priority
6. At random

Parameters Besides the computed priority list, the heuristic scheduling algorithm requires additional input parameters. The user has to provide a *start date* for scheduling. This is the earliest date for which preparing tasks will be scheduled. The start date has to be later or equal to the current date which is at the same time the default value. In the descriptions of the algorithm, the date `start` $\in$ `Dates` is the start date provided by the user.

Critical path analysis is always performed for the whole dynamic task net. All tasks in the project are taken into account when it comes to time constraints which they impose on the (re)scheduled tasks. Resource-constrained scheduling can be restricted to a subprocess. The *root task* of the subnet to be scheduled can be specified by the user, which is by default the root of the whole dynamic task net. Only this root task and its descendants will be (re)scheduled by the heuristic algorithm. Other tasks are not (re)scheduled but may impose constraints on the planned dates of the scheduled tasks.

Planned start and end dates The first step of the algorithm is to initialize the planned start and end dates and the workload distributions of the specified root task and all its descendants.

- The planned start and end times of scheduled tasks, i.e. descendants of the root task which are in the execution state *InDefinition*, are reset so that their values are undefined. The workload distributions of these tasks and their task assignments are reset as well.

- For partially scheduled tasks, i.e. tasks in one of the execution states *Active* or *Replanning*, only the planned end times are reset. The workload distributions are retained but may be overwritten during scheduling.

All not scheduled tasks keep their possibly defined planned start and end times as well as their workload distributions. Their constraint dates and planned dates constrain the respective dates of the scheduled tasks, and their planned workload distributions impose resource constraints on the scheduled tasks.

7.3.2 Task Durations

During resource-constrained scheduling, resources are assigned to the task assignments of a task, and the workload of these task assignments is distributed over several work days.

The first step in scheduling a single task is the assignment of resources to the task assignments. Manually assigned resources are not reassigned during scheduling. The replacement of a resource has to be done manually by the user. The user can e.g. delete an assigned resource and leave the task assignment unassigned before automatic scheduling.

For task assignments without assigned resources, eligible resources are automatically selected. One of the resources who can play the required role is selected automatically. If there are several eligible resources, the resource which will complete the task in the shortest amount of time is selected, i.e. the resource which has the most free working hours. Differences between resources with respect to skill level and performance are not considered.

At this point, the implemented approach could be extended by other resource selection criteria like qualification, past performance, or cost. However, this has not been investigated further in the context of this thesis. The selection of the resource with the most available workload at the following work days aims at two optimization goals at the same time: The duration of the task shall be minimized, and the resources shall be equally used for task assignments.

Work calendars define the available work days for the distribution of workload during scheduling. Every resource and every task has an individual work calendar (cf. Section 5.3.1).

- The work calendar of a task is used when the unassigned total workload is distributed over the duration of the task.
- The work calendar of a resource is used to distribute the workload of its task assignments.

The scheduling of tasks is directly performed on the work calendars of the tasks and resources. The duration of a task or task assignment depends on the respective calendar. The actual availabilities of the resources at the different dates are taken into account when the duration of a task is determined. In contrast, most scheduling approaches found in literature schedule tasks on a generic time scale and afterwards transform the computed times to dates in a calendar [DH02, Zha92]. However, this approach is not feasible, when different calendars are used for different tasks and resources.

After the resource for a task assignment has been selected, the planned workload is distributed. For every work day in the work calendar of the assigned resource, as many working hours as possible are planned which has to be less than the available workload of the resource for the day and less than the maximal daily workload defined for the task assignment. The distributed workload is added to the used workload of the resource's work calendar (cf. Section 5.3.1). The used working hours of the resource are not available anymore for other tasks.

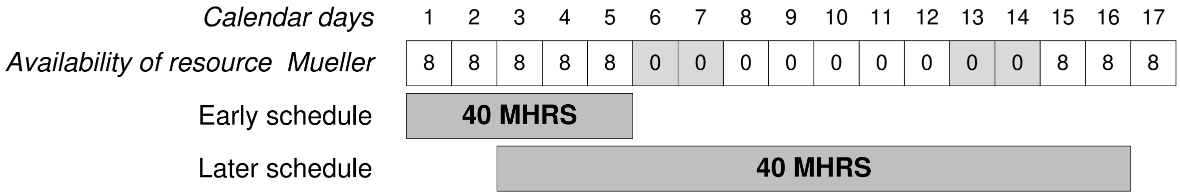


Abbildung 7.10: Example for duration variability.

During scheduling, the full amount of available working hours defined by the work calendar of a resource is used. This default strategy can be adapted by the project manager by defining a general utilization ratio of less than 100%. For example, for 8 available working hours per day and a utilization ratio of 75% only 6 hours are scheduled for task assignments. This way, more realistic schedules can be generated which take into account, that a resource cannot work full time on technical tasks every day, since they need time for administrative work like telephone calls, meetings, etc. However, the utilization ratio is not defined for each resource's work calendar individually but only for the whole project.

The main advantage of scheduling the workload of task assignments on a daily basis is that realistic task durations are obtained which respect the actual resource availabilities in the project. When a resource is assigned which has less free working hours in a certain time frame, then the task will take more work days. If a resource with more available working hours is assigned, this can significantly speed up the task, i.e. decrease its total duration. The consequence of the dynamic calculation of a task's total duration based on the required workload is that the duration of the task may vary depending on the planned start date. Depending on the availability of the assigned resource, the difference can be significant. Figure 7.10 shows an example in which the duration of a task in calendar days varies from 5 to 14 days because the assigned resource Mueller has a vacation week defined in his or her work calendar. This example illustrates the advantage of the approach. A project manager can easily analyze the consequences of vacation times of project team members on the planned end times of their assigned tasks by specifying the unavailabilities in the work calendars and rescheduling the tasks.

The example in Figure 7.11 shows the task Specify Heat Exchanger with a total workload of 294 MHRS which is partly used for two task assignments. The task assignments have been scheduled whereby the maximal daily workload of 6 MHRS for the assignment of resource Bach has been respected. Scheduling of the task assignments resulted in the planned end date 21/05/2010. With respect to the work calendar of the task, which defines the 13th of May as a holiday but the 14th of May as a work day, the resulting total duration of the task is 9 work days.

Algorithm 7.7 shows the method `ScheduleTask` which schedules the workload of the task assignments of task `t` ∈ `Tasks` and sets the the total duration and the planned end time of the task correctly with respect to the task assignments, manually specified total duration, earliest possible start time, and planned end times of

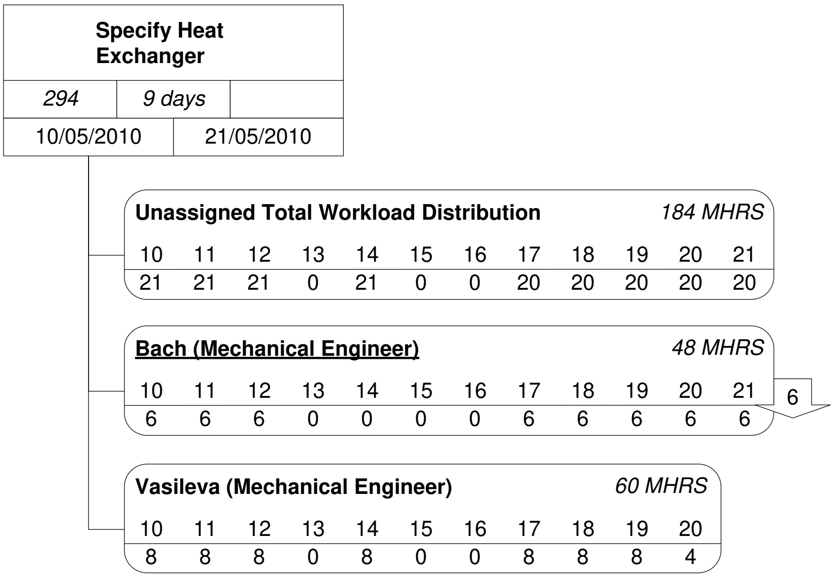


Abbildung 7.11: Example for distributed workload of a task and its task assignments.

Algorithm 7.7 ScheduleTask(t)

```
$\mathtt{t.PlannedEndTime} := \mathtt{ScheduleTaskAssignments(t)}$
if $\mathtt{t.PlannedStartTime} + \mathtt{t.TotalDuration} > \mathtt{t.PlannedEndTime}$ then
  $\mathtt{t.PlannedEndTime} := \mathtt{t.PlannedStartTime} + \mathtt{t.TotalDuration}$
end if
if $\mathtt{t.State} \neq \mathtt{InDefinition}$ then
  if $\mathtt{t.PlannedEndTime} < \mathtt{t.EPET}$ then
    $\mathtt{t.PlannedEndTime} := \mathtt{t.EPET}$
  end if
  if $\neg\mathtt{undef(t.EET)} \wedge \mathtt{t.PlannedEndTime} < \mathtt{t.EET}$ then
    $\mathtt{t.PlannedEndTime} := \mathtt{t.EET}$
  end if
end if
for all $\mathtt{s} \in \mathtt{t.Subtasks}$ do
  if $\neg\mathtt{undef(s.PlannedEndTime)} \wedge \mathtt{s.PlannedEndTime} > \mathtt{t.PlannedEndTime}$ then
    $\mathtt{t.PlannedEndTime} := \mathtt{s.PlannedEndTime}$
  end if
end for
$\mathtt{t.TotalDuration} := \mathtt{t.PlannedEndTime} - \mathtt{t.PlannedStartTime}$
$\mathtt{DistributeAvailableWorkload(t)}$
```

predecessors. The method `ScheduleTask` is invoked during execution of the parallel schedule generation scheme which will be described in the following section.

If the workload distribution of the task assignments of a task requires more work days than are available according to a manually specified total duration of the task, then the total duration is increased accordingly by setting the planned end time to the value determined by the durations of the task assignments. This prolongation of the task may cause scheduling failure due to explicitly set deadlines or successor tasks which cannot be rescheduled. To obtain a time and resource feasible schedule, it would be necessary to schedule more working hours per day for certain task assignments which may even exceed the available workload of the assigned resources. This problem can be resolved by manually increasing the total workload for several workdays in the work calendars of the assigned resources and by increasing the maximal daily workload of the task assignments. In this way, overtime work can be planned manually by the user. Overtime work is never planned automatically by the scheduling algorithm, i.e. the algorithm does not plan more working hours per day for a resource than are available in his work calendar.

The planned end time derived from scheduling the task assignments can be inconsistent with the earliest possible end time or the planned end times of simultaneous or standard predecessors. If the planned end time of the task is too early compared to the EPET or planned end times of predecessors, then the task has been scheduled to early, or in other words, the duration of the task is too short for the planned start time. If the task is preparing, it is scheduled for a later start time as explained in the next section. However, if the task is running, then its planned start time cannot be moved. Therefore, its total duration has to be increased, so that the resulting planned end time is consistent with the earliest possible end time and the planned end times of all predecessors. The latter is achieved by setting the `EET` during-resource constrained scheduling to a consistent date.

Besides the task assignments, also the subtasks of a complex task may influence its total duration. The planned end time and the total duration of a complex task may have to be adapted to the collective duration of the scheduled subtasks at a later point in time. When the resource-constrained scheduling of the subtasks results in a duration of the subprocess which is longer than the total duration of the parent task, then the total duration is increased as well. In the method `ScheduleTask`, all not scheduled subtasks are considered which have been scheduled before and therefore have a defined planned end time. However, the scheduled subtasks will be handled after the parent task. Their planned end times may lead to an increased planned end time of the complex task. This is done during the execution of the parallel scheduling scheme which is described in the following section.

Altogether the total duration which is computed during resource-constrained scheduling is the maximum of the manually specified total duration, the duration of the scheduled task assignments, and the duration of the scheduled subprocess which is defined by the subtasks. Furthermore, the total duration of a running task is long enough for the planned end time to be consistent with the earliest possible end time and the planned end times of all predecessors.

When the duration of a task has been finally determined, the unassigned total workload of the task is equally distributed over all work days of the task according to the task's work calendar. First, the unassigned workload w_u is distributed over the work days $d_1, \ldots, d_n$ of the task so that every work day receives the same amount of $\lfloor w_u / n \rfloor$ man hours. The rest of the workload $(w_u \; mod \; u)$ is distributed with 1 MHR per work day starting from the first work day. The distribution of the unassigned total workload is required to compute an accurate planned value for earned value analysis.

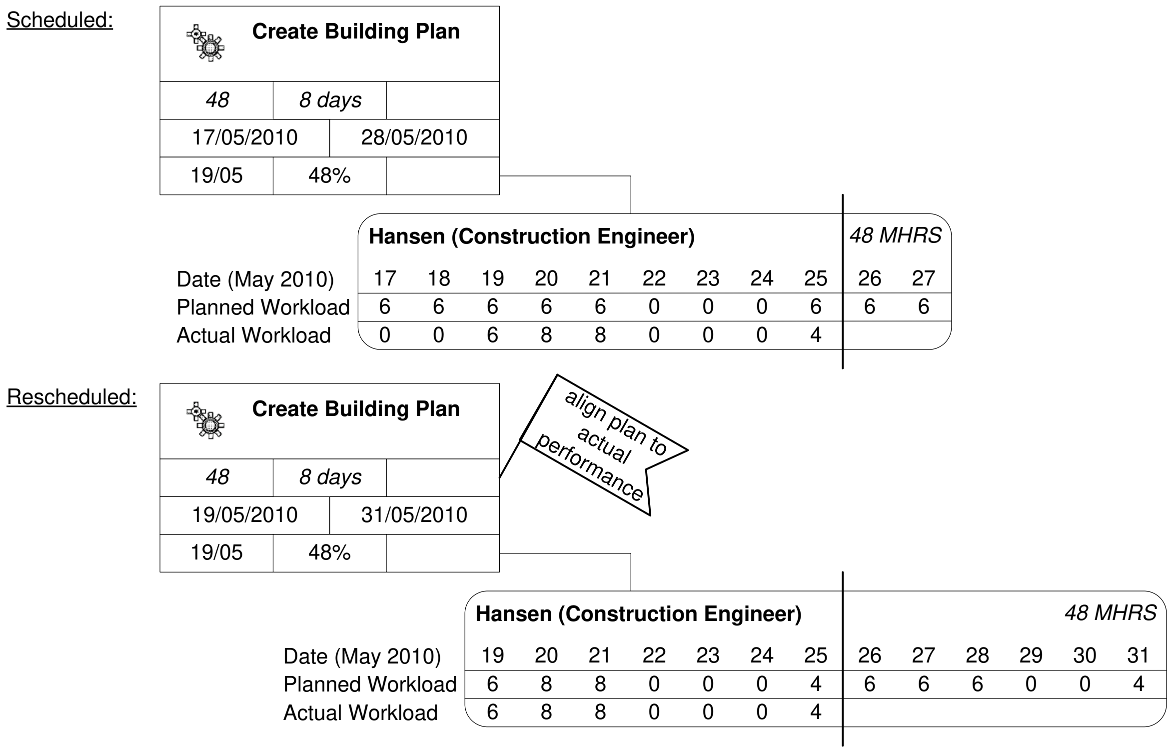
In the example of Figure 7.11, the unassigned total workload of the task has been equally distributed over the 9 work days of the task's work calendar. The workload distribution is visualized like a task assignment in Figure 7.11, however, no task assignment is created to store the workload values. They are stored in a separate workload distribution of the task.

The total duration of a task may exceed the duration of the scheduled task assignments and in particular the duration of the assignment of the responsible resource. The scheduled subtasks together may require more time than the task assignments. The duration may have been increased to achieve consistency with the earliest possible end time and the planned end times of the predecessors. A common approach proposed in the related work is to split a task if its planned end date is constrained in this way. This would be a solution for the problem of meeting an earliest possible end time. However, splitting a task is not an option if it has subtasks. The task needs to be active for the whole duration of the subtasks. In PROCEED, a practical solution has been chosen. The scheduler warns the user about tasks for which the task assignment of the responsible resource is shorter than the total duration of the respective task. The user can then increase the planned workload for the responsible resource or decrease the maximal daily workload of the task assignment, and another scheduling pass will resolve the issue.

If no task assignments and subtasks are defined for a task and the total duration has not been specified, then it cannot be determined. It is not possible to derive the total duration from the total workload since it is not known how much workload should be scheduled per day. Therefore, these tasks have been defined as zero-duration tasks in Section 7.1, although a non-zero workload may be specified.

Tasks which are in one of the execution states *Active* or *Replanning* are always scheduled at their planned start time. If the actual performance of a running task deviates from the plan, there are two possibilities for scheduling. Either the plan is not changed and the task is scheduled as before or the plan is aligned to the actual performance. To align the planned dates of a running task to the actual dates, the planned start time has to be set to the actual start time manually. However, this alone does not ensure that the planned workload distributions of the task assignments are aligned to the actual workload distributions. Therefore, a flag can be set for a task which indicates, that the planned values should be aligned to the actual values.

PROCEED offers the operation *align plan to actual performance* to the user which can be executed for a running task. This operation sets the planned start time to the actual start time and sets the aforementioned flag to `true`. If the flag is set, then for



Date (May 2010)	17	18	19	20	21	22	23	24	25	26	27
Planned Workload	6	6	6	6	6	0	0	0	6	6	6
Actual Workload	0	0	6	8	8	0	0	0	4		

Date (May 2010)	19	20	21	22	23	24	25	26	27	28	29	30	31
Planned Workload	6	8	8	0	0	0	4	6	6	6	0	0	4
Actual Workload	6	8	8	0	0	0	4						

Abbildung 7.12: Example for aligning planned values to actual values.

every day for which actual workload is defined, the actual values are used to compute the new planned workload distributions. The remaining planned workload of a task assignment is distributed according to the work calendar of the assigned resource. Besides the planned start time, the operation *align plan to actual performance* also adapts the total duration and workload of the task. The total duration is set to the number of working days from the actual start time to the forecasted end time which is obtained from progress measurement and earned value analysis. If the progress of the task is measured by estimating the remaining workload until completion (cf. Section 8.1), then the total workload is set to the sum of actual workload and estimated remaining workload.

Figure 7.12 shows an example where an active task is rescheduled at runtime and the user has decided to align the plan to the actual performance. Therefore, the planned start time has been set to the actual start time of the task. The planned workload of the only task assignment is distributed like the actual workload. Because only 26 man hours have been spent on the task assignment instead of the planned 36 man hours, the remaining workload of 22 MHRS has to be distributed over the following working days. In this example, the total duration of the task does not change because the plan is adapted to the late start. However, if the actual workload per day would be smaller, the planned duration would increase during rescheduling. Furthermore, an increase of the planned workload, e.g. due to the estimation of remaining workload, would lead to an increased duration.

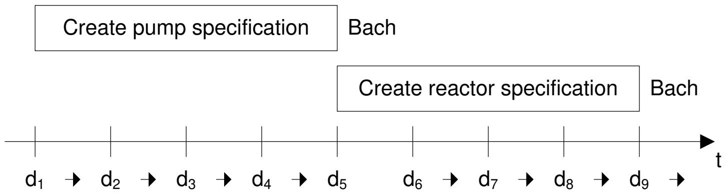


Abbildung 7.13: Illustration of time increments of parallel heuristic.

7.3.3 Parallel Scheduling Scheme

The heuristic for resource-constrained scheduling is based on the general parallel schedule generation scheme presented in [KH98, DH02, Kle00]. Several adaptations of this parallel heuristic were required which will be discussed in Section 7.7.

The heuristic iterates over time as illustrated in Figure 7.13. After every iteration step, the current date $\mathtt{d} \in \mathtt{Dates}$ is increased by one day. In every iteration step, eligible tasks which fulfill all control flow and resource constraints are scheduled. In the example in Figure 7.13, the two tasks are scheduled in sequence because they both require the same resource.

The algorithm operates on several sets which contain the tasks to be scheduled and which are updated in each iteration step. The set $\mathtt{T} \subset \mathtt{Tasks}$ contains all tasks in the project except for the zero-duration tasks which have been removed from the memory representation of the task net before critical path analysis. Critical path analysis is performed for all tasks in $\mathtt{T}$ as described in the previous section. The set $\mathtt{S} \subset \mathtt{T}$ contains the scheduled tasks.

$$\begin{aligned}\mathtt{S} := \{\mathtt{t} \in \mathtt{T} | \; & \mathtt{t.State} \in \{\mathtt{InDefinition}, \mathtt{Active}, \mathtt{Replanning}\} \wedge \\ & (\mathtt{t} = \mathtt{root} \vee (\mathtt{root} \in \mathtt{t.Ancestors} \wedge \\ & \forall \mathtt{a} \in \mathtt{t.Ancestors}(\mathtt{a} = \mathtt{root} \vee \mathtt{root} \in \mathtt{a.Ancestors} \\ & \Rightarrow \mathtt{a.State} \in \{\mathtt{InDefinition}, \mathtt{Replanning}\})))\} \subset \mathtt{T}\end{aligned}$$

This excludes all tasks which are not in the subnet of the specified root task and all not scheduled tasks which are excluded from scheduling due to their execution states. Only tasks in $\mathtt{S}$ are (re)scheduled by the heuristic.

The elements of the following sets are changed at defined points in the algorithm by evaluating the following mathematical definitions. The formulas which define the sets may not be valid for the sets between the explicit updates. The set $\mathtt{A}_\mathtt{d} \subset \mathtt{T}$ is the *active set* which contains all tasks which have already been scheduled and which are active at the date $\mathtt{d}$ according to the schedule.

$$\begin{aligned}\mathtt{A}_\mathtt{d} = \{\mathtt{t} \in \mathtt{T} | \; & \mathtt{t.PlannedStartTime} \leq \mathtt{d} \wedge \neg \mathtt{undef}(\mathtt{t.PlannedEndTime}) \wedge \\ & (\neg \mathtt{endTimeFinal}(\mathtt{t}) \vee \mathtt{d} \leq \mathtt{t.PlannedEndTime})\} \subset \mathtt{T}\end{aligned}$$

The active set does not consist of the tasks which are in execution state *Active*, although the name of the set might suggest it. The actual start and end times are not considered for determining the elements of the active set.

The formula $\mathtt{endTimeFinal(t)}$ is defined as follows for a task $\mathtt{t} \in \mathtt{Tasks}$.

$$\begin{aligned}\mathtt{endTimeFinal(t)} \equiv &\neg\mathtt{undef(t.PlannedEndTime)} \wedge \\ &\forall \mathtt{s} \in \mathtt{t.Subtasks(endTimeFinal(s))}\end{aligned}$$

For an atomic task, it returns `true`, if the task has been scheduled before and a planned end time has been computed. For a complex task, it only returns `true`, if the task itself has already been scheduled as well as all descendants in the hierarchy of the dynamic task net, so that the planned end time of the task will not change anymore in the current scheduling pass.

The set $\mathtt{C_d} \subset \mathtt{T}$ is the *complete set* which contains all tasks which have been scheduled and are terminated at date $\mathtt{d}$ according to the schedule.

$$\mathtt{C_d} = \{\mathtt{t} \in \mathtt{T} | \mathtt{endTimeFinal(t)} \wedge \mathtt{t.PlannedEndTime} < \mathtt{d}\} \subset \mathtt{T}$$

The set $\mathtt{F_d} \subset \mathtt{S}$ is the *forbidden set* which contains atomic tasks which could be started at date $\mathtt{d} \in \mathtt{Dates}$ according to control flow and resource constraints, but whose end dates would be inconsistent with the end dates of standard or simultaneous predecessors. When the planned end date of a task $\mathtt{t} \in \mathtt{S}$ has been calculated, and the following condition is satisfied, then the task is added to the forbidden set.

$$\begin{aligned}\mathtt{FS(t)} \equiv\ &\mathtt{t.PlannedEndTime} < \mathtt{t.EPET} \vee \\ &(\neg\mathtt{undef(t.EET)} \wedge \mathtt{t.PlannedEndTime} < \mathtt{t.EET}) \vee \\ &\exists \mathtt{c}(\mathtt{c} \in \mathtt{t.IStdCFs} \cup \mathtt{t.ISimCFs} \wedge \mathtt{endTimeFinal(c.Pred)} \wedge \\ &\mathtt{c.Pred.PlannedEndTime} + \mathtt{c.LagTime} > \mathtt{t.PlannedEndTime}) \vee \\ &\exists \mathtt{f} \in \mathtt{t.ActiveFeedbacks(endTimeFinal(f.Target)} \wedge \\ &\mathtt{t.PlannedEndTime} < \mathtt{f.Target.PlannedEndTime})\end{aligned}$$

The planned end time of a task may not be earlier than its earliest possible end time. Furthermore, the planned end time of a task has to be consistent with the planned end times of all predecessors as well as all targets of outgoing active feedback flows. If the planned end time of a predecessor is still undefined or may change later due to scheduling its subtasks, then the constraint on the planned end time cannot be checked yet. The formula $\mathtt{FS(t)}$ returns false, no action is taken, and the consistency is checked at the end of the scheduling pass which may lead to backtracking.

The *eligible set* $\mathtt{E_d} \subset \mathtt{S}$ contains all tasks which can be scheduled at the current date $\mathtt{d} \in \mathtt{Dates}$. It is defined as follows.

$$\begin{aligned}\mathtt{E_d} = &\{\mathtt{t} \in \mathtt{S} \setminus (\mathtt{C_d} \cup \mathtt{A_d} \cup \mathtt{F_d}) | \\ &\forall \mathtt{c} \in \mathtt{t.ISeqCFs(endTimeFinal(c.Pred)} \wedge \mathtt{c.Pred} \in \mathtt{C_{d-c.LagTime}}) \wedge \\ &\forall \mathtt{c} \in \mathtt{t.ISimCFs(c.Pred} \in \mathtt{A_{d-c.LagTime}} \cup \mathtt{C_{d-c.LagTime}}) \wedge \mathtt{t.Parent} \in \mathtt{A_d} \wedge \\ &\mathtt{d} \geq \mathtt{t.EPST} \wedge (\mathtt{t.State} \neq \mathtt{InDefinition} \vee \mathtt{d} \geq \mathtt{start}) \wedge \\ &(\mathtt{undef(t.EST)} \vee \mathtt{d} \geq \mathtt{t.EST}) \wedge \\ &((\mathtt{t.State} \neq \mathtt{Active} \wedge \mathtt{t.State} \neq \mathtt{Replanning}) \vee \mathtt{d} = \mathtt{t.PlannedStartTime}) \wedge \\ &\forall \mathtt{a} \in \mathtt{t.TaskAssignments(f(a,d)} > \mathtt{0})\}\end{aligned}$$

A task $\mathtt{t}$ is in the eligible set, when all of its sequential predecessors are terminated and all simultaneous predecessors are active or terminated according to the schedule. Furthermore, the current date $\mathtt{d}$ has to be later or equal to the computed earliest possible start time of the task. If the task is in the execution state *InDefinition*, the current date $\mathtt{d}$ has to be equal or later than the specified start date for resource-constrained scheduling $\mathtt{start} \in \mathtt{Dates}$. In this way, the preparing tasks in state *InDefinition* are moved to planned start dates which are later or equal to the specified start date for scheduling. In contrast, running tasks are scheduled for their defined planned start dates, and only their duration may change. If the earliest planned start time $\mathtt{EST}$ of the task has been set during backtracking because a previously computed planned end time was inconsistent with the end time of a standard or simultaneous predecessor, then the task is only included in the eligible set, if the date $\mathtt{EST}$ has been reached. Finally, to schedule a task at the current date, workload has to be available for its task assignments. Workload is always available for running tasks at their planned start time, since they have been scheduled before for this date.

The subformula $\forall \mathtt{a} \in \mathtt{t.TaskAssignments}(\mathtt{f}(\mathtt{a},\mathtt{d}) > 0)$ does not require, that the resources are available for the whole working day and it does not exclude the case that two task assignments require the same resource. This is the case, when several task assignments require the same role but there are less resources who can play the role available than there are task assignments defined. Also in the case that two task assignments require different roles but there is only one resource available who can play both roles, the task assignments require the same resource. In these cases, the task assignments are scheduled in sequence, or in parallel if maximal daily workload is defined for the task assignments.

When an eligible task has been scheduled, its duration and planned end time have been determined. The calculated planned end time can be inconsistent with the start or end times of not scheduled tasks. For this reason, the following condition has to be checked after scheduling a task $\mathtt{t} \in \mathtt{S}$.

$$
\begin{aligned}
\mathtt{IE}(\mathtt{t}) \equiv\; & \mathtt{t.PlannedEndTime} > \mathtt{t.LPET} \lor \\
& \exists \mathtt{c} \in \mathtt{t.ControlFlows}((\mathtt{c.Succ} \notin \mathtt{S} \land \\
& (((\mathtt{c.Semantics} = \mathtt{Standard} \lor \mathtt{c.Semantics} = \mathtt{Simultaneous}) \land \\
& \neg\mathtt{undef}(\mathtt{c.Succ.PlannedEndTime}) \land \\
& \mathtt{t.PlannedEndTime} > \mathtt{c.Succ.PlannedEndTime} - \mathtt{c.LagTime}) \lor \\
& (\mathtt{c.Semantics} = \mathtt{Sequential} \land \\
& \neg\mathtt{undef}(\mathtt{c.Succ.PlannedStartTime}) \land \\
& \mathtt{t.PlannedEndTime} > \mathtt{c.Succ.PlannedStartTime} - \mathtt{c.LagTime}))) \lor \\
& (\mathtt{c.Succ} \in \mathtt{S} \land \mathtt{c.Succ.State} \neq \mathtt{InDefinition} \land \\
& (\mathtt{c.Semantics} = \mathtt{Sequential} \land \\
& \neg\mathtt{undef}(\mathtt{c.Succ.PlannedStartTime}) \land \\
& \mathtt{t.PlannedEndTime} > \mathtt{c.Succ.PlannedStartTime} - \mathtt{c.LagTime})))
\end{aligned}
$$

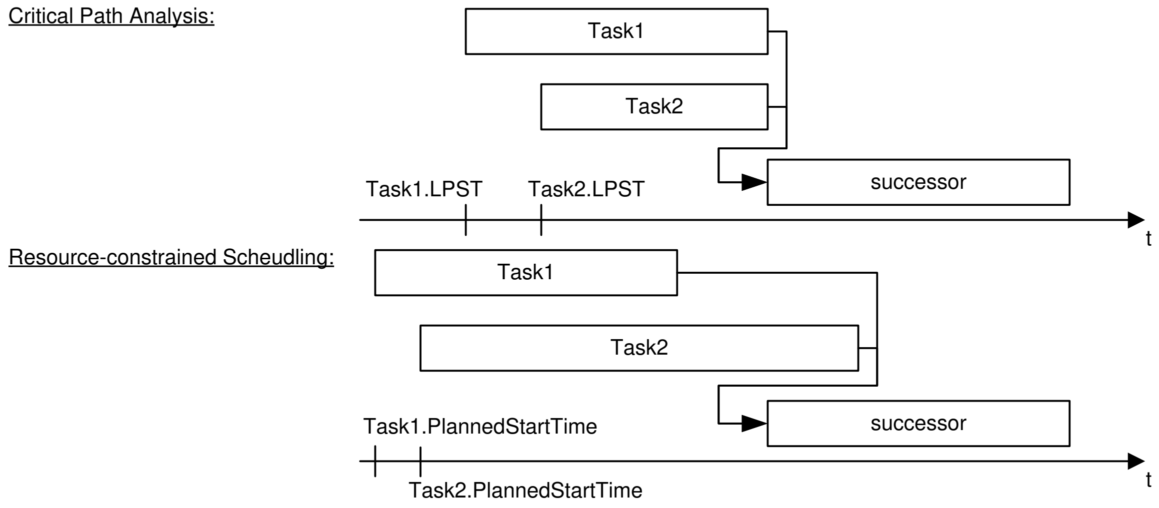


Abbildung 7.14: Example for inconsistent end time with not scheduled successor.

The formula checks whether the planned end time of the task is later than the latest possible end time. Furthermore, it checks whether the planned end time is later than the planned end time of a standard or simultaneous successor or the planned start time of a sequential successor which is not (re)scheduled or only partially (re)scheduled. If the formula $\mathtt{IE}(\mathtt{t})$ evaluates to true, scheduling is aborted because the end date of task `t` is inconsistent. In this case it is not possible to generate a time and resource feasible schedule given the time constraints, the planned dates of the not scheduled tasks, and the priority list. The priority list based on the latest possible start times of the tasks has been chosen to avoid these inconsistencies. Among two tasks, the one with the earlier LPST is more constrained by succeeding tasks and has therefore the higher priority. However, the durations of the tasks which are determined during resource-constrained scheduling can be significantly longer than the durations used for critical path analysis. Therefore, even a task with lower priority may cause inconsistencies although a task with higher priority does not. This is illustrated in Figure 7.14 where Task1 and Task2 have a common sequential successor and the duration of Task1 is longer than the duration of Task2 for critical path analysis. However, during resource-constrained scheduling, the duration of Task2 increases due to resource availabilities which is why its end date is inconsistent with the planned start date of the not scheduled successor.

Backtracking and rescheduling with a higher priority for the conflicting task could resolve some of these cases, but the problem would not be solved in general. The conflicting task may still have an inconsistent end date even when it is scheduled for an earlier start date. Therefore, scheduling is aborted in these cases, and the user is informed about the inconsistency. There are several actions which the user can perform to enable a successful scheduling run. He can for example change the execution state of a waiting successor back to *InDefintion*, so that its planned start time can be adapted during scheduling. Alternatively, he can change the semantics of the connecting control flow or remove the control flow completely.

The method `Schedule` which is shown in Algorithm 7.8 implements the adapted parallel schedule generation scheme. In lines 1 to 5, the start date $\mathtt{d} \in \mathtt{Dates}$ for the scheduling algorithm is set. The task $\mathtt{project} \in \mathtt{Tasks}$ is the root node of the complete dynamic task net. If this task is running, i.e. the project has already been started, the parallel heuristic has to start at the planned project start date in order to cover all active and running tasks which may not be moved but whose workload may be redistributed. If the project has not been started yet, i.e. all tasks of the dynamic task net are still preparing, then the parallel heuristic starts at the manually specified start date. In both cases, tasks which are in the execution state *InDefinition* are not scheduled earlier than the manually specified start date for scheduling.

The algorithm iterates the main `while` loop until all tasks in `S` have been scheduled. If it is not possible to schedule all tasks in `S` so that he planned dates of all tasks in `T` are consistent, then the `while` loop terminates when the date $\mathtt{end} \in \mathtt{Dates}$ is reached. The date `end` is set to the due date of the project, if the latter is defined. Otherwise it is set to a sufficiently late date, so that all tasks of the project can be scheduled in the resulting time frame, e.g. 4 years for a plant design project which is sufficient in most cases.

The first step of every iteration is the update of the complete set, active set, and eligible set. If there are eligible tasks for the current date, the task with the highest priority is selected for scheduling. If the current date is already later than the latest possible start time of the task, scheduling is aborted. Otherwise, the planned start time of the selected task is set to the current date.

The method `ScheduleTask` assigns resources to the task assignments of the selected task and distributes the workload of the task assignments according to the work calendars of the resources as described in the previous section. The distributed workload is added to the used workload of the resources' work calendars. The used working hours of the resources are not available anymore for other tasks. Therefore, the eligible set has to be updated after every iteration in which one of the eligible tasks is scheduled (line 34 of Algorithm 7.8).

The computed planned end date of the scheduled task can be inconsistent with the planned dates of not scheduled successors. This is checked by evaluating the formula $\mathtt{IE}(\mathtt{t})$ for the scheduled task `t`. If the formula evaluates to true, then the computed planned end time is inconsistent with the planned end time of a standard or simultaneous successor or with the planned start time of a sequential successor. In this case, scheduling is aborted and the user is informed about the reason for the failure.

Inconsistencies with planned end dates of scheduled standard successors do not require to abort scheduling completely. These inconsistencies are detected at the end of the full scheduling pass and lead to backtracking as described below. Inconsistencies with planned start dates of scheduled simultaneous or sequential successors cannot occur at this point because these successors will be scheduled later.

When a task is scheduled at its earliest possible start time or later, it is not gua-

Algorithm 7.8 Schedule(start)

```
if $\texttt{project.State} \in \texttt{Running}$ then
  $\texttt{d} := \texttt{project.PlannedStartTime}$
else
  $\texttt{d} := \texttt{start}$
end if
while $\texttt{S} \not\subseteq \texttt{C}_\texttt{d} \cup \texttt{A}_\texttt{d} \wedge \texttt{d} < \texttt{end}$ do
  update $\texttt{C}_\texttt{d}$, $\texttt{A}_\texttt{d}$, $\texttt{E}_\texttt{d}$
  while $|\texttt{E}_\texttt{d}| > 0$ do
    choose $\texttt{t} \in \texttt{E}_\texttt{d}$ with highest priority
    if $\texttt{d} > \texttt{t.LPST}$ then
      abort scheduling
    end if
    $\texttt{t.PlannedStartTime} := \texttt{d}$
    $\texttt{ScheduleTask}(\texttt{t})$
    if $\texttt{IE}(\texttt{t})$ then
      abort scheduling
    else if $\texttt{t.Subtasks} = \emptyset \wedge \texttt{FS}(\texttt{t})$ then
      $\texttt{UnscheduleTask}(\texttt{t})$
      if $\texttt{t.State} = \texttt{InDefinition}$ then
        $\texttt{F}_\texttt{d} := \texttt{F}_\texttt{d} \cup \{\texttt{t}\}$
      else
        $\texttt{t.EET} := \texttt{max}\{\texttt{e} | \exists \texttt{c} \in \texttt{t.IStdCFs} \cup \texttt{t.ISimCFs}$
        $(\texttt{e} = \texttt{c.Succ.PlannedEndTime} + \texttt{c.LagTime})\}$
        $\texttt{ScheduleTask}(\texttt{t})$
        if $\texttt{IE}(\texttt{t})$ then
          abort scheduling
        end if
        $\texttt{UpdateParent}(\texttt{t})$
        add task $\texttt{t}$ to the active set
      end if
    else
      $\texttt{UpdateParent}(\texttt{t})$
      add task $\texttt{t}$ to the active set
    end if
    update $\texttt{E}_\texttt{d}$
  end while
  $\texttt{d} := \texttt{d} + 1$
  $\texttt{F}_\texttt{d} := \emptyset$
end while
if $\texttt{d} = \texttt{end}$ then
  abort scheduling
end if
$\texttt{CheckEndDates}()$
```

ranteed that its planned end time is later than its earliest possible end time, because the time window between the EPST and the EPET may be longer than the task's duration (cf. Section 7.2). Furthermore, it is not guaranteed that its planned end time is consistent with the planned end times of its predecessors because the durations of the predecessors may be longer for resource-constrained scheduling than for critical path analysis. If the planned end date of a task is inconsistent, backtracking is required. While inconsistencies concerning complex tasks are handled at the end of a complete scheduling pass, backtracking is performed directly in case of atomic tasks. The resource assignments, the workload distributions, and the planned dates of an atomic task are undone. If the task is in the execution state *InDefinition*, it is scheduled at a later date, which is realized by means of the forbidden set. The unscheduled task is inserted into the forbidden set and the inner while loop proceeds with scheduling the eligible task with the next lower priority. The task which has been added to the forbidden set is scheduled at the earliest on the next date. If the atomic tasks is in one of the execution states *Active* or *Replanning*, it cannot be scheduled at a later date. Therefore, its earliest planned end time is set and the method `ScheduleTask` is invoked again. This leads to a prolongation of the task as described in the previous section. The planned start time remains unchanged but the planned end time is moved to the earliest planned end time.

The planned end date of a complex task is not final until all subtasks have been scheduled. Whenever a subtask of a complex task has been scheduled, the method `UpdateParent(t)` updates the planned end date of the parent task `t.Parent`. The pseudo code of this method is depicted in Algorithm 7.9. First, the latest planned end time of all subtasks is determined. If the parent task can be rescheduled and the computed value is later than the previously defined planned end time, then the latter is set to the former. The total duration of the parent task is adapted accordingly, and the unassigned total workload is redistributed. If the new planned end time of the parent task leads to a violation of the formula `IE(t)` which checks the consistency with the planned dates of not scheduled successors, then scheduling is aborted. Otherwise, the planned end time of the ancestors is updated recursively. If the parent task is not a (partially) scheduled task and the computed planned end time is later than an existing value, then scheduling has to be aborted as well. This may be the case, if a subprocess is rescheduled leading to a longer duration which is inconsistent with the planned dates of not scheduled tasks outside the subprocess.

At the end of every iteration of the inner `while` loop (line 34 of Algorithm 7.8), the eligible set is updated. The formula which defines the eligible set is evaluated for the current situation after the eligible task has been scheduled. The scheduling of an eligible task may lead to the removal of other tasks from the eligible set because the scheduled task uses resources which are required by the other tasks. These tasks may become members of the eligible set again when the scheduled task has released its resources. As a consequence, tasks which compete for resources are scheduled in sequence.

When the eligible set for the current date $\mathtt{d} \in \mathtt{Dates}$ is empty because all eligible tasks have been scheduled or have become non-eligible, then the current date is

Algorithm 7.9 UpdateParent(t)

```
$\hat{e} := \mathtt{max}\{\mathtt{e} \mid \exists \mathtt{s} \in \mathtt{t.Parent.Subtasks}(\neg\mathtt{undef}(\mathtt{s.PlannedEndTime}) \wedge \mathtt{e} = \mathtt{s.PlannedEndTime})\}$
if $\mathtt{t.Parent} \in \mathtt{S}$ then
  if $\mathtt{undef}(\mathtt{t.Parent.PlannedEndTime}) \vee \mathtt{t.Parent.PlannedEndTime} < \hat{e}$ then
    $\mathtt{t.Parent.PlannedEndTime} := \hat{e}$
    $\mathtt{t.Parent.TotalDuration} := \mathtt{t.Parent.PlannedEndTime} - \mathtt{t.Parent.PlannedStartTime}$
    $\mathtt{DistributeAvailableWorkload}(\mathtt{t.Parent})$
    if $\mathtt{IE}(\mathtt{t.Parent})$ then
      abort scheduling
    else
      $\mathtt{UpdateParent}(\mathtt{t.Parent})$
    end if
  end if
else
  if $\neg\mathtt{undef}(\mathtt{t.Parent.PlannedEndTime}) \wedge \mathtt{t.Parent.PlannedEndTime} < \hat{e}$ then
    abort scheduling
  end if
end if
```

increased by one day, and the forbidden set is emptied. Scheduling proceeds with the computation of the complete, active and eligible set for the new date.

After all tasks in `S` have been scheduled, the method `CheckEndDates` is invoked in line 42 of Algorithm 7.8. This method checks the consistency of all planned end dates in the task net. Inconsistencies between planned end dates can still exist for complex tasks which are sources or targets of standard or simultaneous control flows. The end date of a complex task is not ultimately fixed until all subtasks have been scheduled. Therefore, complex tasks as targets of standard and simultaneous control flows were excluded in line 17 of Algorithm 7.8. In the definition of the formula `FS(t)`, complex tasks whose planned end time is not final, i.e. which may still increase during scheduling, are not considered as sources of incoming simultaneous or standard control flows.

When all subtasks of a complex task have been scheduled, the resulting final planned end time may be inconsistent with the planned end times of simultaneous or standard predecessors or successors. This can only be detected after all subtasks of the source and target of a simultaneous or standard control flow have been scheduled. Therefore, the consistency of the planned end dates of all tasks is checked at the end of the scheduling algorithm by the method `CheckEndDates`.

Figure 7.15 shows example cases where the scheduling of subtasks led to inconsistencies. In Figure 7.15 a), the task assignments of the predecessor P and its subtask S1 have been scheduled which led to the total duration indicated by the gray bar. Scheduling the successor task S did not lead to inconsistent end dates. However, when the subtasks S2 and S3 of task P were scheduled, the duration of P

Algorithm 7.10 CheckEndDates()

```
$reschedule := false$
for all $t \in S$ do
  if $t.PlannedEndTime < t.EPET \land t.State = InDefinition$ then
    $\Delta d := t.EPET - t.PlannedEndTime$
    $t.EST := t.PlannedStartTime + \Delta d$
    $reschedule := true$
  end if
end for
for all $c \in ControlFlows \cap (Tasks \times S)$ do
  if $(c.Semantics = Standard \lor c.Semantics = Simultaneous)$ $c.Pred.PlannedEndTime + c.LagTime > c.Succ.PlannedEndTime$ then
    if $c.Succ.State = InDefinition$ then
      $\Delta d := (c.Pred.PlannedEndTime + c.LagTime) - c.Succ.PlannedEndTime$
      $c.Succ.EST := \max\{c.Succ.EST, c.Succ.PlannedStartTime + \Delta d\}$
    else
      $c.Succ.EET := \{\max\{c.Succ.EET, c.Pred.PlannedEndTime + c.LagTime\}$
    end if
    $reschedule := true$
  end if
end for
for all $f \in FeedbackFlows \cap (S \times Tasks)$ do
  if $f.IsActive = true \land f.Source.PlannedEndTime < f.Target.PlannedEndTime$ then
    $f.Source.EPET := f.Target.PlannedEndTime$
    $reschedule := true$
  end if
end for
if $reschedule = true$ then
  ResetSchedulingResult()
  Schedule(start)
end if
```

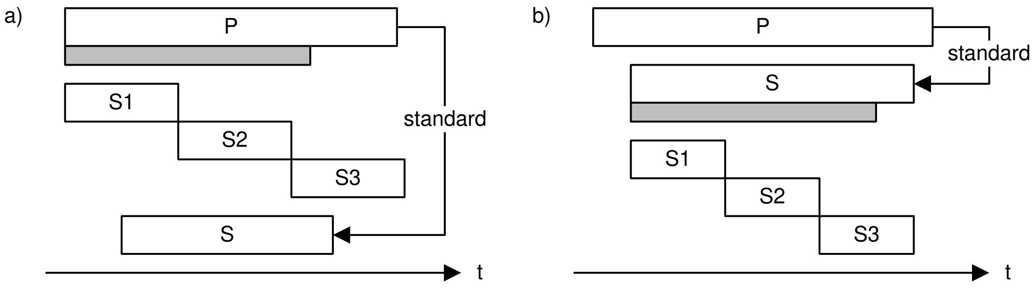


Abbildung 7.15: Examples for inconsistent end times of complex tasks.

was increased which lead to the inconsistent planned end dates. In Figure 7.15 b), the scheduling of the task assignments of task S lead to an inconsistent end date. However, the scheduling of its subtasks S1 to S3 could still resolve this inconsistency, which is why task S was not moved to a later planned start time. In the example, the inconsistency remained unresolved until the end of the scheduling pass although the duration of task S increased due to the scheduled subtasks.

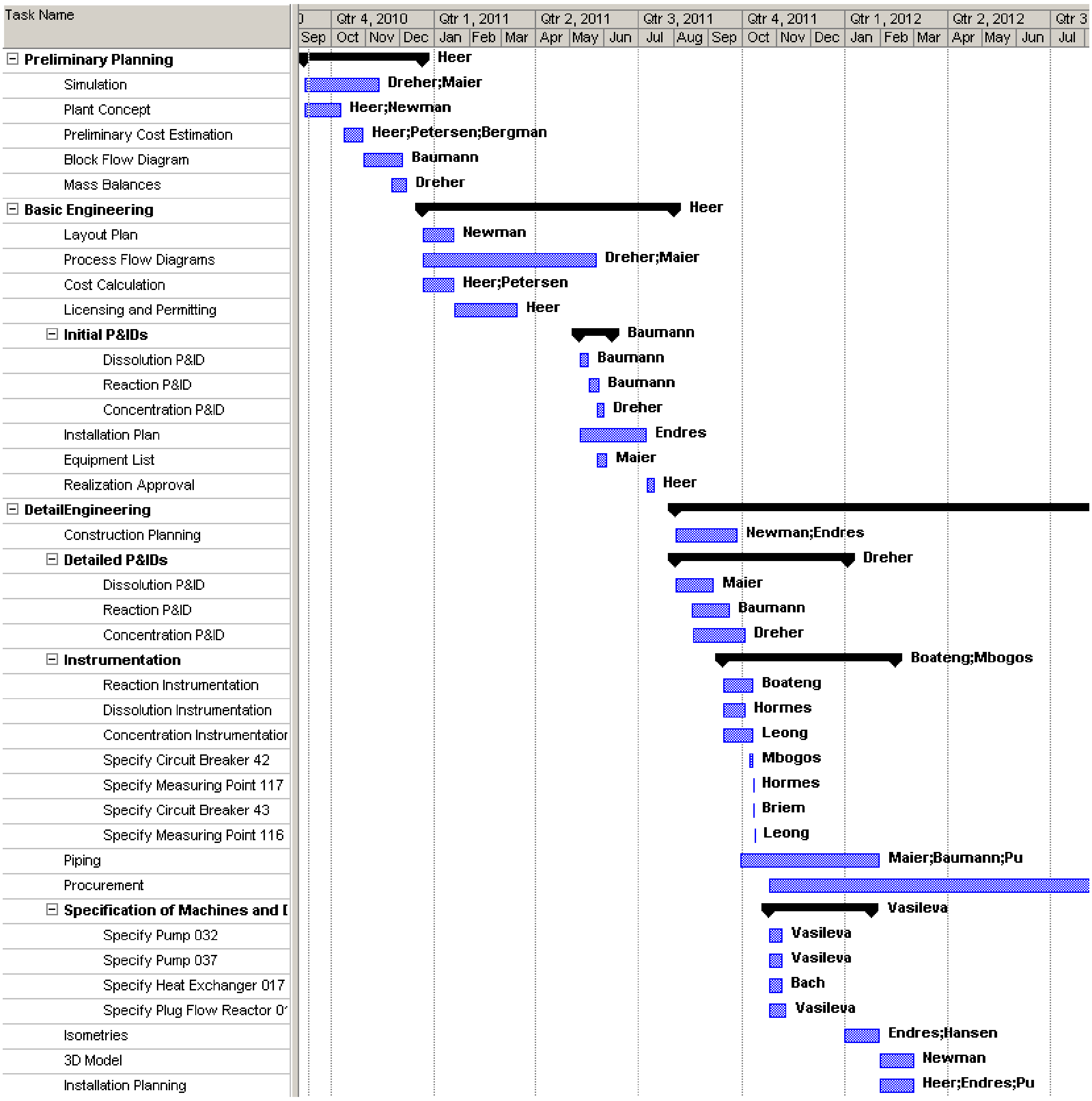
The pseudo code of the method `CheckEndDates` is presented in Algorithm 7.10. First, the planned end times of all tasks are compared to the respective earliest possible end times. For every task for which the dates are inconsistent, the earliest planned start time is set to the planned start time plus the time span $\Delta\mathtt{d} = \mathtt{t.EPET} - \mathtt{t.PlannedEndTime}$. As a consequence, the task will be scheduled $\Delta\mathtt{d}$ work days later during the next scheduling pass. Second, all standard and simultaneous control flows are checked for consistency. If the target task $\mathtt{t} \in \mathtt{S}$ of an inconsistent control flow is in the execution state *InDefinition*, then its earliest planned start time is set to the previously planned start time plus the time span

$$\Delta\mathtt{d} = \mathtt{max}\{\mathtt{c.Pred.PlannedEndTime} + \mathtt{c.LagTime} | \mathtt{c} \in \mathtt{t.IncomingCFs}\} \\ - \mathtt{t.PlannedEndTime}$$

so that the task is scheduled $\Delta\mathtt{d}$ work days later during the next scheduling pass. If the target task $\mathtt{t} \in \mathtt{S}$ is running, then its earliest planned end time is set to a value which is consistent with the planned end times of the predecessors. Third, the planned end times of all tasks are checked for consistency with respect to active feedback flows. The earliest planned end time of a source task is set to prolong the necessarily active task during the next scheduling pass. Finally, backtracking is performed if inconsistent planned end dates have been detected. All computed workload distributions and planned dates of all scheduled tasks are reset and another scheduling pass is started. Depending on their execution states, the conflicting tasks will be either scheduled later or with a longer duration.

7.3.4 Scheduling Example

The example scenario which has been introduced in Section 2.3 has been used to evaluate the correctness and applicability of the described approach for scheduling



Task Name	
Preliminary Planning	Heer
Simulation	Dreher;Maier
Plant Concept	Heer;Newman
Preliminary Cost Estimation	Heer;Petersen;Bergman
Block Flow Diagram	Baumann
Mass Balances	Dreher
Basic Engineering	Heer
Layout Plan	Newman
Process Flow Diagrams	Dreher;Maier
Cost Calculation	Heer;Petersen
Licensing and Permitting	Heer
Initial P&IDs	Baumann
Dissolution P&ID	Baumann
Reaction P&ID	Baumann
Concentration P&ID	Dreher
Installation Plan	Endres
Equipment List	Maier
Realization Approval	Heer
DetailEngineering	
Construction Planning	Newman;Endres
Detailed P&IDs	Dreher
Dissolution P&ID	Maier
Reaction P&ID	Baumann
Concentration P&ID	Dreher
Instrumentation	Boateng;Mbogos
Reaction Instrumentation	Boateng
Dissolution Instrumentation	Hormes
Concentration Instrumentatior	Leong
Specify Circuit Breaker 42	Mbogos
Specify Measuring Point 117	Hormes
Specify Circuit Breaker 43	Briem
Specify Measuring Point 116	Leong
Piping	Maier;Baumann;Pu
Procurement	
Specification of Machines and [	Vasileva
Specify Pump 032	Vasileva
Specify Pump 037	Vasileva
Specify Heat Exchanger 017	Bach
Specify Plug Flow Reactor 0	Vasileva
Isometries	Endres;Hansen
3D Model	Newman
Installation Planning	Heer;Endres;Pu

Abbildung 7.16: Gantt chart of the complete base schedule.

dynamic task nets. Figure 7.16 shows the base schedule of the example project, which results from scheduling the whole dynamic task net. Thereby, resources are assigned to tasks depending on the defined required roles of the task assignments and the resource availabilities. The tasks are scheduled in a way that the computed planned dates are consistent with all defined control flows and task-subtasks relationships as well as resource constraints. The base schedule constitutes the basis for the enactment of the defined development process. However, the actual execution of tasks may deviate from the plan. The plan is only adapted to the actual execution when an authorized user performs according manual change operations and invokes another scheduling pass at project runtime.

In Section 2.3, a cutout of the complete dynamic task net was presented. Figure 7.17 shows the cutout with additional information about the required workload for the tasks and task assignments, the defined total durations of the tasks, and their budget. The total workload and budget of a task are contained in the total workload and budget of the parent task respectively. The maximal daily workload of several task assignments is restricted. This possibility has been used for task assignments which cover administrative work, e.g. the resource Baumann only manages the task Initial P&IDs which has several technical subtasks for the actual creation of the piping and instrumentation diagrams. The date 06/09/2010 in the top left corner of the figure is the date on which the base schedule has been generated. The planned start and end dates of the tasks are displayed in Figure 7.17 below the computed constraint dates. The planned dates are consistent with the computed constraint dates according to the timing consistency constraints (5.76) to (5.79). The planned dates also respect the execution semantics and lag times of the defined control flows. For some control flows, minimal lag times are defined, so that a certain number of work days has to pass between the start and end events of the connected tasks. For example, the planned end time of the task Initial P&IDs is 10 work days later than the planned end time of the task Process Flow Diagrams.

Figure 7.18 shows the corresponding Gantt chart for the cutout of the base schedule. The time frame which will be most affected by replanning and rescheduling activities is depicted. It can be seen that there is a minimal lag time of 10 work days between the planned end times of the tasks Process Flow Diagrams and Initial P&IDs. Since the tasks Initial P&IDs and Installation Plan are connected by a simultaneous control flow, the planned start time of the latter has to be equal or later than the planned start time of the former. Finally, it can be seen that there is a time buffer for the task Basic Engineering which results from the explicitly defined total duration of the task, which is longer than the scheduled duration of the defined subprocess. This time buffer will be used to compensate the effects of task delays in the basic engineering phase up to a certain extent.

In the following, two examples for rescheduling the task net at project runtime are presented to demonstrate the corresponding features of the scheduling algorithm. The procedure which has to be followed for replanning a dynamic task net at runtime will be presented in Chapter 9, and the example will be reviewed there again with respect to this procedure and authorization aspects.

In the given example, the task Process Flow Diagrams is delayed. The delay is identified at the 23rd March 2011, i.e. before the planned end time of the task. This situation is depicted in Figure 7.19. The actual degree of completion of the task Process Flow Diagrams is 52% which is smaller than its planned degree of completion for the given date which is 60%. Earned value analysis yields a forecasted duration of 126 working days which is 16 days longer than the planned total duration of 110 working days.

In this situation, the plan shall be adapted to the actual performance of the task. For this purpose, the command *Adapt plan to actual performance* is executed for the task Process Flow Diagrams (cf. Section 7.3.2), so that the planned duration is set to

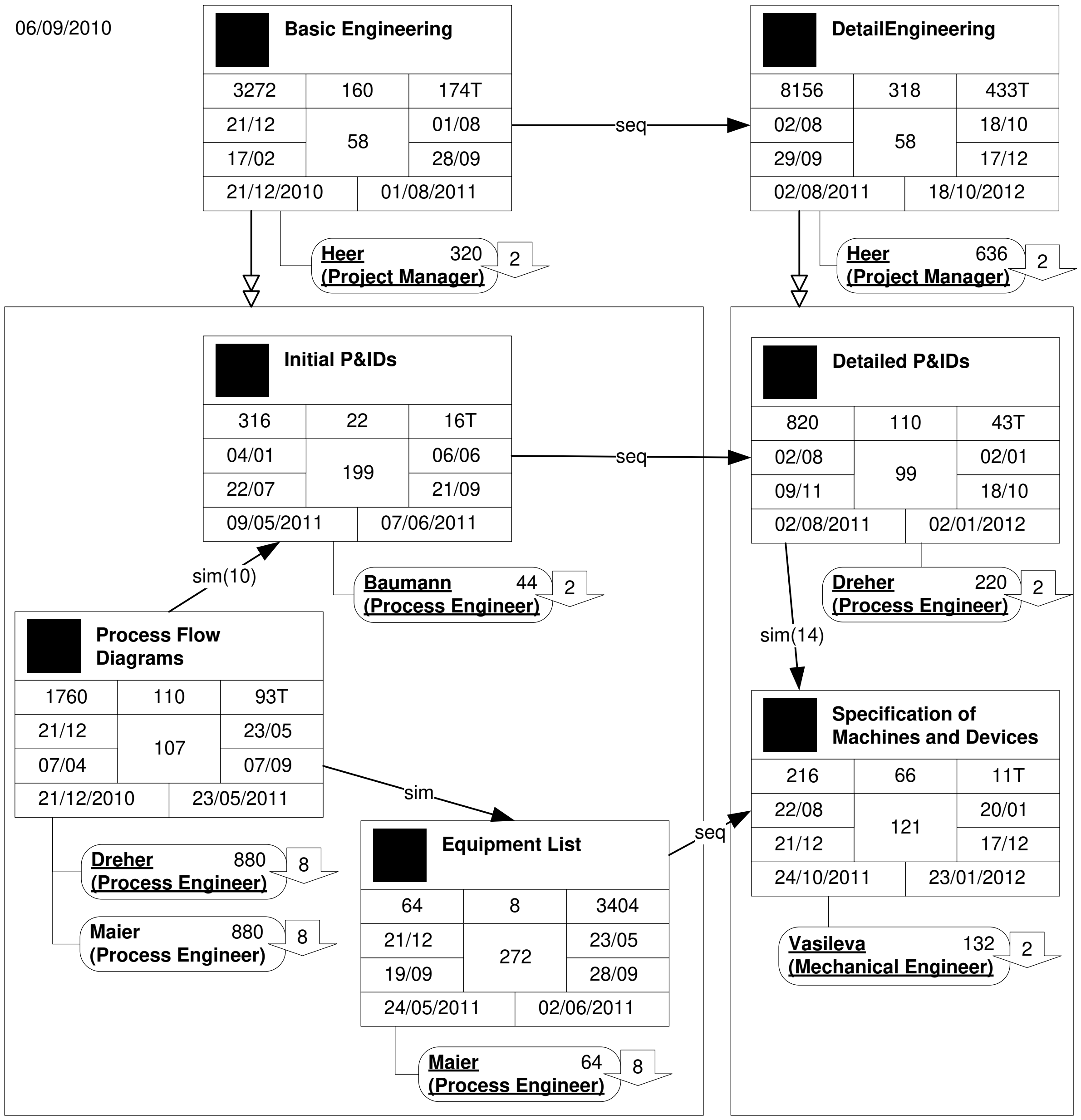


Abbildung 7.17: Planned dates for the base schedule cutout.

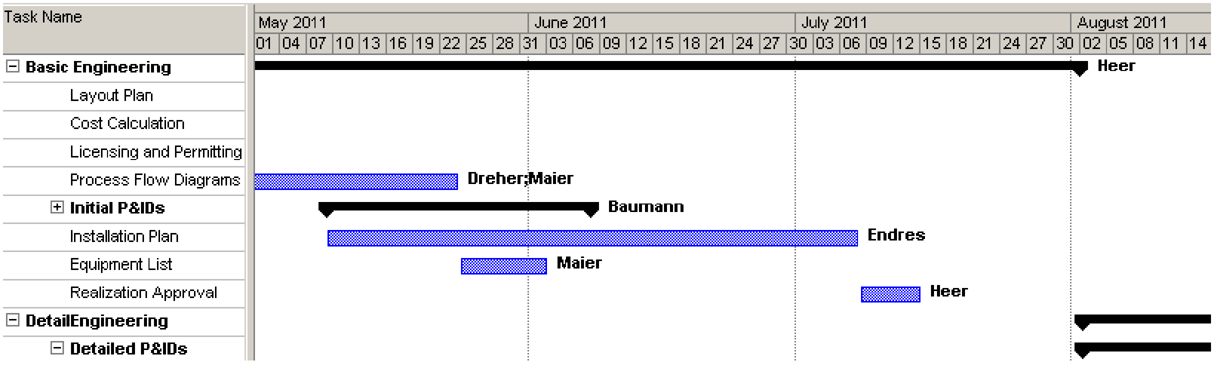


Abbildung 7.18: Gantt chart of the base schedule cutout.

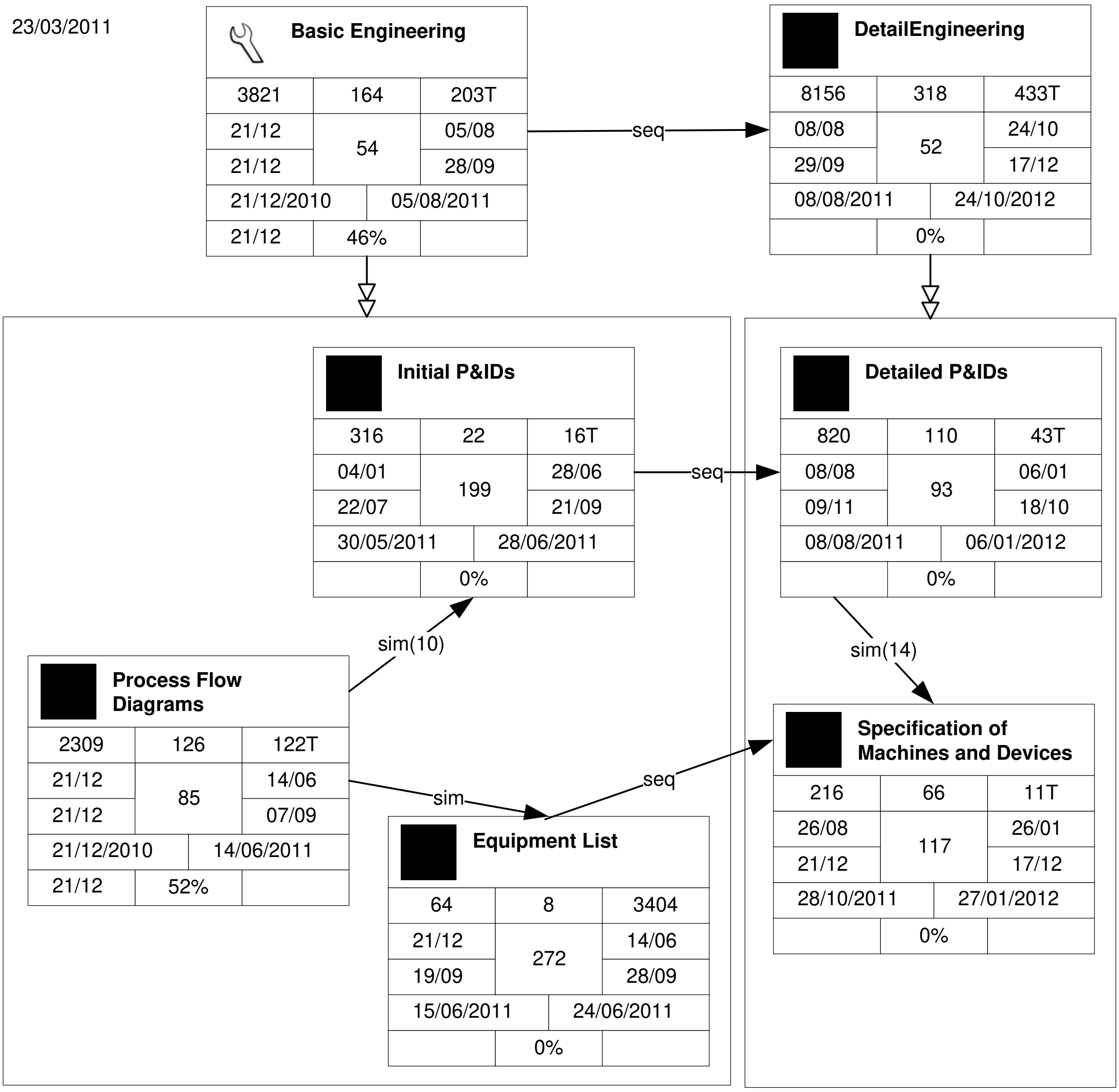


Abbildung 7.19: Computed constraint dates and planned dates after rescheduling.

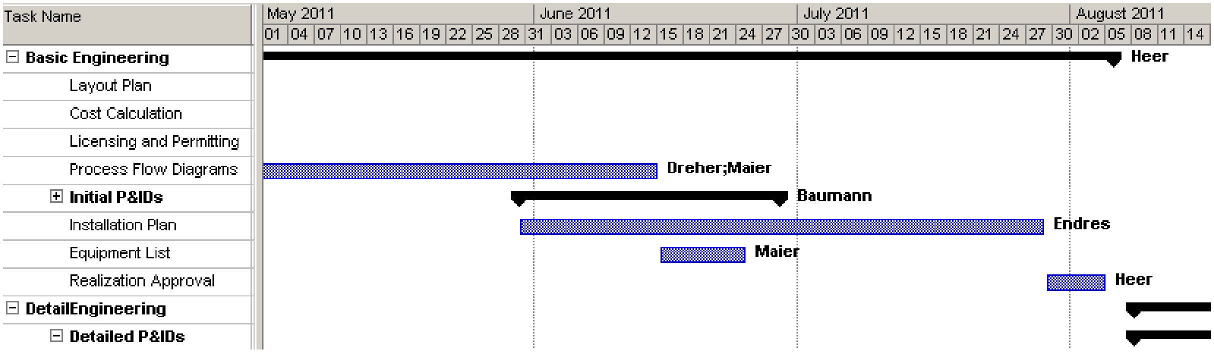


Abbildung 7.20: Gantt chart of rescheduled plan due to task delay.

the forecasted duration and the total workload is increased. Afterwards, the added workload is manually distributed to the task assignments by an authorized user.

The execution states of the task Basic Engineering and the root task of the dynamic task net are changed to *Replanning*, and a scheduling run for the complete task net is initiated. Figure 7.19 shows the result of rescheduling the dynamic task net. This plan state can be compared to the cutout of the base schedule which has been depicted in Figure 7.17.

In Figure 7.19, the total duration of the task Process Flow Diagrams has been increased from 110 to 126 working days. The total workload and total Budget have been increased accordingly. The earliest and latest possible start times of the running tasks Process Flow Diagrams and Basic Engineering have been set to the planned start times. Both tasks have been started at their respective planned start time. The earliest possible end time of the task Process Flow Diagrams has been increased due to the longer total duration. As a consequence, the earliest possible end times of the simultaneous successors have been increased. In case of the task Initial P&IDs, the earliest possible end time results from adding the lag time of the connecting control flow to the earliest possible end time of the predecessor. Due to the increased total duration and workload, the planned end time of the task Process Flow diagrams has been increased. The successors have been scheduled for later planned start times to fulfill the constraints imposed by the control flows on the planned end times. This required backtracking during resource-constrained scheduling. The task Equipment List is scheduled sequentially after its predecessor due to limited resource availabilities. The connecting control flow only demands simultaneous execution. However, the resource Maier which is required for the task Equipment List is also assigned to the task Process Flow Diagrams with 8 MHRS per day. Finally, the total duration of the task Basic Engineering has increased during scheduling because several subtasks have been scheduled for a later date. As a consequence, the planned start times of Detail Engineering and its subtasks have been increased.

The Gantt diagram in Figure 7.20 shows the same cutout of the example scenario as Figure 7.19 and some additional tasks. It can be compared to the Gantt chart of the base schedule cutout which is depicted in Figure 7.18. The previously described changes can be easily comprehended in the Gantt representation. The duration of the task Process Flow Diagrams has increased and the tasks Initial P&IDs and Equipment List have been moved to later dates. Because the task Initial P&IDs is a simultaneous predecessor of the task Installation Plan (cf. Figure 2.6), the latter had to be moved to a later date as well. Together with its sequential successor Realization Approval, the task Installation Plan accounts for the increased duration of Basic Engineering. However, because the originally planned total duration of Basic Engineering incorporated a certain time buffer, it is not increased as much as the duration of Process Flow Diagrams.

After the dynamic task net has been rescheduled at runtime, the development process proceeds. The task Process Flow Diagrams is eventually committed and the successors are started as planned. The task Equipment List is committed two days before its planned end time and the assigned resource Maier takes a leave for two days. The planned end time of the task is automatically set to the actual end time. During the execution of the task Initial P&IDs, at the 23rd June 2011, errors are detected in the process flow diagrams which require to rework the previous results. Because the task Process Flow Diagrams is already terminated, new versions of this task and the terminated successor Equipment List have to be created. Furthermore, a feedback flow from Initial P&IDs to the new version of Process Flow Diagrams is created. Since the revision of the process flow diagrams does not take as much time and workload as the initial creation, the values for the new task versions are adapted accordingly. Furthermore, the lag time between Process Flow Diagrams and Initial P&IDs is reduced to 5 working days. The release dates of the new task versions are automatically set to the date of the plan change. Rescheduling yields computed constraint dates and planned dates for the new task versions. The resulting state of the dynamic task net is depicted in Figure 7.21.

Figure 7.22 shows the Gantt chart representation of the task net cutout. The progress of the terminated and running tasks is visualized by the black bars inside the task bars. The new versions of the tasks Process Flow Diagrams and Equipment List are listed as separate tasks with the same name. The new version of Process Flow Diagrams is scheduled for the 27/06/2011 because the required resource Maier is not available before this date. Likewise, the new version of Equipment List is scheduled sequentially after Process Flow Diagrams because resource Maier is required for both tasks. The duration of the running task Initial P&IDs is prolonged so that the planned end time is consistent with the planned end time of the new predecessor and the lag time of 5 working days of the connecting control flow. Since the task has already been started, it is not moved to a later planned start time. The longer duration of Initial P&IDs does not affect the simultaneous successor Installation Plan and its successor Realization Approval. Consequently, the total duration of Basic Engineering is not affected as well. In this case, local rescheduling of Basic Engineering could be performed because other tasks in the task net have not been affected.

The scheduling example which has been presented in this section has demonstrated several advantages of the developed approach for time management and

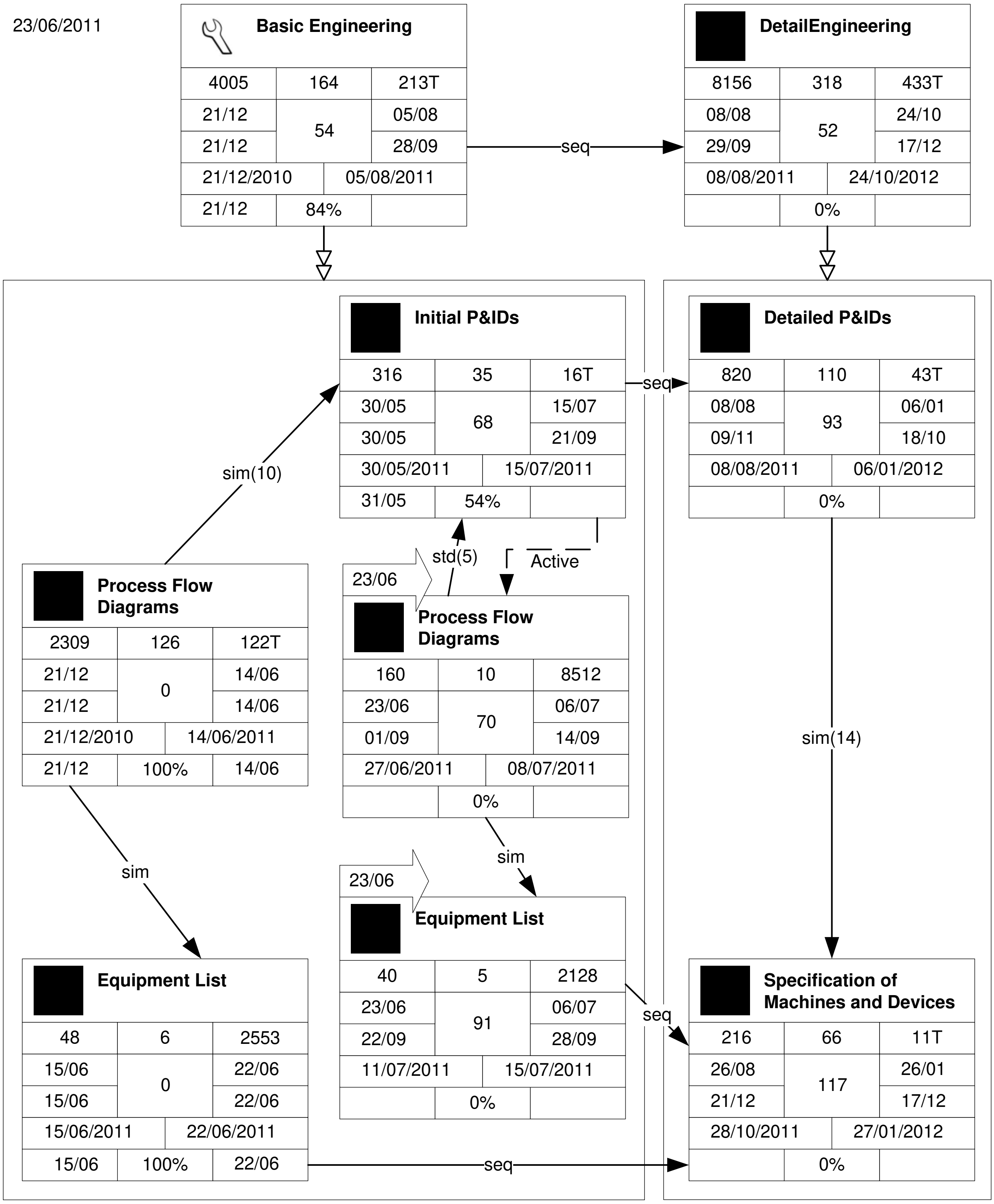


Abbildung 7.21: Rescheduled task net after feedback.

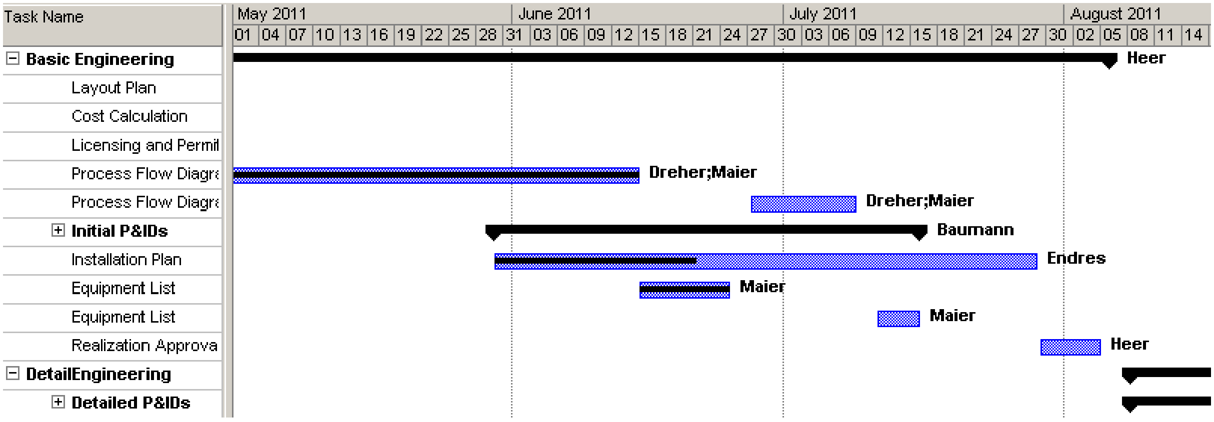


Abbildung 7.22: Gantt chart of rescheduled plan after feedback.

progress control in PROCEED. During scheduling, lag times which are defined for control flows and the availability of assigned resources are taken into account. The degree of completion of a task is used to forecast its expected duration, workload, and budget at completion. The underlying computations will be described in the next chapter. The forecasted values can be directly used for a plan change in order to adapt the plan to the actual performance. The execution states of tasks influence the rescheduling of a dynamic task net at runtime. Running tasks are not moved but only prolonged if required. The planned dates of terminated tasks are not changed at all. Time and workload buffer can be planned for complex tasks which may alleviate the effect of subtask delays during rescheduling. New task versions are scheduled like separate new tasks since there may be a delay between the end of the original task and the start of the new version. Finally, local rescheduling can be performed, if the timing properties of the root task of the rescheduled subprocess are not affected.

7.3.5 Correctness and Time Complexity

In this section, the correctness of the algorithm for resource constrained scheduling is shown, and an upper bound for the time complexity of the algorithm is determined.

Consistency with Constraint Dates The computed constraint dates which are determined by critical path analysis are used during resource-constrained scheduling. The priority of the tasks depends on the computed latest possible start times. The tasks with earlier latest possible start time have a higher priority. Furthermore, computed constraint dates constrain the planned start and end times of the tasks.

- A task cannot be scheduled before its EPST. Therefore, it is not included into the eligible set when the current scheduling date is earlier than its EPST.
- A task must not be scheduled later than its LPST. If an eligible task shall be scheduled but the LPST of the task has already passed, then scheduling is

aborted. However, this condition is only evaluated when the latest possible times have been determined based on an explicitly specified project deadline.

- A task must not finish before its EPET. This is ensured for preparing tasks by putting them into the forbidden set or setting the earliest planned start time during backtracking. Running tasks are prolonged, so that the planned end time is later than or equal to the EPET.
- A task must not finish later than its LPET. This condition is evaluated after the planned end time of a task has been determined. As for the LPST, this condition is only evaluated when the latest possible times have been determined based on an explicitly specified project deadline.

If no project deadline is specified, the latest possible start and end times which are computed starting from the earliest possible end time of the project cannot be used for consistency checks. This is due to the fact that resource-constrained scheduling in general leads to longer task durations and later planned start dates than critical path analysis because the latter does not take resource availabilities into account. This circumstance is illustrated in Figure 7.23. Figure 7.23 a) shows the case where the earliest possible end time of the last task in a task net has been used as the latest possible end time. As a consequence, the resulting time window for the tasks `T1` and `T2` is to short for resource-constrained scheduling, because the same resource is assigned to both tasks and the tasks have to be scheduled in sequence. When the backward scheduling pass of the CPM starts with the explicitly set due date of the last task, this may result in a larger time window for the two parallel tasks, so that the resource-feasible schedule is consistent with the latest possible times. This case is depicted in Figure 7.23 b). A manually set project deadline may still lead to inconsistencies. In this case, the project manager is informed, and he can adapt the deadline as required.

When the computed latest possible end times of tasks are not used for consistency checks because no project deadline has been specified, the planned end times of all tasks are compared to possibly defined due dates of the respective tasks nevertheless. These manually set constraint dates have to be consistent with the generated schedule.

For the presentation of the algorithms for resource-constrained scheduling, it has been implicitly assumed that the latest possible times have been determined based on an explicitly set project deadline. According consistency checks which compare the earliest possible times and planned dates with the latest possible times can be found in the algorithms and formulas. The presentation would have to be extended by alternative comparisons with the manually specified constraint dates and according checks whether the latter have defined values. This has been omitted in the presentation for reasons of clarity.

If a project deadline is explicitly defined, the computed constraint dates are used as strict constraints during resource-constrained scheduling. In this case, the time windows between the earliest possible start times and the latest possible end times of tasks have to be large enough, so that the tasks fit in these time windows during

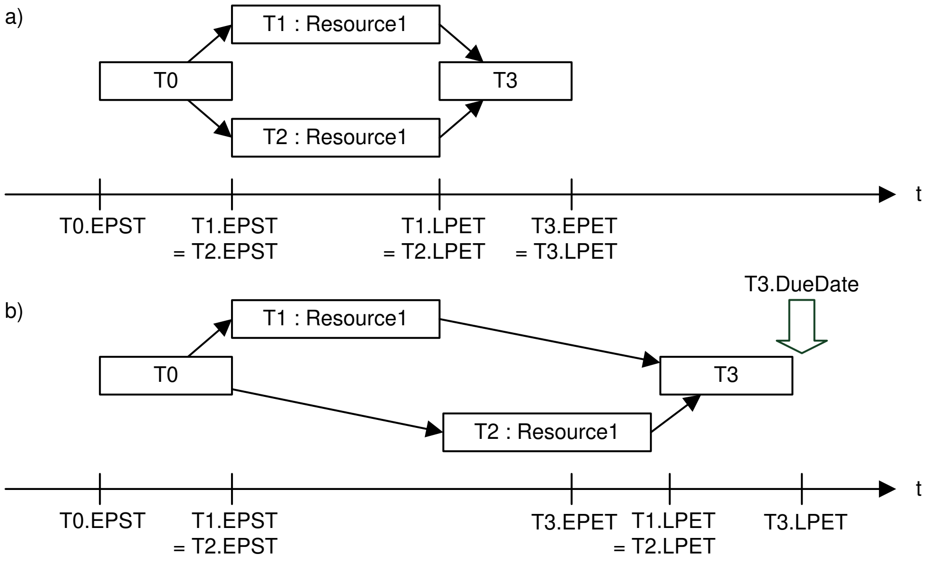


Abbildung 7.23: Latest possible end times with and without explicit project deadline.

resource-constrained scheduling where they usually have a longer duration. The minimal task durations used during critical path analysis are always smaller or equal to the durations derived for the tasks during resource-constrained scheduling. When the computed constraint dates are consistent with the planned dates, most tasks will have a total float which is greater than zero, and hence will not be critical by definition. The definition of the total float presented earlier may therefore not be helpful for assessing the criticality of tasks in the context of resource-constrained scheduling. The alternative of computing activity resource floats as proposed in [Lib01] by performing forward and backward scheduling with the parallel heuristic is not feasible for the GRCPSP [Kle00, p.178].

The solution proposed in this thesis is to perform another critical path analysis, using the computed total durations which result from resource-constrained scheduling. After a complete scheduling run, the total durations of all scheduled tasks are set in the database to the values which have been determined during resource-constrained scheduling. These values are used as minimal task durations for critical path analysis in every following scheduling run. As a consequence, the durations used for CPM do not differ very much from the durations determined during resource-constrained scheduling in these following scheduling runs. The duration of a task which is determined during resource-constrained scheduling will at least be as long as the total duration defined in the database. The duration of a task may still increase during resource-constrained scheduling due to different resource availabilities for different planned start dates or changes to the planned total workload of the task. However, the task durations used during later CPM computations will be close to the durations computed during resource-constrained scheduling.

The constraint dates in Figures 7.9 and 7.17 have been computed based on total duration values which stem from a previous resource-constrained scheduling pass. The general problem of the CPM that no resource availabilities are taken into account remains, so that tasks may be scheduled in parallel during CPM which have to be scheduled in sequence when resources are considered.

Termination The first step to prove the correctness of the developed algorithm is to show that it always terminates. In case of the parallel scheduling scheme, there could be several possible reasons for an infinite run of the algorithm. First, if no time and resource feasible schedule exists or cannot be found by the algorithm, then the scheduled tasks contained in the set `S` $\subset$ `Tasks` will never be completely contained in the complete and active sets. Therefore, the calendar search limit `end` $\in$ `Dates` is used in Algorithm 7.8 to constrain the maximal scheduling timeframe. The algorithm terminates at the latest at the date `end` $\in$ `Dates` which means that scheduling was not successful.

Second, backtracking is performed to correct inconsistent planned end times of tasks. Local backtracking is realized by means of the forbidden set. If, for some reason, a task is put into the forbidden set for every date in the scheduling timeframe and is never scheduled, then no time and resource feasible schedule can be found and scheduling terminates at the date `end` $\in$ `Dates`. Global backtracking is performed if inconsistent planned end times are detected in the method `CheckEndDates`. If the planned end time of the target task of a simultaneous or standard control flow is inconsistent with a planned end time of a predecessor, then the task is either moved to a later date or prolonged. This is realized by increasing its earliest possible start time or setting the earliest planned end time respectively and restarting the whole resource-constrained scheduling algorithm. The earliest possible start time is increased by a non-zero value, i.e. it is at least set to the next work day of the task's calendar. Therefore, the task is scheduled for a later date during every subsequent scheduling pass. Nevertheless, when the EPST of the task has been increased, the planned end time of the task may still be inconsistent after rescheduling with the modified constraint date because the duration of the task may decrease. The EPST has to be increased again and scheduling is performed again. However, as soon as the minimal duration of the task is reached, the increase of the EPST will finally lead to a consistent planned end time. The number of scheduling runs required to achieve the consistency of the planned end date of a task is constrained by a constant value which is smaller than the maximal total duration of all tasks in the dynamic task net. The increase of the planned end time of a task may affect the planned dates of its successors, which may again lead to inconsistent planned end dates. Only successors of the task can be affected in this way but no predecessors. Therefore, the maximal number of iterations of the scheduling algorithm due to global backtracking is constrained by the number of tasks in the dynamic task net. Consequently, resource-constrained scheduling will finally terminate.

Correctness The requirements for a time and resource feasible schedule have been defined in the form of timing consistency constraints in Section 5.3.2. It remains to be shown that the scheduling algorithm yields planned dates which fulfill the timing consistency constraints. This is informally proven in the following by mapping the constraints to those parts of the presented algorithms which ensure their fulfillment.

Table 7.2 shows the timing consistency constraints which are relevant for resource-constrained scheduling in the left column. In the right column, the corresponding definitions, methods and lines in the presented pseudo code fragments are given. The total duration of a scheduled task is updated whenever the planned end time of a task is changed during scheduling, so that constraint (5.61) is fulfilled for all tasks. The planned end time is computed starting from the planned start time of a task in the method `ScheduleTask`. The planned end time is not decreased in the method `UpdateParent`. This ensures the fulfillment of the constraint (5.62). A task can only be scheduled when its parent task is already contained in the active set. Therefore, constraint (5.63) is fulfilled for all tasks. The method `UpdateParent` ensures that the planned end time of a complex task is greater or equal to the planned end times of all subtasks, so that constraint (5.64) is fulfilled. The constraints (5.65) and (5.66) which refer to the consistency of planned end times with respect to control flow relationships are fulfilled because their violation is explicitly checked by the algorithm and backtracking is performed if required. Constraints imposed by simultaneous or sequential control flows imposed on the planned start times of tasks (constraints (5.67) and (5.68)) are fulfilled due to the definition of the eligible set which only includes tasks whose predecessors have been started or terminated, respectively. Constraint (5.69) is implicitly fulfilled when a control flow path exists from the feedback flow’s target to its source. Diagonal feedback flows (cf. Section 5.1) are treated in the formula $\mathtt{FS}(\mathtt{t})$ and the method `CheckEndDates`. The method `ScheduleTaskAssignments` has not been presented in pseudo code. It assigns eligible resources to the task assignments of a task and distributes the planned workload of the task assignments as described in Section 7.3.2. Thereby, the maximal daily workload of the task assignments and the available workload of the assigned resources are respected. As a consequence, the constraints (5.70), (5.73), and (5.72) are fulfilled. The planned end time of a task is increased in the method `ScheduleTask`, so that the timeframe between the planned start and end covers the complete distributed workload of all task assignments. As a consequence, the constraint (5.71) is fulfilled. The timing consistency constraints (5.74) to (5.79) which relate the planned dates to the manually set and computed constraint dates are fulfilled because inconsistencies with respect to the latest possible times would lead to the abortion of the scheduling algorithm, and inconsistencies regarding the earliest possible end times are handled by the backtracking mechanisms. The planned start times are consistent with the earliest possible start times due to the definition of the eligible set.

Constraint	According part(s) of heuristic algorithm
(5.61)	`ScheduleTask` line 18 and `UpdateParent` line 5
(5.62)	`ScheduleTask` and `UpdateParent`
(5.63)	Definition of eligible set
(5.64)	`UpdateParent`
(5.65)	`Schedule` lines 17-29 (`FS`) and `CheckEndDates`
(5.66)	`Schedule` lines 17-29 (`FS`) and `CheckEndDates`
(5.67)	Definition of eligible set
(5.68)	Definition of eligible set
(5.69)	`Schedule` lines 17-29 (`FS`) and `CheckEndDates`
(5.70)	`ScheduleTaskAssignments`
(5.71)	`ScheduleTask` lines 1-4 and `ScheduleTaskAssignments`
(5.72)	`ScheduleTaskAssignments`
(5.73)	`ScheduleTaskAssignments`
(5.74)	fulfilled due to CPM and constraints (5.49) and (5.76)
(5.75)	fulfilled due to CPM and constraints (5.50) and (5.79)
(5.76)	Definition of eligible set
(5.77)	`Schedule` lines 10-11
(5.78)	`Schedule` lines 17-29 (`FS`) and `CheckEndDates`
(5.79)	`Schedule` lines 15-16,24-25 and `UpdateParent`

Tabelle 7.2: Timing consistency constraints fulfillment by heuristic.

Time complexity The heuristic algorithm for resource-constrained scheduling is based on the parallel scheduling scheme presented in [KH98] which has a time complexity of $O(|\mathtt{T}|^2|\mathtt{Roles}|)$. A problem instance for the parallel scheduling scheme presented in [KH98] does only contain sequential control flows, does not have a hierarchical structure, and the task durations are fixed before scheduling. The introduction of PDM task relationships with minimal time lags does not increase the runtime complexity because only the computation of the next decision point, i.e. the date on which a task can become eligible, has to be adapted [Kle00].

However, due to the dynamic calculation of task durations during scheduling and the hierarchical structure of dynamic task nets, backtracking is required for the heuristic presented in this thesis. Backtracking has to be performed when the planned end time of a task is inconsistent with the planned end time of a simultaneous or standard predecessor. These cases cannot be avoided beforehand because the duration of the successor is computed during scheduling and the predecessor or successor can be complex tasks whose respective planned end times are not final yet.

In case of atomic tasks, the forbidden set is used to delay the scheduling of an inconsistent task. Scheduling an eligible task requires a constant amount of time whose upper bound is determined by the maximal value for the total workload of a task in the dynamic task net. After the distribution of the workload and the computation of the total duration and planned end time of the task, it can be decided if the task is still eligible for the current date or if it has to be scheduled at a later date. Therefore, the constant amount of time required for scheduling the task assignments of a task can be regarded as additional time required to determine whether a task is eligible or not. As a consequence, the overall time complexity of the parallel scheduling scheme does not increase by this local backtracking using the forbidden set.

If inconsistent end dates are detected at the end of the scheduling pass, the whole scheduling algorithm has to be executed again after constraint dates have been adapted. As explained before to show the termination of the algorithm, the maximal number of reiterations is in $O(|\mathtt{T}|)$. Therefore, the runtime complexity of the implemented parallel scheduling scheme is $O(|\mathtt{T}|^3|\mathtt{Roles}|)$ which is only slightly worse than the basic version of the algorithm which does not cover end-end task relationships, complex tasks, and durations computed during scheduling.

7.4 Scheduling of Workflow Instances

The scheduling of workflow-managed tasks requires a special approach. Specific problems arise due to the usage of alternative branching and loop constructs in a workflow definition. In classical project plans and dynamic task nets, all defined tasks are eventually executed. In workflow-managed task nets, alternative tasks can be skipped and tasks may be iterated several times.

As described in Section 6.3, a workflow template contains a subtask for every activity in the workflow definition. This means, that subtasks are defined for all activities contained in all alternative branches of an `IfElse`-activity, but only one branch is executed at workflow runtime. Furthermore, for every activity contained in a `While`-activity, exactly one subtask is defined, so that only the first iteration of the loop is explicitly modeled in the workflow template. However, at workflow runtime, several iterations of the `While`-activity may be executed which results in the creation of several versions of the defined tasks as described in Section 6.3.

A workflow-managed task in a dynamic task net is a copy of a workflow template. After its creation, a workflow-managed task and its subtasks may be scheduled. Critical path analysis and resource-constrained scheduling have to be adapted for the case of workflow-managed tasks. Alternative tasks cannot be treated like parallel tasks, and scheduling only the first iteration of a loop does not cover the cases of zero or multiple iterations.

In the following descriptions, the distinction between workflow tasks in a workflow-managed task net and their associated activities in the workflow definition is softened for reasons of readability. For example, it will be written that a workflow task is contained in a control structure, although strictly speaking the associated activity is contained in the workflow structure. The attentive reader will be able to understand the implicitly described relations.

7.4.1 Critical Path Analysis

With respect to critical path analysis, the computation of the earliest possible start times of the workflow tasks which succeed alternative branching constructs and loops have to be adapted. Furthermore, the computation of the latest possible end times of the tasks which precede these control structures have to be adapted.

In PROCEED, the computed constraint dates are used as strict constraint dates for resource-constrained scheduling. Therefore, the minimal task durations are used during critical path analysis to obtain computed constraint dates which are still valid during resource-constrained scheduling (cf. Sections 7.2 and 7.3).

Consequently, the earliest possible start times of the successors of an alternative branching construct have to be computed based on the duration of the shortest alternative branch. At workflow runtime, the shortest path through an alternative branching construct may be chosen and the successors may start after this branch has been terminated. If a longer branch would be used for the computation of their earliest possible start times, the resulting dates would constrain the start of the successors too much because they could start earlier.

In case of a loop structure, the minimal number of iterations has to be assumed to compute the earliest possible start times of the successors. If the loop has to be iterated at least once, the shortest duration of the first iteration of the loop is used. If the loop can be skipped completely, then a duration of zero days is used.

Just like the earliest possible start times of the successors, the latest possible end times of the predecessors of alternative branching constructs and loops have to be computed based on the shortest durations of these control flow constructs. If a predecessor task terminates at its latest possible end time, it may still be possible to finish the workflow in time by executing the shortest alternative branch or the smallest number of iterations.

Besides the earliest start times of the successors and the latest end times of the predecessors, constraint dates have to be computed for the tasks inside control flow structures. This computation is performed for every alternative branch and for the body of a loop construct in the same way as for the whole workflow. Constraint dates are computed for all tasks in all alternative branches, but only the earliest possible end time of the shortest branch is used for the computation of the earliest possible start times of the successors, and only the latest possible start time of the shortest branch is used for the computation of the latest possible end times of the predecessors. Constraint dates are computed for all tasks in the first iteration of a loop, even when the loop may be skipped completely.

Figure 7.24 shows an abstract example for the CPM computation of a workflow-managed task. The task B is executed alternatively to C and D. The task E may be iterated several times but has to be executed at least once. On the right side of the figure, the minimal durations of the tasks and the computed constraint dates are depicted. The EPST of task E is computed based on the EPET of task B, because B is contained in the shortest branch. The 13th of September 2010 is the next working day after the 10th of the same month. Because task E will be executed at least once, the EPST of task F is the next working day after the EPET of task E. The LPET of the

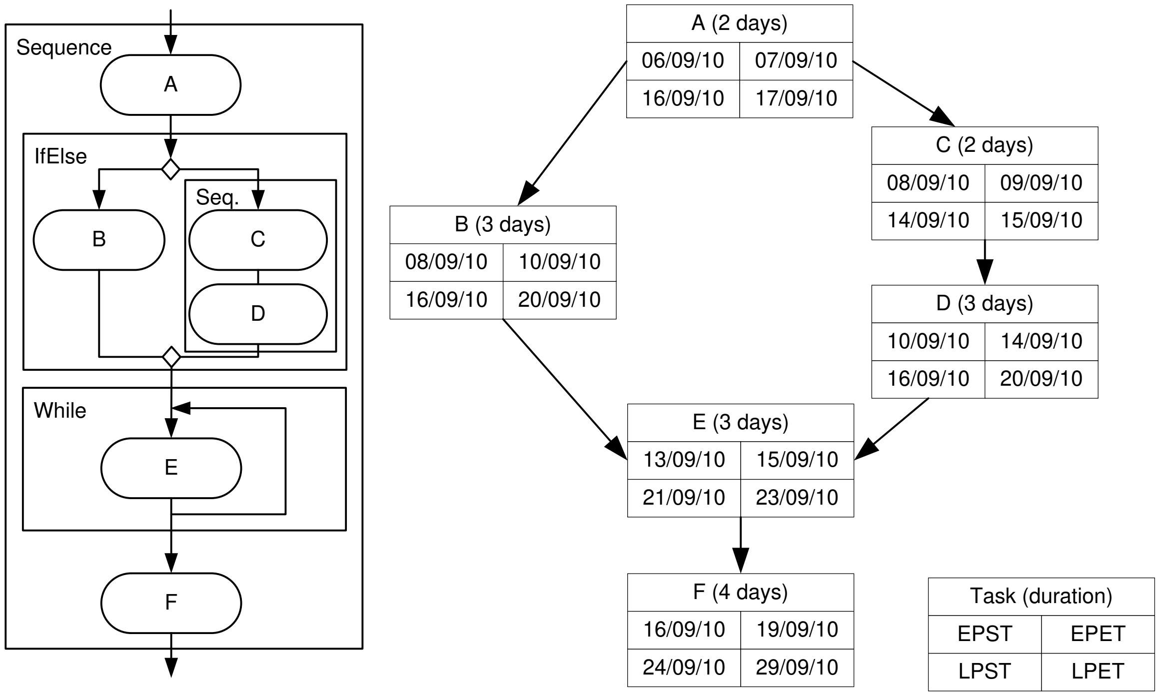


Abbildung 7.24: CPM computations in a workflow-managed task.

last task F is later than its EPET. Starting from this date, the backward scheduling pass is performed analogous to forward scheduling.

In a workflow definition, control flow structures may be nested. The above described computations of constraint dates are still valid for the case of nested control flow structures. For example, if an alternative branch contains a loop construct which may be skipped completely at runtime, then the duration of the first iteration is not taken into account for the computation of the earliest possible end time of the alternative branch, which may implicate that the alternative branch is the shortest branch.

The described CPM computations for control flow structures in workflow-managed task nets are performed before the respective control flow structures have been executed. At workflow runtime, a re-computation of the constraint dates is required when decisions for alternatives or further iterations have been made. When the decision for an alternative branch has been made at workflow runtime, the tasks of all other branches are skipped which makes them zero-duration tasks so that they are eliminated from the memory implementation of the task net before scheduling. The selected alternative branch is therefore handled as if no alternatives ever existed. When the nth iteration of a loop is currently executing, all previous iterations are handled like a ordinary task nets because they are already terminated and can only contain parallel and sequential constructs. The currently executing nth iteration is handled as described above.

Adaptation of CPM algorithm The method `HandleControlFlowForward` is extended as shown in Algorithm 7.11 by the cases that the control flow originates from a last task of an alternative branch or a loop. There may be several parallel last tasks in an alternative branch or a loop.

In case of alternative branching, the EPET of the shortest branch will finally become the EPST of the successor if it is later than the release date. The control flow originates from one last task in one of the alternative branches. Because the CPM algorithm traverses a subnet along the control flows in a depth-first fashion, the earliest possible end times of the other last tasks may still be undefined. In a first step, the EPET of the alternative branch is determined in which the predecessor of the control flow is contained. This is the latest defined EPET of all last tasks in the branch. Second, the EPET of the whole alternative branching construct is computed as the earliest EPET of all branches for which an EPET can already be determined. If the computed EPET is later than the release date of the successor, it is set as its EPST. The computed EPET is compared to the release date of the successor and not to a possibly defined EPST because the latter may have been derived from a longer alternative branch before and therefore may be later than the computed EPET. However, in this case, the EPST of the successor has to be set to the earlier date.

A control flow which originates from a last task of a loop construct is handled in the same as a sequential control flow in Algorithm 7.2, if the loop is executed at least once. If the loop may be skipped completely, then the EPST of the successor is set to the EPST of the whole loop construct.

The adaptation of the method `HandleControlFlowBackward` for the latest possible end time of the predecessor is analogous.

Adaptation of timing consistency constraints The computation of constraint dates for tasks which are contained in an alternative branching construct but not in its shortest branch as well as the computation of constraint dates for tasks which are contained in a loop which may be skipped completely may result in a violation of the timing consistency constraints (5.57) and (5.58).

The EPST of a successor of an alternative branching construct is computed based on the shortest branch and can therefore be earlier than the EPET of one of the last tasks in a longer branch. This is the case in Figure 7.24 for the task E whose EPST is earlier than the EPET of the predecessor D. Analogously, the LPET of a predecessor of an alternative branching construct may be inconsistent with the LPST of a task in a longer branch, e.g. task A in Figure 7.24. The same problem may occur for successors and predecessors of loop constructs which may be skipped completely.

Therefore, the constraint checking has to be adapted for workflow-managed tasks as well. A task which is a successor of an alternative branching construct has only tasks as predecessors which are contained in one of the alternative branches. The constraint (5.57) is only evaluated for the predecessor which is contained in the shortest branch. Likewise, a task which is a predecessor of an alternative branching construct has only tasks as successors which are contained in one of the

Algorithm 7.11 `HandleControlFlowForward(c)`

```
if c.Pred is a last task in an alternative branch then
    set EPET of the branch to the latest EPET of all last tasks in the branch
    EPET := the earliest defined EPET of all branches
    if undef(c.Succ.ReleaseDate) $\vee$ EPET $>$ c.succ.ReleaseDate then
        c.Succ.EPST := EPET
    end if
    if c.Succ.Parent $=$ c.Pred.Parent then
        ScheduleForward(c.Succ)
    end if
else if c.Pred is a last task in a loop then
    if the loop will be executed at least once then
        if undef(c.Succ.EPST) $\vee$ c.Pred.EPET $>$ c.Succ.EPST then
            c.Succ.EPST := c.Pred.EPET
        end if
    else
        EPST := earliest start time of the loop
        if undef(c.Succ.EPST) $\vee$ EPST $>$ c.Succ.EPST then
            c.Succ.EPST := EPST
        end if
    end if
    if c.Succ.Parent $=$ c.Pred.Parent then
        ScheduleForward(c.Succ)
    end if
else
    Algorithm 7.2
end if
```

alternative branches. The constraint (5.58) is only evaluated for the successor which is contained in the shortest branch

If a loop is iterated at least once, the earliest EPET of the last tasks in the loop is used to compute the EPST of the successors, and the latest LPST of the first tasks in the loop is used to compute the LPET of the predecessors. No inconsistencies can occur in this case. However, if the loop may be skipped completely, the earliest possible end times of the predecessors of the loop are used to compute the earliest possible start times of the successors of the loop. Likewise, the latest possible start times of the successors are used for the computation of the latest possible end times of the predecessors of the loop. Therefore, the timing consistency constraints (5.57) and (5.58) are evaluated with respect to the EPET of the loop's predecessors and the LPST of the loop's successors, respectively.

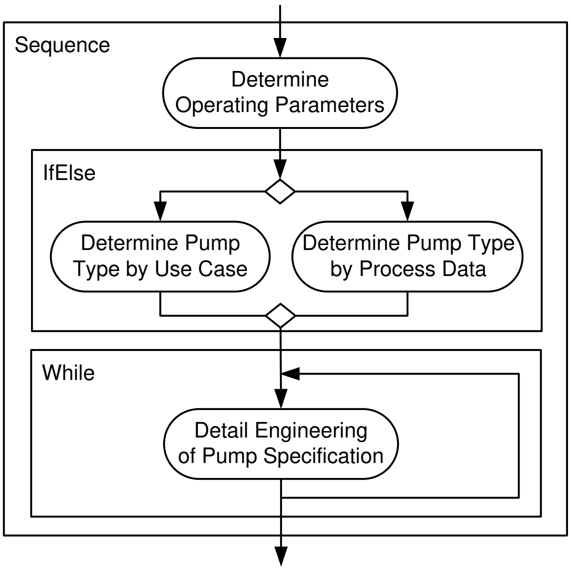


Abbildung 7.25: Example workflow for resource-constrained scheduling

7.4.2 Resource-Constrained Scheduling

In the context of resource-constrained scheduling, the problems involved with alternative branching and loop constructs extend to the assignment of eligible resources and the distribution of workload. If workflow-managed tasks would be scheduled like all other tasks in a dynamic task net, the planning of workload and resource requirements would be incorrect. In case of alternative branching, too much workload would be planned and resources would be unnecessarily bound for those tasks which are skipped at runtime. In case of loop structures, too little workload would be planned, and required resources would possibly not be available when several iterations of a loop are executed.

Alternative branching Figure 7.26 shows the realization of a workflow-managed task before its start. The associated workflow definition is depicted in Figure 7.25. The two tasks to determine the pump type shall be executed alternatively. Both require the same role and shall be performed by the same resource. If the assigned resources are not specified manually before automatic scheduling, different resources are assigned to the alternative tasks, because the resource which is assigned to one task is not available for the other. This situation is depicted in Figure 7.27 a) where the tasks are scheduled in parallel with assigned resources Meyer and Dreher. The assignment of different resources for alternative branches has the disadvantage that one of the resources is unnecessarily blocked for the planned duration of the task and cannot be assigned to other tasks in the project.

When a maximal resource usage per day is specified for the task assignments, the same resource may be selected by the scheduling algorithm for both tasks. This

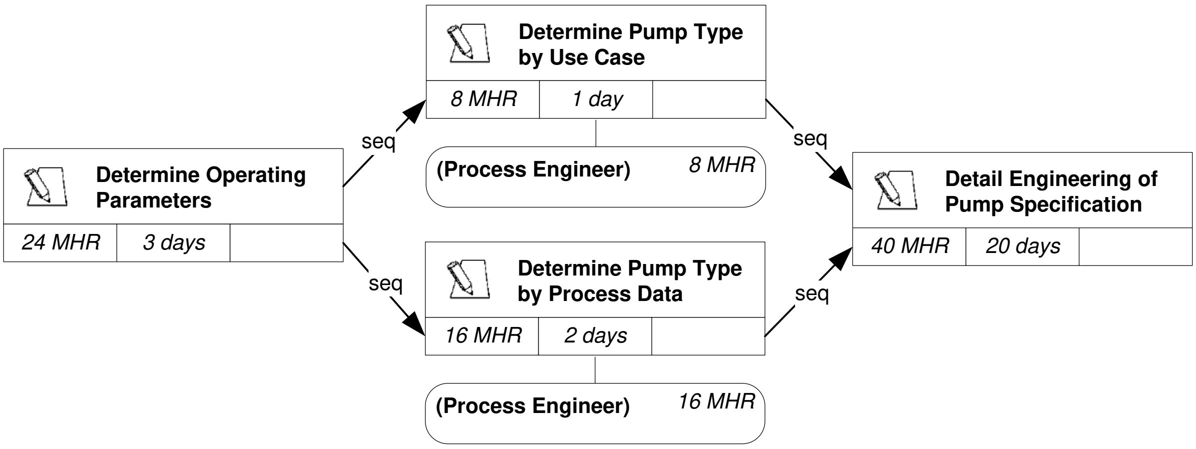


Abbildung 7.26: Example realization of a workflow-managed task.

		16	17	18	10/2010
a)	Determine Pump Type by Use Case	8			Meyer
	Determine Pump Type by Process Data	8	8		Dreher
b)	Determine Pump Type by Use Case	4	4		Dreher
	Determine Pump Type by Process Data	4	4	8	Dreher
c)	Determine Pump Type by Use Case	8			Dreher
	Determine Pump Type by Process Data		8	8	Dreher

Abbildung 7.27: Different unsatisfying possibilities for scheduling alternative tasks

case is depicted in Figure 7.27 b) where the resource Dreher is assigned to both tasks. However, PROCEED does not offer the possibility to define a binding between two tasks which demands that they are necessarily executed by the same resource. Therefore, the only way to ensure that both tasks are executed by the same resource is to manually assign the resource before automatic scheduling.

If the same resource is assigned to both tasks but no maximal resource usage per day is specified for the task assignments, another problem arises. The tasks have to be scheduled in sequence because the resource is only available for one task at a time as depicted in Figure 7.27 c).

In both cases in which the same resource is assigned to the tasks, the planned workload for this resource is to high since he or she only has to execute one of the tasks. Furthermore, the planned workload for the workflow-managed parent task is to high, since workload is defined for all alternative tasks. In the example of Figure 7.26, the sum of 24 MHRS is added to the used workflow of the parent task. This value is in any case too high because only one of the tasks will be executed and the other task will be skipped. Therefore, the planned workload of the workflow-managed parent task will be too high until the decision for one of the alternative branches is made.

Several possible solutions to resolve the described problems concerning the scheduling of alternative branches in a workflow have been evaluated.

- Workload is planned for all alternative branches and all tasks are scheduled. This is the case which has been described above. As a consequence, to much workload is planned and scheduled, and resources are unnecessarily bound.
- All tasks in the alternative branches are scheduled according to their duration but no workload is planned and no resources are assigned. In this case the tasks may be correctly timed, but too little workload is planned for the workflow-managed task, and the required resources may not be available when the tasks shall finally be executed.
- The tasks in alternative branches are not scheduled at all until the decision is made at runtime which branch shall be executed. In this case the same problems arise as in the previous case.
- One of the alternative branches is scheduled—preferably the most probable one if it can be determined—and the workflow-managed task net is replanned and rescheduled in the case that a different branch is chosen at workflow runtime.

The last solution has been selected because it has several advantages over the other cases which will be described in the following. When the realization of a workflow-managed task shall be scheduled and the workflow definition contains alternative branches, only the tasks which belong to one of the branches are scheduled. The workflow-managed task net may be scheduled before the workflow has reached the decision point in which one of the alternative branches is selected. Before the decision for one of the branches has been made, the branch to be scheduled is selected as follows.

- If statistical data about the enactment of previous instances of the same workflow type exists, it is used to determine the most probable branch, which is then selected to be scheduled.
- If no statistical data is available, the branch with the highest total workload in the sum of its tasks is selected. In this way, it is assured that resources with enough free working hours are available for the workflow tasks when the decision is made for another branch at workflow runtime.

The planning data for the tasks which belong to the alternative branches is set in a way, that the algorithm for resource-constrained scheduling only schedules the selected branch.

- The tasks of the selected branch are in the execution state *InDefinition* and have a total workload greater than zero man hours, and their total duration is either undefined or greater than zero work days. Therefore, they are taken into account during resource-constrained scheduling.

- The tasks of the not-selected branches are also in the execution state *InDefinition* but have a total workload of zero man hours and the duration is undefined. Therefore, they are not taken into account during resource-constrained scheduling (cf. Section 7.1).

At workflow runtime when the decision has been made for one of the alternative branches, rescheduling may be required. In the case that the previously selected branch has also been selected at runtime, no change to the schedule is required. In the case that a different branch has been selected at runtime, the workflow-managed dynamic task net is replanned and rescheduled automatically.

- The tasks of the previously selected branch are skipped. Thereby, the planned total workload is reset to zero and the assigned resources are released.
- Likewise, the tasks of all other not-selected branches are skipped.
- The planning data of the tasks which belong to the branch selected at runtime are retrieved from the workflow template.
- If task assignments of the branch selected at runtime require the same roles as task assignments of the skipped tasks of the previously selected branch, the freed resources are assigned to these task assignments. Otherwise, no resources are assigned at this point.
- The workflow-managed task is rescheduled whereby the workload is distributed and resources are assigned to task assignments of tasks in the branch which has been selected at runtime. This rescheduling is local and does not affect other parts of the project plan.
- The execution states of the scheduled tasks are changed from *InDefinition* to *Waiting* as described in Section 6.3.

The solution for resource-constrained scheduling of alternative branches is illustrated in Figure 7.28. Figure 7.28 a) shows the situation before the decision for one of the alternative branches has been made at workflow runtime. The branch with the higher total workload has been scheduled whereby the resource Dreher has been assigned and the workload has been distributed over two days. Figure 7.28 b) shows the situation at runtime, after the decision has been made for the previously scheduled branch. No rescheduling was required and the task Determine Pump Type by Process Data can be executed as scheduled. In the case depicted in Figure 7.28 c), the decision has been made for the other alternative branch at runtime. The task Determine Pump Type by Process Data was skipped which is why no resource is assigned anymore, and no workload is planned for the given days. The freed resource Dreher was assigned to the task Determine Pump Type by Use Case and the workload was distributed by the algorithm for resource-constrained scheduling.

	16	17	18	10/2010
a) Before decision has been made				
Determine Pump Type by Use Case				
Determine Pump Type by Process Data	8	8		Dreher
b) After decision for scheduled branch				
Determine Pump Type by Use Case				
Determine Pump Type by Process Data	8	8		Dreher
c) After decision for not-scheduled branch				
Determine Pump Type by Use Case	8			Dreher
Determine Pump Type by Process Data				

Abbildung 7.28: Scheduled alternatives at different stages of workflow enactment.

Because the task of the previously selected branch was skipped, the resource Dreher was available and the task Determine Pump Type by Use Case could be scheduled for the 16th October.

Altogether, the planned duration of the alternative branching construct was at any time at most two days because the tasks could be scheduled with fully available resources assigned which are not blocked by alternative tasks. In contrast to that, the different possible solutions for scheduling all alternative branches depicted in Figure 7.27 lead to an overall duration of three days of the alternative branching construct.

The rescheduling of tasks which belong to an alternative branch of a workflow at runtime is local, i.e. only the planned dates, workload and resource assignments of the workflow tasks are changed. This is achieved by setting the workflow-managed parent task as the root task for scheduling (cf. Section 7.3). If required resources are not available this may lead to delays in the selected branch, or resource-constrained scheduling may even fail. In the first case, a suboptimal schedule is computed which is nevertheless feasible. In the second case, the project manager is informed about the failure. In both cases, a manually invoked scheduling pass, in which also other tasks in the project may be rescheduled, can improve or repair the schedule respectively. In the case that resources which were assigned to tasks of the previously scheduled branch can be used for the branch which is selected at runtime, scheduling of the selected branch will most probably not fail and lead to good results, in particular when the previously scheduled branch required more workload than the branch selected at runtime.

Loops In the example of Figures 7.25 and 7.26, the task Detail Engineering of Pump Specification is iterated several times whereby the pump design is more and more refined. If the planned duration, total workload, and total budget of the task only account for the first iteration of the loop, additional work days, workload and budget have to be planned for the subsequent versions of the task. Therefore, the planned values are too low until the loop is actually iterated.

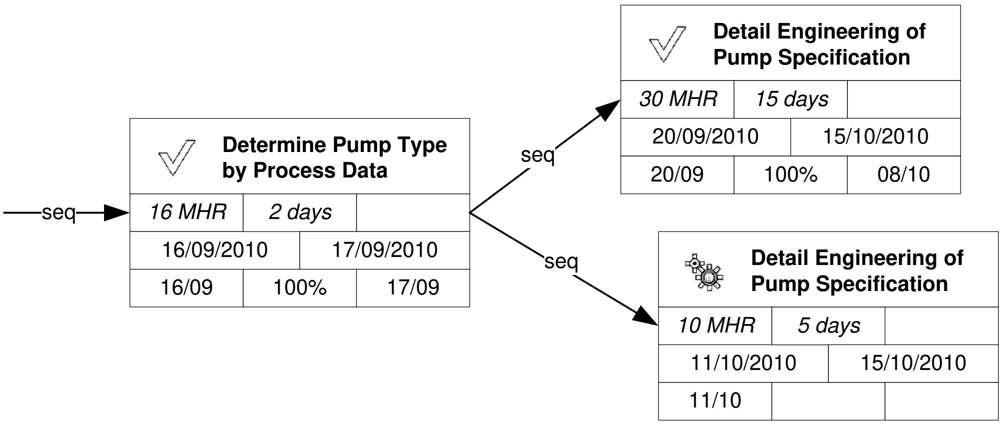


Abbildung 7.29: Scheduled iterated activity in a workflow-managed task.

This problem is comparably simple to solve. The duration and total workload of workflow tasks which represent activities in a `While`-loop of the workflow definition are set in a way that they incorporate multiple iterations of the loop. In the example of Figure 7.26, the task Detail Engineering of Pump Specification is planned with a duration of 20 days and a required workload of 40 MHRS. These 40 MHRS are the required workload for all iterations of this task according to the values stored in the workflow template. Figure 7.29 shows a part of the workflow-managed dynamic task net at runtime. The aforementioned task has been committed after 15 days, whereupon the remaining planned workload has been deleted as described in Section 5.3.1, which leads to a planned workload of 30 MHRS. The loop construct has already entered the next iteration and a new version of the terminated subtask has been created. The total workload of this new task version is automatically set to 10 MHRS which are defined for the remaining iterations in the workflow template. The total duration of the new task version is set to 5 days which is the value defined by the workflow template. If the iterated task is executed approximately as planned in every iteration, then the initially planned values are finally reached by the subsequent task versions together. In the example, the two versions of the task together account for 20 work days and 40 MHRS workload.

When the number of iterations and the activity durations in a workflow instance are close to the average values derived from statistical data about all instances of the workflow type, then the originally planned duration and required workload will be close to the actual duration and workload.

Zero-duration subtasks of workflow-managed tasks The described solution for resource-constrained scheduling of workflow-managed tasks is required when a workflow definition is used to enact a long-running task on a higher level of the hierarchical dynamic task net. However, workflow definitions are often used to define procedures for individual engineers. The corresponding workflow-managed tasks often have a short duration and are defined on the lowest levels of the hierarchical dynamic task net. In these cases, it is not useful to schedule the subtasks of the workflow-managed tasks. Low-level workflows can be excluded from resource-constrained scheduling in different ways. As described in Section 7.1, the following tasks are not taken into account during scheduling.

- Tasks with a duration of 0 days,
- Tasks with an undefined duration and a workload of 0 MHRS,
- Tasks whose parent task has a duration of 1 day only,
- Tasks with granularity level *Work Step*.

The granularity of the subtasks defined in a workflow template can be set to *Work Step*. This way, the automatic scheduling algorithm will schedule all instances of this workflow template as atomic tasks and will neglect the subtasks which represent the workflow activities. If a workflow-managed task itself has the granularity *Work Step*, it is not scheduled at all. Furthermore, if the duration or workload of a workflow is too little, it is not scheduled at all (0 days) or only as an atomic task (1 day). This way, low-level workflows can be enacted as part of the overall dynamic task net, but they are neglected when it comes to scheduling.

7.5 Export to Database

After a successful scheduling run, the memory representation contains the earliest and latest possible start and end times of all tasks in the project as well as the updated planned dates, resource assignments, and workload distributions of the (partially) scheduled tasks. These values and distributions are written to the database after a successful scheduling pass when the user, who initiated the scheduling pass, accepts the results.

The computed constraint dates are exported to the Comos database for all tasks in the project and override possibly existing values. Even not scheduled tasks receive the new computed constraint dates. These new values are necessarily consistent with the constraint dates and planned dates of the not scheduled tasks because otherwise scheduling would have failed. The local rescheduling of a subnet may influence the earliest and latest possible start and end times of not scheduled tasks in other parts of the project. This information is relevant for the responsible resources of these tasks to assess the criticality of their tasks after the plan changes.

In contrast to computed constraint dates, planned dates of not scheduled tasks are never changed during scheduling and are therefore not exported to the Comos database. The local rescheduling of a subnet does not affect the planned dates of tasks which are not part of this subnet.

The planned start and end times of scheduled tasks and the computed workload distributions are exported to the Comos database. The planned start times of partially scheduled tasks remain unchanged and are therefore not exported. However, the planned end times and the workload distributions of partially scheduled tasks may be changed by the scheduling algorithm. Therefore, they are updated in the Comos database.

Resource-constrained scheduling assigns resources to task assignments of (partially) scheduled tasks which do not have resources assigned yet. These assignments are exported to the Comos database. Accordingly, the workload which is scheduled for task assignments is added to the used workload in the work calendars of the resources.

The total duration of a task specifies the number of work days required for the task with respect to the work calendar of the task. It should be consistent with the planned start and end times of the task at any time, i.e. between the specified dates for planned start and end, the work calendar of the task should contain as many work days as specified by the total duration. The total duration can be manually set for tasks in the persistent dynamic task net. The defined values are imported into the memory representation of the task net. During resource-constrained scheduling, the total duration of a task may increase due to the scheduling of its task assignments and subtasks as described in Section 7.3.2. This increased value of the total duration is exported to the Comos database after scheduling. The user who initiated the scheduling pass is informed about all cases in which the total duration of a task has been increased.

The update of the total duration reveals a common issue related to the adaptation of workload, budget and duration. If the workload, budget or duration of a task are automatically increased due to a modification of the task assignments or subtasks or due to rescheduling of the subprocess, then the adapted values cannot be distinguished from manually set values afterwards. If a later plan change or a scheduling pass leads to a state in which the increased value of workload, budget or duration is not necessary anymore, the values are not reduced automatically, because they could be intentionally specified by the user. In all these cases, the user has to reduce the workload, budget or duration of a complex task manually to account for the decrease of the respective values for the task assignments or subtasks. In this sense, the automatic increase of the total duration of a task during scheduling is in line with the update of the total workload and budget of a task in case of manual changes to the planning data. The only difference to the case of workload and budget is that the computed duration of the task assignments and subtasks depends on the number of work days and available resources in the timeframe for which the parent task is scheduled. If the task had been scheduled for a different timeframe, the computed duration could have been shorter, so that an increase of the total duration of the parent task would not be required.

Altogether, computed constraint dates, planned dates, durations, workload distributions, and assigned resources are exported to the Comos database after scheduling. For not scheduled and (partially) scheduled tasks, only those properties are exported which can generally be changed by the scheduling algorithms.

7.6 Related Work

The related work on temporal analysis and scheduling can be coarsely divided into two different areas of research. First, there is the large scheduling community which is concerned with project scheduling, production scheduling in manufacturing systems, and the like. Second, there is a part of the process management community which has tried to incorporate the aspect of time into workflow management. The latter has adopted some techniques from the former, tackled problems which are specific to workflows, but also reinvented established solution concepts.

7.6.1 Resource-Constrained Scheduling

The scheduling research community has been quite successful in recent years, and scheduling research has had an increasing impact on practical problems [Smi03]. A large number of different approaches and algorithms exist for solving the different variants of the resource-constrained scheduling problem. Current scheduling techniques are capable of solving large problems and they are capable of generating schedules under broad and diverse sets of temporal and resource constraints. Furthermore, various different optimization goals can be handled. However, scheduling is not a solved problem as Smith argues in [Smi03]. The open research questions relate inter alia to the management of change and the integration of planning and scheduling. These problems have been addressed in this thesis by the solution for resource-constrained scheduling presented in this chapter and a change management procedure which will be presented in Chapter 9.

In this section, related work on scheduling is presented. First an overview over common general solution approaches and the different problem classes is provided. After a short coverage of robust scheduling approaches, related work on schedule repair is presented which addresses the problem of rescheduling at project runtime. Finally, the scheduling functionality provided by commercial project management systems is shortly reviewed.

A quantitative comparison of the scheduling algorithms presented in this thesis with related solutions from literature regarding the quality of the generated schedules has not been performed for two reasons. First, the algorithm presented in this thesis differs significantly from scheduling algorithms found in literature in that it takes the hierarchical structure of a task net and the execution state of the tasks into account. Therefore, the problem instances are not comparable and consequently neither are the generated solutions. Second, the optimization of the generated schedule with respect to the makespan of the project or the tardiness of tasks has not been the focus in this thesis.

Overview

Over the last decades, several different approaches have been developed for solving the resource-constraint project scheduling problem and its numerous variants. In Section 3.2, the general approaches have already been introduced. There are exact methods which aim at finding the optimal solution for the respective optimization problem. On the other hand there are heuristic approaches, which aim at finding good feasible solutions efficiently. The heuristics can be distinguished into constructive heuristics and improvement heuristics.

In [Haj97], Hajdu presents network scheduling techniques for construction project management covering the steps of network-based project planning, critical path analysis, and the precedence diagramming method. Finally, resource leveling and resource allocation are treated.

Kolisch and Padman provide in [KP01] an integrated survey of project scheduling. Different optimization objectives for project scheduling are presented and optimal approaches as well as heuristic approaches to solve the respective optimization problems are reviewed. In [KH98], Kolisch and Hartmann review the heuristic algorithms for solving the resource-constrained scheduling problem in detail. In particular, the parallel and serial schedule generation schemes are presented as examples for constructive heuristics. Simulated annealing, tabu search, and genetic algorithms are shortly described as examples for meta-heuristics. Finally, a computational analysis of the different algorithms is performed.

Herroelen, De Reyck, and Demeulemeester provide in [HRD98] a survey over developments with respect to the research on resource-constrained scheduling until the year 1998. In particular, research work addressing the generalized resource-constrained project scheduling problem (GRCPSP) and the resource-constrained project scheduling problem with generalized precedence constraints (RCPSP-GPR) is reviewed (cf. Section 3.2). A much more elaborate overview is provided in the research handbook on project scheduling by Demeulemeester and Herroelen [DH02]. It can be considered as a fairly complete overview over the different scheduling problems, optimization goals, exact and heuristic solution approaches, which even covers stochastic, robust, and reactive scheduling. A similarly but less extensive overview is provided by Klein in his dissertation [Kle00] in which he performs several computational experiments to compare the different approaches for solving the RCPSP and the GRCPSP with respect to their runtime efficiency.

The algorithm for resource-constrained scheduling presented in this thesis can be categorized as a constructive heuristic, which does not necessarily yield an optimal schedule with respect to the project’s makespan, but efficiently generates a good and feasible schedule. All related work on project scheduling does not explicitly address hierarchically structured task networks. It is generally assumed that precedence relationships are only defined for the tasks on the lowest level of a work breakdown structure and that the duration of complex tasks cannot be specified independently of their subtasks.

In contrast to the previously reviewed established approaches for solving the GRCPSP, the algorithm presented in this thesis can not only be used to generate an initial baseline schedule, but also to repair a project schedule at runtime. In project management and other application domains for scheduling algorithms, generated baseline schedules often have to be revised at project or process runtime. The possible disruptions which may occur at runtime cannot be determined it advance, which is subsumed under the term uncertainty. Surveys over the current research regarding scheduling under uncertainty are presented in [HL05, ALM$^+$05, OP09]. In contrast to approaches which aim at generating an optimal baseline schedule before the start of the project, the surveyed works particularly target possible disruptions at project runtime. The reviewed publications are classified into five fundamental approaches: reactive scheduling, stochastic project scheduling, proactive robust scheduling, fuzzy project scheduling, and sensitivity analysis, where the latter two are not related to the approach of this thesis.

Predictive-reactive scheduling is concerned with repairing the baseline schedule in case of disruptions. Simple techniques aim at restoring the consistency of the schedule and are referred to as schedule repair actions. An example is the well-known right shift rule which may however lead to poor results. At the other end of the spectrum of reactive scheduling approaches is the full rescheduling of the project part which remains to be executed. Match-up scheduling matches up with the baseline schedule within a given time frame. In contrast to predictive-reactive scheduling, completely reactive scheduling does not create a baseline schedule in advance but decisions are made locally in real-time, e.g. by using priority dispatching rules. In the following, the term reactive scheduling is used for predictive-reactive scheduling. The approach presented in this thesis can be categorized as a reactive scheduling approach. In case of dynamic changes, the task net is (partially) rescheduled to reestablish the consistency of the timing data.

Stochastic project scheduling is mainly concerned with solving the stochastic resource-constrained scheduling problem which aims at scheduling project activities with uncertain durations in order to minimize the expected project duration subject to zero-lag finish-start precedence constraints and renewable resource constraints. Due to the presence of both, resource constraints and random activity duration, no baseline schedules are used, but schedules are generated on-the-fly by applying scheduling policies. In that sense, stochastic project scheduling is a form of completely reactive scheduling. Without an explicit baseline schedule no advance commitments to subcontractors and customers can be made in a project. Hence, merely applying scheduling policies is not feasible for engineering projects, which is why the scheduling of dynamic task nets could not be based on stochastic project scheduling approaches.

Proactive robust project scheduling is an interesting approach for achieving ex ante stability in contrast to ex post stability which is achieved by reactive scheduling. With proactive robust scheduling, the baseline schedule is generated in a way that in case of disruptions during runtime like overdue activities, reactive measures are in most cases not necessary. For example, slack time can be distributed to protect several activities. The schedules generated by PROCEED for dynamic task nets are robust to some extent when buffers for the duration and workload of complex tasks have been planned. The hierarchical structure of a dynamic task net together with explicitly defined durations and workload for complex tasks which exceed the respective aggregated values of the subtasks leads to a stable baseline schedule for which changes to the dynamic task net at project runtime have only local impact.

Robust Scheduling

In [SW00, DGB01, HL04, VDH05, DDHVdV06, dVDH08], different algorithms for solution robust scheduling are presented. Solution robust scheduling aims at the generation of baseline schedules which have to change as little as possible in case of disruptions at runtime. The common optimization goal is to minimize the expected weighted deviation in task start times in the repaired schedule from those in the baseline schedule. This is achieved by scheduling activities later than their earliest possible start times in the baseline schedule to ensure that at runtime they can be started as scheduled even if preceding activities have been delayed. The presented algorithms include time buffers in a given schedule while the project due date remains respected.

Three slack-based techniques for generating robust schedules are reviewed in [DGB01]. Temporal protection simply extends task durations before scheduling to obtain so-called protected durations. Time window slack modifies the scheduling problem definition to ensure that each activity will have at least a specified amount of slack rather than extending task durations before scheduling. In this way, slack times are explicitly represented and can be reasoned about during scheduling. With the focused time window slack technique, tasks obtain more slack time when it is more likely that a disruptive event will occur before their execution, i.e. intuitively that tasks which are scheduled later receive more slack time. In [VDH05] resource flow networks which were introduced in [AR00] are used to determine the required time buffers. The remaining solutions for proactive robust scheduling include constructive heuristics, algorithms based on linear programming and constraint programming, as well as combinations of the latter two.

Related work on solution robust scheduling always assumes a flat task network. In this thesis, the hierarchical structure of dynamic task nets has been used as a means to increase the stability of the generated schedule. Instead of planning a time buffer between activities by scheduling tasks late, the approach presented in this thesis allows to incorporate a time buffer into the total duration of a complex task which enables the timely start of succeeding tasks even if subtasks have been delayed. In a way, the chosen approach can be considered as temporal protection in a hierarchically structured task net.

Reactive Scheduling

According to [Smi94], scheduling is an ongoing reactive process where evolving and changing circumstances continually force reconsideration and revision of pre-established plans. Traditional scheduling research has ignored this process view of the problem, focusing instead on optimization of performance under idealized assumptions of environmental stability and solution executability. Reactive scheduling is concerned with reestablishing the feasibility of a schedule which has been flawed due to disruptions at runtime. Several approaches also try to optimize the repaired schedule.

Sadeh et al. In [SOS93], the approach for reactive scheduling implemented in the MicroBoss scheduler is presented. Two levels are distinguished on which disruptions to the schedule can be handled. On the control level, small disruptions are handled by simple control rules. On the scheduling level, the schedule is repaired or re-optimized from a more global perspective. Inter alia two conflict propagation procedures are introduced which can be used as control rules for handling disruptions on the control level, but which can also be used to determine a set of tasks which shall be rescheduled by the global scheduling algorithm implemented in MicroBoss. The so-called *right shift rule* (or *right shifting heuristic*) moves forward in time all those tasks which are affected by the failure of a resource, either because they were assigned to the resource, or due to incoming control flows from shifted tasks. This technique may produce poor solutions, since it does not re-sequence tasks. A more sophisticated heuristic is the *right shifting and jumping* procedure which bumps forward tasks in time while jumping over some tasks which do not need to be moved. This procedure identifies a smaller number of tasks to be rescheduled but may still lead to poor results. The right shift rule has been one of the first heuristic operations for schedule repair. It is an example for a rule which identifies all affected tasks in case of a disruption.

The rescheduling approach implemented in PROCEED does not make use of this kind of simple repair rules. Defining a complete set of repair rules which covers all possible disruptions would have been tedious and futile since their application generally leads to poor results. Therefore, the constructive heuristic used for initial schedule generation has been extended to be applicable for schedule repair. The parallel heuristic generates a new schedule to handle all occurred disruptions, whereby preparing tasks can be moved forward in time. However, only those preparing tasks which are directly or indirectly affected by the disruptions are actually moved because for all other tasks the time and resource constraints have not changed and they are consequently scheduled for the same dates as before.

Bean et al. A heuristic approach for match-up scheduling was originally introduced in [BBMN91]. The heuristic for schedule repair tries to repair a schedule in case of disruptions in a way that the repaired schedule equals the original schedule after a specified time frame, i.e. only start times and resource assignments of tasks which lie in this time frame are changed. The application domain of [BBMN91] are automotive manufacturing problems. Therefore, resources are machines which perform jobs in sequence. The general procedure of the match-up scheduling algorithm (MUSA) consists of four steps. First, for every disrupted resource (machine), a minimum match-up time T is determined. Then, the jobs on each machine are re-sequenced before T whereby the deviation from the original schedule which is measured by a cost function may not exceed a specified threshold. If this is possible for all machines, then the algorithm is successfully terminated. Otherwise, the match-up times are incrementally increased and resequencing for individual machines is performed again. If the match-up time of a resource exceeds a global maximal value, jobs are reallocated between machines, i.e. the resource assignments are changed. The algorithm underlies the assumption that match-up with the original schedule is eventually possible. This is theoretically proven for the case that there is sufficient slack time in the original schedule. Since this cannot be guaranteed in general, the practical solution is to use a time limit after which the algorithm stops.

Some aspects of the described approach can be found in the approach for schedule repair presented in this thesis. The local rescheduling of a subprocess in PROCEED implicitly defines a time frame after which the original project schedule is left unchanged. In this way, match-up scheduling can be performed for a fixed match-up time. However, the match-up time cannot be arbitrarily chosen but the possible values are implicitly defined by the hierarchical structure of the dynamic task net. Furthermore, rescheduling in PROCEED does not automatically change resource assignments if the schedule cannot be repaired. The user has to change or reset certain task assignments manually.

Smith In [Smi94], Smith presents the OPIS scheduling system which has been designed to automatically revise schedules in response to changed solution constraints. The application domain of the OPIS system is manufacturing production management, i.e. the scheduling of tasks on a production line where the available resources are machines which perform the steps of the production process. Changes to solution constraints may arise due to machines break down, delayed arrival of materials, unexpected production demands, and the like. Due to the focus on production management, the rescheduling approach does not rely on human interaction with the system. The integration with user decision-making processes and the development of flexible interactive scheduling tools is mentioned as intended future research in the conclusion. The problem of reactive scheduling is broken down to two steps. First, the feasibility of the schedule has to be reestablished. Second, certain optimization objectives have to be met, e.g. compliance to due dates, minimization of tardiness of tasks, maximization of resource utilization. Schedule repair is achieved by incrementally revising schedules in response to changes to solution constraints. An approach from AI planning is applied where a control cycle is iterated until all conflicts have been resolved. One iteration consists of the steps event aggregation, event prioritization, event analysis, and subproblem formulation, where the last step yields a complex task which is performed to change the schedule. Different subtasks can be performed to repair the schedule. The so-called strategic alternatives include the application of the right shift rule to achieve feasibility and the application of the left shift rule to optimize the schedule. A specific characteristic of the approach is that the existence of a feasible solution is guaranteed by assuming that constraints are infinitely relaxable, e.g. due dates do not necessarily have to be met.

The approach for schedule repair presented in [Smi94] differs from the approach presented in this thesis in several ways. In [Smi94] a schedule is repaired by incrementally applying local repair actions, while the heuristic for resource-constrained scheduling described in this thesis is globally applied to reschedule complete dynamic task nets or subnets thereof. As a consequence, the latter can also be applied to initially generate a baseline schedule. The application domains of [Smi94] and this thesis differ. While the former addresses production management, the latter addresses scheduling as part of project management. As a consequence, the approach for schedule repair presented in [Smi94] does not involve human interaction, while the the approach presented in this thesis relies on manual management decisions in case that no feasible schedule can be generated. In particular, time constraints can only be relaxed manually.

Zhu et al. In [ZBY05] different possible disruptions at project runtime are analyzed and categorized, and a solution for schedule recovery is presented which is based on linear programming. Only the RCPSP is addressed, i.e. PDM and generalized precedence relationships are not considered but only sequential control flows. The proposed classification scheme for the different types of disruptions will be reviewed in the related work section of Chapter 9. Several recovery options are available to repair a disrupted schedule. Rescheduling of the defined tasks assigns new planned start and end times which takes into account changed constraints. Changes to the so-called resource-duration mode of tasks may resolve inconsistencies which include subcontracting and task cancellation. Finally, so-called resource alternatives increase resource availabilities. The recovery problem is defined as getting back on track as soon as possible at minimum cost, where cost is a function of the deviation from the original schedule. The problem is formulated as an integer linear program and is solved with a so-called hybrid mixed-inter programming/constraint programming procedure. A recovery window is defined which determines the time period after which the project shall again be executed as originally planned, i.e. the part of the original plan after the time window is not changed. Computational experiments have been conducted to determine the effects of different factors related to the recovery process. It has been found that schedule repair is harder to solve when the original schedule is optimal instead of just a good schedule. The earlier a disruption is detected, the easier is schedule repair.

The approach presented in [ZBY05] is related to the approach for resource-constrained scheduling presented in this thesis in several ways. The possible disruptions at project runtime which have been identified in [ZBY05] may occur in the same way in a dynamic task net, and most of the recovery options can be mapped to dynamic changes of a timed dynamic task net, as it will be shown in Chapter 9. The solution approach based on linear programming cannot be directly compared to the constructive heuristic presented in this thesis. However, the heuristic also allows to repair a disrupted schedule at project runtime. It addresses a variant of the GRCPSP while in [ZBY05] only the RCPSP is addressed. The local rescheduling of a subprocess in PROCEED implicitly defines a time window after which the original project schedule is left unchanged. Finally, the experimental results of [ZBY05] have been one of the major motivations why an optimal solution of the scheduling problem has not been aimed at in this thesis.

Wang Wang presents in [Wan05] a heuristic approach for repairing schedules which constitute solutions to the RCPSP. The scheduling problem is modeled as a dynamic constraint satisfaction problem where the due date constraint is considered as a hard constraint, i.e. the project deadline may not be violated. Unexpected changes during project runtime are regarded as additions or deletions of constraints to the problem. These changes include the shift of a task, a changed task duration, a change in certain resource capacities, and the addition or removal of a temporal constraint, e.g. the due date of a task. Two types of schedule conflicts may arise due to unexpected changes. Temporal conflicts are the violation of precedence constraints, earliest or latest start time constraints, or milestone constraints. A resource conflict exists when the aggregated resource demand exceeds the available capacity of a resource for a certain time. A reactive scheduling methodology based on meta-heuristic approaches is developed to repair a disrupted schedule. Thereby, the objective is to find a schedule that satisfies the temporal constraints and minimizes the resource constraint violation, i.e. resource constraints are not regarded as strict constraints. Two different meta-heuristics are applied for schedule repair. On the one hand a repair algorithm based on simulated annealing, and on the other hand a genetic algorithm. Computational studies are performed to test the performance of the proposed approaches. The computational time of the simulated annealing approach grows exponential as the time constraints become tighter while the computational time of the genetic algorithm remains constant. With respect to the quality of the solution, the simultaneous annealing approach performs slightly better.

The paper does not clarify why an approach based on meta-heuristics is particularly advantageous for schedule repair. Furthermore, it remains unclear in which way the implemented algorithms differ from similar improvement heuristics for solving the RCPSP. A common disadvantage of meta-heuristics is that it is difficult for the user to comprehend the applied changes for repairing the schedule and to track the problem solving process. While [Wan05] addresses schedule repair for the RCPSP, the resource-constrained scheduling algorithm presented in this thesis can be applied to instances of the GRCPSP. In both cases, the project deadline is a hard constraint. However, limited resource capacities are also handled as hard constraints by the scheduling algorithm implemented in PROCEED. From a practical point of view, it seems to be questionable that resource capacities can be arbitrarily exceeded in [Wan05].

Zweben et al. An approach for schedule repair similar to that of [Wan05] was presented much earlier in [ZDDD93]. The proposed system uses constraint-based iterative schedule repair, a technique that starts with a complete but possibly flawed schedule and iteratively improves it by using constraint knowledge within repair heuristics. The applied meta-heuristic is based on simulated annealing. In every iteration step, violations of resource constraints and so-called state constraints are detected and one violation is selected. Repair heuristics are applied to decide how to repair the schedule with respect to the selected constraint violation. The changed schedule is used for the next iteration or it is discarded depending on its quality and the escape function of the meta-heuristic.

In [ZDDD93], the authors argue why they applied iterative repair instead of constructive methods. The main argument against constructive methods is that previously scheduled tasks which are not directly affected by a constraint violation have to be unscheduled and rescheduled in order to repair the schedule, because they depend on tasks which are directly affected. According to the authors, determining these tasks is not straightforward. The heuristic for resource-constrained scheduling which is implemented in PROCEED solves this problem by using the execution states of tasks. Preparing tasks which have not been started yet by the assigned resources can easily be rescheduled without introducing to much nervousness due to a constantly changing schedule.

Van de Vonder et al. Van de Vonder et al. present in [dVBDH07] heuristic reactive project scheduling procedures. The objective of these procedures is to minimize the deviations between the original baseline schedule and the repaired schedule. The authors argue that solution robustness present in a predictive baseline schedule should also be maintained when the schedule is repaired due to disruptions at runtime. Reactive procedures should try to repair the predictive schedule in such a way that the included safety is preserved. For this purpose, the existing serial and parallel scheduling schemes for heuristic resource-constrained scheduling have been adapted. The new robust parallel scheduling scheme does not schedule tasks before their original planned start time. This is called railway scheduling. The new robust serial scheduling scheme tries to schedule the tasks as close as possible to their original planned start times but possibly earlier. The robust heuristics for schedule repair do not introduce any safety cushion against future disruptions themselves, but they merely try to maintain existing ones. In this sense, none of the procedures has a proactive nature.

During the review of related work on robust scheduling it has been argued that time buffers incorporated in the durations of complex tasks in a dynamic task net may lead to a certain schedule stability and are therefore a means for proactive robust scheduling. The heuristic for resource-constrained scheduling implemented in PROCEED naturally maintains the stability which results from these time buffers during schedule repair. In this sense the implemented heuristic could also be regarded as a procedure for robust reactive scheduling. Due to the hierarchical structure of the dynamic task net, it is in many situations not required to explicitly demand that tasks are not rescheduled to earlier dates. The successors of a complex tasks will not be scheduled earlier even if subtasks of the complex tasks may have been moved backwards in time. This surely does not cover all situations and it may very well happen that preparing tasks are scheduled for earlier dates. However, for running tasks the same rule applies as it has been introduced for the robust parallel heuristic in [dVBDH07].

Hao et al. The approach to resource-constrained project scheduling which is most related to the approach presented in this thesis has been developed at the same time and has just recently been published in [HSXW10]. The focus of this article is on multi-project scheduling, but also a solution for single project scheduling is presented. The problem of solving conflicts which arise at project runtime is addressed by proposing algorithms for rescheduling and interactive conflict resolution, so that the approach can be categorized as a reactive scheduling approach. A task network is defined containing the tasks in the project and the precedence relationships. Only sequential precedence constraints are defined in a task network which may however specify lag times. Furthermore, resource constraints and exclusive constraints can be defined for tasks where the latter specify that the respective task may not be executed in parallel to any other task in the project. A task network in [HSXW10] is not hierarchically structured but flat. The scheduling algorithm presented in [HSXW10] is related to the approach presented in this thesis in that task execution states are taken into account for scheduling. Scheduling is performed for a specified start date which is not necessarily the start date of the project. This scheduling start date is related to the date $\texttt{start} \in \texttt{Dates}$ used in Section 7.3 in that tasks which have not been started yet are moved to this date. Tasks which were previously scheduled earlier are moved forward in time. Tasks which were previously scheduled later are possibly moved backwards in time if this is consistent with all defined precedence constraints. Scheduling is only performed for a part of the overall task network. The first not yet started tasks in the topology of the task net are determined. They constitute the so-called heads of the rescheduled part. All succeeding tasks are also part of the task net to be rescheduled. In contrast, an arbitrary subprocess can be rescheduled by the scheduling algorithm presented in this thesis. The subprocess is simply specified by the parent task in the task net hierarchy. Furthermore, not scheduled tasks may exist which constrain the latest possible end time of the rescheduled subprocess, which is not possible in [HSXW10]. In [HSXW10], the earliest possible start times of tasks are computed which results in a project plan. An early schedule is computed, but earliest possible times and planned dates are not clearly distinguished. The algorithm for resource-constrained scheduling is only informally described in [HSXW10] and it is not based on any established scheduling scheme. Resource constraints are handled after time constraints, i.e. at first, only a time-feasible schedule is generated which is then corrected with respect to resource constraints by shifting conflicting tasks to later start dates. This approach produces suboptimal schedules with respect to the project's makespan and the resource usage which are probably worse than schedules generated by the parallel or serial scheduling scheme. Resource usage is defined per task but not for individual dates, so that only a uniform resource usage can be modeled.

With respect to multi-project management, an incremental and interactive algorithm is proposed in [HSXW10] which consists of the steps scheduling, conflict detection, and manual conflict resolution, which are performed iteratively, i.e. after the manual resolution of conflicts, the affected task network is rescheduled, conflicts are detected, and so on. The considered conflicts exist between tasks of different projects, and the goal of the algorithm is to align the schedules of different projects with each other, which have previously been generated by the algorithm described above. The iterative and interactive procedure can however be compared with the scheduling approach developed in this thesis. If resource-constrained scheduling fails for a part of a dynamic task net in PROCEED, then the user is informed about the reason for the failure which are in most cases unresolvable conflicts between time and resource constraints of different tasks. The user has to modify the dynamic task net in order to enable a successful scheduling run. Furthermore, the change management procedure for dynamic task nets, which will be introduced in Chapter 9, is also iteratively performed until all time and resource conflicts are resolved. The manual changes to the task network available in [HSXW10] are crushing the task duration, shifting tasks, releasing task constraints, prioritizing tasks, and overtime arrangement. All of these changes can also be performed in PROCEED for tasks in a dynamic task net.

Altogether, the approach for resource-constrained scheduling presented in this thesis goes beyond the solution presented in [HSXW10] in several ways. Scheduling is supported for hierarchically structured task nets and arbitrary subprocesses thereof. The life cycle of a task is more elaborate and particularly includes the state of replanning. The scheduling algorithm is based on the well established parallel scheduling scheme for solving the RCPSP. Resource usage of tasks can be specified for individual dates, and tasks and resources can have individual work calendars.

Scheduling Functionality of Commercial Project Management Systems

Soon after the introduction of the personal computer, software tools for project management emerged. Since the beginning of the 90s, project management software packages provided functionality for resource-constrained project scheduling. Nowadays, there is a huge number of different project management solutions available on the market which differ significantly in their range of functions and with respect to the quality of the generated schedules. Since PROCEED is effectively an extended project management system, according commercial software packages are shortly reviewed in this section.

Due to the time complexity of optimal solutions to the resource-constrained scheduling problems, heuristic algorithms are implemented in commercial project management systems (PMS). Which algorithms are implemented exactly is proprietary information. Therefore, a comparison of commercial PMS can only be performed by experimental analysis but not with respect to the details of the implemented algorithms. Several publications describe the results of comparative studies of commercial PMS [Kol99, MT01, TB09b, TB09a]. Kolisch compared in [Kol99] the resource allocation capabilities of several commercial project management software packages, including Microsoft Project 4.0, Primavera Project Planner 1.0, CA Super Project 3.0, and Time Line 6.0. The criterion for comparison was the quality of the generated schedule with respect to the project's makespan, i.e. the degree of variance from the optimal solution. An example input dataset was generated by varying several problem parameters including the number of activities, the number of available resources, and the network complexity of the projects to be scheduled. The input data was loaded into the PMS and a so-called full leveling procedure was performed in each of the PMS. For all evaluated software packages, the schedule quality decreased for an increasing number of activities, and an increasing number of resources. However, the network complexity did not have a significant influence. While different tools were superior for different parameter settings, on average the Primavera Project Planner performed best, followed by CA Super Project and Time Line, while Microsoft Project generated only average solutions. A similar comparison of commercial project management systems has been performed and presented in [MT01] for a different selection of PMS. The compared systems are Acos Plus 8.2, CA SuperProject 5.0, CS Project Professional 3.0, Microsoft Project 2000, and Scitor Project Scheduler 8.0.1. The input data has been generated by a systematic variation of the same problem parameters as in [Kol99]. The generated schedules are compared with the respective optimal solutions regarding the project's makespan. With respect to the mean and maximal deviation from the optimal solutions over all problem instances, Acos Plus performed best, followed by Scitor's Project Scheduler. Finally, the most recent comparison of commercial project management software packages has been performed by Trautmann and Baumann and presented in [TB09b, TB09a]. The experimental setup and the criterion for comparison are the same as in [Kol99, MT01]. The most recent versions of the following PMS are evaluated: Acos Plus, AdeptTracker Professional, CS Project Professional, Microsoft Office Project 2007, Primavera P6, Sciforma PS8, and Turbo Project Professional. With the respective default resource-allocation options chosen, Sciforma PS8 computes the best project schedules, followed by AdeptTracker Professional and Microsoft Project, while Primavera P6 produces below average results. Besides the default settings, the impact of different priority rules have been evaluated. In contrast to the situation where the default resource-allocation options were selected, Primavera P6 and Acos Plus outperform the other packages when for every problem instance the best results are taken from several different schedules obtained by applying different priority rules.

For the comparison of commercial PMS with PROCEED, the available features for project planning and scheduling provided by the PMS are more interesting than the quality of the generated schedules, because the research on project scheduling in this thesis did not focus the optimization of the project's makespan. In [MT01], some distinguishing features of the evaluated PMS have been described. Almost all software packages allow to specify the full set of PDM precedence constraints between tasks with the exception of CA SuperProject and Project Scheduler which do not support start-end relationships. All packages provide tools for managing resource-related costs including fixed costs per task and overtime costs. All evaluated project management systems allow to define different calendars for tasks and resources including working time and overtime per day, per week, and per month, as well as holidays. In this regard, PROCEED offers the same functionality as common commercial solutions. Acos Plus is the only tool which allows to specify maximal lag times for task dependencies but generally fails to find a feasible schedule in the presence of maximal lag times. This suggests that even the heuristic solution of the RCPSP-GPR is still beyond the capabilities of commercial project management systems. Nevertheless, commercial PMS are widely used in practice and are effectively applied for project management although maximal lag times cannot be defined. This suggests the assumption that the definition of maximal time lags is not essential in practice, in particular when due dates can be explicitly defined for tasks. For most tools evaluated in [MT01], resource allocation can be restricted to a subset of the defined tasks or to a specific time frame. Furthermore, Project Scheduler allows to restrict resource allocation to selected resources. In PROCEED, resource-constrained scheduling can only be restricted to a coherent subprocess but not to an arbitrary selection of tasks. Finally, different priority rules for resource-constrained scheduling including user-defined priority values can be selected in most tools. In PROCEED, user-defined priority values can be defined for tasks as well. The priority rules for resource-constrained scheduling however are fixed and cannot be changed by the user. In this respect, the scheduling functionality of PROCEED could be extended as already mentioned in Section 7.3.

In [Kle00], Klein presented an analysis of the main features common to all commercial project management systems at that time without referring to concrete products. In the project planning phase, a PMS can support the user in structuring, scheduling, resource allocation, and budgeting. With respect to structuring, the work breakdown structure and the organizational breakdown structure of a project can be defined. Furthermore, task durations, release and due dates, as well as precedence relationships can be specified. Most PMS support the PDM precedence relationships but no maximal lag times. The project can be represented as a activity-on-node network or as a Gantt chart. Consistency tests are performed during planning which detect, e.g., cycles in the project network. Scheduling in PMS refers to temporal analysis of the project network. A project calendar can be developed defining working periods and non-working periods. Critical path analysis can be performed considering release and due dates. Slack time calculations are usually restricted to computing the total float of the tasks. Regarding resource allocation, different resources can be defined and associated with their respective corresponding organizational units in the OBS. For renewable resources, individual calendars can be defined. Resource requirements of tasks can be specified, where only few PMS allow the resource usage of tasks to vary over their duration. A resource loading profile is generated for every resource based on the schedule obtained from critical path analysis. Resource conflicts can be resolved automatically by heuristically solving the GRCPSP. With respect to budgeting, the per period cost of human resources can be defined as well as costs for overtime work. In the project execution phase, the performance data of tasks can be input into the system. The progress of tasks is usually visualized in a progress Gantt chart. Some PMS are able to estimate the expected end times of delayed tasks. Most PMS are also able to track the cost performance. The budgeted costs of work performed are calculated automatically, and earned value analysis can be applied. Earned value analysis is only performed based on costs but not based on the planned workload. Finally, there are several functionalities provided by PMS which are not related to a specific project management phase. In some systems, what-if analyses can be performed for comparing the effects of different plan changes. Multi-project management can be supported in different ways, where the most convenient approach is based on a central resource data base for all projects. Finally, some PMS implement access control mechanisms to grant the rights for certain operations only to a subset of the users.

PROCEED offers most of the functionalities provided by state-of-the-art project management systems. In this sense, it reconciles the AHEAD approach with the common standards for project management in practice. In contrast to commercial PMS, PROCEED is based on a formally defined management meta-model. Dynamic task nets are superior to ordinary project plans in that they represent the current enactment state of an explicitly defined process. The incorporation of task execution states into project network diagrams has been the basis for an innovative approach for resource-constrained scheduling and schedule repair which is not supported by existing PMS. Furthermore, the modeling of actual data flow enabled the definition of a new progress measure for tasks in a development project which will be introduced in Chapter 8. With respect to resource allocation, PROCEED is superior to most commercial solutions with respect to automatic resource-constrained scheduling with non-uniform resource availabilities and requirements. Finally, workflow management has been integrated with project management in PROCEED, and specific solutions for the scheduling and monitoring of workflow-managed task nets have been developed. This functionality is not provided by commercial PMS to date.

7.6.2 Temporal Analysis and Scheduling of Workflows

The workflow paradigm emerged in the 1990s. Around the turn of the millennium, several research groups started to investigate the timing aspects involved with the enactment of workflows. Thereby, several established techniques from the domain of project scheduling were adapted and transferred to the domain of workflow management. Several related works are concerned with temporal analysis of workflow definitions, in particular critical path analysis [PEL97, EPR99, CSK02, SKK05].

A distinguishing characteristic of workflow management is that an explicit distinction is made between the build time and the runtime of a workflow. The creation of a workflow definition is considered as the build time, while the runtime covers the enactment of a concrete workflow instance. In between, there is the instantiation time when a new workflow instance is initially created from a workflow definition. Some related works are concerned with consistency checks at workflow build time only [CSK02, SKK05, CP09], while others also cover the scheduling and timely execution of tasks in running workflow instances [EPPR99, KK99, ZCP01, CC02]. Only few related approaches can be found which address rescheduling of workflow instances at runtime [CC02].

Marjanovic et al. describe in [MO99, SMO00] some challenges for temporal management in dynamic workflows such as temporal uncertainty and dynamic modification of a running workflow instance. They identify time management support that is required by a user of a WfMS to be able to effectively face these challenges, e.g. explicitly modeling unknown task durations, identification of affected temporal constraints by intended dynamic changes, as well as re-computation of expected start and end dates in case of performed changes. However, they do not describe how to realize this time management support.

Very little work has been performed with respect to resource constraints present in workflow definitions and instances [LYC04]. In this context, it is required to investigate the dependencies between different workflow instances [LY05].

Scheduling in the context of workflow management systems is sometimes understood as applying scheduling policies for assigning pending tasks to resources on-the-fly [BWE04, CP06]. These approaches are related to common solutions to the job-shop problem. In this case, tasks are not scheduled for specific dates. Therefore, the corresponding works are not reviewed in detail here.

Eder et al. Eder et al. made several contributions on how classical CPM and PERT computations have to be adapted for workflow definitions. Since workflow definitions can define alternative execution paths, the classical PERT approach is not directly applicable to workflows. Therefore, the extended PERT approach is presented in [PEL97]. Earliest and latest possible times are computed for the best case and the worst case execution, i.e. for the case that always the shortest path is selected at runtime, or respectively the longest path. Consequently, four dates are computed for the start and end events of tasks in a workflow. The authors claim, that the approach also covers loop structures in a workflow but they do not show how the ePERT computations work on a workflow definition which is not a directed acyclic graph. The timing information can be used in several ways at build time and at workflow runtime. At build time, static time failures, i.e. inconsistencies, can be detected and prevented. Furthermore, the workflow structure can be optimized with respect to timing issues. At workflow runtime, the time information can be used for pro-active and reactive scheduling as well as workflow controlling. Pro-active scheduling of workflows includes the selection of shorter alternative paths in case of delays and the skipping of non vital tasks. Reactive scheduling includes warnings and automatic compensation, e.g. by means of backtracking to a decision point where a shorter path can be selected. Finally, workflow controlling is concerned with the identification of late tasks but also with the collection of statistical data for the purpose of process improvement.

In [EPR99], the ePERT approach has been advanced. External deadlines are introduced which are defined by so-called fixed date constraints. Furthermore, minimal and maximal lag times between tasks can be defined. An extended critical path method computes task deadlines from external deadlines and lag times. At workflow build time, all dates are computed relative to the starting time of the workflow. These dates are transformed to actual dates upon workflow instantiation. During workflow runtime, the earliest and latest times are recomputed based on the actual dates of completed tasks.

In addition to the earliest and latest times, and the best case and the worst case performance, another dimension is introduced in [EPPR99]. Optional tasks can be defined, and a path through a workflow is called a fast path if all optional tasks are skipped, while on a slow path all optional tasks are executed. As a consequence, eight values are computed for the start and end dates of tasks.

In [EGP00], Eder et al. present a procedure for partially unfolding a workflow graph in order to solve some issues involved with explicit time constraints defined for tasks on alternative execution paths. In particular, superfluous time constraint violations are avoided which were detected by the approach presented in [EPR99] which treated parallel and alternative paths in the same way. [EP00] summarizes the results of the previous publications of the group. Furthermore, the concepts of early scheduling and late scheduling of tasks are introduced for workflow instances.

The results of Eder et al. can be compared to the approach to workflow scheduling and monitoring presented in this thesis. Like the original PERT and CPM approaches, no resource constraints are considered in the work of Eder et al. In PROCEED, the critical path analysis of a workflow-managed task net is only the preprocessing step for resource-constrained scheduling. The critical path computations for workflow-managed task nets presented in this thesis chose the shortest path from alternative paths instead of the longest. As a consequence, the computed constraint dates can be used as strict constraints during resource-constrained scheduling. If the longest paths were used, the computed constraint dates would be too restrictive. The computation of constraint dates for workflow-managed task nets in PROCEED can be performed at workflow runtime after workflow tasks have been completed. Thereby, neglected alternative branches are handled differently compared to the approach presented by Eder et al. Skipped tasks in a dynamic task net become zero-duration tasks and are not considered anymore during critical path analysis.

Son, Kim et al. In [CSK02, SKK05], a method is presented to systematically determine the critical path from a workflow definition. Block structured workflow definitions with synchronization edges are used as the underlying model. From every control flow block, the path with the longest average execution time is extracted and the control flow block is replaced by this sequential path. The replacement proceeds from the innermost to the outermost control flow blocks until all alternative branching and loop constructs are transformed. The resulting critical path is the path with the longest average execution time through the workflow. The approach allows to assess the criticality of workflow tasks in a workflow definition. However, it does not allow to compute earliest possible start times which can be used as strict constraints for workflow instance scheduling. Since the paths with the longest average execution times are extracted from control flow blocks, resulting earliest possible start times of succeeding tasks would be too late and succeeding tasks could be started earlier if shorter paths are selected at workflow runtime.

The intentions of temporal analysis of workflows in [CSK02, SKK05] and in this thesis are different. While in the cited articles the workflow definition is analyzed to identify critical tasks in a workflow definition independently of an actual workflow instance, in this thesis critical path analysis is performed for workflow instances as a preprocessing step for resource-constrained scheduling. In this context, it is important that the computed constraint dates are still valid during resource-constrained scheduling which is not necessarily the case for the approaches of [CSK02, SKK05]. Therefore, the computation of earliest and latest possible times for workflow instances presented in this thesis uses the alternative paths with the shortest average computation times.

Kafeza and Karlapalem Another early publication on time management in workflows is [KK99]. Temporal constraints can be defined between the start and end events of tasks which cover the PDM precedence constraints but without explicit lag times. This is one of few approaches to workflow management which is not restricted to sequential control flows. The consistency of a workflow definition with respect to the temporal constraints is checked by generating the corresponding event graph which connects the start and end events according to the time constraints. If the event graph is acyclic, then the temporal constraints are consistent. Besides a general system architecture for a temporal workflow management system, different scheduling policies are proposed. Thereby, global schedulers are distinguished from agent schedulers. A global scheduler assigns tasks to agents, i.e. resources, and the resources have their own scheduling policies to determine the order in which they execute the assigned tasks. A pseudo-static global scheduler schedules the tasks until the next alternative branching construct statically. After that, it waits until the decision for one of the alternatives has been made and then schedules the next portion of the workflow until the next alternative branching. In contrast, a dynamic global scheduler continuously computes the set of enabled but not yet assigned tasks and dispatches them to resources. The proposed agent schedulers are a first-in-first-out scheduler, an earliest-deadline-first scheduler, and a scheduler that selects the task with the shortest duration first. Different combinations of the dynamic scheduler and the agent schedulers achieve different scheduling goals, e.g. meeting the temporal constraints while missing some deadlines.

The presented approach is interesting insofar as automatic workflow enactment is realized in the presence of PDM precedence constraints. In PROCEED, the usage of control flow types is constrained to sequential control flows in workflow-managed task nets, because the Windows Workflow Foundation is used for workflow enactment. Furthermore, the scheduling algorithm assigns planned dates to all tasks and the assigned resources are expected to start and commit their tasks according to the planned dates. In the context of human-intensive development processes it would not make sense to define scheduling policies for agents, i.e. resources, because the human resources decide for themselves when to start a task. The dispatching of tasks to resources comparable to the dynamic scheduler in [KK99] is realized by setting the execution states of the tasks to *Waiting*, so that they are not scheduled for concrete dates.

Zhuge et al. In [ZCP01], the aspect of globally distributed workflow enactment over different time zones is addressed. Several time axes are modeled which are mapped to one common reference axis. The notion of flow duration is introduced which incorporates the lag times between tasks which arise due to the transmission of control from one site to another. An approach for temporal consistency checking of workflows during build time and runtime is presented.

The aspect of different time zones has not been explicitly addressed in this thesis. A global reference calendar would have to be introduced to make the dates of individual resource calendars comparable. For certain time zones, the working hours scheduled for a particular date in the global calendar would have to be mapped to two half days in the global calendar.

Combi et al. Combi and Posenato introduce in [CP09] the concept of *controllability* for workflow definitions. A workflow definition is controllable if every possible workflow execution is consistent with all defined time constraints as long as all task durations lie between specified minimal and maximal values. This is related to the existence of a free schedule as it has been introduced in [BWJ02]. In [CP09], a conceptual model for workflows is introduced in which time constraints between the start and end times of different tasks can be defined. The introduced constraints together have the expressiveness of generalized precedence constraints with minimal and maximal lag times. An algorithm is presented, how the controllability of a block-structured workflow definition without loops can be verified efficiently. Because controllability is a stronger property of a workflow definition than the consistency of the time constraints, the algorithm implicitly checks the latter, too.

The approach is related to the evaluation of timing consistency constraints for workflow-managed task nets as defined in this thesis. However, the intentions of the two approaches differ significantly. While in [CP09], a workflow definition is analyzed in order to obtain propositions about all possible instances, in PROCEED concrete workflow instances are analyzed by evaluating timing consistency constraints and performing critical path analysis. The verification of controllability has not been the focus in this thesis. In PROCEED, expected durations of the defined tasks are used to determine their earliest and latest possible start times in order to decide whether the tasks can be scheduled in a time-feasible way. The fulfillment of timing consistency constraints is checked for the computed constraint dates and the planned dates of a workflow instance with respect to the expected durations. In [CP09], no fixed task durations are assumed. On the other hand, strict upper bounds for the task durations are defined and it is assumed that they are never exceeded. In the context of development processes it is infeasible to enforce a maximal task duration when the task is performed by a human resource. Therefore, a more practical approach has been followed in this thesis, where the constraint dates are recomputed at workflow runtime if the expected durations are exceeded. Finally, the approach for temporal analysis of workflows presented in this thesis explicitly takes loops in a workflow into account which has not been done in [CP09].

Li, Yang et al. Li and Yang cover two aspects regarding time management in workflows which have been neglected by other research groups, namely resource requirements of workflow tasks and the interdependencies between concurrently executing workflow instances. In [LYC04], they investigate resource dependencies between different tasks in a workflow. They present an algorithm for checking the consistency of time and resource constraints in a workflow definition at build-time. The algorithm is based on computing the active interval of each task which is defined as the time period between the earliest start and latest end time of a task. In [LY05, LY04], the approach is extended to checking the feasibility of time constraints for concurrently executing workflow instances which require the same resources.

In PROCEED several workflow instances requiring the same resources may be executed concurrently as well. All workflow instances are embedded into the overall dynamic task net which represents the project plan. Resource dependencies between workflow instances are taken into account during resource-constrained scheduling. A time- and resource-feasible schedule defines a possibly interleaved sequencing of workflow tasks belonging to different workflow instances. If such a schedule exists, then all defined time constraints for the running workflow instances can be satisfied in the current situation. In this sense, the parallel heuristic solves the same problem as the algorithm of Li and Yang as presented in [LY05, LY04] while it is based on an established approach for project scheduling.

Chan and Chung The IPPM system which has already been reviewed in Section 6.4.1 integrates project and workflow management functionality. In [CC02], different methods are proposed to determine the duration of alternative branching constructs and loops. For alternative branching and parallel constructs, the branch with the maximal duration can be scheduled, or the average duration of all branches can be used. Finally, the weighted average of the durations can be computed weighted by the probabilities for the selection of the respective branches. In PROCEED, another possibility is added, namely to use the duration of the most probable branch. Furthermore, different durations are used for CPM and resource-constrained scheduling in PROCEED. For CPM, the shortest durations are used while for resource-constrained scheduling the longest or most probable durations are used. In [CC02], a so called prudent-estimated task replaces all optional tasks and future iterations of a loop. Its duration is defined as the expected duration of all expected further iterations. This is handled differently in PROCEED, where the first versions of iterated tasks are scheduled with expected durations which incorporate possible future iterations. Every workflow task in a loop carries the information about possible further iterations instead of introducing one additional task which covers the expected further iterations of the whole loop.

7.7 Conclusion

In this chapter, algorithms for critical path analysis and resource-constrained scheduling of dynamic task nets have been presented. The algorithms are implemented in PROCEED and enable the user to perform automatic scheduling of the defined tasks based on the specified planning data, resource and time constraints. In the following, the specific characteristics of the developed algorithms are reviewed which distinguish them from related work. Furthermore, a short outlook is provided on how the approach for scheduling dynamic task nets could be extended.

Characteristics of scheduling approach The algorithm for critical path analysis is based on the classical approach which involves forward and backward scheduling of a task net [DH02]. The heuristic for resource-constrained scheduling is based on the parallel schedule generation scheme presented in [KH98, DH02, Kle00]. Several adaptations and extensions of the algorithms were required for their application to dynamic task nets. These modifications cover the following aspects.

- Hierarchical structure of dynamic task nets,
- End-end task dependencies,
- Individual work calendars for tasks and resources,
- Dynamic computation of task durations,
- Utilization of CPM results for resource-constrained scheduling,
- Heuristic schedule repair considering task execution states,
- Partial rescheduling of dynamic task nets,
- Scheduling of workflow instances.

Dynamic task nets are structured hierarchically and timing constraints apply for the task-subtask relationship. Therefore, the critical path method as well as the heuristic for resource-constrained scheduling have been extended to cover hierarchical task net structures. The developed hierarchical critical path method performs a depth-first traversal of the dynamic task net, and the computed constraint dates of a complex task depend on the respective computed constraint dates of the subtasks. During resource-constrained scheduling, a task only becomes eligible when its parent task is already scheduled for its start date. The planned end time of a complex task is updated to the planned end time of a subtask if the latter is later than the former. Furthermore, control flows between tasks contained in different subnets, i.e. task realizations, are taken into account during scheduling. Finally, task assignments and workload can be defined for complex tasks in a dynamic task net. The workload which is associated with complex tasks is distributed during resource-constrained scheduling. This is useful to model administrative work for a complex task, or to model an incomplete work breakdown structure where the total workload of a complex task includes the workload of subtasks which have not yet been defined yet.

Critical path analysis and resource-constrained scheduling for the cases of PDM task relationships and generalized precedence constraints have been addressed in related work [EK92, Wie81, DH02, Kle00]. However, simultaneous and standard control flows require specific attention during resource-constrained scheduling if the task net is hierarchically structured and if the task durations are dynamically determined. In both cases, the final planned end time of a task is not known when the task is scheduled and backtracking has to be performed when the final planned end time is inconsistent with the planned end time of a simultaneous or standard predecessor. In case of atomic tasks, the forbidden set is used to delay the scheduling of an inconsistent task. The planned end times of complex tasks are checked after a complete scheduling pass which may lead to another scheduling pass with additional constraint dates.

During resource-constrained scheduling, the planned workload of tasks and task assignments is distributed according to individual work calendars of the tasks and resources. The availability of the assigned resources may vary for different dates. As a consequence, the resource consumption of task assignments may result in non-uniform workload distributions.

The duration of a task is derived during resource-constrained scheduling from the workload distributions of its task assignments, the duration of the subprocess defined by its subtasks, and its total duration. As a consequence, the duration of a task is not fixed before scheduling but may vary depending on resource availabilities.

The results of critical path analysis are used as constraints during resource-constrained scheduling when CPM has been performed for a manually specified project deadline. Thereby, all manually set release and due dates of tasks are taken into account. If no time and resource feasible schedule can be found which respects the project deadline, scheduling fails. This accounts for the common requirement in plant design projects that the project deadline must not be violated.

A dynamic task net can be partially rescheduled at project runtime. The rescheduling capabilities of PROCEED address one of the open problems of scheduling research mentioned in [Smi03], namely the management of change during project execution. The planned and actual performance of a process are explicitly distinguished in PROCEED. To adapt the plan to the actual performance of the process, manual management decisions are required which may, e.g., align the total duration and total workload of delayed tasks. After these operations, the schedule can be automatically adapted to the changed planning data. A dynamic task net can be partially rescheduled at runtime. The root of the subnet to be rescheduled can be explicitly specified. Furthermore, certain tasks are excluded from scheduling due to their planning data, granularity, purpose, or execution states. In particular, the exclusion of waiting, suspended and terminated tasks from rescheduling distinguishes the developed approach from related work on schedule repair. These tasks are not rescheduled but impose constraints on the rescheduled tasks. Among the rescheduled tasks, active and replanned tasks are treated in a special way in that their planned start times are not changed. This accounts for the practical requirement that the planned start time of a started task must not be changed anymore because the assigned resource already started working on the task.

Finally, the scheduling of workflow-managed task nets takes the existence of alternative branches and loops into account. Related work on workflow scheduling is often confined to temporal analysis of the workflow definition. The approach presented in this thesis addresses the scheduling of workflow instances thereby taking resource constraints into account. Workflow-managed tasks are automatically rescheduled at workflow runtime depending on the decisions made by the workflow engine.

Open problems The rules to compute the priority list for the parallel scheduling scheme are fixed in the current implementation of PROCEED, which is an unnecessary restriction. Different alternatives for the prioritization of tasks could be provided and the user could even be allowed to define his own comparison methods for the priorities of tasks. Different priority lists have an effect on the scheduling result. It was out of the scope of this thesis to investigate, which prioritization rules lead to better results of the scheduling algorithm.

The implemented parallel heuristic for resource-constrained scheduling may not lead to an optimal schedule in terms of the project makespan. It still remains to be shown how good the generated initial baseline schedules are compared to optimal solutions with respect to the project's makespan. Furthermore, it has not been quantitatively measured, how good the scheduling results are with respect to the robustness and flexibility of the schedule. This kind of computational studies was out of the scope of this thesis.

The implemented resource selection during resource-constrained scheduling aims at a balanced resource usage but an optimal result is not guaranteed. Optimization goals which aim at a uniform resource utilization and a balanced resource usage have not been investigated.

Altogether, the computation of an optimal schedule with respect to different optimization goals was not the focus of this thesis. The focus of the thesis lied on investigating the possibilities for dynamic rescheduling of task nets at project runtime while taking task execution states into account.

Kapitel 8

Monitoring a Development Process

A key feature of process management systems is the *monitoring of running processes*. Especially in plant design projects it is essential to have accurate information about the current status of the enacted development process. This information is even more valuable if it is available at short notice. The PROCEED system supports incremental progress measurement of engineering design projects. At any time during the project, the current status can be retrieved, in which the most recent developments are reflected. The progress measurement functionality of PROCEED facilitates quantitative statements about the actual progress of tasks and allows comparing these to the planned progress which is derived from the project schedule. In this way, critical delays can be detected early.

The general approach for time management and progress measurement realized in PROCEED is illustrated in Figure 8.1. The estimated workload and budget of tasks and their expected durations are planned top-down in a dynamic task net. From this planning data, a baseline schedule is computed. At project runtime, the degree of completion is determined for every task in the project, and these values are aggregated on higher levels of the hierarchically structured dynamic task net.

In plant design projects, often reliable estimates are available for the required workload and costs for the different engineering phases and main tasks. The workload and costs can be distributed top-down over the work breakdown structure in the planning phase of the project. Thereby, the workload of a task is distributed to the subtasks. However, not the whole workload has to be distributed. On the one hand, workload can be associated with task assignments of a complex task. On the other hand, workload can remain at a task without being assigned to either subtasks or task assignments. This workload can be used at later stages of the project planning phase and even during project runtime for new task assignments, additional subtasks or as additional workload for existing subtasks. In addition to workload and budget, the durations of the main tasks in the project and the number of required resources can be estimated.

The scheduling algorithms implemented in PROCEED generate a time and resource feasible baseline schedule from the planning data. Thereby, eligible resources are assigned to the planned tasks in the dynamic task net. The schedule is the basis for progress measurement.

The *degree of completion* (DOC) of tasks in a dynamic task net can be calculated by means of several different progress measures which differ regarding the trade-off

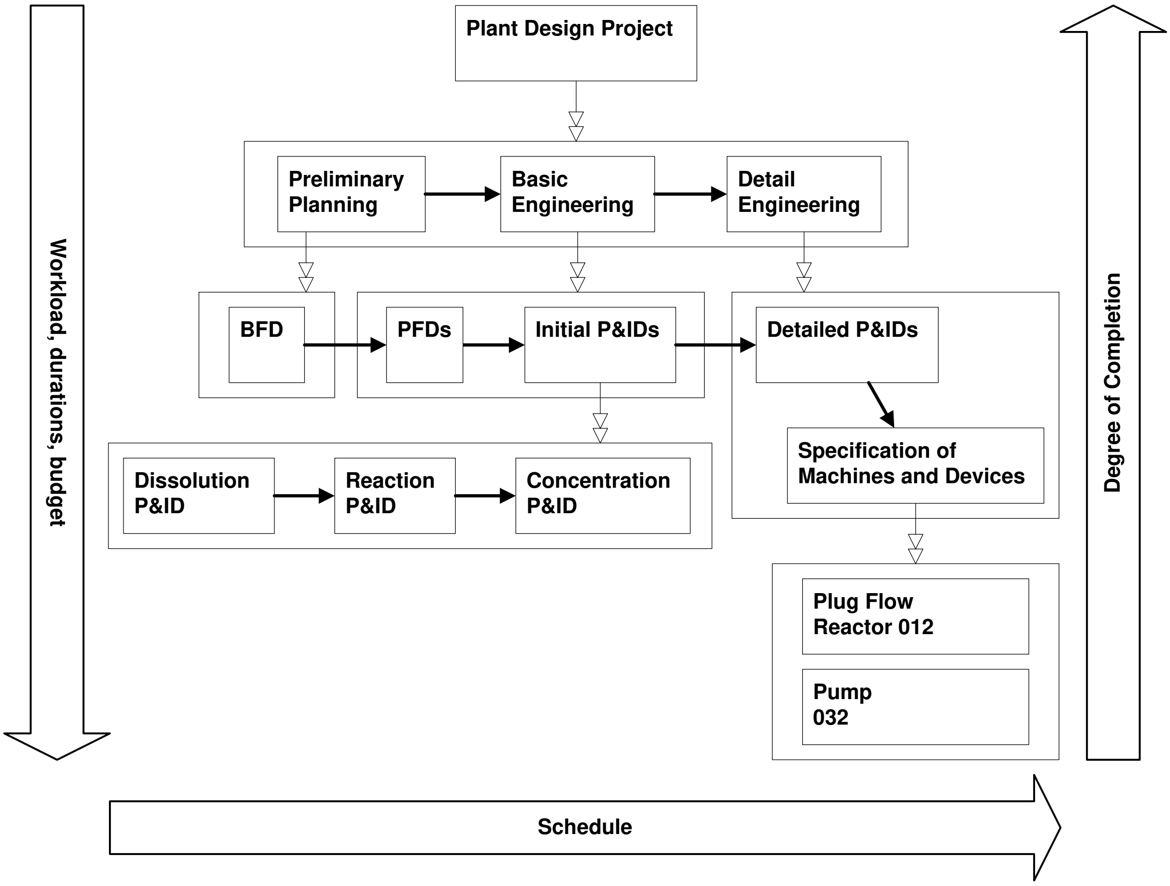


Abbildung 8.1: Integrated approach to planning, scheduling, and monitoring.

between accuracy and measurement effort. In general, the actuality of the DOC of a task is more important than its accuracy [PR05]. Therefore, it would not make sense to spend much effort for measuring the progress of rather small and short-lived tasks. However, if the DOC of a complex, long-running task is measured by a too simple measure, then its accuracy is too low to be of any use. Consequently, progress measures with little measuring effort but low accuracy are generally used for small tasks in the project, which occur in large numbers, e.g. the specification of measuring points for the design of a chemical plant. Progress measures with high accuracy but with a comparably high measuring effort are used for critical and complex tasks in the project, e.g. the creation of a process flow diagram.

In PROCEED, the decision of which progress measure to use can be made for individual tasks or it can be specified for task types. As for other property values like the expected duration, the progress measure defined for a task type serves as the default value for all instances of this type. However, the selected measure can be changed for each task instance individually—even at runtime of the task.

Specific progress measures are used to aggregate the DOCs of subtasks at complex tasks. In this way, the DOCs of individual tasks are aggregated and propagated upwards in the hierarchical structure of the dynamic task net to finally arrive at DOCs for the main project phases. Progress measurement may reveal bad performance of the development process which may require replanning and rescheduling at project runtime. Thereby, the structure of the dynamic task net may be modified, the scheduled dates may have to be adapted, and the DOCs of tasks may change. However, certain structural changes to a dynamic task net at project runtime may leave the DOCs of the respective parent tasks unchanged depending on whether time and workload buffer is available.

The DOC only measures the progress of a task in terms of its defined scope. It does not directly show, if the schedule will be met and the task will keep to the budget. Therefore, the actual performance of tasks has to be compared with the plan. In PROCEED, *earned value analysis* is applied for this purpose (cf. Section 3.3.2). For every task in a dynamic task net, the *Schedule Performance Index* (SPI) and the *Cost Performance Index* (CPI) are calculated. Depending on user specified thresholds for the SPI and CPI, tasks are marked in the PROCEED management views according to their degree of delay or budget overrun. Based on the SPI and CPI, the forecasted end time and the expected budget at completion are computed for every task, respectively.

Altogether, the approach for monitoring of engineering design processes is integrated in two ways. The planning and scheduling functionality is integrated with the progress measurement functionality, and several different progress measures are integrated to measure the progress of a process model instance.

Besides the calculation of performance indicators and their visualization in the management views, PROCEED additionally comprises a dedicated user interface for project status control. The development of this user interface was motivated by the specific characteristics of plant design processes: a huge amount of rather small engineering tasks and a high degree of simultaneous engineering. As a consequence, the presentation of tasks in network diagrams, Gantt charts, and task lists, and the indication of delays in these representations is not sufficient to assess the status of a project. Therefore, a concept for the *multidimensional visualization* of project management data has been developed to provide a condensed overview over all tasks in a project with their planned and actual workload and their current status. The processing of the management data for visualization is done in a *project data warehouse* to which the management data is exported in regular intervals. Additional views are provided to analyze the history of plan changes at project runtime which can be derived from the project plan snapshots in the data warehouse.

This chapter is structured as follows. Section 8.1 describes and compares the different progress measures which can be used for individual tasks in a dynamic task net to compute their degree of completion. Section 8.2 describes, how earned value analysis is applied in PROCEED to determine the degree of delay and budget overrun of a task. Section 8.3 deals with the project data warehouse and the user interface for visual project status analysis. Related work on all aspects of process monitoring in PROCEED is discussed in Section 8.4.

8.1 Progress Measures

In project controlling theory and practice, several different progress measures are used, which range from simple heuristics to more accurate, complex measures [PR05]. The approach to progress measurement in PROCEED incorporates the most common of these techniques and complements them by new measures which are specific for dynamic task nets [Dre09, HW09, HW11].

A progress measure is basically a calculation method for the *Degree Of Completion* (DOC) of a task. The terms progress measure and calculation method will be used synonymously in the following. Common to all different calculation methods for the DOC of a task is, that the value is 0% if the task has not been started yet, and 100% if it has been terminated. However, they differ in how they calculate the DOC of an active task. Hence, in this section each progress measure is defined by a formula which is used to compute the DOC of an active task.

In the following, the different progress measures available in PROCEED are described in detail. For each measure an example case is given for which the measure is most appropriate. The measures are evaluated regarding their required effort and measuring accuracy and rated with one of the values *Lowest*, *Very Low*, *Low*, *Medium*, *High*, *Very High* or *Highest*. Finally, the measures are compared with each other.

The set of progress measures which are available in PROCEED is coarsely divided into two subsets. On the one hand there are measures, which do not take the progress of possibly existing subtasks into account. On the other hand there are measures, which rely on the execution states and DOCs of the subtasks. In the following, we call the former *black-box* and the latter *white-box progress measures*. Even for black-box progress measures, details of the respective task like planned and actual workload or the estimated remaining workload are taken into account. The term black-box here only refers to the fact that the progress of the subtasks is neglected.

8.1.1 Black-Box Progress Measures

The majority of progress measures found in literature fall into the category of black-box progress measures. In the following, the adopted measures are reviewed and formally defined in the notation used for all measures presented in this section. Furthermore an additional progress measure is presented which integrates the common practice to measure the progress of a plant design project by means of document states with the concept of dynamic task nets.

Established progress measures from practice The simplest technique is the calculation method *Absolute*. Here the DOC of a task is 0% as long as it is not completed. Hence we have for an active task $\mathtt{t} \in \mathtt{Tasks}$

$$\mathtt{DOC}_{\mathtt{Absolute}}(\mathtt{t}) = 0$$

This heuristic is suitable for small tasks in a project which occur in large numbers as subtasks of a common parent task, e.g. the specification of measuring points of a process plant. If the DOCs of these subtasks are aggregated at the parent task, this results in a fairly accurate DOC for the parent task. There is no measuring effort required for this measure. However, for the individual task it also has the lowest accuracy.

The *Start/End* heuristic returns a constant value $\mathtt{k}$ with $0 < \mathtt{k} < 1$ for the DOC of an active task.

$$\mathtt{DOC}_{\mathtt{Start/End}}(\mathtt{t}) = \mathtt{k}$$

This method is slightly more accurate than the method *Absolute*, since active tasks are reflected with a non-zero contribution in the aggregated DOC of the parent task. At the same time, this method requires more effort, since the constant value $\mathtt{k}$ has to be defined. Nevertheless the effort is very low, since $\mathtt{k}$ can be defined for a task type and does not need to be changed for individual instances of the type.

Another common technique is the *Time Proportional* calculation of the DOC of a task. This technique underlies the assumption that the work defined by the task, for which the progress is measured, is carried out exactly as planned. This is a reasonable assumption for continuously performed work during the project like management activities. These activities do not have a specific goal or deadline, but the work is carried out as required. If a task is defined for this type of work in the dynamic task net, the *Time Proportional* calculation method is the most adequate. Let $\mathtt{w}_\mathtt{t}$ be the working days of a task $\mathtt{t} \in \mathtt{Tasks}$ between its planned start and end times and let $\mathtt{p}_\mathtt{t}$ be the number of actually passed working days between the planned start and end times of $\mathtt{t}$ until the current date. Then the DOC of an active task is computed as follows.

$$\mathtt{DOC}_{\mathtt{TimeProportional}}(\mathtt{t}) = \frac{\mathtt{p}_\mathtt{t}}{\mathtt{w}_\mathtt{t}}$$

If the planned end time of the task has already passed, the DOC is 100%. During the execution of the task, the progress increases linearly over time. This is the only calculation method which does not take the actual performance of the task into account but only the planned dates and the passed time. If the method is applicable for a task, then its accuracy can be evaluated as *Medium* while there is no measurement effort involved.

The DOC of a task can also be determined simply by an estimation made by a domain expert. The corresponding measure is called *Expert's Estimate*. In theory, this method can be very accurate if sufficient information about the current status of the task is available and if the expert has enough experience. However, there are several social aspects which may bias the estimate [SKW07]. Therefore the accuracy of this measure may vary from *Low* to *High*. The measure requires comparably much effort for providing a good estimate, so that it has to be rated with *High*. Finally, an expert usually cannot provide an estimate on each working day over the complete duration of the task, which means that an up-to-date DOC is not available at any time. Let $\mathtt{K}_\mathtt{t} = \{\mathtt{k}_1, \ldots, \mathtt{k}_\mathtt{n}\}$ be the set of subsequent DOC estimates for a task $\mathtt{t} \in \mathtt{Tasks}$, where $\mathtt{k}_\mathtt{n}$ is the most recent one. The current DOC of the active task $\mathtt{t}$ is

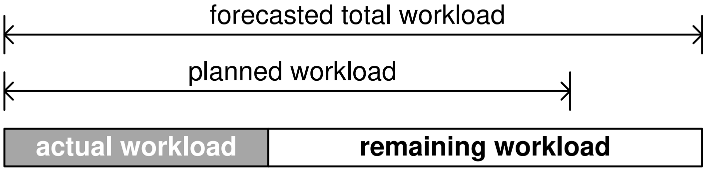


Abbildung 8.2: Actual, planned, and forecasted total workload of a task.

then calculated as:

$$\mathtt{DOC}_{\mathtt{ExpertsEstimate}}(\mathtt{t}) = \mathtt{k}_\mathtt{n}$$

The direct estimation of the DOC of a task may only be done by the resource that is responsible for managing the measured task or one of the superior tasks in the task net hierarchy (cf. Section 5.5). The progress measure is appropriate for long-term, critical tasks in the project whose DOC cannot be calculated based on the DOCs of the subtasks and where the responsibility for the estimated DOC has to be clearly defined.

The calculation method *Estimate to Completion* calculates the DOC of a task based on the estimated remaining workload and the accumulated actual workload up to the current date. It requires a time registration system with which the assigned resources of the task can track their actual workload. As described in Section 5.1.3, the assignment of resources to tasks is explicitly modeled in the form of task assignments in PROCEED. Actual workload can be booked on these task assignments. For a task $\mathtt{t} \in \mathtt{Tasks}$ with task assignments $\mathtt{a}_1, \ldots, \mathtt{a}_\mathtt{n} \in \mathtt{t.TaskAssignments}$ let $\mathtt{A}(\mathtt{t}) = \sum_{\mathtt{i}=1}^{\mathtt{n}} \mathtt{A}(\mathtt{a}_\mathtt{i})$ be its actual workload consisting of the registered actual workload of the task assignments, and $\mathtt{R}(\mathtt{t})$ the estimated remaining workload for the whole task. Then the DOC of the task is computed as

$$\mathtt{DOC}_{\mathtt{EstimatetoCompletion}}(\mathtt{t}) = \frac{\mathtt{A}(\mathtt{t})}{\mathtt{A}(\mathtt{t}) + \mathtt{R}(\mathtt{t})}$$

The sum $\mathtt{A}(\mathtt{t}) + \mathtt{R}(\mathtt{t})$ is the *forecasted total workload* of the task $\mathtt{t}$ assuming that no subtasks exist. Figure 8.2 shows an example where the forecasted total workload of the task is greater than the planned workload. In general, the forecasted total workload can be less, equal or greater than the planned workload, where the latter case indicates, that the actual performance does not meet the plan. As for the measure *Expert's Estimate* the remaining workload of a task may only be set by an authorized resource. The estimation requires much effort, comparable to the direct estimation of the current DOC. Together with the tracking of the actual workload this results in even more effort compared to the measure *Expert's Estimate*. Hence the required effort is rated as *Very High*. The accuracy of the measure is comparable to the measure *Expert's Estimate* since an estimation has to be made and this may be equally biased. Its accuracy may therefore range from *Low* to *High*. As for the measure *Expert's Estimate*, the progress degree of the task is not always up-to-date. However, it may be more practical to give an estimate of the remaining workload

P&ID (150 PH)				
DC	PC	IC	IFR	IFC
20%	40%	70%	80%	100%
30 PD	60 PD	105 PD	120 PD	150 PD

Tabelle 8.1: States and progress degrees for a P&ID

on a daily basis than to estimate the DOC directly with this frequency. The actual workload of subtasks in not considered in this progress measure. For tasks with subtasks, an aggregation method should be used. Therefore, the estimated remaining workload of a task is incorporated in one of the white-box progress measures as well.

Progress measurement based on document states In plant design projects, a common practice is to measure the status of the project by means of the documents which have to be created. For the different types of documents like flow sheets, device specifications, equipment lists, layout plans, etc., states are defined. Every document state is associated with a degree of completion of the document. The document states and progress degrees are usually defined in company-wide standards [Tec10]. In a concrete engineering project, a so called list of deliverables is assembled. The documents which have to be delivered are weighted by their required workload. During the runtime of the project, a document can reach a new state when a new revision is produced and released. At regular intervals, the progress values of the different documents which are derived from the current states are aggregated which results in a degree of completion of the whole project.

The disadvantage of this way of progress measurement is that it is completely detached from project planning. It is possible to derive a degree of completion for the whole project from the current document states, but if the calculated value indicates a delay of the project, the reason for this delay cannot be associated with specific tasks in the project plan. In the approach presented in this thesis, this problem has been solved by integrating the described progress measurement based on document states with the actual data flow which can be modeled in dynamic task nets by means of document revisions. The corresponding progress measure is called *Document States*.

For a document like a Piping and Instrumentation Diagram (P&ID) several states can be defined together with according progress degrees. Furthermore, an estimate for the required workload for the document can be derived from reference data. From the progress degrees and the overall workload for the document, the accumulated workload can be calculated for each state. An example is given in Table 8.1 where five states are defined for the document type P&ID: Devices complete (DC), Piping complete (PC), Instrumentation Complete (IC), Issued for Review (IFR) and Issued for Construction (IFC).

In PROCEED, documents and tasks are managed in an integrated way. A document contained in the Comos database can be associated with an output parameter of

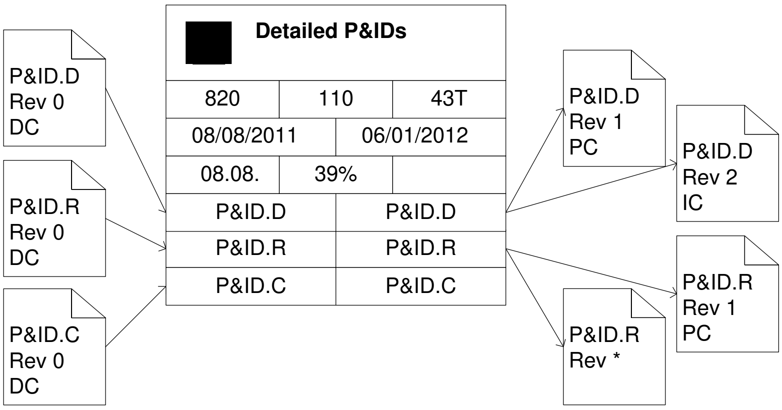


Abbildung 8.3: Task with produced document revisions.

a task meaning that the document is created or modified in the scope of this task. Furthermore input parameters can be defined for tasks and can be connected with output parameters of other tasks via data flows. These data flows define how the documents are routed from one task to another. Figure 8.3 shows an example where four document revisions have been produced in the task Detailed P&IDs. The progress measure *Document States* builds on these modeling capabilities.

Let $\mathtt{s}_0^{\mathtt{d}}, \mathtt{s}_1^{\mathtt{d}}, \ldots, \mathtt{s}_k^{\mathtt{d}}$ be the possible states of a given document $\mathtt{d} \in \mathtt{Documents}$, where $\mathtt{s}_0^{\mathtt{d}}$ denotes the initial state of the document right after its creation. Let $\mathtt{W}_{\mathtt{d}}(\mathtt{s})$ be the accumulated workload required to reach state $\mathtt{s}$ of the document $\mathtt{d}$. For every document $\mathtt{d} \in \mathtt{Documents}$ is $\mathtt{W}_{\mathtt{d}}(\mathtt{s}_0^{\mathtt{d}}) = 0$. For each output parameter of a task, the target state of the respective document is defined, i.e. the state which the document should reach in the scope of this task. For a given task $\mathtt{t} \in \mathtt{Tasks}$ which has an output parameter for document $\mathtt{d}$, we denote $\mathtt{o}_{\mathtt{d}}^{\mathtt{t}}$ for the defined target state of that output parameter. The current state which a document $\mathtt{d}$ has reached in task $\mathtt{t}$ is denoted as $\mathtt{a}_{\mathtt{d}}^{\mathtt{t}}$. For each input parameter of task $\mathtt{t}$, the input state which the corresponding document $\mathtt{d}$ has before it is modified in the task can be determined as the target state of the connected output parameter. It is denoted as $\mathtt{p}_{\mathtt{d}}^{\mathtt{t}}$. If the document is created in the task, then the input state is the initial state of the document $\mathtt{p}_{\mathtt{d}}^{\mathtt{t}} = \mathtt{s}_0^{\mathtt{d}}$. Let $\mathtt{t}$ be a task with $\mathtt{m}$ output parameters associated with documents $\mathtt{d}_1, \ldots, \mathtt{d}_{\mathtt{m}}$. Then, the DOC of task $\mathtt{t}$ is calculated as follows.

$$\mathtt{DOC}_{\mathtt{DocumentStates}}(\mathtt{t}) = \frac{\sum_{j=1}^{m} \left(\mathtt{W}_{\mathtt{d}_j}(\mathtt{a}_{\mathtt{d}_j}^{\mathtt{t}}) - \mathtt{W}_{\mathtt{d}_j}(\mathtt{p}_{\mathtt{d}_j}^{\mathtt{t}}) \right)}{\sum_{j=1}^{m} \left(\mathtt{W}_{\mathtt{d}_j}(\mathtt{o}_{\mathtt{d}_j}^{\mathtt{t}}) - \mathtt{W}_{\mathtt{d}_j}(\mathtt{p}_{\mathtt{d}_j}^{\mathtt{t}}) \right)}$$

The sum of the accumulated workload for the actually reached document states is divided by the sum of the planned accumulated workload for reaching the target states.

In the example in Figure 8.3, the output documents P&ID.D,P&ID.R and P&ID.C of the task Detailed P&IDs have to reach the target state IFR. The documents are in state DC before they are modified in the task Detailed P&IDs. For the document

P&ID.D, two revisions have been created and released which represent the document states PC and IC as depicted in Figure 8.3. For the document P&ID.R, one revision has been released and the state PC has been reached. Another revision has been created but not yet released. In this case, the degree of completion for the task Detailed P&IDs computes to

$$\frac{(105-30)+(60-30)+(30-30)}{(120-30)+(120-30)+(120-30)} = \frac{105}{270} = 38,89\%$$

Since it is a common practice to work with document states and workload estimates for documents in plant design projects, the used reference values often stem from long-year experience. Hence, the progress measure *Document States* can achieve a reasonably high accuracy regarding the progress of the defined tasks. However, the fact that for one task usually only few documents are considered may lessen the accuracy of the progress measure. Therefore the accuracy is rated as *Medium*. As long as no document revisions have been produced in a task, no meaningful DOC is available for the task. Until the next state has been reached, the DOC does not exactly represent the actual progress of the task.

The measure *Document States* is most appropriate for tasks, which produce complex and essential documents like flow sheets or layout plans, where reliable data for the document states and estimated workload is available. Since this data is required and target states for output parameters have to be defined, the effort for measuring the progress of a task by means of document states is rated as *Very High*.

8.1.2 White-Box Progress Measures

As described in Section 5.1.1 and Section 6.3, a task in a hierarchical dynamic task net can be refined by a subnet, which is either managed manually or by the workflow engine depending on whether a workflow definition exists for the subprocess. In the following, white-box progress measures are described which take the subtasks of the measured task into account. First, progress measures are described which are applicable for manually managed and workflow-managed tasks. These measures either use defined milestones or aggregate the DOC values of the subtasks. Second, a progress measure is presented which requires the measured task to be workflow-managed because it relies on reference data derived from completed workflow instances.

Milestones If milestones have been defined for a subnet of the overall dynamic task net, then the DOC of the parent task of the subnet can be calculated based on the *Milestones* measure. Each milestone is associated with an overall degree of completion which is set for the parent task as soon as the milestone has been committed (cf. Section 5.4). Let $\mathtt{M_t} = \{\mathtt{m_1}, \dots, \mathtt{m_n}\}$ be the set of DOCs associated with the completed milestones which are contained in the realization of task $\mathtt{t} \in \mathtt{Tasks}$. Then the DOC of task $\mathtt{t}$ is calculated as follows.

$$\mathtt{DOC_{Milestones}}(\mathtt{t}) = \mathtt{max}(\{\mathtt{m_1}, \dots, \mathtt{m_n}\})$$

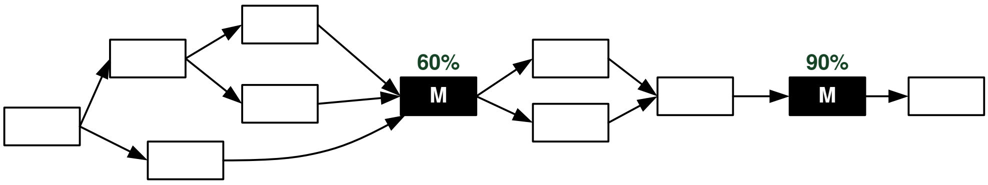


Abbildung 8.4: Required graph structure for progress measure *Milestones*.

For the *Milestones* measure to return a meaningful DOC value, the dynamic task net containing the milestones should be structured in a particular way. The tasks and control flows have to form a weakly connected directed graph and the milestones should be cuts in this connected graph, i.e. removing a milestone would result in two disconnected graphs. Furthermore, all milestones should lie on a all paths from the first to the last tasks in the subnet. Figure 8.4 illustrates the required graph structure, where the black boxes labeled with M represent milestone tasks.

The *Milestones* measure is most appropriate for complex tasks in the project where the DOCs of subtasks are not available or not reliable and where positions in the respective subprocess can be identified with which domain experts can associate progress degrees for the complex parent task. Furthermore, progress measurement based on milestones is common in practice, which is why a practical approach for progress measurement like the one presented in this thesis should incorporate this measure.

The disadvantage of the basic *Milestones* measure is that the DOCs of the subtasks between milestones are not taken into account for the DOC of the parent task. If reliable DOCs for the subtasks are available, then these should not be neglected. Hence, a variant of the *Milestones* measure has been defined, which also takes the subtasks after the last completed milestone into account. It is called *Milestones+* because additional tasks are taken into account and the measure is slightly more accurate than the pure *Milestones* measure. However, to calculate the DOC based on this measure, the above mentioned graph structure is mandatory for the dynamic task net. Otherwise, the computation would fail. Let $\mathtt{M_t} = \{\mathtt{m_1}, \ldots, \mathtt{m_n}\}$ be the set of DOCs associated with the completed milestones, which are part of the subnet of task $\mathtt{t} \in \mathtt{Tasks}$, with $\mathtt{m_i} \leq \mathtt{m_j}$ for $\mathtt{i} < \mathtt{j}$. Furthermore let $\mathtt{m_{n+1}}$ be the DOC of the next milestone which will be reached in the task net (due to the above mentioned graph structure, this milestone is unique) or the value of 100% if $\mathtt{n}$ is the last milestone in the task net. The set of all tasks between two subsequent milestones $\mathtt{i}$ and $\mathtt{j}$ is denoted as $\mathtt{B_{ij}} \subset \mathtt{Tasks}$. Then the DOC of a task $\mathtt{t} \in \mathtt{Tasks}$ is calculated as follows.

$$\begin{aligned}\mathtt{DOC_{Milestones+}}(\mathtt{t}) = \mathtt{m_n} + &\left(\frac{1}{\sum_{\mathtt{i} \in \mathtt{B_{n,n+t}}} \mathtt{W}(\mathtt{i})} \sum_{\mathtt{s} \in \mathtt{B_{n,n+t}}} (\mathtt{s.DOC} \times \mathtt{s.TotalWorkload}) \right) \\ &\times (\mathtt{m_{n+1}} - \mathtt{m_n})\end{aligned}$$

The DOCs of the tasks after the last completed milestone are aggregated, weighted by their total workload. The difference between the overall degrees of completion defined by milestones $\mathtt{n}+1$ and $\mathtt{n}$ is multiplied by the calculated degree of completion of the subnet between these milestones, and the result is added to the overall DOC defined by the milestone $\mathtt{n}$. In this way, the progress of the tasks between the milestones $\mathtt{n}$ and $\mathtt{n}+1$ is taken into account.

For both measures which rely on the definition of milestones, there is considerable effort required for the estimation of the overall DOCs for the milestones in a task net. These estimates are specific for the subprocess defined by the subnet. Accurate estimates require reference values from previous projects and sufficiently experienced domain experts. Therefore, the measuring effort is rated as *High*. The accuracy of the Measures *Milestones* and *Milestones+* range from *Low* to *Medium*, since the DOC-values defined for the milestones are based on estimates, and the progress of the other subtasks in the process is either not taken into account at all or only partly.

Aggregation One of the progress measures available in PROCEED has been particularly defined for aggregating the progress degrees of subtasks and is therefore called *Aggregation*. This measure is usually used on higher levels of the work breakdown structure, when other measures are not appropriate or applicable. The earned value of a task $\mathtt{t} \in \mathtt{Tasks}$ in terms of workload is defined as $\mathtt{E}(\mathtt{t}) = \mathtt{t.TotalWorkload} \times \mathtt{t.DOC}$. If task assignments are defined for a task $\mathtt{t} \in \mathtt{Tasks}$, then their planned workload sums up to $\mathtt{W}_{\mathtt{Ass}}(\mathtt{t}) = \sum_{\mathtt{a} \in \mathtt{t.TaskAssignments}} \mathtt{a.Workload}$. The DOC of task $\mathtt{t}$ is computed as follows according to the measure *Aggregation*

$$\mathtt{DOC}_{\mathtt{Aggregation}}(\mathtt{t}) = \frac{\left(\frac{\sum_{\mathtt{s} \in \mathtt{t.Subtasks}} \mathtt{E}(\mathtt{s})}{\mathtt{t.TotalWorkload} - \mathtt{W}_{\mathtt{Ass}}(\mathtt{t})} \times \mathtt{W}_{\mathtt{Ass}}(\mathtt{t})\right) + \sum_{\mathtt{s} \in \mathtt{t.Subtasks}} \mathtt{E}(\mathtt{s})}{\mathtt{t.TotalWorkload}}$$

Since no DOC is computed for task assignments, their planned workload is multiplied with the aggregated degree of completion of the subtasks. The accuracy of the measure *Aggregation* can be rated as *Medium* to *High*, since it combines the DOCs of the subtasks in a reasonable way. There is no additional measuring effort required for this measure.

If the actual workload of the task assignments of a task $\mathtt{t} \in \mathtt{Tasks}$ has been tracked and there is an estimate for the remaining workload for the task assignments of task $\mathtt{t}$, then the calculation can be refined to

$$\mathtt{DOC}_{\mathtt{Aggregation+}}(\mathtt{t}) = \frac{\left(\frac{\mathtt{A}(\mathtt{t})}{\mathtt{A}(\mathtt{t}) + \mathtt{R}(\mathtt{t})} \times \mathtt{W}_{\mathtt{Ass}}(\mathtt{t})\right) + \sum_{\mathtt{s} \in \mathtt{t.Subtasks}} \mathtt{E}(\mathtt{s})}{\mathtt{t.TotalWorkload}}$$

where $\mathtt{A}(\mathtt{t})$ and $\mathtt{R}(\mathtt{t})$ are defined as for the measure *Estimate to Completion*. The variant *Aggregation+* of the measure *Aggregation* is more accurate. The estimation of remaining workload for the task may slightly improve the calculated progress degree. If the estimation is bad, it does not bias the aggregated value too much. Therefore, its accuracy is rated with *High*. Time tracking and estimation of remaining workload lead to a *very high* measuring effort as for the measure *Estimate to Completion*.

In the special case, where no task assignments with planned workload exist for task $\mathtt{t} \in \mathtt{Tasks}$, its DOC is computed as

$$\mathtt{DOC}_{\mathtt{Aggregation+}}(\mathtt{t}) = \mathtt{DOC}_{\mathtt{Aggregation}}(\mathtt{t}) = \frac{\sum_{\mathtt{s} \in \mathtt{t.Subtasks}} \mathtt{E}(\mathtt{s})}{\mathtt{t.TotalWorkload}}$$

which is the weighted average of the DOCs of the subtasks, weighted by their total planned workload.

In conventional work breakdown structures, the workload of a complex task always equals the sum of the workload of its subtasks. Hence, every change to the planned workload of a task inevitably leads to a change of the planned workload of the parent task.

The total workload of a task is used as the normalizing factor in the progress measures *Aggregation(+)*. If the total workload was always equal to the used total workload, then a change of the workload of a subtask would always lead to a change of the degree of completion of the task even if the DOC of the subtask was 0%. This would lead to unnecessary disturbance and confusion regarding the progress of the task during replanning. Therefore, the total workload of a task may exceed the used total workload in a dynamic task net. The creation of a new subtask or the increase of the total workload of a subtask can be covered by the unassigned total workload of a complex task. In these cases the total workload of the task remains unchanged and the dynamic change to the realization of the complex task has no effect on its degree of completion.

Progress measurement of workflow instances For workflow-managed tasks, a specific progress measure is available. The measure *Workflow* calculates the DOC of a workflow-managed task based on reference values for the expected durations of the workflow activities. The current state of the workflow instance determines the expected duration of the workflow. The degree of completion of a workflow-managed task $\mathtt{t} \in \mathtt{Tasks}$ is calculated as the quotient of the actual duration and the expected duration of the workflow instance.

$$\mathtt{DOC}_{\mathtt{Workflow}}(\mathtt{t}) = \frac{\text{Actual duration of workflow instance}}{\text{Expected duration of workflow instance}}$$

Reference values for activity durations are collected for all activities in a workflow definition—atomic activities as well as complex activities. The reference value for the duration of an activity is calculated as the mean duration of all occurrences of the activity in the completed instances of the workflow template. For an IfElse activity this has the effect that its mean duration equals the weighted average of the mean duration of its branches, weighted by the number of workflow instances which have chosen the respective branch during execution. This is illustrated in Figure 8.5. The mean duration of the activity If is 40 days and the mean duration of activity Else is 8 days. Nevertheless, the mean duration of the whole IfElse activity is only 10 days, because in the example only two workflow instances chose the If-branch while 30 instances chose the Else-branch ($10 = [(40 \times 2) + (8 \times 30)]/32$).

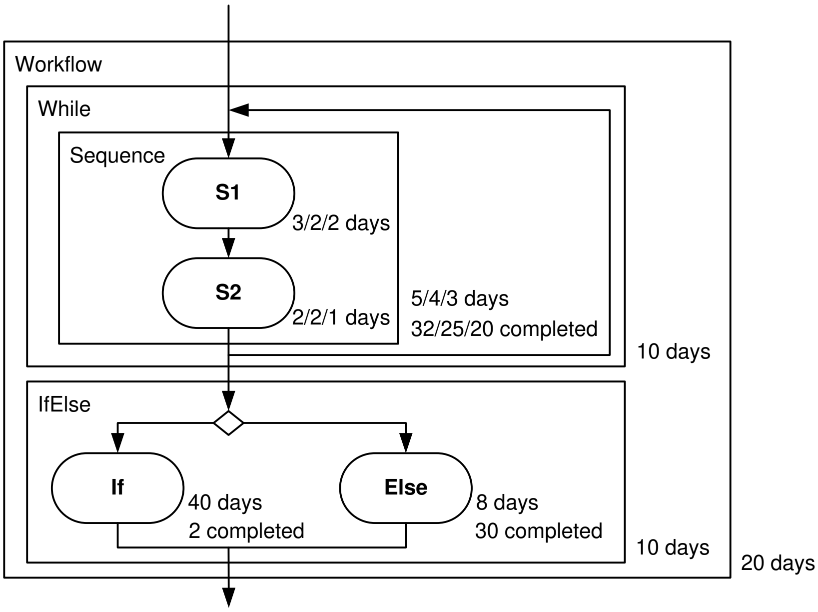


Abbildung 8.5: Reference values for a workflow definition.

For activities directly or indirectly contained in a While activity, more information is needed about their mean durations. For these activities, the mean duration is determined for every iteration of the surrounding loop construct separately. Figure 8.5 illustrates this. On average, activity S1 took 3 days in the first iteration, 2 days in the second and another 2 days in the third (3/2/2 days in Figure 8.5). For a While activity we furthermore store for each iteration the number of workflow instances which actually completed the respective iteration (e.g. 32/25/20 completed in Figure 8.5). In the example of Figure 8.5, the reference data only covers workflow instances with up to three iterations of the While activity. For more than three iterations, there are no reference data available. When reference values are required for further iterations of the While-loop, a simple heuristic is applied, which assumes that every further iteration will have approximately the same duration as the last one for which reference values are available. It is reasonable to assume that every further iteration of a loop does not take longer than the previous iterations, because the iteration of tasks in the context of a development project is usually performed to rework previous results. Therefore, later iterations will probably be shorter than the previous ones. However, since no information is available about the time saving in a later iteration compared to the previous one, the same value as for the previous iteration is used.

The tracking service of the workflow engine determines the total durations of workflow activities including work time and idle time. For progress measurement, only the effective work time is considered. This is achieved by suspending and resuming the workflow instance after and before working on the tasks respectively. The suspension time is subtracted from the total activity durations. This applies for the workflow instances which are used to compute the mean activity durations, as well as for the measured workflow instance. Consequently, the activity durations reflect when the assigned resources were actually working on the respective tasks.

The quality of the reference values for activity durations depends on the number of workflow instances which are taken into account and the standard deviation of the measured durations. Only those workflow instances should be considered, which have been executed in a comparable project. Dynamically changed workflow instances are not taken into account for the computation of reference data for the original workflow template. Reference values for activities, which were dynamically added or removed, cannot be used or collected respectively. If an activity was dynamically removed, incorporating a duration of zero time units into the mean duration of this activity would bias the reference value for its duration. If an activity was dynamically added, the calculated reference value cannot be associated with an activity in the original workflow definition. Consequently, dynamically changed workflow instances have to be handled as new variants of the workflow definition and—if sufficiently many instances with the same dynamic changes exist—separate reference values can be calculated for these variants.

At runtime of a workflow instance, the reference data which has been collected for the workflow template is used for progress measurement. The DOC of a running workflow instance is recalculated after every closing event of an activity. The actual duration of the workflow instance is calculated as the time span between the start of the workflow and the closing event, where suspension times are subtracted. The expected duration of the workflow is calculated based on the actual durations of closed activities and the expected durations of all activities which still have to be executed. The algorithm updates the expected durations of all complex activities which directly or indirectly contain the recently closed activity. This leads to an updated expected duration of the whole workflow instance. The degree of completion can then be computed based on the updated expected duration.

The essential step of the algorithm is the recalculation of the expected durations of all complex activities directly or indirectly containing the recently closed activity. For `Sequence` and `IfElse` activities this is quite straight forward. The expected duration of a `Sequence` activity is the sum of the actual durations of all closed activities, the recursively computed expected duration of the activity containing the recently closed activity, and the expected durations of all subsequent activities which are taken from the reference data of the workflow template. For an `IfElse` activity, the expected duration of the selected branch is calculated which is at the same time the expected duration of the whole `IfElse` construct.

In case of a `While` activity, future iterations have to be taken into account. Similar to the `Sequence` activity, the actual durations of all completed iterations and the expected duration of the currently executing iteration are summed up. Furthermore, a time span is added to the result which covers the expected duration of probably executed future iterations of the `While` loop.

Figure 8.6 shows on the top the iterations of a `While` activity in a running workflow instance. The `While` activity contains only the single activity `S`. Activity `S` has been

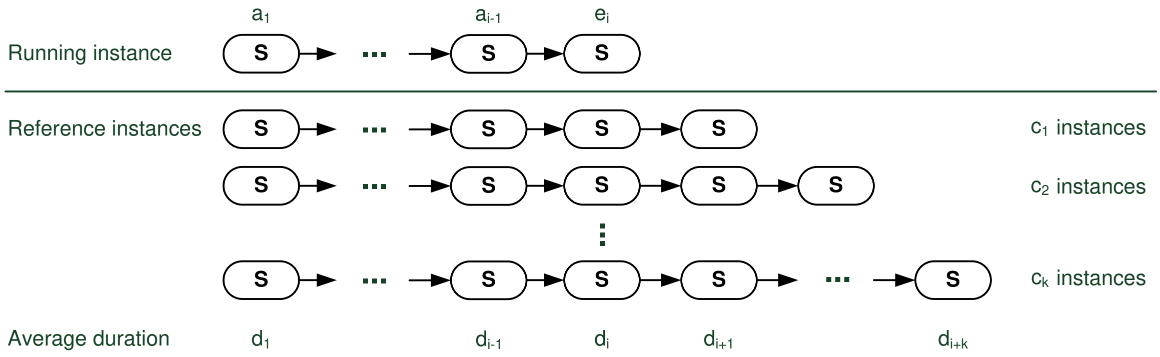


Abbildung 8.6: Reference data for an iterated activity.

iterated $i-1$ times and the surrounding While activity is now in iteration i. The expected duration e_i of iteration i is calculated based on the reference values for this iteration. For this purpose, reference values are stored for the activities contained in a `While` activity for each number of iterations separately. To obtain the expected duration of the complete `While` activity, the expected remaining time for all following iterations has to be determined. Since it is not known how many further iterations the `While` loop will have in the workflow instance, the remaining time of the `While` loop is computed using the probabilities for the additional iterations. For every number of additional iterations, the probability is determined that the `While` loop will iterate that often. These values are multiplied with the corresponding expected remaining durations for the respective number of iterations. The sum of these products is the expected remaining duration of the `While` loop. The probability for a certain number of additional iterations is derived from the reference data, i.e. from the number of workflow instances which completed this many iterations. Altogether, the expected duration of a While activity in iteration $\mathtt{i}$ is computed as follows.

$$e^{\mathtt{i}}_{\mathtt{While}} = \sum_{j=1}^{i-1} a_j + e_i + \frac{1}{\sum_{j=1}^{k} c_j} \sum_{j=1}^{k} \left(c_j \times \sum_{t=1}^{j} d_{i+t} \right)$$

The semantics of the variables in the formula is illustrated in Figure 8.6. For a certain number j with $1 \leq j \leq k$, the variable c_j denotes the number of workflow instances in the reference data which have completed $i+j$ iterations. The sum of the average durations of the iterations $i+1$ to $i+j$ is the expected remaining duration of the While loop for j further iterations.

A concrete example can be provided based on the workflow definition and reference data of Figure 8.5. Here, it is assumed that an instance of this workflow type has just completed activity S1 after 5 days, which is 2 days longer than expected. The expected duration of the Sequence activity in the first iteration is therefore 7 days, and the expected duration of the whole While activity is

$$0 + 7 + \frac{((25-20)\times 4) + (20 \times (4+3))}{25} = 13.4$$

The degree of completion of the whole workflow instance computes to 5/23.4=21.4% which is less than the planned progress after five days 5/20=25%.

By using the expected remaining duration, the measure *Workflow* reaches an accuracy comparable to the measure *Estimate to Completion*. However, while for the latter manual time tracking and estimation are required this is not necessary for a workflow-managed task. Time tracking is done automatically by the tracking service of the workflow engine and the estimation of the remaining durations is done automatically based on the available reference data. Since workflow definitions are required for this measure, and resources assigned to workflow-managed tasks have to suspend and resume their tasks correctly, the measuring effort is rated as *Medium*. The measure *Workflows* seems to be preferable to the measure *Estimate to Completion*. However, it is only applicable for workflow-managed tasks, for which reference data is available. Finally, it is important to mention that the project controller is not obliged to choose the measure *Workflow* for a workflow-managed task. It is very well possible to choose any other measure while the task is managed by the workflow engine. The detailed description of the algorithms which are required to determine the degree of completion of a workflow instance are described in [Bri08].

Progress measurement for incomplete work breakdown structure By means of the white-box progress measures, which take the progress of subtasks into account, the DOC can be aggregated on higher levels of the work breakdown structure. In this way, degrees of completion for the different engineering phases and finally the whole project can be derived.

Due to the top-down planning of workload which has been described in Section 5.3, a valid DOC can be derived for the project even if the work breakdown structure has not been completely elaborated. Furthermore, if workload buffer has been planned for a complex task, dynamic structural changes to the realization of the complex task do not necessarily affect its degree of completion. This is illustrated in Figure 8.7 which shows an abstract example of a task net hierarchy.

If the measure *Aggregation* is used for calculating the DOC of task A, it returns a value of 10% because the planned total workload of the tasks C and D is taken into account, although they have a DOC of 0%. The DOCs of all subtasks are weighted by their planned total workload. Since the expected total workload is already defined for task D although its realization is not yet elaborated, i.e. no subtasks have been defined, the DOC of task A already reflects that there will be subtasks which will amount for 500 MHRS and which have not been started yet.

Similarly, the measure *Workflow* works with the expected durations of tasks. If the DOC of task A would be measured by means of the measure *Workflow*, then the expected duration of D would already incorporate the durations of subtasks which have not been planned yet. Even if task D would be iterated several times in a `While` loop, the expected duration of the whole While loop could be derived from the reference data, and the structural changes as a consequence of the iteration would already be incorporated into the DOC of task A before the iteration would

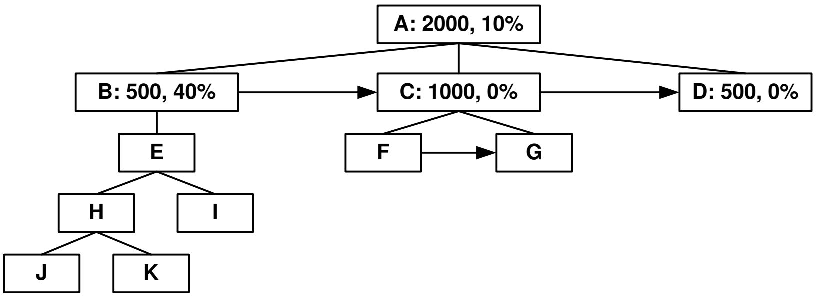


Abbildung 8.7: Accurate progress measurement despite incomplete WBS.

take place.

Finally, the measure Milestones explicitly defines points in the realization of a task where a certain DOC is reached and it does therefore not rely on the detailed planning of future tasks. For example, if the task C in Figure 8.7 would be defined as a milestone task with an overall degree of completion of 45%, then this DOC is reached upon committing task C regardless of whether task D has already been completely elaborated.

8.1.3 Comparison of Progress Measures

For every task in a dynamic task net, an individual progress measure can be chosen. In the example in Figure 8.1, the progress of the task Pump 032 may be measured by means of the measure *Workflow*. If for short-lived tasks like the specification of a Measuring point no workflow definition is available, then the measure *Start/End* may be most appropriate. For complex tasks like Specification of Machines and Devices, the measure *Aggregation(+)* can be chosen. If milestones are defined in a task, as it is conceivable for an engineering phase like *Basic Engineering*, then the measure *Milestones(+)* can be chosen. In this way, the most appropriate measure can be selected for every task. The selection of a progress measure for a task constitutes process knowledge which should be reused for future projects. Therefore, progress measures can be specified for task types which serve as default selection for all instances of the respective type. Furthermore, progress measures can be specified for the subtasks of process templates. This is particularly useful for the *Milestones* measure because the overall degree of completion determined by a milestone depends on the subprocess in which it is contained.

Table 8.2 gives an overview over all available progress measures in the PROCEED system together with the rating of their required measuring effort and accuracy. The rationale for the selected values has been given in the description of the different measures in the previous section. Furthermore, Table 8.2 shows the timeliness of the different measures, i.e. if up-to-date values for the DOC of a task are available at any time, or if the values may be outdated until a certain event takes place. Anyway, with PROCEED a project controller is not limited to assembling a progress report once a month, but he can retrieve a sufficiently accurate and up-to-date report from

Measure	**Effort**	**Accuracy**	**Timeliness**
Absolute	Lowest	Lowest	Immediately available
Start/End	Very Low	Very low	Immediately available
Time Proportional	Lowest	Medium	Immediately available
Expert's Estimate	High	Low–High	Outdated between estimations
Estimate to Compl.	Very High	Low–High	Outdated between estimations
Milestones(+)	High	Low–Medium	Immediately available
Workflow	Medium	Medium	Immediately available
Document States	Very High	Medium	Outdated between creation of document revisions
Aggregation	Lowest	Medium–High	Immediately available
Aggregation+	Very High	High	Outdated between estimations

Tabelle 8.2: Evaluation of progress measures.

the system at any time.

The different progress measures can be positioned against each other regarding their measuring effort and accuracy as depicted in Figure 8.8. The progress measures can be coarsely grouped into three sets. First, there are measures with low effort and low accuracy (*Absolute* and *Start/End*). For individual tasks, these measures return too inaccurate results and other measures are usually preferred if they are applicable. Second, there are measures with low effort and comparably high accuracy (*Aggregation*, *Time Proportional* and *Workflow*), which are only applicable for some tasks when certain prerequisites are fulfilled. Finally, there are measures with high accuracy, but which require a comparably high measuring effort. Although in most cases, these measures return the best results, they cannot be applied to all tasks in an engineering project, since this would lead to a huge measurement overhead.

The measures with the highest accuracy are close to each other, both in terms of effort and accuracy. However, the calculation of the DOC relies on different sources of information. If reference data about the output documents of a task is available one may decide to choose the measure *Document States*. If no such data is available, one has to resort to measures based on estimates. If time tracking is performed for a task, one can use the measure *Estimate to Completion*, otherwise *Expert's Estimate* is the alternative choice. As a consequence, none of the different measures is dispensable.

In some rare cases a low effort configuration is feasible. In these cases, only measures without or with very low measuring effort are used, i.e. *Absolute* or *Start/End* for atomic tasks, *Aggregation* for complex tasks, and *Time Proportional* for accompanying management tasks. In some cases this may lead to valid progress degrees at the project level because of the balancing effect of the aggregation. However, if several long-running atomic tasks exist which are critical for the project then this approach will fail in terms of accuracy and timeliness.

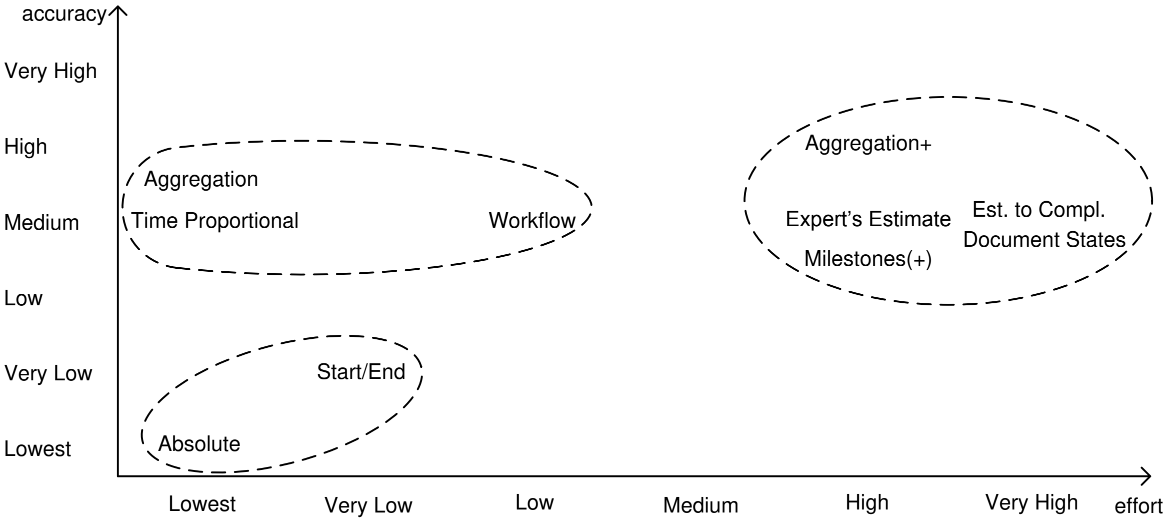


Abbildung 8.8: Comparison of progress measures regarding effort and accuracy.

The discussion leads to the conclusion, that the proposed approach to use all described measures and for every task the most suitable one is the only practicable way of measuring the progress of an engineering project with an acceptable measuring effort and a sufficiently high accuracy.

8.2 Earned Value Analysis and Forecasts

The calculation of the degree of completion for every task in the project is only one step on the way to a useful project status report. In general, projects are constrained by time, cost and scope. The DOC only measures the progress of a task in terms of its defined scope, i.e. it measures to which extent the final result of the task is completed, e.g. a process flow diagram can be completed to 60%. The DOC does not directly show if the schedule will be met and whether the task will keep to its budget. Therefore, the actual performance of tasks has to be compared with the plan.

In PROCEED, *Earned Value Analysis* [Anb03] is applied for this purpose. Earned Value Analysis (EVA) can be performed based on either workload or costs. The former allows a simplified planning and performance analysis without a detailed budget. However the latter is the more accurate method which is generally used in practice, because base costs and different resource costs can be taken into account. The PROCEED system supports both methods. The following explanations will refer to EVA based on the planned and actual workload of tasks.

To compare the actual performance of a task with the plan, the planned value has to be determined. The *Planned Value* $\mathtt{P(t)}$ of a task $\mathtt{t} \in \mathtt{Tasks}$ is the workload that has been planned for the task from its planned start time to the reporting date. Resource constrained scheduling of a dynamic task net distributes the planned workload of tasks and task assignments over several working days. The planning of

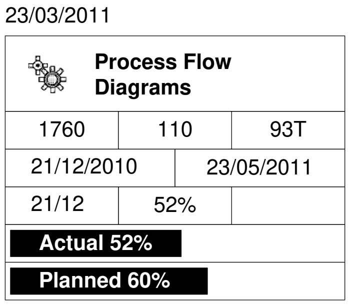


Abbildung 8.9: Comparison of planned and actual degree of completion.

workload on a daily basis allows for a non-uniform distribution over the duration of a task, i.e. the working hours which are scheduled per day for a task assignment may differ for different dates. As a consequence, the planned value of a task does not necessarily increase linearly over the duration of the task.

From the planned value, the *planned degree of completion* of a task can be derived. It is calculated as the quotient of planned value and the total planned workload.

$$\mathtt{DOC}_{\mathtt{planned}}(\mathtt{t}) = \frac{\mathtt{P}(\mathtt{t})}{\mathtt{t.TotalWorkload}}$$

The planned DOC of a task can be directly compared to the actual DOC to check whether the task is behind schedule. In Figure 8.9 the task Process Flow Diagrams is delayed because the planned DOC is smaller than the actual DOC.

For tasks whose actual degree of completion is determined by means of the progress measure *Absolute* or *Start/End*, the planned DOC is computed in a different way. Since these progress measures provide only very rough estimates for the actual DOC of a task, the actual DOC would nearly always differ from the planned DOC if it was calculated as described above. Therefore, the planned DOC is calculated like the actual DOC but with respect to the planned dates, i.e. when the task should be preparing, running, or terminated according to the plan, then the degree of completion defined for the progress measure is used as the planned DOC. This value may nevertheless differ from the actual DOC, if the task is started or terminated too late or too early.

The comparison of the actual DOC with the planned DOC of a task does not allow any quantitative statements about the expected delay of the task. For this purpose, earned value analysis has to be applied. The *Actual Value* $\mathtt{A}(\mathtt{t})$ of a task $\mathtt{t} \in \mathtt{Tasks}$ is the workload that was spent on the task from its actual start time to the reporting date. It is determined by means of a time tracking system in which resources can book their actual working hours per day on their task assignments. A resource can spend every number of working hours between zero and the maximally available time for a particular day on a task assignment. Like the planned workload, the actual workload is not necessarily uniformly distributed over the working days of a task. Consequently the actual value may also increase non-linearly over the runtime of the task.

The actual value of a task may deviate from the planned value. This indicates that either more or less effort and money has been spent on the task than planned. However, even if more money has been spent on the task than planned, this does not necessarily mean that the task is over budget, because the earned value may also be higher than the planned value, i.e. the work has simply been completed earlier than planned but with the planned effort and costs. The *Earned Value* $\mathtt{E(t)}$ of a task $\mathtt{t} \in \mathtt{Tasks}$ is calculated by multiplying the DOC of the task $\mathtt{t}$ with its total workload.

$$\mathtt{E(t)} = \mathtt{t.DOC} \times \mathtt{t.TotalWorkload}$$

Performance indices To quantify the amount of delay of a running task and to decide whether a running task has already exceeded its budget limits, performance indices have to be computed which allow forecasting the expected duration and budget of the task.

For this purpose, the *Schedule Performance Index* (SPI) and the *Cost Performance Index* (CPI) are calculated for every task in a dynamic task net. In contrast to the common practice, performance indices are not only calculated for project phases or the whole project but for all tasks in the project. This enables the detection of delayed tasks on lower levels of the hierarchical dynamic task net.

The SPI indicates, how much a task $\mathtt{t} \in \mathtt{Tasks}$ deviates from the schedule It is defined as

$$\mathtt{SPI(t)} = \frac{\mathtt{E(t)}}{\mathtt{P(t)}}$$

where $\mathtt{P(t)}$ is the *Planned Value* of task $\mathtt{t}$. An SPI value greater than one means that the task is ahead of schedule, a task with SPI=1 is exactly on schedule, and a delayed task has an SPI smaller than one. The project management is interested in tasks which perform worse than planned because these tasks may require intervention. Tasks which do not perform as planned are marked in the management views of PROCEED. A running task is marked with one out of three colors depending on whether the schedule performance index falls below a certain threshold. The colors represent increasingly severe levels of delay and required different actions by the project management.

Green The task is on schedule or only a little bit behind.
No corrective measure is required.

Yellow Corrective measures may be necessary and should be discussed.

Red The task is considerably behind schedule and corrective measures are required.

The thresholds for the different levels of delay can be configured for each project individually. Examples for SPI thresholds are listed in Table 8.3.

While the SPI indicates a delay of a task in terms of the schedule, the CPI indicates whether the task will eventually exceed its planned budget. It is defined as

$$\mathtt{CPI(t)} = \frac{\mathtt{E(t)}}{\mathtt{A(t)}}$$

Level	**Threshold**
Green	$\mathtt{SPI} \geq 0.9$
Yellow	$0.9 > \mathtt{SPI} \geq 0.7$
Red	$0.7 > \mathtt{SPI}$

Tabelle 8.3: Examples for SPI thresholds.

where $\mathtt{A}(\mathtt{t})$ is the *Actual Value* of task `t`. To determine the actual value of a task, the assigned resource have to register their actual working hours in the time tracking system. A CPI greater than or equal to one means that the task is in budget limits, and a CPI smaller than one means that the task will probably exceed its budget. The color markings of tasks which are used to provide an overview over the delay of the tasks in a project can alternatively be used to represent different levels of budget overrun. The project management can switch between the two modes, i.e. the colors of tasks either represent the SPI levels or the CPI levels. For the cost performance index, the meaning of the colors is similar to the SPI.

Green The task is in budget limits or only a little bit over budget. No corrective measure is required.

Yellow Corrective measures may be necessary and should be discussed.

Red The task is considerably over budget and corrective measures are required.

The example thresholds for the SPI which are presented in Table 8.3 could also be used for the CPI.

Forecasts The SPI provides quantitative information about the delay of a task. It is furthermore possible to forecasted the expected duration of a task $\mathtt{t} \in \mathtt{Tasks}$ based on the SPI as follows.

$$\mathtt{t.ForecastedDuration} = \frac{\mathtt{t.TotalDuration}}{\mathtt{SPI}(\mathtt{t})}$$

It is implicitly assumed, that the performance of the task—whether good or bad—will not change until the task is terminated. From the forecasted duration and the actual start time, the forecasted end time can be derived.

$$\mathtt{t.ForecastedEndTime} = \mathtt{t.StartTime} + \mathtt{t.ForecastedDuration}$$

Figure 8.10 shows the delayed task Process Flow Diagrams with its schedule performance index. The bars below the task illustrate the difference between the planned total duration and the forecasted duration. The delay of a task may require different actions of the project manager or another resource which is responsible for managing the subprocess. The forecasted end time does not directly influence the planned dates of the task and its successors. Adapting the plan to the actual performance requires an explicit management decision of an authorized resource. These plan changes due to task delays are discussed in Chapter 9.

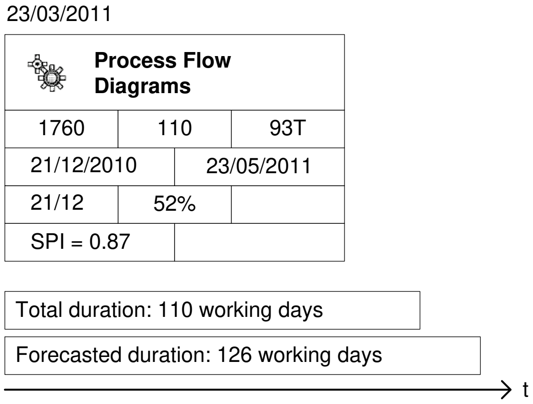


Abbildung 8.10: Duration forecast based on SPI.

Analogously to the forecasted duration, the estimated budget at completion (EAC) of a task can be forecasted based on the CPI.

$$\mathtt{t.EAC} = \frac{\mathtt{t.TotalBudget}}{\mathtt{CPI(t)}}$$

If the costs for a task exceed the planned budget too much, it may be necessary to release assigned resources of the task.

Summary Altogether, the performance of an enacted development process can be evaluated in different ways in PROCEED. First, the DOC of a task can be directly compared to the planned DOC. Second, the performance indices SPI and CPI of the earned value analysis are computed, and tasks are marked in the management views if their respective values of the performance indices exceed specified thresholds. Finally, the duration, end time, and budget at completion of a task can be forecasted and compared to the currently planned values to obtain the expected amount of delay or budget overrun. The performance status evaluation of a process model instance by means of the earned value analysis is not an original contribution of this thesis, but it is rather an application of the established project controlling technique to process management.

8.3 Visual Project Status Analysis

The Comos system is used during all phases of a plant design project for the management of the engineering data. While the AHEAD system has been applied to support the early phases of the plant design process [NM08], the PROCEED prototype also has to be applicable to later process phases, in particular to the detail engineering phase (cf. Section 2.1). Compared to the early phases of preliminary planning and basic engineering, the processes in the detail engineering phase are less dynamic but the degree of simultaneous engineering is even higher. Furthermore, a large number of rather small, similar engineering tasks is executed in the detail engineering phase for the specification of all devices, pipes, instruments, etc. of the designed plant. As a consequence, network diagrams and Gantt charts are insufficient for the assessment of the current project status for the following reasons. The visualization of tasks in the form of a network diagram is not sufficient since most tasks are executed in parallel. If all engineering tasks would be displayed, network diagrams and Gantt charts would quickly grow too big. Finally, both diagram types focus the tasks and control flow perspective, while resources, roles, and process templates are not adequately reflected.

The project management data in plant design projects is inherently *multidimensional* where dimensions are among others the tasks, resources, functional roles, but also the different parts of the chemical plant which is designed. For this reason, software tools for the monitoring of design projects in plant engineering need to be able to handle and to visualize multidimensional project management data in an adequate way. Business Intelligence technologies like Online Analytical Processing (OLAP) [CCS93] in a data warehouse together with appropriate visualization techniques [Kei02, Tuf86] can be applied for this purpose. The information in data warehouses is multidimensional, meaning for the user that it can be visualized in grids. OLAP functionality is characterized by dynamic multidimensional analysis of consolidated data that supports end user analytical and navigational activities [JLVV00].

In this thesis, a *multidimensional visualization approach* for project management data has been developed to provide a condensed overview over all tasks, resources, and roles in a project with respect to their associated workload and costs, as well as the status and progress in case of tasks. The current project status can be analyzed from different perspectives, e.g. the planned workload per role can be visualized, or the status of all workflow instances of a certain template can be inspected. It is furthermore possible to analyze the history of plan changes in a project and to compare it with the progress of tasks. These functionalities complement the project monitoring functionality provided by the management views of PROCEED where performance indices are visualized by color markings as described in the previous section. The developed approach for multidimensional visualization of project management data has been presented in a workshop on business process intelligence [HAW10].

The analysis views of PROCEED serve several different purposes. First, project monitoring is supported by the visualization of delays, overtime work, or resource bottlenecks. Second, project planning is supported by the visualization of resource availabilities and the identification of unbalanced resource usage. Third, the traceability of plan changes during a project is ensured, and plan changes can be compared to progress changes of individual tasks or the whole project. The current plan can be compared to the initial plan at the start of the project. This leads to the fourth possible usage of the analysis views, namely *process improvement*. After the completion of a project, the actual workload and cost can be compared to the initially planned values, and reference data defined for task types and process templates can

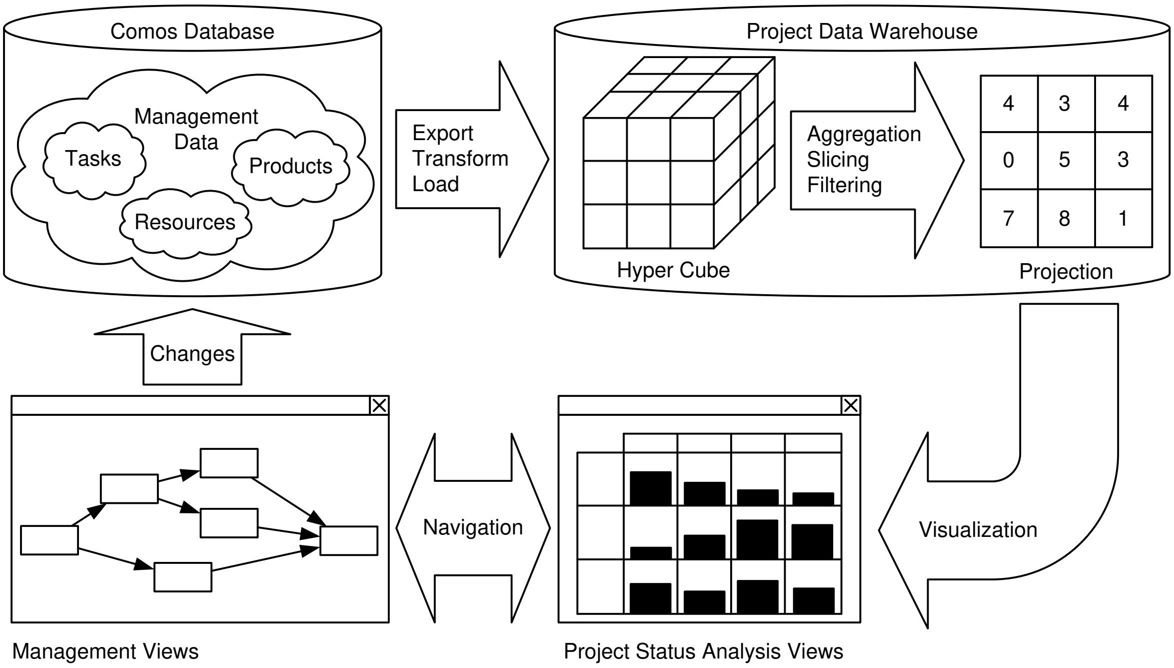


Abbildung 8.11: Overview over the visualization approach.

be adapted accordingly.

An overview over the visualization approach is given in Figure 8.11. Measures are applied to the management data in the Comos database, and the measured values are exported to a separate data warehouse [JLVV00]. In the *project data warehouse*, the measured values are arranged along several dimensions of a *hyper cube*. OLAP operations on the hyper cube lead to projections which are visualized in a flexibly configurable pivot table. The views for visual project status analysis are *coupled* with the management views of PROCEED, so that the user can *navigate* between the analysis and management views. The approach to multidimensional project status analysis in PROCEED has been described in [Auß09, HAW10].

For the management of development projects in PROCEED, an *object-oriented management data model* is used, which has been introduced Chapter 5. Figure 8.12 shows a simplified cutout of the complete TNT meta-model for dynamic task nets extended by the entity PlantPart. A chemical plant is hierarchically structured into several plant parts. Documents as well as tasks in a plant design project can be associated with certain parts of the chemical plant. For example, the task to specify the properties of a pump as well as the resulting specification are associated with the part of the chemical plant in which the pump is contained.

The object-oriented data model is suitable for process management, but does not support the multidimensional visualization of the project management data. To facilitate the latter, a transition from the object-oriented model to a *multidimensional data model* is required. This is the transformation step of the so-called ETL process (export, transform, load) in which data from different data sources is loaded into a

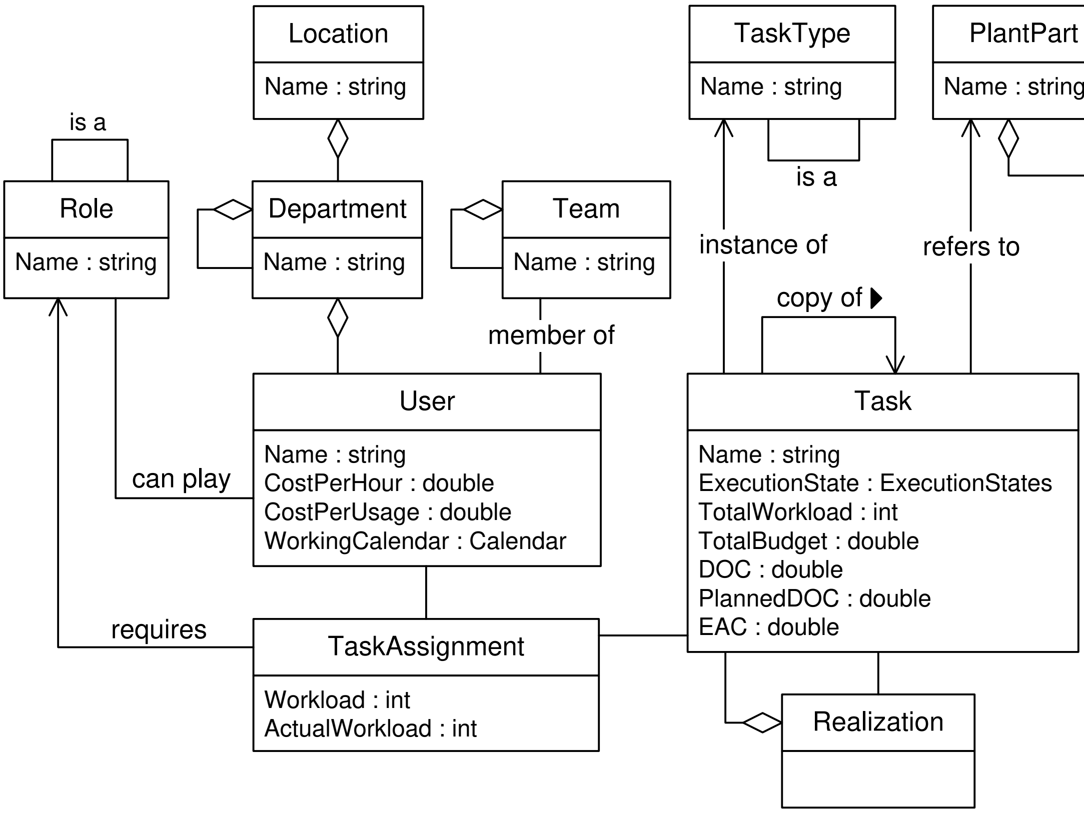


Abbildung 8.12: Simplified cutout of the TNT meta-model.

data warehouse [KC04].

The multidimensional data model is an n-dimensional array, also called a *hyper cube*, which stores all measured values in its cells, i.e. at its coordinates [JLVV00, Leh03]. A hyper cube is spanned by several *dimensions* which act as indices for identifying values in the hyper cube. *Measures* associate values with points in a hyper cube. They correspond to columns in a relational database table whose values functionally depend on the values of other columns.

8.3.1 Measures

In PROCEED, different measures are used to analyze the status of a running project. The measured values are extracted from the management data in the Comos database and are exported to the separate data warehouse. The following measures are used for project status analysis.

- Workload
- Cost
- Degree of completion
- Schedule performance index

- Cost performance index

The measure *workload* refers to the planned and actual workload of tasks and task assignments, as well as the workload of resources and roles. The measure *cost* refers to the total budget of tasks and the costs of task assignments which are derived from the respective planned workload as described in Section 5.3.1. The degree of completion (DOC), schedule performance index (SPI), and cost performance index (CPI) is exported to the data warehouse for every task in the project.

8.3.2 Dimensions

The different measures can be associated with tasks, users, roles, plant parts and dates. For this reason, the data in the project data warehouse is structured along the following major dimensions.

- Time
- Tasks
- Roles
- Resources
- Plant parts

While the measures workload and cost are associated with all dimensions, the progress related measures are only associated with time, tasks and plant parts but not with roles or resources. The set of associated dimensions is called the *granularity* of a measure [Leh03].

Every major dimension has an associated hierarchy of levels of consolidated data, i.e. the coordinates of each major dimension are *structured hierarchically*. The hierarchy defined for the dimension *time* is day-month-year. For the dimension *tasks*, several hierarchies are defined. The tasks can be structured according to the task-subtask relationship in the dynamic task net, according to the task types from which they have been instantiated, or according to the process templates from which they have been copied. In case of the task types, the generalization relationship defines the upper part of the hierarchy. Functional *role*s in a project are structured according to the generalization relationship. The dimension *resources* has the human resources at the lowest level, the hierarchy of departments to which they belong on the higher levels, and the different locations on the highest level. Finally, the hierarchy of the dimension *plant parts* reflects the composition hierarchy of the designed plant.

Besides the major dimensions there are additional dimensions which are not hierarchically structured. The dimension *modus* has the coordinates *actual* and *planned* which enables the distinction between the actual and planned workload and cost of tasks and task assignments. The dimension *execution states* has the coordinates *preparing*, *running*, and *terminated* and is used to separate the planned

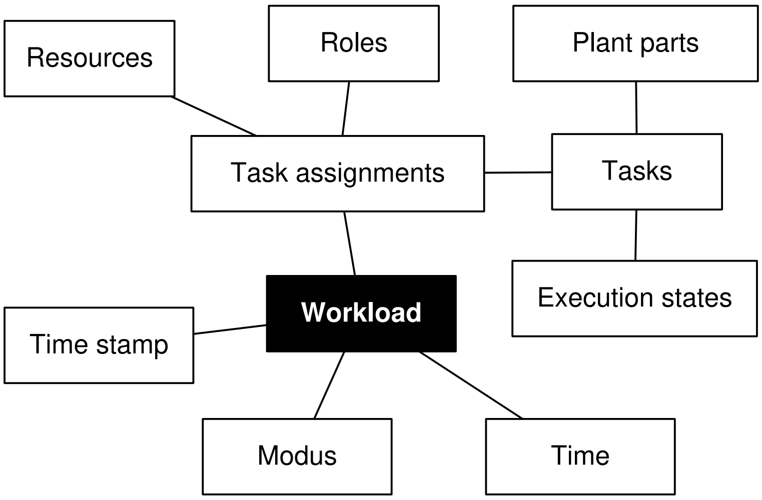


Abbildung 8.13: Multidimensional database schema of project data warehouse.

and actual workload of tasks in the respective execution states. Finally, the dimension *time stamp* defines the date and time of the data export of the measured values to the project data warehouse. It allows to distinguish between different subsequent plan states.

Figure 8.13 shows a cutout of the *multidimensional data model* of the data warehouse. Every box corresponds to a table in the underlying relational database according to the ROLAP data model [JLVV00]. There is a *fact table* for the measure workload and there are *dimension tables* for all major and additional dimensions. For reasons of clarity, only the fact table for the measure workload is shown in Figure 8.13 while the fact tables for cost, DOC, and the performance indices are omitted. The resulting schema is a so-called *snowflake schema* [JLVV00], because the fact tables refer to several dimension tables which themselves refer to other dimension tables.

Figure 8.14 shows a cutout of a four-dimensional hypercube which holds the planned and actual workload in man hours. The depicted cube is a subcube of the complete hypercube containing all dimensions and all measured values. It is the result of a so-called *slicing operation* by which the coordinate on the dimension *time* has been fixed to the date 13/09/2011. For the coordinate planned on the dimension *modus*, the cells of the cube hold the planned workload of task assignments for the specified date. In the example, resource Boateng has to perform 8 man hours in the task Instrumentation in his role as Instrumentation Engineer. This does not necessarily mean that resource Boateng is assigned to the task Instrumentation but the value for the planned workload for the task aggregates all values of the subtasks. The depicted values are taken from the example scenario where resource Boateng is assigned to the task Instrumentation with 2 man hours and to the subtask Reaction Instrumentation with 6 man hours which together amount for the 8 man hours. On the right side of Figure 8.14, the actual working hours performed by the resources are depicted, some of which deviate from the respective planned values.

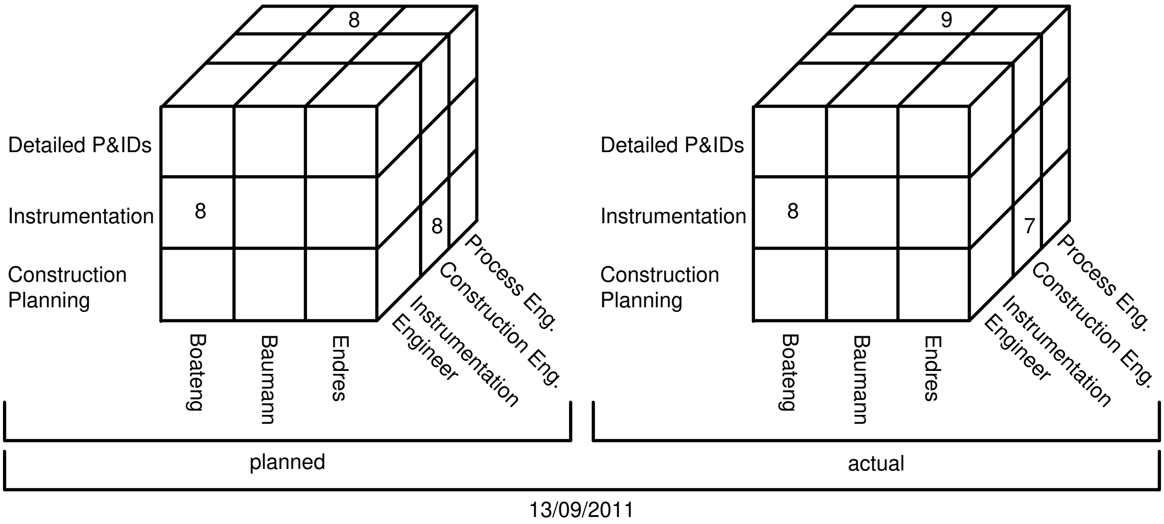


Abbildung 8.14: Example for a subcube of the complete hypercube.

The values in the hypercube are *aggregated* along all dimensions which are not displayed, and which were not subject to a slicing operation. Every measure defines an aggregation method which is applied when measured values are aggregated, either on a higher level of a dimension hierarchy or along a whole dimension. For the measures *workload* and *cost*, the summation is defined as the aggregation method. For the progress related measures, no aggregation method is defined. The aggregation of the respective values along the task net hierarchy is already performed for the management data in the Comos database.

8.3.3 Configurable Pivot Table for Project Status Analysis

The main project status analysis view is a *flexibly configurable pivot table* for the *multidimensional visualization* of the measured values. The coordinates of the pivot table show *stacked-bar charts*. The *configuration* of the pivot table is done by mapping the dimensions and measures of the data cube to the axes of the pivot table and the properties of the stacked bars, respectively. The measured values are represented by the height of the stack layers. One dimension can be mapped to the color of the stack layers. Furthermore, two stacked bars can be displayed in one cell to visualize the values for two different coordinates of a dimension. Consequently, four dimensions can be visualized in the two-dimensional grid.

Figure 8.15 shows an example configuration of the pivot table. (In this section, schematic figures are used whereas screenshots of the prototype will be presented in Chapter 10.) The *tasks* dimension is mapped to the y-axis where the task-subtask relation defines the hierarchy. The dimension *plant parts* is mapped to the x-axis. Each cell of the pivot table holds one stacked bar for the planned workload on the left and for the actual workload on the right side. The roles are mapped to the colors of the stack layers. The time frame is set from the beginning of the project

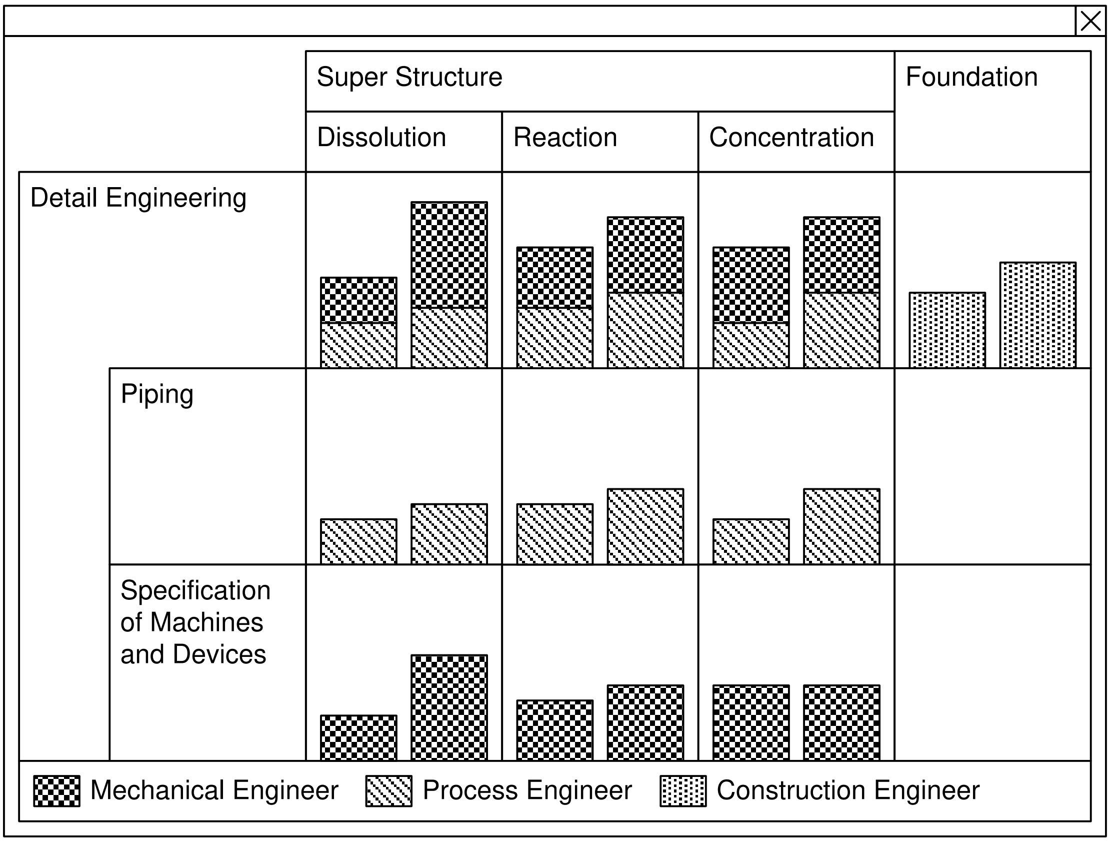


Abbildung 8.15: Example configuration of the pivot table.

to the current date. By means of this view configuration, the project responsible can inspect, how much workload has been planned for the tasks in the project distributed over the different plant parts, and how much effort has actually been spent on the respective tasks. The values in the row for the task Detail Engineering are the aggregated values of all subtasks and task assignments.

The project status analysis view allows virtually any *combination* of dimensions and measures and thereby provides many *different perspectives* on the project management data. It supports all common operations on a multidimensional dataset [JLVV00]: Drill-down, roll-up, pivot, slicing and filtering. With *drill-down* and *roll-up*, a dimension is added or removed from the visualization respectively, e.g. no dimension is mapped to the x-axis or the colors of the stacked bars. The *pivot* operation changes the mapping of dimensions to axes while the number of displayed dimensions stays the same. A *slicing* operation fixes the coordinates on one or more dimensions, e.g. it sets a fixed date for which the management data shall be displayed. *Filtering* excludes certain coordinates of a dimension from being displayed, e.g. only the basic and detail engineering phases could be displayed on the y-axis while the preliminary planning phase is filtered. The values in the hypercube are *aggregated* along all dimensions which are not displayed, and which were not subject to a slicing operation.

A view configuration can be *manually assembled* by selecting the dimensions and measures individually. Furthermore, a configuration can be changed into another configuration by applying the aforementioned operations on multidimensional data sets. However, some configurations are more useful than others for project status analysis. The most *common configurations* have been identified and can be directly selected by the user. On the one hand, common views provided by conventional project management systems (PMS) like Microsoft Project can be configured.

- Gantt Chart
- Task Usage
- Resource Graph
- Resource Usage

The *Gantt Chart* configuration maps the *tasks* dimension to the y-axis, the *time* dimension to the x-axis, and shows the planned workload in the cells of the pivot table. The configuration *Task Usage* additionally maps the *resources* dimension to the colors of the stacked-bar layers. The *Resource Graph* shows the resources on the y-axis, the timeline on the x-axis, and the planned workload in the cells. The *Resource Usage* configuration additionally maps the *tasks* dimension to the colors of the stacked-bar layers.

On the other hand, there are view configurations, which are not provided by conventional PMS. These configurations take specific modeling concepts of dynamic task nets as well as domain-specific aspects into account, namely functional roles, task execution states, progress degrees, and plant parts. The following configurations are available.

- Task Workload
- Technical Crews
- Task States

The *Task Workload* configuration has been depicted in Figure 8.15. It can be used to analyze the planned and actual workload of the tasks with respect to the different plant parts. The view configuration *Technical Crews* maps the *roles* dimension to the y-axis and the *time* dimension to the x-axis. The resources are mapped to the colors of the stacked-bar layers. The configuration can be used to analyze the scheduled workload for the different technical crews in the upcoming weeks.

A different perspective on the tasks in the project is provided by the view configuration *Task States* which gives insight into the *current execution states* of the tasks in the project This configuration is depicted in Figure 8.16. The time frame is set to the current date. As in Figure 8.15, the tasks dimension is mapped onto the y-axis of the pivot table. However, this time the tasks are grouped by the process templates of which they are copies. Consequently, the values displayed in a row for a certain process template are the aggregated values of all copies of this template

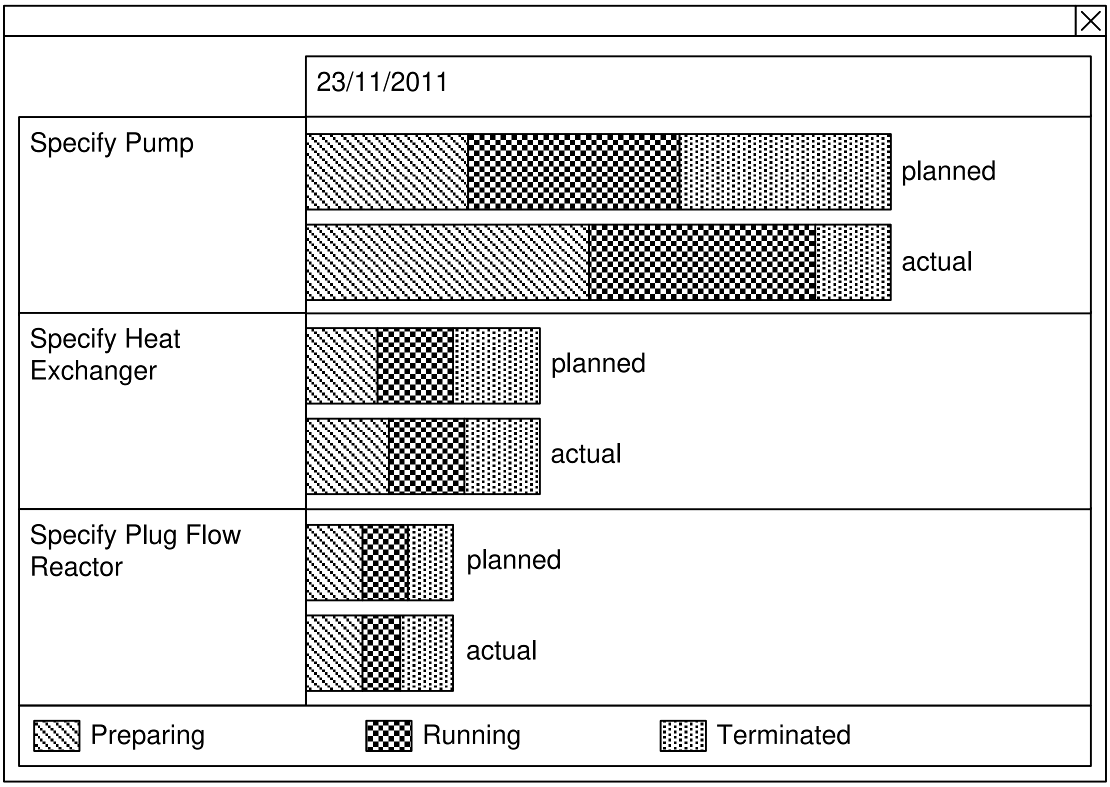


Abbildung 8.16: Task States configuration of the project status analysis view.

in the dynamic task net. The colors of the stacked bars indicate the execution states of the tasks. The measure is the planned total workload of the tasks. This measure has been chosen instead of the tasks count, so that tasks with a high total workload are adequately represented. By means of the view configuration *Task States*, the project responsible can inspect, how much work is already successfully completed, how many tasks—as measured by workload—are currently running, and how much effort still remains for the preparing tasks. These actual values are compared with the planned values which are derived from the planned start and end dates of the tasks. This view configuration is useful for project controlling to analyze the overall performance of all process instances derived from certain templates.

In the example of Figure 8.16, the measured values which are mapped to the bar stack widths indicate that for the process template *Specify Pump* less tasks have been terminated than planned, mainly because several tasks have not been started as planned. At the same time, the enactment of the tasks derived from the process templates *Specify Heat Exchanger* and *Specify Plug Flow Reactor* goes nearly as planned. The project responsible could gain even more insight into where the bottleneck with respect to the specification of pumps is by mapping the resource dimension to the x-axis. This would reveal, if certain resources are responsible for the delay.

Project controlling and process improvement When the view configuration of Figure 8.15 is used after the end of the project or the end of a certain project phase, it can reveal that the planned effort defined in task types or process templates was unrealistic, and that the process knowledge has to be improved.

In general, the monitoring view can be used for two different purposes: for *process controlling* and for *process analysis*. While the former may lead to corrective measures or plan changes at runtime of a project, the latter can be used for *process improvement*. During the course of a development project, the different view configurations can reveal, if certain tasks exceed their time limits, or if the count of completed process instances of a certain type lies below the planned number. After the completion of an engineering project, the actual workload for tasks and subprocesses can be compared with the required workload defined in the task types and process templates. The latter can be adapted if necessary. Furthermore, actual resource allocation data can be evaluated and compared with the defined assignments in the process templates.

8.3.4 Analyzing the History of Plan Changes

Besides the pivot table, an additional view is provided to analyze the past performance of the enacted development process and the history of plan changes. For every task in the project, the values of key properties can be displayed in line diagrams which show how these values developed over time. The planned total workload of a task can be displayed as it was planned at certain dates in the past. Accordingly, the planned degree of completion can be visualized. Regarding process performance, the accumulated actual workload of a task can be displayed which yields a monotonically nondecreasing function. In contrast, the actual degree of completion may also decrease compared to earlier values. Finally, the performance indices SPI and CPI can be displayed.

Planned values and the values of derived properties show the planning state for a certain date. Therefore, the dates refer to the time stamp that is assigned to each record in the data warehouse. The management data can be exported to the data warehouse in regular intervals, e.g. every night or once a week. Furthermore, it is technically possible to export changes to the management data incrementally after every change operation, e.g. after the creation of a new task. Based on the time stamp, it is possible to visualize how the planned values have changed over time. This allows for a detailed analysis of the planning process itself, e.g. it is possible to see if the plan had to be adapted several times in the course of the project. In case of actual values, the dates refer to the time dimension in the data warehouse. In this case, only the most recent time stamp is relevant. Actual values which have been determined for a date in the past cannot change anymore.

The successive property values can be visualized in separate line diagrams and can even be combined in one diagram. This combination allows, for example, to investigate whether a decrease of the actual progress of a task is related to plan changes. This situation is illustrated in Figure 8.17 where the degree of completion

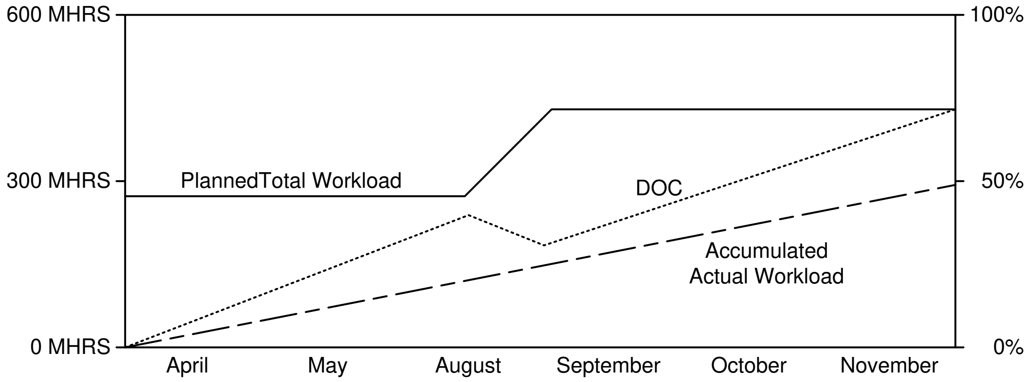


Abbildung 8.17: Visual comparison of the development of property values.

of a task decreased because its total workload was increased.

8.3.5 Coupling with Management Views

The views for project management are *coupled* with the project status analysis view by navigation functionality. To take immediate action based on the analysis of the running process, a user can navigate from the project status analysis view to specific tasks in the task net or task list view. For example, if the view configuration *Task States* reveals that a task has not been completed as planned, the user can directly navigate to this task in the task net view to take corrective measures or to adapt the plan. Vice versa, the project manager often needs additional information when he is replanning the project. For example, if he is looking for an additional resource for a task, he can *navigate* from the dialog for the definition of a task assignment to the adequately configured analysis view, which shows him the utilization of the resources that can play the required role. General purpose multidimensional visualization tools fail to provide this tight coupling with project management views.

8.4 Related Work

Research work that is related to the project monitoring approach presented in this chapter can be divided into two categories. First, there are approaches that specifically address progress measurement of development processes. Second, research has been undertaken regarding the processing and visualization of project management data for project status analysis.

8.4.1 Progress Measurement of Development Processes

Liefeldt et al. A specific approach for measuring the progress of a plant design project in Comos has been presented in [LGB$^+$05]. Liefeldt et al. describe how the project's status can be automatically retrieved from the Comos database by analyzing the engineering data. Progress measurement can be continuously performed, simultaneously to the performance of the design process. It does not depend on reports issued by assigned resources, and it is not restricted to certain time points, e.g. once every month. The progress data is not stored in a separated database but together with the engineering data in the Comos database. The engineering data is systematically analyzed and evaluated to determine the project's status. The basis for this analysis is the object-oriented approach to plant design in Comos. Additional objects are inserted into the database for the purpose of progress measurement. So-called check objects perform checks on the engineering data and can be divided into three different types. An attribute check determines whether a certain attribute value is set. A document check determines whether required revisions of a document have been created. Finally, a reference check determines whether certain objects are linked with each other. The check objects can be associated with the main engineering phases in a plant design process, i.e. to successfully complete a certain phase, all associated checks have to be successful. Checkpoint objects in the Comos database search for check objects, perform the corresponding checks, and visualize or propagate the results. Various search criteria can be specified including the device type, plant part, functional role, and the engineering phase. Checkpoints can be structured hierarchically, so that the results of lower checkpoints are aggregated by superior checkpoints. Checks and checkpoints can only be performed and evaluated when the corresponding engineering objects have already been instantiated, which is not necessarily the case for all objects in the early phases of a plant design project. Therefore, a quantity structure of the expected number of engineering objects is created at project start. In this way, the existing objects can be related to the expected total amount of objects, even in early phases of a plant design project.

The proposed approach to progress measurement has the advantage that it does not require any additional measuring effort at project runtime. Members of the project team do not have to explicitly report their progress. On the other hand, there is a comparably large effort involved with customizing a Comos project to enable progress measurement. The aggregation of check results at checkpoint objects and the use of a quantity structure to incorporate not yet existing objects are related to the aggregation method for the DOC of subtasks at a complex parent task in PROCEED, where the total workload of the parent task may incorporate the workload required for not yet existing subtasks. The filters which can be used for the evaluation of checkpoint objects are related to the dimensions in the project data warehouse of PROCEED, where the results can also be structure according to plant parts, functional roles, etc. The significant difference of the progress measurement approach of Liefeldt et al. compared to the approach presented in this thesis is that the former does not rely on an explicitly defined process model. This has the disadvantage that the progress of the project and possible delays cannot be associated with specific tasks in the project plan. The same disadvantage has been identified for the common progress measurement approach from practice which is only based on document states. For this reason, this approach has been integrated with dynamic task nets in

PROCEED as described in Section 8.1. In the same way, the approach of Liefeldt et al. could be integrated with the progress measurement capabilities of PROCEED. Check objects would not only be associated with the engineering phases but could be associated with individual tasks as well. This integration has not been performed because the technical details of the approach presented in [LGB^{+}05] have not been available. On the conceptual level, only document checks have been integrated into the progress measurement approach implemented in PROCEED.

Gupta and Buddhdeo An early publication on progress measurement in plant engineering projects is [GB83]. Gupta and Buddhdeo present an approach which awards progress at the completion of job steps which are physically measurable. The approach implicitly takes into account the efficiency of the assigned resources. In this regard, it differs from the common methods for progress management which were used at that time including estimate to completion. The main project phases are divided into activities which are further divided into so-called job steps. The job steps of a common parent activity are sequentially executed. The progress of a project phase is computed as the weighted average of the progress degrees of the activities, weighted by their estimated required workload. The progress of an activity is determined by the last job step which has been completed. Every job step defines an overall progress of the parent activity. The overall progress of a job step is determined before project runtime based on the artifacts which have to be produced in the activity. The man hours required to create the various artifacts are distributed to the work steps. The proportion of the cumulative man hours required to complete a job step and its preceding job steps of the total required workload to complete the activity determines the overall progress defined for the job step.

The progress measurement approach presented in [GB83] is related to the approach presented in this thesis in many ways. In both cases, the actual physical progress of tasks is measured instead of the mere progress in time. The aggregation method used in [GB83] is similar to the one presented in this thesis with the exception that in [GB83] no workload can be assigned to the complex task. Gupta and Buddhdeo restrict their approach to the first four levels of the work breakdown structure comprising project phases, activities, and job steps. The approach presented in this thesis can be applied to task net hierarchies of arbitrary depth. Job steps in [GB83] correspond to work steps in PROCEED in that they define steps in a procedure which is performed to execute a task. However, the progress measurement based on job steps is related to the progress measure *Milestones* which has been introduced in this chapter. Job steps and milestones define an overall degree of completion of the parent task which is reached when the job step/milestone is completed. The determination of the overall progress associated with job steps is related to the progress measure *Document States*. A job step implicitly defines development states of the artifacts produced in the corresponding activity. The workload required for the various artifacts to reach the respective development states is combined to obtain the workload required for the job step. In a sense, the concepts underlying the progress measures *Milestones* and *Document States* are combined in the approach for measuring the progress of an activity in [GB83]. Altogether, the basic ideas of Gupta and Buddhdeo have been picked up in this thesis and have been extended resulting in a more elaborate and flexible progress measurement approach.

Daubner et al. In [Dau05, DHW06, DWH06, Dau08] Daubner et al. present an approach for progress measurement in software development projects, which anchors the progress measures to elements of the instantiated process model. This allows to define measures independently of a concrete project, in particular before the start of a project. The term process model is solely used for process model definitions [DWH06]. The V-Model XT is used as an example for a process model. It defines roles, activities, and products. The work breakdown structure (WBS) of a project is used to connect the actual products produced in a project with the activities of the process model. A product is associated with a work package in the WBS which is in turn an instance of an activity defined in the process model. In this way, the lines of code of source files which belong to a certain process activity can be determined at project runtime. Further measures can be defined by the process manager and selected by a project manager. To make the measured values of different projects comparable, a standard WBS is defined and tailored for individual projects. The approach to software process measurement has been implemented as an extension of the well-known tool Maven, which supports the management and the development process of Java projects, resulting in the Maven Measurement Framework (MMF). The unique WBS code of a work package is used in a time recording system to track the effort spent by resources. Furthermore, these references to the work breakdown structure are also maintained in the configuration management system and the bug tracking system to relate the progress on products to work packages in the WBS.

The described approach for software process measurement allows to define progress, productivity, and quality measures in advance before the start of a project. Several measures can be automatically evaluated at project runtime. However, to track the effort spent by the assigned resources on work packages, a time recording system has to be used which requires manual data input. With respect to progress measurement, it does not become clear how the reference for 100% completion of a work package is defined, and whether the effort required for future process phases is taken into account. A degree of completion of running work packages cannot be determined. Only the deviation of the manually estimated expected duration from the planned duration is determined in [Dau08]. Consequently, the support for project controlling is limited. The main disadvantage of the approach of Daubner is that no explicit representation of a process model instance is maintained. Only the WBS of a project is used for measurement but not the project plan with scheduled dates. Execution states of tasks and actual data flow are not modeled. As a consequence, the connection between product versions and activities of the process model definition has to be established by the WBS codes of work packages. In contrast, timed dynamic task nets are used in PROCEED to capture all aspects of a running development process. The enactment state including the actual data flow is explicitly modeled and is used for progress measurement.

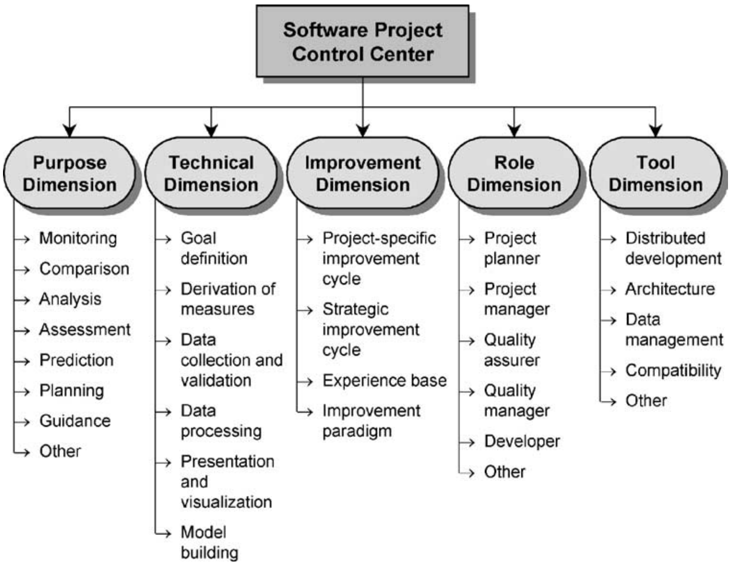


Abbildung 8.18: Dimensions of the SPCC taxonomy [MH04].

Software project control centers In [MH04], Münch and Heidrich present the concept of a software project control center (SPCC) which is a means for collecting, interpreting, and visualizing measurement data in order to provide purpose- and role-oriented information to all involved parties during the execution of a software development project. The involved parties include the project manager and the quality assurer but also the developers. The input information of an SPCC includes information about project goals and characteristics, project plan information, measurement data of the current project, and empirical data from previous projects. An SPCC visualizes measurement data depending on the context of the project, the purpose of the usage (e.g. monitoring), and the role of the user. A reference model for concepts and definitions around SPCCs is presented in [MH04]. An SPCC can be classified according to five dimensions: purpose, technical, improvement, role and tool dimension, which are presented in Figure 8.18. The purpose of an SPCC can, e.g., cover the monitoring of a running project and the prediction of future project behavior. On the technical dimension, the different purposes of data collection and presentation/visualization are distinguished among others. Münch and Heidrich also describe, how an SPCC can be integrated into a software engineering environment. This integration is illustrated in Figure 8.19 where four different levels of a software development model are distinguished: roles, services, tools, and information. The SPCC is neither used for project planning nor for the collection of measurement data, but it relies on the information provided by these tools.

The concepts and definitions regarding software process control centers presented in [MH04] can be transferred to development processes in general. This allows to characterize the functionalities provided by PROCEED using the terminology introduced in [MH04]. With respect to the dimensions for classifying an SPCC, it can be stated that PROCEED fulfills the purposes of monitoring, prediction, planning, and guidance. On the technical dimension, the functionality of PROCEED covers data

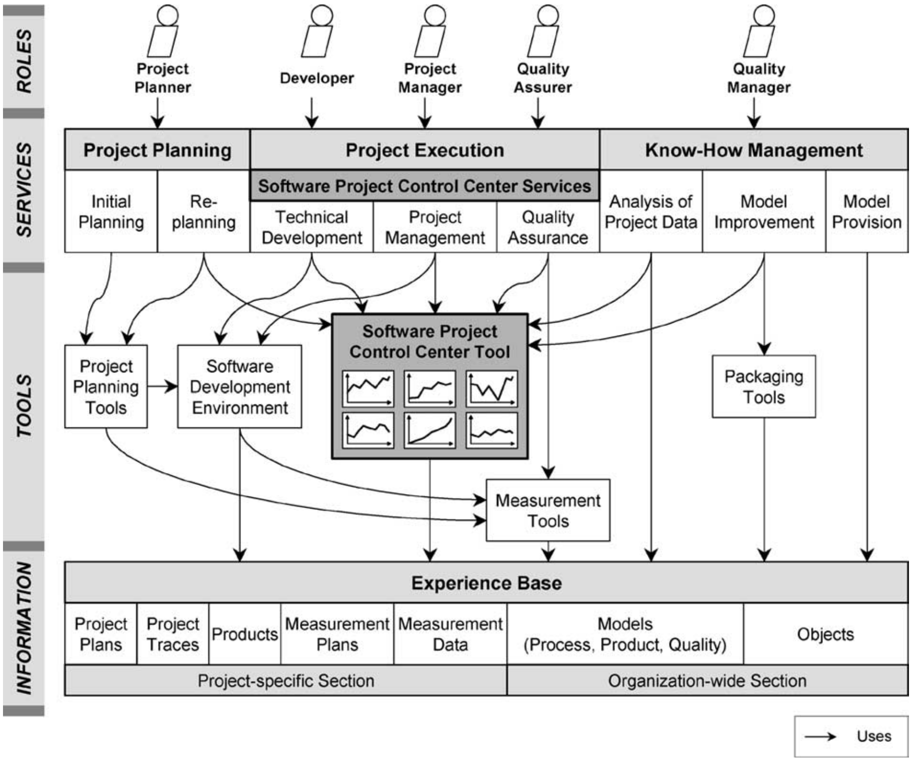


Abbildung 8.19: Software development model [MH04].

collection and validation, data processing, presentation and visualization, as well as model building. Because knowledge gained in a project about the enacted processes can be reused in future projects, PROCEED can support the project-specific improvement cycle as well as the strategic improvement cycle. It provides an experience base containing process models and historical planning data. Finally, different roles may use the PROCEED system, including project manager and developer. When the software development model of Figure 8.19 is transferred to the general case of a development process, it can be seen that PROCEED and Comos together cover the whole functionality provided by all depicted tools including project planning and progress measurement, where the software development environment has to be replaced by the design and specification functionalities of Comos. The Comos database and the PROCEED project data warehouse together provide the knowledge which is contained in the experience base of an SPCC.

Progress measurement of workflow instances The publications of Eder et al. have already been reviewed in Section 7.6 with respect to temporal analysis of workflows. In [EPPR99], Eder et al. furthermore present an approach to monitor the performance of running workflow instances. At workflow runtime, the internal deadlines which are computed for every task are used for the early detection of delays. Traffic light colors are used to visualize the status of a running workflow instance with respect to the defined deadline. If a workflow is green, it is expected to finish in time without skipping tasks or choosing shorter paths (cf. Section 7.6). A yellow workflow can still be finished in time by skipping certain tasks or choosing shorter paths. A red workflow can only be finished in time if certain tasks are completed with shorter durations than planned.

The warning system implemented in PROCEED also uses three different color values to visualize the delay or budget overrun of a task. However, while the approach presented by Eder et al. is merely based on internal deadlines for workflow tasks, the warning system implemented in PROCEED is based on a resource-feasible schedule used as a reference and earned value analysis for extrapolating the amount of delay and budget overrun at task completion. The approach of Eder et al. can only be applied to complex, workflow-managed tasks, while the approach based on earned value analysis can be applied to all tasks in a project.

8.4.2 Visualization of Project Management Data

The visualization of multidimensional data and especially the application of these techniques to project management have been tackled in some related research projects which are described in this section. However, none of the related works covers the full cycle depicted in Figure 8.11. Furthermore, domain specific measures and dimensions are neglected.

Stolte et al. Polaris [STH03] has been a research project at Stanford University concerned with the visualization of multidimensional data from a data warehouse. The developed prototype offers a configurable pivot table whose axes can be associated with the dimensions of a data cube. The entries of the pivot table can be numbers or even diagrams, and their color can also be associated with a dimension. The approach is closely related to the multidimensional visualization of project management data in PROCEED. However, since Polaris is a general purpose visualization tool, it does not integrate with a process management system, i.e. no navigation from an analysis view to corresponding management views is possible.

Songer et al. In [DBC04], an application with different multidimensional views for project status control of construction projects is presented. The analysis is limited to budget data, and there is no pivot table among the visualizations. The focus lies rather on a comparative study about the usefulness of different diagrams than on the development of a user interface concept for a project management and controlling tool.

Nie et al. Another approach which applies OLAP technology to project status analysis can be found in [HN07]. A multidimensional data model for a data warehouse is developed, which comprises five dimensions and the measures person hours, actual costs and planned costs. A pivot table is used to generate different views on the project data using MDX-queries. The approach uses a simple star schema for the data model instead of a more complex snowflake schema like in the approach presented in this thesis. No information about plan changes in the monitored project is stored in the data warehouse. No graphical visualization techniques are applied to present the data in the pivot table. Only the standard functionality of the SQL-server is used. The focus of [HN07] lies on the evaluation of the OLAP technology for project management, but not on a suitable visualization of the data.

Eder et al. In [EOG02], Eder et al. present a general architecture for a workflow log data warehouse. The enactment data which is collected by a WfMS at runtime of the workflow instances is exported to the data warehouse and processed. The measured values stored in the cells of a hypercube which is spanned by several dimensions. The data warehouse can be queried in several different ways to analyze the performance of the workflow instances, e.g. the number of workflow instances started by a certain agent for successive dates can be determined.

The approach presented by Eder et al. is related to the project data warehouse of PROCEED. In both cases, runtime data of process instances is exported to a data warehouse, processed, and queried for monitoring and controlling as well as process improvement. Both warehouses store the data for subsequent points in time for trend analysis. However, the two approaches have different purposes which has lead to significantly different warehouse architectures. While the data warehouse of Eder et al. collects runtime data of independent workflow instances which represent business processes in an organization, the workflow instances in the PROCEED data warehouse are all embedded in a development project. In [EOG02], no planned values for the workload and duration of workflows and no scheduled dates are considered. Only the actual durations of workflows and workflow tasks are determined, as well as the estimated and actual overtimes with respect to defined deadlines. A comparison of planned and actual values beyond the end times of workflows is not possible because the workflow instances are not scheduled. The visualization of the consolidated data in the data warehouse is not treated in [EOG02]. Merely, line diagrams are used to display the trends of individual measured values. In contrast, the multidimensional visualization approach realized in PROCEED allows to compare different key figures of the enacted processes. Finally, the data warehouse is not connected with the WfMS in [EOG02], i.e. the systems are uncoupled and it is not possible to navigate from the analysis views to the workflow management views to take corrective measures.

Statistical process control The *statistical process control (SPC)* has been applied in production management systems for decades but only recently for controlling software development processes [FC99, DCA04]. Several different chart types or diagrams are used for SPC. A run chart can be used to track trends of measured values over a period of time. A control chart is a run chart in which upper and lower levels of tolerance are defined for the measured values. The monitoring system informs the process owner if a value is above the upper or below the lower level. Early warning messages can be generated if a certain number of successive values are continuously increasing or decreasing. Histograms are used to present data by frequency, i.e. each bar represents the number of observations in a certain time frame that fit in the indicated range. A Pareto chart is a specific bar chart that presents prioritized in some fashion. Finally, scatter diagrams plot data points, allowing trends to be observed between one variable and another.

The measured values which are exported to the project data warehouse of PROCEED can be visualized in different diagrams. Run charts can be used to visualize the planned workload, actual progress, etc. of tasks. The SPI and CPI of a task can be visualized in control charts with lower levels of tolerance. The pivot table for project status analysis can be configured to show bar charts which resemble histograms, although in PROCEED there are no measures which merely count the occurrences of events. Besides the diagram types, the SPC defines several statistical methods of data analysis which are, however, not related to the approach for progress measurement presented in this thesis.

8.5 Conclusion

In this chapter, the capabilities of PROCEED with respect to project monitoring have been described. Several different progress measures are available to determine the degree of completion of a task. The most common measures from practice have been integrated with new computation methods which exploit the specific modeling capabilities of the TNT meta-model. One of the new measures takes the actual data flow into account, i.e. the produced document revisions which are explicitly modeled in dynamic task nets. The second new measure uses reference data available for workflow templates to automatically determine the degree of completion of an enacted workflow. Earned value analysis is applied to compare the actual performance of tasks with the plan and to quantitatively measure deviations. In this way, project planning, scheduling and monitoring are integrated.

The multidimensional visualization of the project management data complements the project monitoring functionality based on the degree of completion and performance indices of tasks. The project management data is exported to a project data warehouse to be processed for visualization. The export is performed in regular intervals, so that subsequent planning states are stored and can be analyzed. This ensures the traceability of plan changes.

Altogether, this chapter demonstrated the integrated approach to project monitoring implemented in PROCEED. Depending on the analysis of the current project status, replanning and rescheduling may be required. In the next chapter will describe how changes to a timed dynamic task net are performed in a controlled fashion.

Kapitel 9

Change Management

The authorization model for dynamic task nets which has been presented in Section 5.5 ensures that project team members can only make changes to the dynamic task net according to their personal permissions and their task assignments. However, avoiding unauthorized modifications of a task net does not support the responsible resources in performing their management activities. Therefore, the workflow approach has been applied to define and enact management processes in addition to technical processes. The management processes have to be handled differently compared to the technical processes in a development project. The solution for the controlled enactment of management processes in PROCEED is described in Section 9.1.

During project runtime, when the defined processes are enacted, many disruptions can occur which require changes to the calculated schedule. These disruptions include bad performance of subprocesses, unexpected unavailability of resources, and additional work due to changed requirements. An overview over possible disruptions in a project and according compensating actions is given in Section 9.2. The project management control cycle which has been introduced in Section 3.3 shows that replanning of a project may be required due to performance analysis. This includes the rescheduling of tasks at project runtime. Managing changes to a project plan and integrating planning and scheduling are two of the open research questions with respect to resource-constrained scheduling [Smi03]. The scheduling algorithm presented in Chapter 7 allows to repair a disrupted schedule. However, it does not define how manual plan changes to a dynamic task net can be performed at runtime and when rescheduling is performed in the process. Section 9.3 describes the general change management procedure, which defines how changes to a timed dynamic task net have to be performed during enactment in order to avoid violations of timing consistency constraints and authorization rules. In particular, it is shown how the procedure is applied to change the values of dependent task properties.

9.1 Enactment of Management Processes

Project management covers various different activities including project planning, reporting, change management, and quality management (cf. Section 3.1.1). Project management processes are often standardized within an organization, to ensure

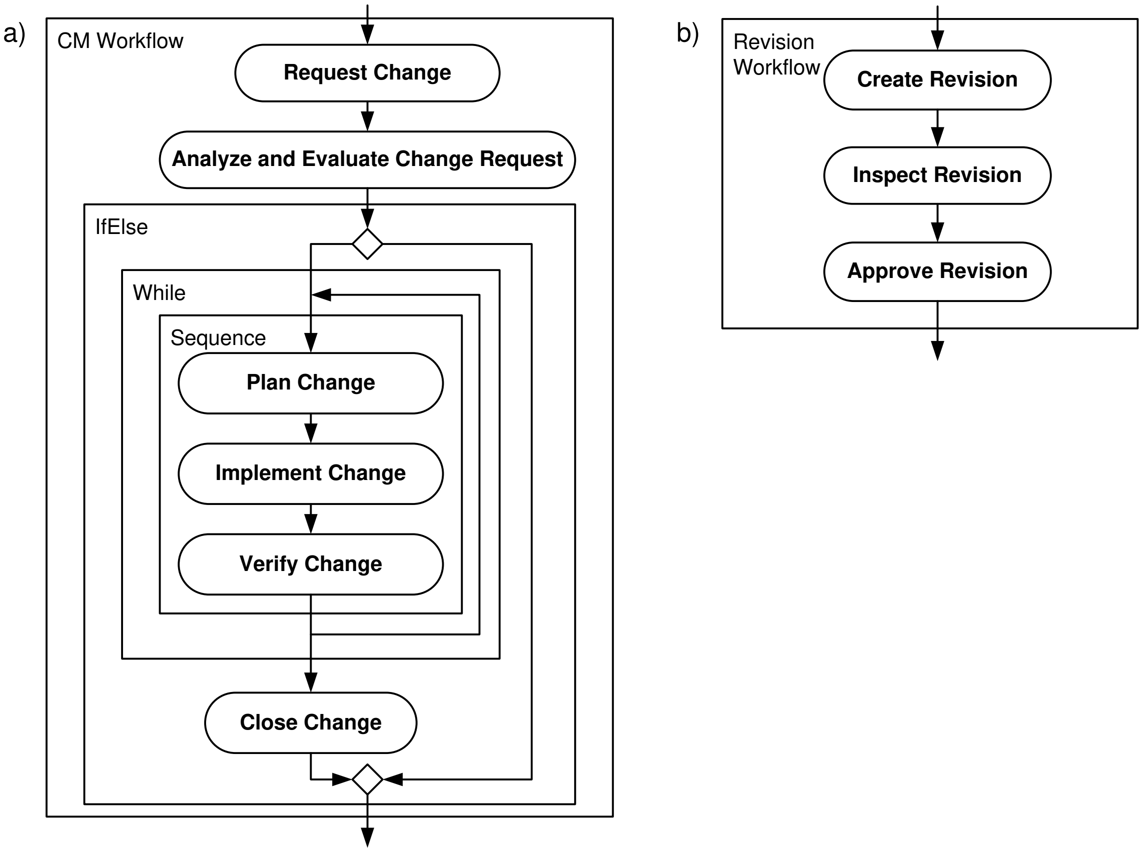


Abbildung 9.1: Examples for management workflows.

organization-wide quality standards. In particular, the following project management activities are usually performed according to predefined procedures.

- Reporting
- Change management
- Quality management

Reporting refers in this case to the submission of problem reports by people responsible for individual work packages or tasks. *Change management* includes changes to the final product of the development process as well as changes to the project plan, where the former usually involves the latter. Finally, *quality management* refers to processes in a development project which ensure the quality standards demanded for the final product, e.g. reviews of documents. Processes which belong to the listed project management activities can be *explicitly defined* as process model definitions. Instances of these process model definitions are enacted whenever a problem occurs, a change is requested, or the quality of an intermediate product has to be verified in a development project.

In Figure 9.1 a), an example for an explicitly defined change management process is depicted, which has been adapted from [Wik10]. In general, change management has two main objectives [CAD03]. The first is to provide support for the processing of changes, which includes requesting, determining attainability, planning, implementing, and evaluating of changes. The second objective is traceability, i.e. it should be possible at any time to list all active and implemented changes. Both objectives can be achieved by enacting change management processes in a process management system. In the change management process depicted in Figure 9.1 a), a change is identified and a change request (CR) is created in the first task. The reason for a change can be the correction of an error or the improvement of the (design of the) product. Authorized resources analyze the CR and determine the action to be taken. In large development projects, the evaluation of a change request is often performed by a so-called *change control board* [CAD03]. If the change is approved, the required tasks for the implementation of the change are planned and assigned to qualified resources, which are then responsible for implementing the change. When the assigned resources have completed the implementation, the change has to be verified, e.g. by means of reviews. If the verification is not successful, additional work has to be performed to finally implement the change correctly and completely. In the end, the change request is closed.

The enactment of such change management processes enables the traceability of changes in a development project. A general change management procedure can be prescribed for all change management cases in the development projects of an organization. Such a procedure ensures that every change request is handled in a structured way and that the responsibilities for the evaluation, implementation, and verification of changes are clearly defined. For this reason, a change management procedure is not adapted for individual cases. Although the actual cases differ, i.e. the changes to the product and the involved changes to the project plan are different from case to case, the general procedure which specifies how to proceed in these cases is not changed.

The same argument holds for quality management processes like document reviews. Figure 9.1 b) shows an example of a review process which consists of three steps: creation, inspection, and approval of a document revision. This process reflects the common steps which have to be performed in the Comos system before a revision of a document can be released. The enactment of the process ensures that only authorized resources may perform the inspection and approval tasks of the process.

All in all, management processes for project controlling, change management, and quality management are rather static, in contrast to development processes. Furthermore, management processes comprise alternative courses of action which depend on management decisions made at process runtime. The alternatives are predefined before process runtime. As a consequence, the workflow approach is suitable for the modeling and enactment of management processes. Therefore, the workflow management functionality of the PROCEED prototype, which has been described in Section 6.3, is not only used to support technical processes but also management processes.

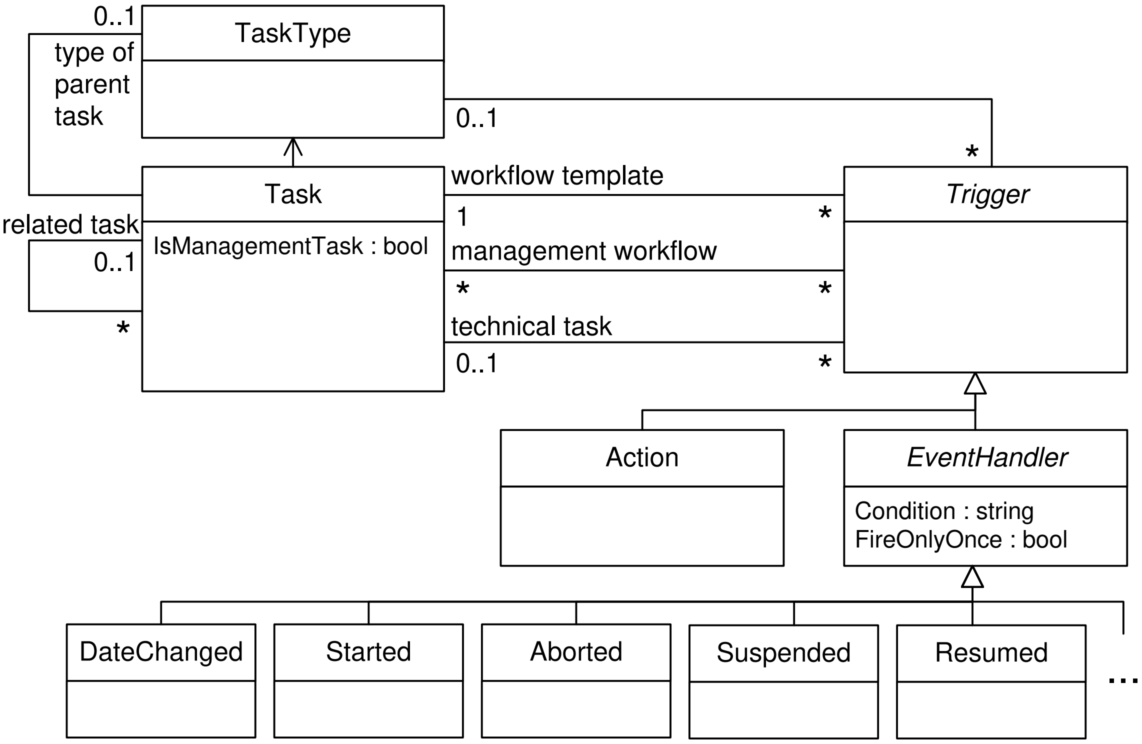


Abbildung 9.2: Management extensions of the TNT meta-model.

Figure 9.2 shows the *management extensions* of the TNT meta-model which are required to model and enact management processes in addition to development processes in PROCEED. The new classes, properties, and associations defined in the class diagram will be explained in the following sections.

9.1.1 Management Tasks

Management tasks define work which has to be performed to manage a development project. They are distinguished from technical tasks in a task net by setting the property `IsManagementTask`, which is defined for the class `Task`, to the value `true`. Management tasks can be atomic, but they can also define complex processes. Workflow templates can be defined for management processes in the same way as for technical processes, including task assignments with required roles and data flows between the workflow tasks. A workflow template for a management process is enacted in the form of a dynamic task net as described in Section 6.3. Both, the workflow-managed task net and its workflow tasks are management tasks. A workflow-managed management task is called a *management workflow*. Its subtasks may be manual management tasks or again management workflows.

Although the same tools and mechanisms are applied for the modeling and enactment of management workflows as for technical workflows, they represent processes of a different kind. Management processes which are enacted in a development project are not subprocesses of the overall development process. The subtasks of enacted management workflows are not part of the project plan. On the contrary, the execution of management tasks may involve changes to the project plan, e.g. to handle a change request. The successful completion of quality management processes may be a prerequisite for certain state changes in the dynamic task net representing the development process, e.g. several review workflows may have to be completed before a milestone task may be committed. Management tasks are usually not scheduled in a development project. They are rather executed as soon as possible and are therefore maintained in todo-lists. The explicit distinction between development processes and management processes in a project was already made in [NW94]. Management processes were enacted to change a development process. The conceptual framework presented in [NW94] will be reviewed in Section 9.4.1.

Although management tasks are not part of the project plan and are not scheduled, they involve costs which have to be incorporated in the project's budget. A common approach in project management is to define a first level element in the work breakdown structure of a project which subsumes all project management activities [Bur00, Hau01]. As explained in Section 3.1, the costs of a project are distributed over the work breakdown structure which covers the full amount of work that has to be conducted in the project. Therefore, also project management is incorporated as a task in the WBS. This approach has been applied for management tasks in PROCEED. All management tasks are located in the task net hierarchy under the task Project Management which is defined by default on the first level below the root task representing the whole project. Figure 9.3 shows the task net hierarchy of the example scenario which is extended by the task Project Management. Subtasks of the task Project Management can be defined to structure the management tasks according to different topics, e.g. Change Management and Quality Management as depicted in Figure 9.3. The task Project Management is a management task itself, i.e. the property `IsManagementTask` is set to `true` by default. Furthermore, a management task can only be created as a subtask of another management task. This ensures that management tasks are always located somewhere below the task Project Management in the task net hierarchy.

The majority of management workflows and tasks in a project are somehow related to the project plan. In PROCEED, the project plan is the scheduled part of the dynamic task net without the task Project Management and its subtasks. The enactment of change management workflows involves changes to the project plan, the successful completion of quality management processes is a prerequisite for certain state changes in the dynamic task net, and reporting workflows are invoked to report problems which are detected in certain tasks of the development process. Therefore, the management tasks are linked to the tasks in the project plan to which they refer. When a management workflow is invoked by a user in his role as the responsible resource of a certain task, then the management workflow is linked to that task. The technical task is then called the *related task* of the management workflow. The corresponding association which has been defined for the class Task is depicted in Figure 9.2.

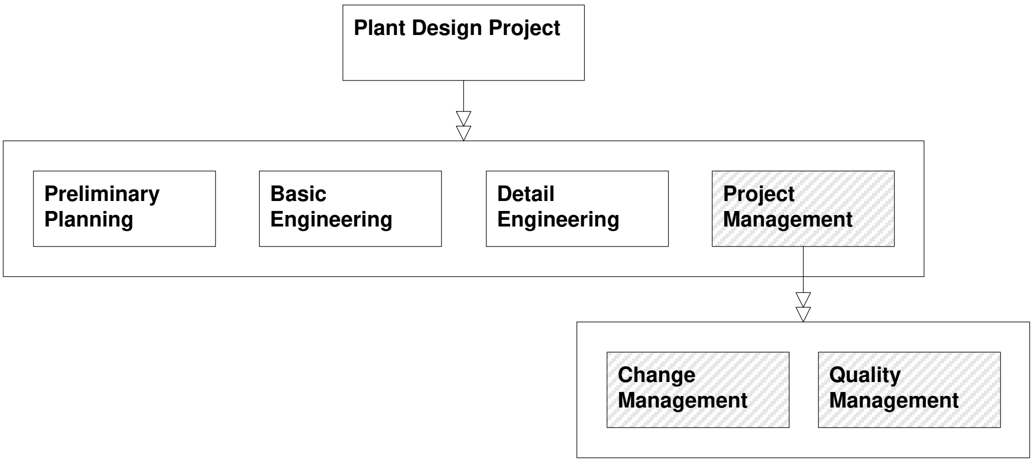


Abbildung 9.3: Management tasks in work breakdown structure.

The procedures which have to be followed in a project to perform the project management activities vary from organization to organization. Within an organization, these processes may be standardized to increase the performance of projects. If individual projects in an organization require higher quality and safety standards than others, then specific management processes may be defined for these projects. As a consequence, the management processes in development projects can be *organization-specific* as well as *project-specific*. This requires the parameterization of management processes and development processes for an organization and its projects.

PROCEED provides the means to define and parameterize management processes for organizations and projects individually. The task types for development tasks as well as the management workflow templates and their subtasks can be parameterized. The definition of management workflow templates is also part of the parameterization process. The examples for management workflows which have been presented in Figure 9.1 are not predefined in PROCEED but could have been defined by a process manager of a plant engineering company which uses Comos and PROCEED in its design projects.

9.1.2 Parameterization of Task Types

In PROCEED, domain-specific task types can be defined which can be instantiated in process model instances in a development project (cf. Section 6.1). These task types for development tasks can be parameterized to specify which management workflows can be started from the respective task instances. This means, that the responsible resource of a task which has been instantiated from a type can invoke those management workflows for the task which have been specified for the type. For example, the responsible resource may want to send a problem report to the responsible resource of the parent task and starts a reporting workflow for this purpose. In addition to manual invocation, management workflows may also be triggered automatically for a task upon certain events which are related to the task. Altogether, there are the following two ways to parameterize a task types with respect to management workflows.

- Actions
- Events

Actions and events are both possible triggers of management workflows. In Figure 9.2 the classes Action and EventHandler are inherited from the abstract class Trigger for which associations to the class Task are defined. Objects of the these classes always belong to a unique technical task or to a task type. Upon instantiation of a task, the Trigger objects associated with the type are cloned for the instance. Furthermore, task instances refer to a workflow template which defines the management workflow to be started. The root task of a workflow template is an object of the class Task (cf. Section 6.2). If one or several management workflows have been started for an action or an event, then they are connected to the respective Action or EventHandler objects.

Actions For a task type, arbitrary user-defined *actions* can be specified and associated with management workflow templates. Performing such an action for a task instance in a concrete project is equivalent to the invocation of a management workflow which is derived from the associated template. Because the actions and associated management workflows can be freely defined during parameterization, only examples for possible actions can be given here.

Report problem The responsible resource of a task may report a problem related to his task, e.g. technical problems may have occurred which will cause a delay of the task. The invoked management workflow routes the information to the resource which has to be informed about the problem. This may be the responsible resource of the parent task or the work package to which the task belongs. It may as well be the group leader of the responsible resource, the project controller, or the project manager. The definition which resource has to be informed is organization-specific and can be specified in the workflow template during the parameterization process.

Request plan change With this action, a responsible resource can request a change to the project plan which is related to his task. For example, if the resource estimates that the task cannot be completed within the planned duration, he may request a duration increase and with it an increase of the planned workload and budget of his task. The management workflow routes the request to the person who is authorized to decide over the plan change. This may be the responsible resource of the parent task or the work package to which the task belongs, or the project manager, depending on how plan change requests shall be handled in the project.

Operation	**Event Parameters**
`Start(t)`	`t`
`Commit(t)`	`t`
`Abort(t)`	`t`
`CreateSubtask(p, out s)`	`s`
`CreateOutputParameter(t, out o)`	`o`
`ProduceRevision(o, out r)`	`o, r`
`DeleteTaskAssignment(a)`	`a.Task, a.Resource`

Tabelle 9.1: Selected events which may trigger management workflows.

Request product change In this case a problem is reported and a change is requested. However, this action differs from the previous examples in that the requested change refers to other tasks in the project. The responsible resource of a task has detected an error in a technical document produced in a previous task which requires the revision of the document. Therefore, a workflow like the one depicted in Figure 9.1 a) is started to handle the change request. The enactment of the workflow involves changes to the project plan.

Events Management workflows can also be invoked automatically upon certain events. An *event mechanism* for dynamic task nets was already introduced in [Kra98]. This approach has been adopted and extended in PROCEED. Every structural and behavioral change operation on a task (net) raises a corresponding event. Table 9.1 lists some operations which raise events. The events carry parameters which describe the context of the event, e.g. which tasks have been involved or which document revision has been created. The corresponding parameters are listed in the second column of Table 9.1.

A task type can be parameterized, so that a management workflow is automatically started when a certain event occurs for an instance of the type. The workflow can be considered as the *event handler*. For example, if a task assignment is deleted, then a management workflow can be started which informs the released resource as well as his team leader about the freed working hours. Furthermore, if a responsible resource of a task aborts his task, then a management workflow can be started which informs the responsible resource of the parent task about the task's failure, so that he may decide on compensating actions or plan changes.

The event mechanism introduced in [Kra98] has been extended by *conditions* for triggering event handlers, i.e. a management workflow which is associated with an event is not invoked in any case, but only if a defined condition evaluates to true. The associations of management workflows to events are specified in the form of *event-condition-action (ECA) rules* [DGe95]. When the event of an ECA-rule is raised and the condition evaluates to true, then the action is performed. The action is the invocation of the associated management workflow. The condition may refer to properties of the task or to its work context. In the following examples, ECA-rules are presented as triples (`Event`, `Condition`, `Action`). The previously described example

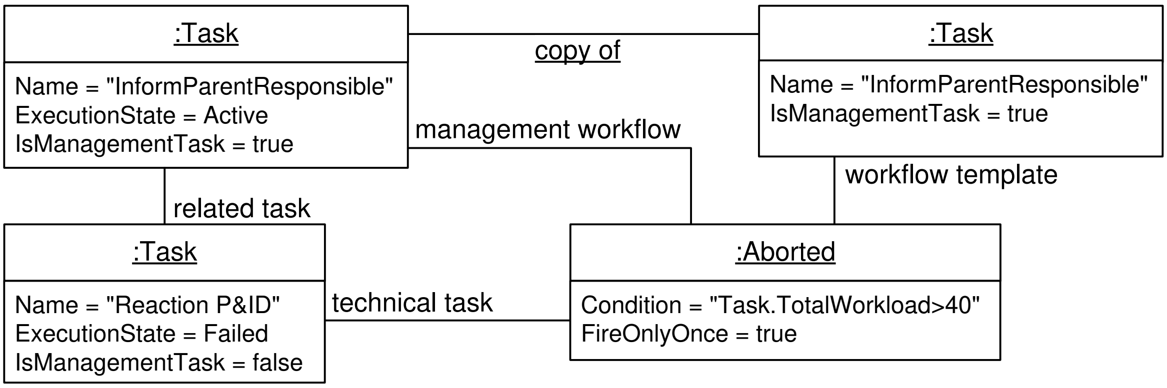


Abbildung 9.4: Management workflow triggered by an event.

for the notification about an aborted task can be refined by a condition.

$$(\texttt{Abort}(\texttt{Task}), \texttt{Task.TotalWorkload} > 40, \texttt{InformParentResponsible})$$

In this case, the notification only takes place if the total workload of the aborted task is larger than 40 man hours. This covers all tasks for which a single resource works full-time for a complete work week. If shorter tasks fail, then no notification workflow is started because this would constitute a disproportionate management overhead. The ECA-rule including the threshold of 40 man hours as well as the management workflow `InformParentResponsible` are project or organization specific and would be defined in the course of the parameterization process.

An ECA-rule is represented by an object of the class EventHandler which is depicted in Figure 9.2. The condition is defined as the textual value of the property Condition. In this section, the conditions are shown in the formal notation introduced in Chapter 5. In the actual implementation, boolean conditions in the programming language C# are used as conditions, which are compiled and evaluated at runtime of PROCEED. The boolean property FireOnlyOnce specifies whether more than one management workflow of the associated template can be started. In Figure 9.4 the situation is depicted in which the previously described ECA rule for the abortion of a task has fired and a management workflow has been started. The management workflow is a copy of the associated workflow template whose root task is also an object of the class Task. The started management workflow is linked to the related technical task. The EventHandler object is also linked to the management workflow. This information is used to avoid that several management workflows are started for an event for which the property FireOnlyOnce is set to `false`.

In addition to events which are raised upon change operations, a *timing event* has been introduced which is raised whenever the current date changes and is therefore named `DateChanged`. This event is required to react to delays of tasks or other deviations from the plan, i.e. whenever a monitoring constraint is violated (cf. Section 5.3.2). Monitoring constraints may be violated only because time proceeds. The timing event was not defined for the event mechanism presented in [Kra98] because the AHEAD system did not cover timing issues. The event `DateChanged` cannot be associated with specific tasks in a project. It applies for all tasks in the same way. Nevertheless, task types can be parameterized by the definition of ECA-rules, so that actions are performed for the task instances on certain dates. In the following example, the responsible resource of a task is notified whenever a subtask is started late, i.e. the planned start time has already passed but the task is still preparing.

$$\begin{aligned}\big(&\texttt{DateChanged}, \texttt{Today} > \texttt{Task.PlannedStartTime} \wedge \texttt{Task.State} \in \texttt{Preparing},\\ &\qquad\qquad\qquad\qquad\qquad\qquad \texttt{InformParentResponsible}\big)\end{aligned}$$

This example of an ECA-rule should only fire once in a project. However, the event `DateChanged` is raised every day, and if the condition is still satisfied on the next day, then another management workflow would be started, so that the same deviation from the plan would be reported over and over again. To avoid this behavior, ECA-rules can be explicitly defined to fire only once. Altogether, there are two different modi for ECA-rules in PROCEED, those which fire only once and those which fire as often as possible. The former modus is mainly used in conjunction with the event `DateChanged`. The latter modus could for example be used for an ECA-rule which should fire whenever a task is resumed from suspension. This may happen several times in a project, and every time a new management workflow should be started.

9.1.3 Parameterization of Management Workflow Templates

Every management workflow which is enacted in a development project is derived from a workflow template. A management workflow template can be defined just like a workflow template for an engineering process. However, additional information is required to define who may invoke the management workflow and which resources shall be assigned to the workflow tasks.

Data flow and decisions Like for technical workflow templates, input and output parameters can be defined for workflow tasks, and data flows can be defined which connect these parameters. In management workflows, data flows are used to specify how reports and other management documents are routed between the management tasks at runtime. Decisions in a management workflow are made by setting the values of decision variables which are defined for management tasks. These decision values are evaluated in the conditions of the workflow's control structures as described in Section 6.3.

External management workflows As described before, workflow templates can be associated with tasks via actions or events. Management workflows which are triggered by tasks in the dynamic task net are called *internal management workflows*. In contrast, *external management workflows* cannot be associated with tasks in the project plan. They are invoked by members of the project team which have a certain role or position in the project. These management processes are enacted to handle changes to the scope of the project and the objectives of project management. For example, customer requirements or external deadlines may be changed. In these cases, the respective management workflows cannot be associated with specific tasks in the project plan. For a management workflow template, it can be explicitly specified whether it defines an internal or an external management workflow. For an external management workflow, the required functional role or position in the project team has to be specified. Only resources which have this role or position may invoke a management workflow of the respective type.

Organization of management tasks Management workflows can be started automatically by the PROCEED system upon certain events. If a responsible resource of a task invokes an action for his task, then the associated workflow is started. By default, a management task representing the management workflow is created below the task *Project Management*. Management tasks can be organized according to different project management activities, e.g. classified into quality management and change management processes like in Figure 9.3. This can be achieved by the parameterization of the management workflow templates. The type of the intended parent task can be specified. For this purpose, the association type of parent task is defined in Figure 9.2 between the classes Task and TaskType. If there is a unique subtask of the task *Project Management* which is of the specified type, then every management workflow which is derived from the template is automatically created as a subtask of this task. This situation is depicted in Figure 9.5 where the management workflow template Document Revision is associated with the task type Quality Management. The management workflow Revision of PFD.D is therefore created automatically below the task Quality Management.

Resource assignment rules For the workflow tasks in a management workflow template, task assignments with required roles can be defined. However, in contrast to technical tasks, management tasks are usually not assigned by means of the pull pattern (cf. Section 5.1.3) where only the required role for a task is specified and an eligible resource can pick up the task, because resource with the same functional role may have different authorizations and responsibilities for managing a project. Instead, management tasks are directly assigned to resources of the project team which are responsible for handling the change management or quality management cases. Management tasks must be assigned to actual resources at any time to avoid that essential management decisions are delayed or overlooked.

The assignment of actual resources to management tasks cannot be done during the parameterization of the workflow templates. Workflow templates are defined independently of any concrete project for which resources could be allocated. However, at project runtime, management tasks have to be directly and automatically assigned to actual resources. For this purpose, so-called *assignment rules* can be defined for task assignments of management tasks. These rules specify how the correct actual resource for a task assignment of a management task is determined automatically at runtime. The following assignment rules are predefined in PROCEED and can be

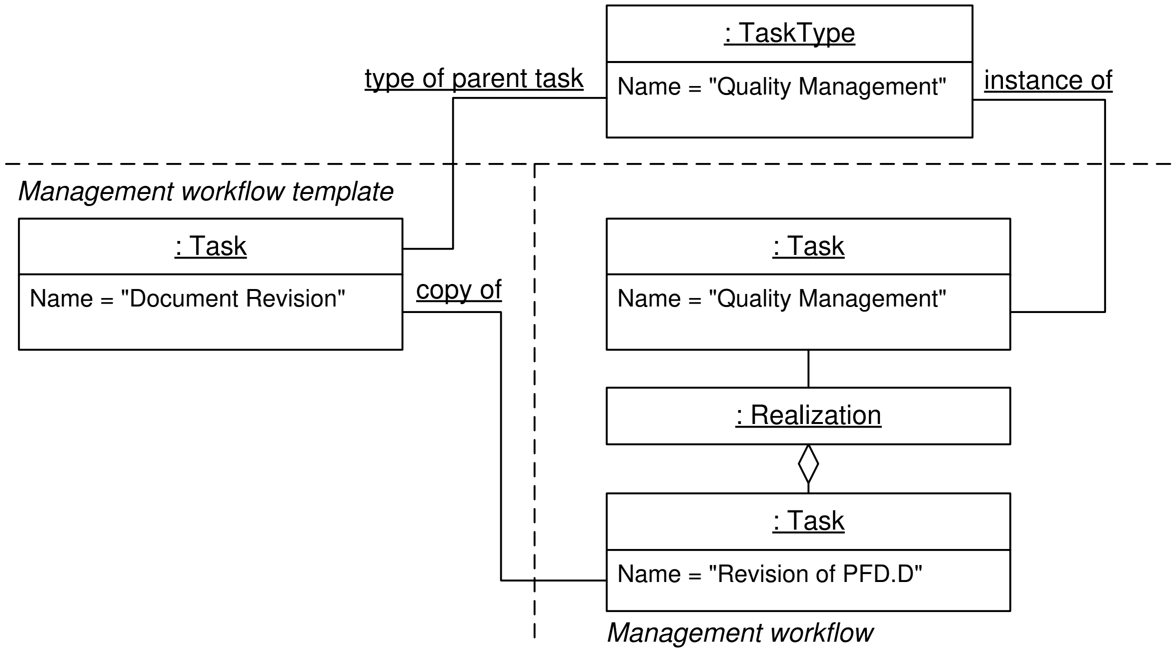


Abbildung 9.5: Example for the automatic organization of management workflows.

used to parameterize management workflows and management tasks. In any case, the selected resource has to be eligible with respect to the required role of the task assignment.

Random A resource who can play the required role is randomly selected from the project team.

Responsible resource of related task The resource who is responsible for the related task of the management workflow is assigned to the management task.

Responsible resource of parent task The resource who is responsible for the parent task of the related task is assigned to the management task.

Responsible of management workflow The resource who is responsible for the whole management workflow is assigned to the management task.

Team leader The leader of the team to which the responsible resource of the related task belongs is assigned to the management task.

If no assignment rule is specified for a task assignment of a management task or no resource can be determined by evaluating the specified assignment rule, then the task is assigned to the responsible resource of the parent task by default. This means a workflow task of a management workflow is assigned to the responsible resource of the whole workflow, and a management workflow is assigned to the responsible resource of the collective parent task which subsumes several management workflows. The automatically assigned resource can then manually reassign the management task if necessary.

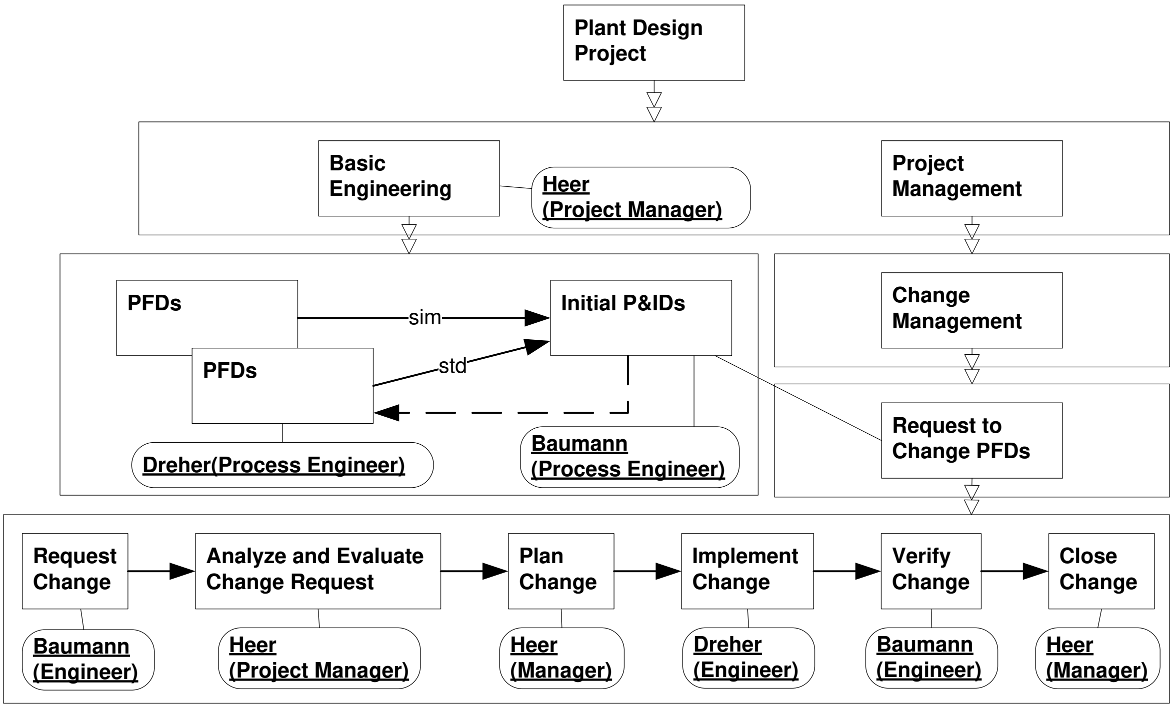


Abbildung 9.6: Example for a change management workflow instance.

9.1.4 Example Case

When a management workflow is copied from a template, it is automatically inserted into the dynamic task net below the task Project Management as described before. If the workflow has been started from a technical task in the project plan, it is linked to this related task. The assignment rules which have been defined for the task assignments of the management tasks in the workflow are evaluated and the determined resources are assigned to the tasks. Afterwards, the management workflow is automatically started.

Figure 9.6 shows an example of a change management workflow and its relation to other tasks in the project. The responsible resource Baumann of the tasks Initial P&IDs has detected errors in the previously created process flow diagrams which need to be corrected. Because resource Baumann is not authorized to change the realization of the task Basic Engineering, he issues a change request by starting a change management workflow. The management task Request to Change PFDs, which represents the management workflow, is automatically created below the management task Change Management and is linked to the related task Initial P&IDs. Different assignment rules have been specified for the subtasks of the management workflow. The management task Request Change has to be performed by the responsible resource of the related task who issued the change request, which is resource Baumann. He has to create a document which describes the encountered problems and his propositions for changes. The task Analyze and Evaluate Change Request is assigned to a random resource with the role Project Manager. In most projects, there is a unique resource with this functional role in the project. In the example, the project manager Heer decides that the changes shall be performed. The planning of the changes is left to the responsible resource of the parent task of the related task, which is in this example also resource Heer. During the execution of the task Plan Changes, resource Heer creates a new version of the terminated task PFDs and introduces the feedback flow from Initial P&IDs to PFDs. The implementation of the changes is the responsibility of the resource Dreher who is assigned to the new version of the task PFDs. The problem report of resource Baumann can be transferred to resource Dreher via a data flow defined along the feedback flow. After new revisions of all erroneous process flow diagrams have been released, resource Dreher commits the task Implement Change. The changes are then verified by the resource Baumann who has been assigned to the task Verify Change according to the assignment rule *responsible of related task*. After all changes are verified, the change request is finally closed which involves the deactivation of the feedback flow. If the changes were not verified, the change management workflow would enter another iteration of the while loop. New revisions of the process flow diagrams would be created until they would finally be verified.

Conclusion This section showed how management tasks are handled in PROCEED, and how they are related to the development process. The clear distinction between technical tasks and management tasks is required because the latter are not part of the project plan, which is why management tasks are excluded from scheduling as described in Section 7.1. Some aspects of the approach have not been described in this section. For example, so-called recursive management tasks have been introduced, which allow for the recursive invocation of management workflows at runtime. Further specific assignment rules have been defined which were not presented in this thesis. All details of the approach are described in [Vas10]. With respect to the definition of assignment rules, the approach could be extended. A domain-specific language [KKP+09] could be introduced to enable the flexible definition of assignment rules as part of the parameterization. This has not been investigated in more detail in the context of this thesis.

9.2 Possible Disruptions and Compensating Actions

No development project is executed exactly as planned. At project runtime, many disruptions may occur which cause deviations from the original plan [PR05, NW94]. The project goals may change, which includes changes to deadlines, milestones and the budget. Additional customer requirements or changes to the existing requirements and specifications for the product to be developed may require to perform additional work to extend or rework the results. Disruptions in the narrow sense include the unavailability of resources, technical problems, and delivery problems, which generally lead to task delays. Finally, the actual execution of tasks may deviate from the plan because of bad estimates for the required time and workload or because of poor performance. Task delays and budget overruns can be detected by means of progress measurement as described in the previous chapter. The forecasted end time of a task, which is computed by means of the schedule performance index, may reveal delays already during the execution of the task. The cost performance index is used to forecast the total budget at completion of the task.

Disruptions which occur at project runtime have to be addressed by the project management to ensure that the project will be completed within time and budget limits. Corrective control measures can be taken to increase the performance of the resources, e.g. by increasing the motivation or qualification of the project team members or by eliminating existing conflicts [Bur00]. If corrective measures are insufficient, the project plan has to be changed with respect to resource assignments. Additional resources can be assigned to delayed tasks or the availability of assigned resources can be increased which usually involves overtime work. Corrective measures and resource related plan changes aim at bringing delayed tasks back on schedule. If this cannot be achieved, the project schedule has to be adapted to reflect the actual performance and the expected end times of the tasks. In some projects it may be possible to reduce the scope of the project to avoid plan changes. Certain requirements for the product are deleted, so that the remaining requirements can be met within the given time and budget limits. However, this is usually the last resort and often infeasible.

In the following, possible disruptions at project runtime are reviewed. These disruptions may require certain plan changes which are performed manually by authorized users. Manual plan changes may in turn require automatic rescheduling of parts of the dynamic task net. Corrective measures and changes to the scope of the project are not considered here because they do not directly influence the project plan.

9.2.1 Disruptions at Project Runtime

Possible disruptions and entailed plan changes are the following.

Requirements are added or changed. Complex structural changes to the dynamic task net may be required. New task (versions), control and feedback flows may have to be added. It may even be necessary to abort or skip certain tasks which are not relevant anymore.

Errors are detected in produced artifacts If an erroneous artifact has been produced in a task which is already terminated, a new task version of this task and all terminated successors has to be created. A feedback flow is introduced which connects the task in which the error has been detected with the new version. The new task version has to be scheduled like a new task.

Key artifacts are produced enabling the refinement of the plan When a revision of a key artifact has been released, new tasks are introduced into the task net and existing tasks may have to be aborted. In plant design projects, the different flow sheets are key artifacts. For every device contained in such a diagram, a task should be defined for the specification of the device. Similarly, in software development projects, the software architecture determines the implementation tasks.

Deadlines for tasks, milestones, or the project are tightened. These changes may require to add additional resources to certain tasks to shorten their duration. Certain tasks may have to be aborted or skipped to meet the new tighter deadlines. In any case, rescheduling will be required.

The project budget is decreased. This may implicate that task assignments have to be deleted and resources have to be removed from the project team to save labor costs. The deletion of task assignments may lead to increased task durations.

A resource becomes temporarily or permanently unavailable. The resource has to be replaced by one or several substitutes to avoid a delay of his assigned tasks. The substitutes may possibly be reassigned from other tasks. If the resource cannot be substituted for certain tasks, these tasks may become delayed.

A task is (expected to be) delayed. There are many possible reasons for a task delay: Bad estimates for the planning data, poor performance at runtime, changed or additional requirements for the completion of the task, unavailable or reassigned resources, technical problems, or delivery problems. The assignment of additional resources may speed up the task. If the delay cannot be avoided, the plan should be aligned to the actual performance. This may require to relax the semantics of control flows to successors to preserve the consistency of the later planned end time with the planned dates of the successors.

9.2.2 Change Operations

In most cases, project management will try to avoid plan changes. However, sometimes plan changes are necessary, so that the plan reflects the actual project status and can be used to forecast the end times of running and preparing tasks. When task delays stem from bad estimates which were made during the project planning phase, adapting the plan to the actual performance is the only reasonable solution. The bad estimates for the planning data of tasks may be defined in task types. These values should be updated after the completion of the project. In this way, the process knowledge about the timing of tasks is improved, which leads to better planning results in future projects.

The manual plan changes which are performed to compensate disruptions at project runtime can be classified into three categories depending on whether they are related to the task net structure, the tasks, or the resources. Manual plan changes may require rescheduling of the task net depending on whether timing consistency constraints are violated or planned workload has to be redistributed.

- Task net structure related

Create a new task (version). The new task (version) requires workload and resources, and it has a certain duration. There has to be enough unassigned total workload at the parent task. The duration of the new task has to be consistent with the duration of the parent task. The planned workload has to be distributed over several work days which requires scheduling of the task. Resources can be assigned manually beforehand or automatically during scheduling. It may be required to reschedule the parent task and all of its subtasks as well to obtain a consistent schedule.

Abort or skip a task. The (remaining) planned workload of the task is deleted and its duration is reduced as described in Section 5.3.1. This does not lead to violations of timing consistency constraints. However, the user may want to reschedule parts of the dynamic task net to schedule successors earlier and to reduce the duration of the parent task if possible.

Change the semantics of a control flow. If a new control flow is introduced or the semantics of an existing control flow is tightened, e.g. changed from simultaneous to sequential, this may lead to inconsistent planned dates of the connected tasks, so that rescheduling is required. Deleting a control flow or relaxing the semantics of a control flow cannot lead to inconsistencies. However, rescheduling may be desired to benefit from the gained degree of freedom, i.e. the successor of the control flow may possibly be scheduled earlier.

- Task related

Change the total workload of a task. This does not directly affect the duration of the task as long as the additional workload is not used for task assignments. Therefore, rescheduling is not required at this point. However, the changed unassigned total workload of the task has to be redistributed over the duration of the task.

Change the total budget of a task. Changes to the budget of a task do not affect the schedule.

Change the total duration of a task. The new duration and planned end time may be inconsistent with the planned dates of predecessors, successors, the parent task, scheduled subtasks, and scheduled task assignments. In these cases, rescheduling is required. In any case, the unassigned total workload of the task has to be redistributed for the new duration.

Align planned values of a task to actual values. The planned start time is set to the actual start time, and the planned workload of all task assignments is re-distributed during rescheduling according to the actual workload distributions as described in Section 7.3.2. This may result in a changed duration and a changed planned end time with the consequences discussed before.

Change planned dates of a task. The planned dates are usually computed by the scheduling algorithm but can be manually adapted. If the task has task assignments and has been scheduled before, then rescheduling is required to redistribute their planned workload. If no task assignments are defined and the new planned dates are consistent with respect to all timing consistency constraints, then rescheduling is not required. However, a later scheduling pass may override the manually set planned dates. The user can alternatively change the constraint dates of the task and initiate rescheduling.

Add or change manually set constraint date of a task. The new date must not be inconsistent with other constraint dates, i.e. the timing consistency constraints (5.34) to (5.40) can never be violated. The constraint date can however be inconsistent with the planned dates of the task. In this case, rescheduling of the parent task is required.

- Resource related

Assign an additional resource to a task. The total workload and the total budget of the task may have to be increased. On the other hand, workload can also be transferred from another task assignment. In any case, rescheduling of the task is required to distribute the planned workload of the new task assignment.

Replace a resource for a task assignment. Rescheduling of the task is required to redistribute the planned workload of the task assignment because the availability of the new resource may be different.

Change the planned workload of a task assignment. Rescheduling of the corresponding task is required to (re)distribute the new planned workload of the task assignment which may be less or greater than the previously planned workload.

Change the total workload of a resource. If the total workload—and thereby implicitly the available workload—in the work calendar of the resource is decreased for a particular date, then rescheduling is required because task assignments of this resource may become longer. An increase of the total workload for a particular date does not affect the dynamic task net directly. However, the intention of this change is usually to shorten the duration of tasks which are assigned to the resource, which is only achieved by rescheduling these tasks.

Task related plan changes may have the effect that formerly scheduled tasks become zero-duration tasks which are excluded from scheduling, or the other way round zero-duration tasks become scheduled tasks. A task may become or cease to be a zero-duration task due to changes to its total duration, total workload, or granularity level, or due to changes of the duration of the parent task (cf. Section 7.1). In these cases, scheduling of the parent task is required to update the scheduled dates of the predecessors and successors as well as the parent task.

In this section it has been shown which disruptions may occur at project runtime, which dynamic changes to a task net can be performed in response, and whether these manual changes require automatic rescheduling of (parts of) the task net. The integration of planning and scheduling is based on a change management procedure which is presented in the following section.

9.3 General Change Management Procedure

Structural and behavioral invariants constrain the possible changes to dynamic task nets at runtime (cf. Sections 5.1 and 5.2). Furthermore, a user of the PROCEED system may only be authorized to modify certain parts of a dynamic task net, depending on his permissions and task assignments. Finally, timing consistency constraints, which have been defined in Section 5.3.2, further constrain the allowed change operations to a dynamic task net. These constraints are evaluated to check whether the planning data is consistent, whether there are no contradictory constraint dates, and whether the planned dates represent a time and resource feasible schedule.

Replanning a dynamic task net may violate timing consistency constraints, e.g. performing a structural change operation or a change to a timing property. The reason for a violation of a timing consistency constraint is always a user action, i.e. a manual modification of the management data. In order to maintain a time and resource feasible schedule, timing consistency constraints may not be violated permanently. However, it would be impractical to prohibit every change operation which would violate a timing consistency constraint. Therefore, the change management procedure implemented in PROCEED allows that certain timing consistency constraints are temporarily violated during replanning of a task net. A (partial) dynamic task net is replanning if the root task is in one of the states *InDefinition* or *Replanning*. All inconsistencies are eventually resolved by automatic rescheduling when replanning is completed. The replanning of a task net is finished when the execution state of the root task is changed to *Waiting* or *Active*, respectively. In this way, planning and scheduling is integrated in PROCEED.

Before a property change or a structural change operation, the PROCEED system checks whether timing consistency constraints would be violated by the operation. Depending on which timing consistency constraints would be violated, there are four different ways to handle the expected constraint violations.

- The action which would lead to the inconsistency is *prohibited*.
- An *alternative action* is performed, e.g. a consistent property value is set instead of the user-specified value.
- *Additional compensating changes* to the dynamic task net are made which preserve consistency.
- The inconsistency is *temporarily accepted* until the replanning of the subprocess is finished.

The change management procedure ensures that the timing data of a dynamic task net is eventually in a consistent state after several dynamic changes have been performed. It is divided into two parts. First, when a change operation is invoked

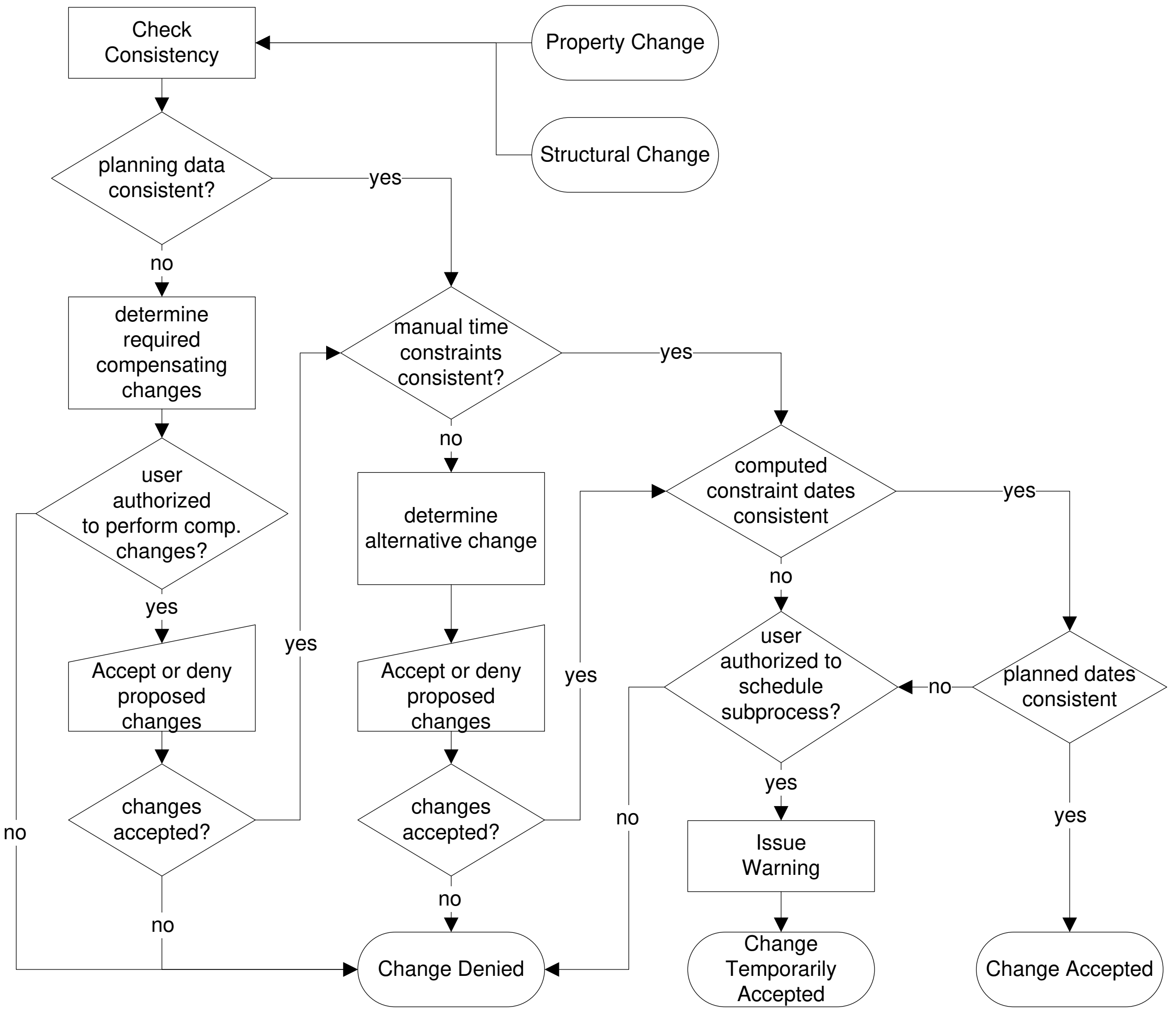


Abbildung 9.7: Procedure for consistency checking before a change operation.

by the user, it has to be decided whether the operation is prohibited, alternative or additional changes are performed, or the change is temporarily accepted. Second, when a complex task shall be defined (state change from *InDefinition* to *Waiting*) or restarted (state change from *Replanning* to *Active*), it has to be checked whether temporarily accepted changes exist for the realization of the task. The corresponding inconsistencies have to be resolved by rescheduling the realization before the task can be defined or restarted.

9.3.1 Consistency Checks Before Change Operations

The flowchart depicted in Figure 9.7 defines the procedure which is executed upon the invocation of a change operation by the user. Beforehand, PROCEED verifies that all structural and behavioral consistency constraints are fulfilled and that the user is authorized to perform the intended change operation.

Structural changes to the realization of a task as well as property changes to the interface of a task have to be handled by the procedure. Structural changes include the creation and deletion of tasks, control flows, data flows, and feedback flows, the creation of new task versions, and all changes to properties of control flows. Property changes include changes to timing properties but exclude execution state changes. Structural changes to the realization of a complex task and changes to the timing property values of one of its subtasks may cause several constraint violations at once. For example, the manual change of a planned date may at the same time be inconsistent with other planned dates, with constraint dates, and with the workload distributions of task assignments. A change to the semantics or the lag time of a control flow may cause inconsistencies between planned dates as well as inconsistencies between constraint dates. Therefore, the post-conditions defined for structural change operations in Section 5.3.2 evaluate several timing consistency constraints with respect to the modified entities. The operation $\texttt{ModifyTask}(\texttt{t})$ covers all changes to timing properties of the task $\texttt{t} \in \texttt{Tasks}$. The post-conditions of the different change operations are evaluated by the change management procedure before the invoked operation takes effect. In the definition of the change management procedure shown in Figure 9.7, the different structural change operations and property changes are not distinguished. For every operation it is checked whether one of the following inconsistencies would occur in the dynamic task net.

- Inconsistent planning data,
- Inconsistent manually set constraint dates,
- Inconsistencies between computed constraint dates and manual time constraints or task durations,
- Inconsistencies between planned dates and computed constraint dates, manual time constraints, or planning data.

Thereby, the inconsistencies may concern any task in the work context of a changed task, i.e. the parent task, the subtasks, as well as predecessors and successors. Changes to control or feedback flows may even affect tasks of different realizations. In the following, the reactions to the expected inconsistencies defined by the change management procedure are described for every case.

Inconsistent planning data If timing consistency constraints related to the planning data of tasks would be violated by the change operation, the PROCEED system determines compensating changes to the task net, which have to be performed in addition to the intended change to ensure consistency. The relevant constraints are the constraints (5.31) to (5.33). Violations to these constraints have to be resolved immediately by performing alternative changes or additional compensating changes.

If increased values for the total workload, budget or duration of a task would be inconsistent with the respective values of the parent task, then PROCEED proposes to adapt the conflicting values of the parent task as well. The latter changes compensate the intended changes. If decreased values for the total workload or budget of a

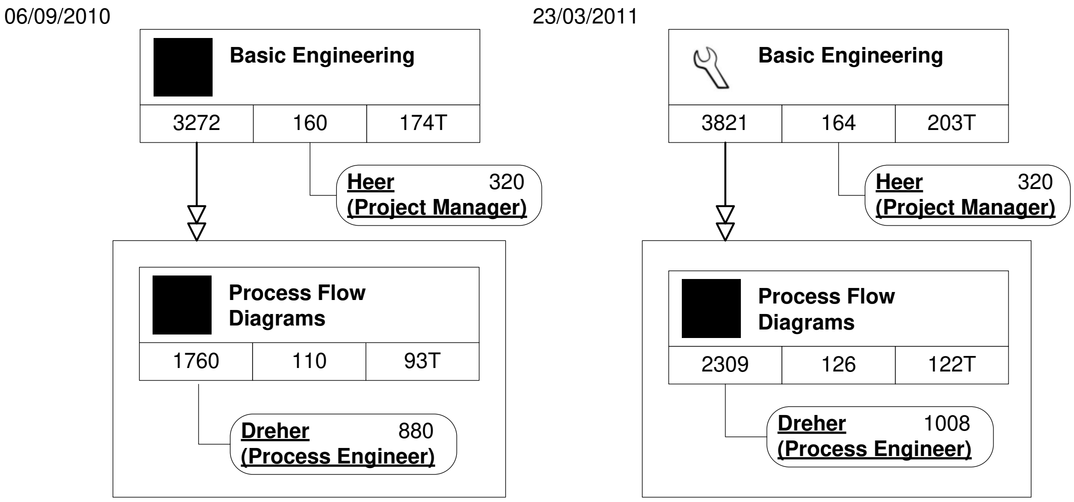


Abbildung 9.8: Changes to planning data and compensation.

task would be inconsistent with the respective used total workload or budget of the same task, then PROCEED proposes to decrease the properties only to values which are still consistent, e.g. setting the total workload to the used total workload. The same holds for the total duration which may not be shorter than the durations of the subtasks.

Changing the total duration of a task involves an automatic adaptation performed by PROCEED. If the planned start time is defined, then the planned end time is automatically set to the consistent date which fulfills constraint (5.61) with respect to the work calendar of the task. Inconsistencies which may arise due to the changed planned end time are resolved afterwards when the planned dates of the task and its context are checked.

For example, if the user wants to increase the total workload of a task, but there is not enough unassigned total workload available at the parent task, so that the used workload would exceed the total workload of the parent task in contradiction to the timing consistency constraint (5.31), then the PROCEED system offers the possibility to increase the total workload of the parent task accordingly.

However, the option to perform additional compensating changes is only available if the user is authorized to perform the changes, e.g. if he may increase the total workload of the parent task. If the user is not authorized, then his initial change operation is discarded and he has to invoke a management workflow to request the necessary changes from an authorized resource. If the user is authorized, he may still reject the proposed compensating changes. In this case, his initial change operation is discarded as well.

In the example of Figure 9.8, the total workload and budget of the task Process Flow Diagrams is increased by resource Heer. The responsible resource Dreher of the task Process Flow Diagrams is not authorized to perform this change because it involves an increase of the total workload and total budget of the parent task Basic Engineering, which is a compensating change. If resource Dreher tried to perform the change operation, the change management procedure would deny the change. Therefore, resource Dreher has invoked a management workflow to request the plan change from resource Heer.

Inconsistent manually set constraint dates When no inconsistencies with respect to the planning data exist or compensating changes could resolve the inconsistencies, then the manual time constraints are checked for inconsistencies afterwards. Inconsistencies between manually set constraint dates may lead to the violation of one of the constraints (5.34) to (5.40). The reason for a constraint violation is either a date change or a change of the semantics or lag time of a control flow.

Inconsistent manually set constraint dates are not compensated by additional changes to the task net but alternative values are proposed instead. The PROCEED system proposes an alternative, consistent value for the changed constraint date or the semantics or lag time of a control flow. For example, if the release date of a task is set to a date which is earlier than the release date of the parent task, PROCEED proposes to set the release date to the release date of the parent task instead. If the lag time of a control flow shall be increased to a value which is inconsistent with the due dates defined for the predecessor and successor, then an alternative consistent value is proposed.

Since the user is authorized to perform the initially intended change operation, he is also authorized to set the alternative value. Therefore, an additional verification as in the case of compensating changes is not required. However, like for compensating changes, the user may still decide whether he accepts the proposed alternative changes or not. Depending on his decision, the alternative change is applied or the intended change operation is denied.

Inconsistencies with respect to computed constraint dates Inconsistencies regarding computed constraint dates refer to the violation of the constraints (5.41) to (5.60). This includes inconsistencies of computed constraint dates with the total duration of tasks and with manually set time constraints (constraint dates and control flow properties). Computed constraint dates cannot be changed manually by the user but are computed automatically by critical path analysis. The consistency can be reestablished by performing critical path analysis. Therefore, the inconsistencies are temporarily accepted until the replanning of the subprocess is finished.

In the example of Figure 9.9, resource Heer increased the total duration of the task Process Flow Diagrams which lead to a violation of the timing consistency constraints (5.41) and (5.44) because the earliest and latest possible start and end times were computed based on a shorter duration before. The compensating adaptation of the total workload and budget of the parent task Basic Engineering also led to a violation of the same timing consistency constraints. The changes can be temporarily accepted because the parent task Basic Engineering is in the execution state *Replanning* and the resource Heer who performed the changes is authorized to reschedule the subprocess defined by this task.

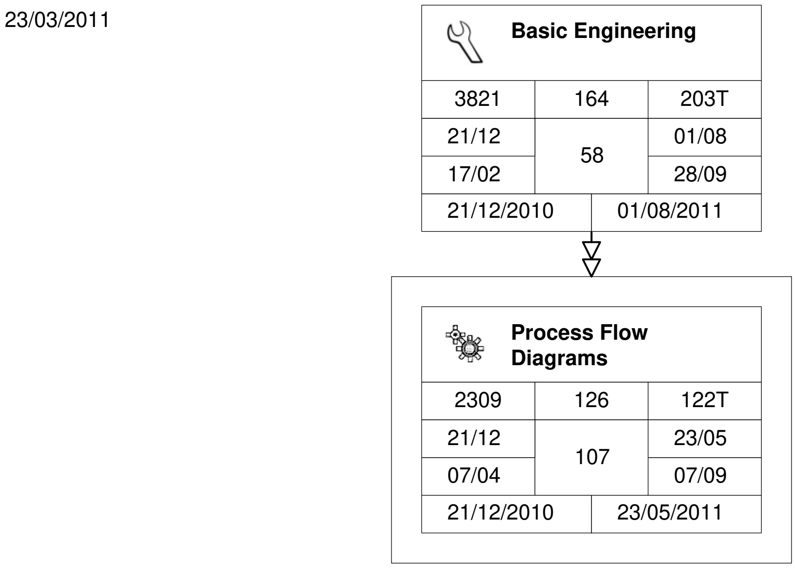


Abbildung 9.9: Temporally accepted inconsistent changes.

Inconsistencies with respect to planned dates Inconsistencies are also accepted temporarily in the case that planned dates are inconsistent with planning data, manually set time constraints, or computed constraint dates. These cases refer to the timing consistency constraints (5.61) to (5.79). In particular, constraint (5.70) checks whether the planned workload of a task assignment has been changed or not yet distributed by scheduling. In this case, scheduling is required and the change which led to the violation of the constraint is only temporarily accepted. Similarly, rescheduling is required when the planned dates of a task have been changed in a way that they are inconsistent with the distributed workload of the task assignments (constraint (5.71)). If a new subtask has been created during replanning, then its planning data, constraint dates, and planned dates are set by default but can also be adapted by the user. If the defined values are consistent with the timing data of the parent task, and if no task assignments have been defined, then no scheduling is required yet. However, if task assignments are defined for a new task with according planned workload, then constraint (5.70) is necessarily violated, because the planned workload of the task assignments has not been distributed yet. Therefore, the change is only temporarily accepted and the realization of the parent task has to be rescheduled before the parent task can be restarted. Likewise, all changes to the planned workload of task assignments are only temporarily accepted and require rescheduling. In contrast, the unassigned workload of a task is automatically redistributed without scheduling whenever its value or the planned dates of the task are changed (cf. Section 5.3.1.

In the case that inconsistencies are temporarily accepted, the complex task has to be determined, which contains all tasks which have conflicting computed constraint dates or planned dates. For example, if two tasks are contained in different realizations but are connected by a control flow whose semantics and lag time are inconsistent with the planned dates of the tasks, then the common ancestor of both tasks has to be found in the task net hierarchy. This task will have to be rescheduled to resolve the inconsistency.

When the complex task has been identified which has to be rescheduled, it has to be checked whether this task is plannable, i.e. in one of the execution states *InDefinition* or *Replanning*, and whether the user who performed the change operation is authorized to schedule the task. If one of these conditions is not fulfilled, then the initial change operation invoked by the user is discarded. because it would lead to inconsistencies which could only be resolved by rescheduling but this is not possible. If the conditions are fulfilled, then a flag is set at the identified root task of the subprocess to indicate that it has to be rescheduled before it can be defined or restarted. The user is warned about the circumstance that rescheduling will be required.

In the example of Figure 9.9, The responsible resource Dreher of the task Process Flow Diagrams has estimated that the forecasted duration is actually required to complete the task. Since he is neither authorized to perform compensating changes to the planning data of the task Basic Engineering nor to reschedule this task, he has requested additional time, workload and budget for his task Process Flow Diagrams from the resource Heer who is responsible for the task Basic Engineering and at the same time the project manager. Resource Heer has changed the execution state of the task Basic Engineering to *Replanning* and has executed the command *Adapt plan to actual performance* for the task Process Flow Diagrams (cf. Section 7.3.2), so that the planned duration is set to the forecasted duration, and the planned end time is automatically changed to a consistent date. Furthermore, resource Heer has increased the total workload of the task Process Flow Diagrams and has distributed the added workload to the task assignments. Because the planned workload of task assignments has changed and is therefore inconsistent with the previously computed workload distributions, and because the new total duration and planned end time are inconsistent with the planned dates of succeeding tasks, the changes are only temporarily accepted. The corresponding flag is set for the task Basic Engineering.

9.3.2 Resolving Inconsistencies After Replanning

The second part of the change management procedure concerns the moment, when a complex task shall be defined or restarted, i.e. when its execution state shall be changed from *InDefinition* to *Waiting* or from *Replanning* to *Active*, respectively. In these situations rescheduling has to be performed if temporally accepted changes exist in the realization of the complex task, i.e. when the corresponding flag is set. Figure 9.10 shows the flow chart of the second part of the change management procedure.

After the initial planning of a task net, rescheduling is always required to initially distribute the planned workload of the defined tasks and to compute their planned dates. Therefore, a complex task and its realization are always scheduled when its execution state is changed from *InDefinition* to *Waiting*. For a complex task which

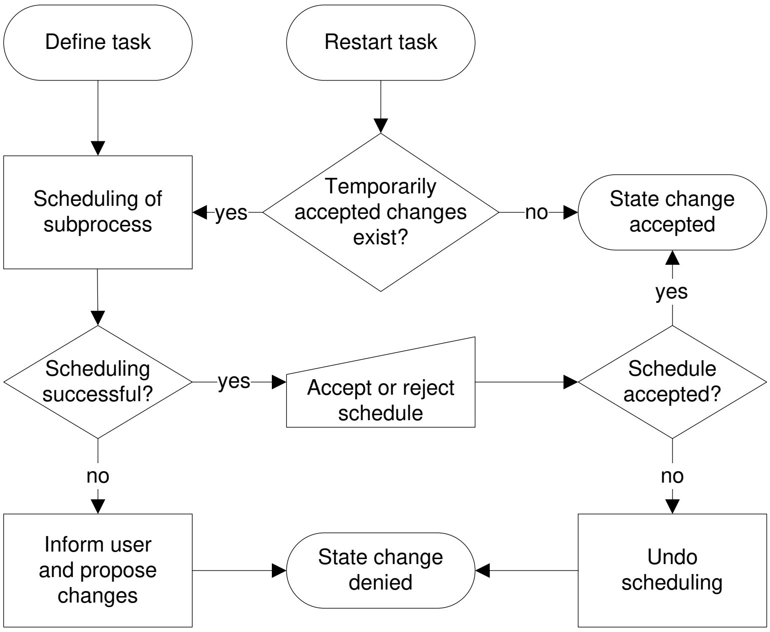


Abbildung 9.10: Procedure for (re)starting a task.

shall be restarted, i.e. whose execution state shall be changed from *Replanning* to *Active*, scheduling is only required if temporarily accepted changes exist which caused inconsistencies of the management data. Temporarily accepted changes lead to inconsistencies which are related to computed constraint dates and planned dates of tasks. These inconsistencies can be resolved by a local scheduling run of the complex task and its realization.

Local scheduling may fail due to task and resource dependencies to other tasks in the dynamic task net which are not contained in the rescheduled subprocess. In this case, it is required to start a local scheduling pass on a higher level of the task net hierarchy or even a full scheduling pass for the whole project. For this purpose, the resource who wants to (re-)start his task may have to request the rescheduling of the parent task from the respective responsible resource. According to the change management procedure, PROCEED informs the user about the failure of the scheduling pass and prohibits the change of the execution state for the moment.

Scheduling may also fail for other reasons. Given a consistent set of user-defined time constraints, planning data for tasks, and resource availabilities it may not be possible to schedule the tasks in a time and resource feasible way. This can be the case for local scheduling but also for a full rescheduling pass of the whole project. In these cases, the user is informed about necessary changes to the dynamic task net which are required for successful scheduling. Until these modifications are made by the user, the change of the execution state of the complex task to *Waiting* or *Active* is prohibited.

When scheduling is successful, the user is asked if he accepts the computed schedule. If he does, the state change is accepted and the new scheduled dates are

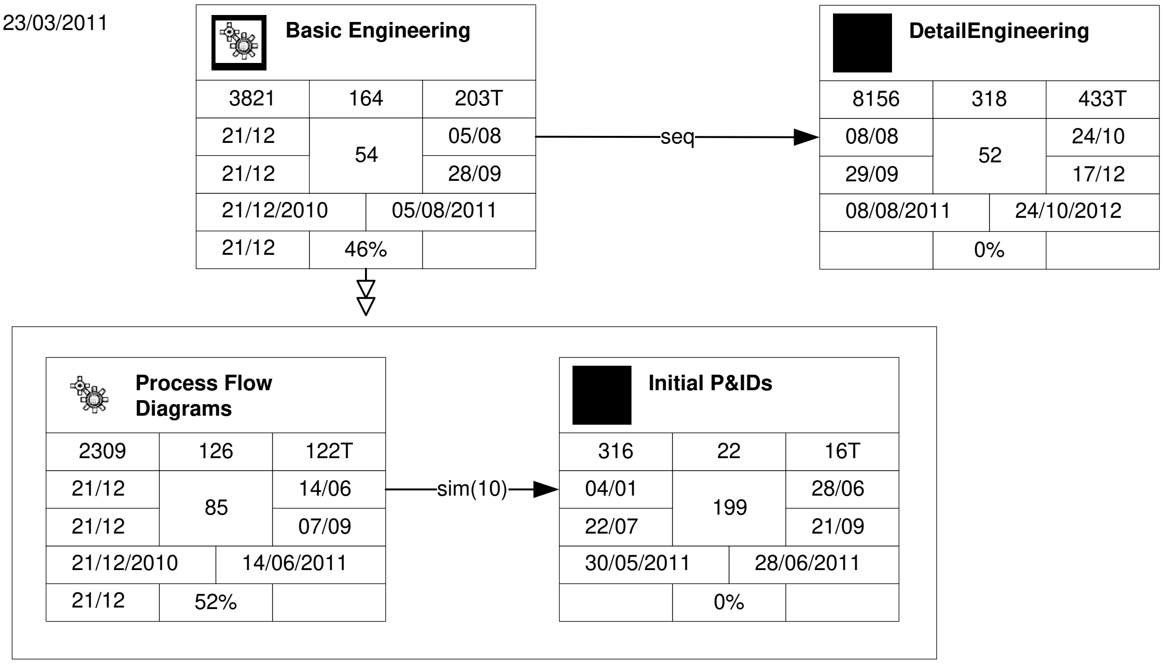


Abbildung 9.11: Replanned and rescheduled task net.

saved for the tasks in the subprocess. There may be reasons for the user to reject a successfully computed schedule, e.g. if he realizes that certain tasks have been scheduled too late. If he rejects the computed schedule, the scheduled dates are not written to the database and the state change is denied. The complex task for which temporarily accepted changes are pending remains plannable and the user may introduce further manual time constraints (constraint dates and control flows) or change existing ones to influence the automatic generation of the schedule. Finally, when scheduling has been successful and the user has accepted the generated schedule, then the flag which indicates that temporarily accepted changes exist is reset for all scheduled tasks, so that their execution state can be changed to either *Waiting* or *Active*.

In the example of Figure 9.11, resource Heer has invoked the operation to change the execution state of the task Basic Engineering back to *Active*. According to the change management procedure, partial rescheduling of the task Basic Engineering has been performed. Local rescheduling of Basic Engineering has failed at first, because the planned end time of Basic Engineering would have to be increased to the date 05/08/2011 which is later than the planned start time of the succeeding task Detail Engineering (02/08/2011). Therefore, resource Heer has changed the execution state of the root node of the dynamic task net to *Replanning* and has initiated a scheduling run for the complete task net. He is authorized to do so because he is the project manager and responsible of the project root task. The task Detail Engineering is now rescheduled and moved to a later planned start time because it is in the execution state *InDefinition*. Rescheduling of the dynamic task net is successful and all inconsistencies are resolved. The flag that temporarily accepted changes exist for the task Basic Engineering is reset to false. Finally, the state of Figure 9.11 is reached in which the planned dates and computed constraint dates are consistent with the time constraints and planning data.

9.3.3 Rescheduling of Workflow-Managed Task Nets

As described in Section 7.4, the enactment of a workflow-managed dynamic task net may involve structural changes and local rescheduling of the task net. When the decision for one of several alternative branches is made, the workflow-managed task net is rescheduled. When a loop is iterated once more, new subtasks are created and scheduled. The structural changes and the local rescheduling are performed automatically by PROCEED according to the execution of the workflow instance. As long as the changes to the task net and the timing properties remain local, i.e. they do not influence other tasks in the context of the workflow, the enactment of the workflow-managed task net can proceed normally. If the automatic rescheduling requires to adapt planned dates of the workflow-managed task in a way that is inconsistent with timing properties of the task itself or its context, then scheduling fails. As a consequence, workflow enactment fails when the automatic local rescheduling fails, and the workflow-managed task is aborted. To compensate the abortion, a new version of the workflow-managed task has to be started manually. Since the replanning and rescheduling of workflow-managed tasks is performed automatically, it is not covered by the change management procedure.

9.3.4 Violations of Monitoring Constraints

In contrast to timing consistency constraints, violations of monitoring constraints merely indicate that the actual execution of the development process deviates from the plan. Violations of monitoring constraints may result from manual changes to the dynamic task net, but they may also result from the passing of time as discussed in Section 5.3.2. Monitoring constraints may be permanently violated. The PROCEED system informs the user about the constraint violations, but the user is not obliged to resolve the inconsistencies. Therefore, violations of monitoring constraints are not covered by the change management procedure.

9.3.5 Changes to Dependent Task Properties

Common project management systems handle inconsistencies with respect to the time management properties of a task. When the user performs a change operation to one of the properties of a task, the system reacts with an adaptation of other properties to re-establish consistency or to automatically make changes intended by the user. For example, when the user assigns an additional resource to a task in the well known project management system MS Project [Mic10a], then the software offers different possibilities: Either the duration of the task is shortened, its total workload is increased, or the resource usage per day is decreased for all assigned resources.

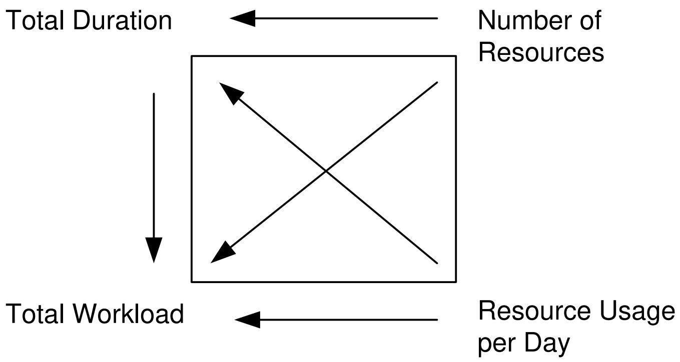


Abbildung 9.12: Dependent time management properties.

The total workload, the total duration, the number of assigned resources, and their planned workload per day are depended properties of a task in a timed dynamic task net. These properties are depicted in Figure 9.12. A change to one of the properties influences the other properties and may require corresponding changes to ensure the consistency of the management data. This section elaborates on how the consistency of the dependent time management properties of an individual task is ensured by means of the change management procedure implemented in PROCEED. It is shown that the procedure covers all cases of inconsistencies between the properties.

In PROCEED, the total workload of a task is explicitly defined, independently of the used total workload of the task. The total workload of a task does not necessarily equal the sum of the planned workload of the task assignments and subtasks. Therefore, the dependencies between workload, duration, number of resources and their usage are handled differently in PROCEED compared to other project management systems like MS Project.

The following description reviews manual changes to one of the four properties depicted in Figure 9.12. The arrows in Figure 9.12 indicate which property may have to be adapted in case of changes to the property at the source of the respective arrow. For every property change, the timing consistency constraints which may be violated are identified, and the responses in terms of the constraint handling procedure are described. In this way, it is shown that the defined constraints together with the defined change management procedure cover all possible cases of inconsistencies between the four properties of a task.

First, the effects of changes to the number of assigned resources are considered.

Creation of a new task assignment A user may create a new task assignment for a task by specifying the required role, the planned workload, and optionally the assigned resource. If workload should be transfered from an existing task assignment to the new one, this has to be done manually by reducing the workload of the other task assignment beforehand. Two consistency constraints may be violated:

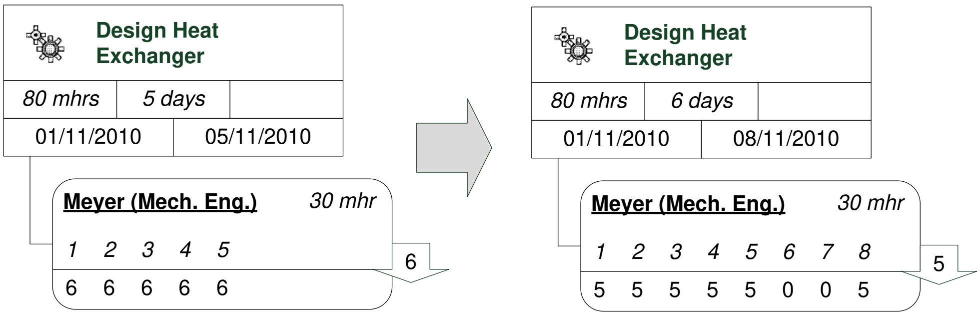


Abbildung 9.13: Decrease of maximal resource usage per day for a task assignment.

- Since the new task assignment still has to be scheduled to distribute the planned workload, constraint (5.70) is violated, and the change to the task net is only *temporarily accepted* until the parent task is rescheduled. Rescheduling assigns an eligible resource if necessary and distributes the planned workload according to the work calendar of the resource.
- If the used total workload of the task including the workload of the new task assignment exceeds the total workload, the constraint (5.31) is violated. This constraint violation is *resolved immediately*. PROCEED offers the possibility to increase the total workload of the task.

Deletion of a task assignment When a task assignment is deleted, no adaptation of other property values is required and the change can be *accepted permanently*. The used total workload of the task is implicitly reduced while the total workload remains unchanged. If the intention of the deletion was to save workload for the task, the total workload has to be reduced manually afterwards.

Second, the effects of a change to the resource usage per day are considered. In PROCEED, only the maximal resource usage per day can be manually defined for a task assignment. The planned workload of task assignments is distributed during scheduling and the individual values cannot be changed manually because they would be overridden during the next scheduling pass anyways.

Decrease maximal resource usage per day for a task assignment If the distributed planned workload exceeds the decreased maximal resource usage for a particular day, the constraint (5.72) is violated. This change is *temporarily accepted* until the parent task is rescheduled. Rescheduling may fail when the adapted total duration of the task is inconsistent with the timing properties of other tasks. In this case, the decrease of the maximal resource usage per day may have to be undone. Figure 9.13 shows an example for the situation in which the task has been rescheduled to comply to the reduced maximal resource usage of 5 hours per day for the task assignment which led to an increased total duration of the task.

Increase maximal resource usage per day for a task assignment This change

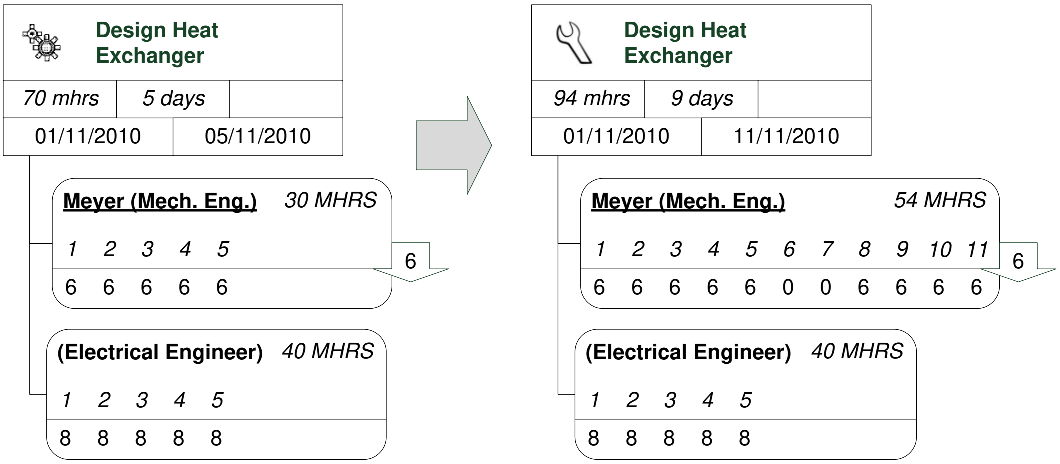


Abbildung 9.14: Increase of total duration.

can be *permanently accepted* since it does not lead to any inconsistencies. It may however shorten the duration of the task assignment during rescheduling.

Third, the effects of a change to the total duration of a task are considered. The total duration of a task may be inconsistent with the planned dates and the distributed planned workload of task assignments.

Increase of total duration The unassigned total workload of the task is automatically redistributed. The planned end time is automatically adapted to be consistent with the new total duration. If the new planned end time of the task is consistent with the planned dates of the other tasks in the task net, then this change is permanently accepted. However, PROCEED warns the user that the new total duration of the task is longer than the scheduled task assignments. The user can for example increase the workload for the responsible resource, so that the task assignment will span over the full duration of the task after rescheduling. This situation is depicted in Figure 9.14 where 24 man hours have been added to the task assignment of the responsible resource and distributed by rescheduling over the additional days.

Decrease of total duration If the duration of a previously scheduled task assignment, which is implicitly defined by the distributed workload, exceeds the new total duration of the task, then the constraint (5.71) is violated. The change is *temporarily accepted* until the parent task is rescheduled. If the task assignment is not changed as well, then rescheduling will increase the total duration again, as required for the task assignment. To avoid this, the user has to reduce the planned workload or increase the maximal daily workload of the conflicting task assignment.

Finally, the effects of a change to the total workload of a task are considered.

Increase of total workload In this case, no adaptation of other property values of the same task is required. The change can be *accepted permanently* if it is consistent with the total workload of the parent task. An inconsistency between the total workload and the total duration of the modified task cannot be identified at this point since the total workload may be distributed to multiple task assignments later on.

Decrease of total workload If the used total workload of the task exceeds the new total workload, the constraint (5.31) is violated. This constraint violation has to be *resolved immediately*. PROCEED proposes to decrease the total workload only to the used total workload. Otherwise, the change is prohibited.

9.4 Related Work

The concepts which have been presented in this chapter are related to two different research directions. First, there is the enactment of project management processes, in particular change management processes. Second, there is the integration of planning and scheduling at project runtime, in particular the identification and classification of possible changes to the project plan.

9.4.1 Enactment of Project Management Processes

Nagl and Westfechtel The distinction between technical processes and management processes was already made in [NW94]. A layered approach for the realization of a process management system is presented in which different levels of administration information are distinguished. On the lowest level, layer 0, there is only technical information, i.e. the details of the technical documents in a development project. The fine-grained processes on the technical level are not explicitly represented in the system. In the so-called extended technical configuration, technical artifacts and their dependencies are represented on a more abstract level. The development process and its subprocesses, the corresponding resources, and the extended technical configuration together form the so-called administration configuration which is administrated on layer 1. The processes which are enacted on layer 1 are comparable to the management processes in PROCEED. They define how the administration configuration including the development process is built up and modified. The highest level, layer 2, is called the management administration. On this level, the parameterization of the process management system for the usage in different domains, organizations and projects takes place. Process model definitions for technical processes as well as management processes can be defined and customized. This resembles the parameterization of management workflows in PROCEED. The three levels are compared with each other [NW94, p.45]. Different users of a process management system are associated with the different layers. Level 0 is the activity level of technical developers, level 1 that of the administrator of a project, level 2 that of the manager of the parameterization process. From the lowest level to the highest level, the number of different users of the system and the number of different user roles decreases. With respect to the enacted processes, the degree of dynamics decreases from the lowest level to the highest level. For the modification of the technical data on layer 0, no processes are explicitly prescribed. For the management of the development process, dynamic task nets are defined on layer 1, and complex dynamic changes can be applied. Finally, the parameterization process defined on layer 2 is usually not changed at project runtime. In this thesis, management processes have been compared to development processes, and it has been argued that the former are less dynamic than the latter. This corresponds to different degrees of dynamics within layer 1 of the approach presented in [NW94]. The enacted processes on layer 1 are less dynamic than the managed processes.

Joeris Change management processes in the software engineering domain are addressed in [Joe97], where the author Gregor Joeris presents a conceptual framework for the integration of process and configuration management. Management of changes means managing the process of change as well as managing all artifacts of an evolving software system. Joeris argues that process modeling languages provide only poor concepts for managing the process of change. The presented approach tries to integrate the underlying representation formalisms of process and version models. For this purpose, basic concepts of dynamic task nets are used: Input and output parameters of tasks, data flow relationships, and actual data flow, i.e. modeling released document revisions in the process model instance. Feedbacks in a process are related to changes to the product. Change processes shall be reflected in a software development process by adapting the process model instance. The adaptation is performed in several steps: modeling and initiating of feedbacks, impact analysis and consequences, managing the flow of change. Joeris defines the requirement that management processes like change request authorization or approval of changes have to be integrated with the technical processes. This leads to the concept of reflexivity which is required when "the process of process (model) changes is defined in the process space". Altogether, Joeris describes on a conceptual level how change management and development process management can be integrated, but he does not provide a concrete solution for the problem. The approach for the enactment of management processes presented in this thesis can be regarded as a concrete realization of this integration.

Ivins et al. In [IGM04] an approach is presented which uses change processes to manage changes to products. The key activities that are required for changing a product are divided into human and technical activities. Human activities are finding developers affected by a change, obtaining permission for a change, notifying relevant users of a change, and following a predefined approval procedure for releasing performed changes. Technical activities are the actual changes to the artifacts and product configurations. The change processes provide explicit support for the coordination of human and technical activities. They are modeled using UML activity diagrams. The authors emphasize that a behavioral model was required to clearly show the sequence of activities in a process, i.e. the model of a change process has to be capable of representing sequential and concurrent activities, alternative paths and iteration. Several requirements for a system to enact the process models are defined including the modeling of process participants and required roles, the traceability of the process, and the adaptability of the process which refers to dynamic changes of a process model instance at runtime. A research prototype has been developed which integrates workflow technology with a development system that supports versioning. In contrast to the approach presented in [IGM04], human and technical activities are not represented in the same process model in PROCEED. The human activities are the subtasks of management workflows, while the technical activities are contained in the part of the dynamic task net which represents the project plan. Both approaches have in common, that change management processes are explicitly modeled and enacted by a workflow system. This provides standardized, auditable change processes which support the coordination of several process participants in a change management case.

9.4.2 Replanning and Rescheduling

According to Smith [Smi03], two of the open research questions with respect to scheduling are the management of change and the integration of planning and scheduling. When speaking of the integration of planning and scheduling, Smith mainly thinks of planning systems from the domain of artificial intelligence. These systems autonomically generate a plan to execute and schedule the individual tasks. However, the integration of manual planning and scheduling in the context of a project is also an open field of research. Current scheduling techniques are best suited for highly predictable scheduling environments while most practical applications tend to comprise highly uncertain and dynamic scheduling environments. Solutions for reactive scheduling have already been reviewed in Section 7.6.1.

In the following, two works are reviewed in which the possible disruptions at project runtime have been systematically determined in order to develop reactive scheduling algorithms. Finally, the common functionality of project management systems for handling manual plan changes is reviewed by means of an exemplary tool.

Zhu et al. In [ZBY05], a classification of possible disruptions at project runtime is provided which defines four different classes. Under the term disruption, reasons for plan changes and actual plan changes are subsumed, e.g. resource shortage as well as the increase of a task’s duration. First, there are disruptions which are related to the project network like the creation or deletion of tasks or changes to precedence relationships. Second, task related disruptions like delays, increased task durations, and deviating resource usages may occur. The third class covers resource related disruptions like unexpected resource shortage or unavailability. Finally, milestone disruptions do not affect the feasibility of the schedule but it may be desirable to revise the schedule to meet a new or changed milestone. Several recovery options are available to repair a disrupted schedule, which have been reviewed in Section 7.6.1. Recovery options are applied by changing, adding, or removing constraints of a integer linear programming (ILP) model. A solution to the ILP problem instance is a repaired time and resource feasible schedule for the whole project.

The classification for disruptions introduced in [ZBY05] has been used to classify the possible changes to a dynamic task net in this thesis. Thereby, milestone disruptions have not been considered. In contrast to [ZBY05], reasons for plan changes and actual plan changes have been explicitly distinguished in this thesis. Zhu et al. solve an integer linear programming problem for the whole project schedule. In this way, no local schedule repair is possible. The effects of the applied recovery options may be local in many cases, but there is no way to explicitly constrain scheduling to a part of the project plan. This has been required in this thesis due to the possibly limited authorization of a user to perform changes as well as for enabling automatic rescheduling of workflow instances. The approach of Zhu et al. does not take execution states of tasks into account because project planning is treated independently of project execution. The change management procedure presented in this chapter ensures that replanning and rescheduling is performed consistently with process enactment.

Wang Wang describes in [Wan05] several possible disruptions which may occur at project runtime. Four different cases are identified. The shift of a task refers to changing the start or end time of the task. Second, the duration of a task may change. Third, a change in certain resource capacities may occur. Finally, a temporal constraint may be added or removed. Temporal constraints as defined by Wang include release dates, due dates, precedence constraints, and milestone constraints. A milestone constraint defines a fixed point in time for the start or end time of a task. Disruptions lead to changed constraints of a dynamic constraint satisfaction problem which is then solved by means of meta-heuristics (cf. Section 7.6.1). As a consequence, the same limitations apply for the approach of Wang with respect to local rescheduling as for the approach of Zhu et al. which has been discussed before. The disruptions identified by Wang do not include the addition or deletion of tasks.

Changes to dependent task properties in project management systems As an exemplary case, the alternative adaptations offered by the project management system MS Project [Mic10a] are described in detail in the following. MS Project provides all basic functionalities required for project management in small to medium-sized projects. It provides tool support for the manual scheduling of tasks. In particular, it ensures the consistency of the planned workload of a task, its duration, the number of assigned resources, and the individual resource usage per day for the task. The dependencies are visualized in Figure 9.15 by the square with the properties at the edges. The arrows in Figure 9.15 indicate, which automatic changes MS Project proposes to handle a manual change to one of the properties, i.e. a change to the property at the source of an arrow may require a change to the property which is the target of the arrow.

The total workload of a task is not explicitly defined in MS Project and can

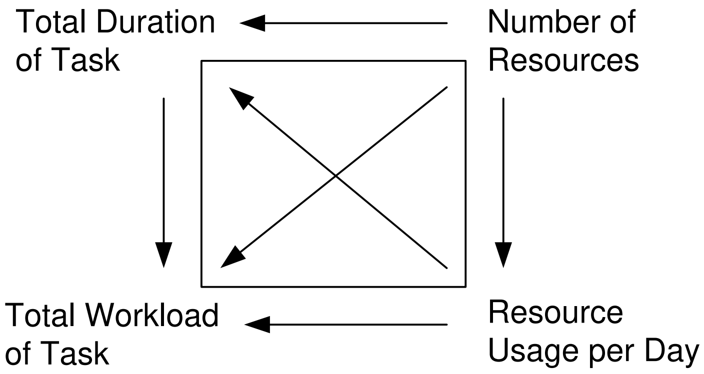


Abbildung 9.15: Dependent time management properties in MS Project

therefore not be changed manually. When the user performs a change operation to one of the properties, MS project performs a default change to re-establish consistency and offers the alternative change operations to the user to select one of those instead. This way, the consistency of the dependent property values is ensured. The semi-automatic adaptations of the management data are necessary because the workload of a task equals the sum of the workload of all task assignments and this equality has to be preserved at any time. The workload of subtasks is not taken into account. There are different possibilities for re-establishing the equality after a user action whereby the total workload remains unchanged. On the other hand, the user may want to change the workload of a task by performing an action like adding a new resource or increasing the resource usage per day. These alternative changes are also provided to the user.

Like in PROCEED, the planned workload of task assignments can be defined on a daily basis in MS Project. When the workload values for individual days are manually adapted by the user, the workload or the task is automatically adapted accordingly. The duration of a task is connected with the planned start and end time in MS Project. A change of the planned start or end time of a task moves the task on the time line. Thereby, the duration remains fixed. The scheduled workload of the resources is moved accordingly. It is not checked, whether the resources are assigned to other tasks in the new time frame as well. If so, this leads to overtime work for overlapping days. The project manager has to resolve the resulting inconsistencies manually.

In contrast to MS Project, the total workload of a task is explicitly defined in PROCEED and is not necessarily equal to the sum of the workload of all task assignments. Therefore, it is not required to make any adaptations in PROCEED, when the used total workload is reduced. PROCEED supports automatic resource-constrained scheduling which can resolve inconsistencies between planning data, time constraints and planned dates. As a consequence, necessary adaptations to the management data can be delayed while a user is replanning a task, and are only performed just before the task is restarted.

9.5 Conclusion

This chapter showed how changes to a timed dynamic task net are performed at project runtime. Changes to a project plan or to the scope of a project have to be performed according to defined processes in an organization. Section 9.1 showed how project management processes can be defined and enacted in PROCEED. The organization-specific and project-specific parameterization enable organizations to define their own management processes for reporting, change management, and quality management. In practice, a scheduled dynamic task net has to be continuously replanned in the course of a project. Section 9.2 described the possible disruptions which may occur at project runtime, which manual plan changes can be applied to react to the respective disruptions, and whether the plan changes require rescheduling. Section 9.3 showed how replanning is performed in a consistent way which respects the current enactment state of the development process. In this way, planning, scheduling, and enactment of dynamic task nets are integrated.

Kapitel 10

Prototypical Implementation

This section describes the PROCEED prototype which is an implementation of the concepts and algorithms described in the previous chapters. First, a coarse overview over the system and its main components is given. In the following section, relevant technical details with respect to the design and implementation of the prototype are described. The main part of this chapter is the presentation of the graphical user interface of PROCEED. The different views for process and project management and monitoring are described. Finally, key figures regarding the size of the implementation are provided.

10.1 System Overview

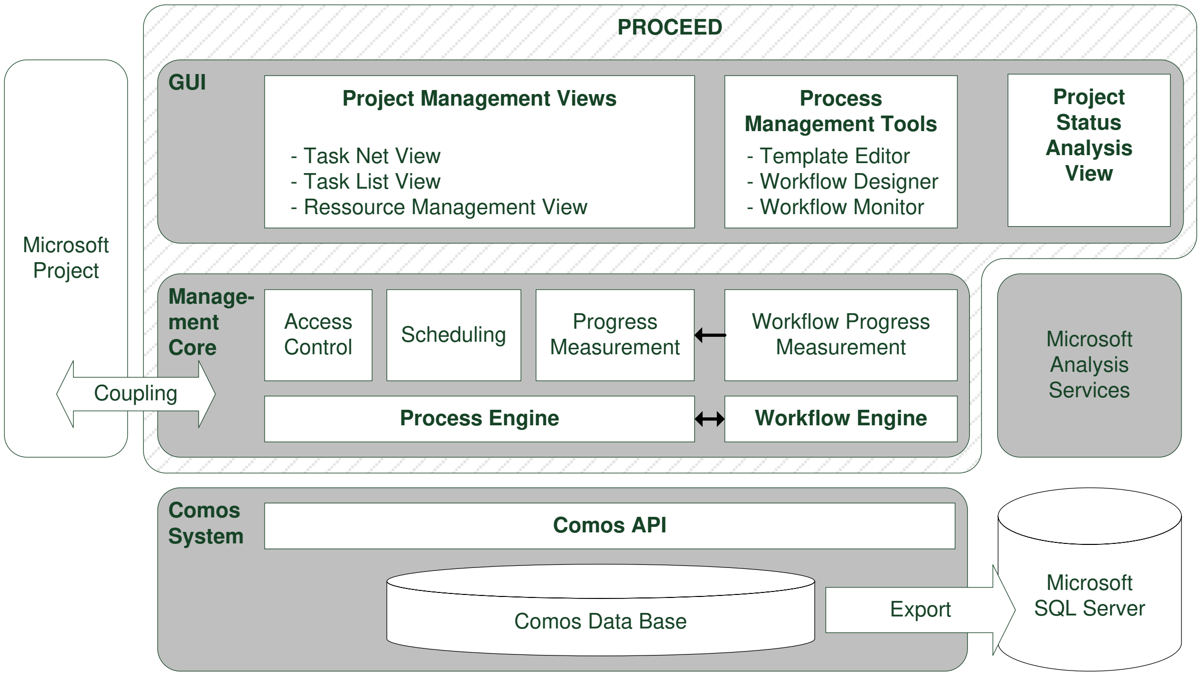
The process management environment PROCEED has been implemented as an extension to the life cycle asset information system Comos which is widely used in the plant engineering industries. PROCEED adds new functionality to Comos which covers process and project management in engineering design projects. It is a prototypical implementation of the concepts and algorithms presented in this thesis.

Since the infrastructure consisting of PROGRES, UPGRADE and GRAS, based on which the AHEAD prototype was built, could not be used for the development of PROCEED, the PROGRES graph schema defining the DYNAMITE meta-model had to be translated to classes and associations defining the data model of the PROCEED prototype. Graph transformations and graph queries were translated to according methods in the PROCEED source code.

PROCEED has been implemented using the programming language C# and several libraries of the Microsoft .NET framework. The Windows Presentation Foundation (WPF) has been used to realize the graphical user interface. The Windows Workflows Foundation (WF) has been used to realize the workflow management functionality.

The integration of the PROCEED extension module with Comos has been realized by means of the Component Object Model (COM) []. COM interfaces have been defined to allow the invocation of the different tools provided by PROCEED from Comos. PROCEED accesses the data in the Comos database and the Comos functions via the Comos API which provides several COM interfaces.

The Comos database is an object oriented database. The planning objects in the Comos database which represent the actual engineering data in a project are derived from base objects which constitute templates for planning objects (cf.



Section 2.2). Several base objects have been created as templates for the entities of the management data model of PROCEED. A Comos database has to contain these base objects, so that PROCEED can be used.

Section 10.1 shows the coarse grained architecture of the PROCEED system and its relation to Comos. A three-tier architecture has been implemented. On the presentation tier, the different views and tools which together amount to the graphical user interface (GUI) of PROCEED are located. These views will be described in detail in Section 10.3. They are divided into three different categories. Project Management Views of PROCEED are used for managing a running project including enacted process model instances. The Project Status Analysis View is used at project runtime for monitoring. The Process Management Tools are used for the creation of process model definitions, and for dynamically changing the definitions of running process model instances.

On the logic tier, the Management Core of PROCEED is located. The Process Engine is responsible for the enactment of dynamic task nets, which includes the evaluation of all structural, behavioral, and timing consistency constraints. The Workflow Engine enacts workflow instances which control the enactment of workflow managed task nets. The bidirectional arrow indicates the coupling between the two engines. The Access Control module realizes the authorization model for dynamic task nets presented in Section 5.5. The algorithms for critical path analysis and resource-constrained scheduling of dynamic task nets have been implemented in the module Scheduling. The evaluation of the progress measures defined for tasks takes place in the Progress Measurement module. The degree of completion of workflow instances is computed in the module Workflow Progress Measurement which is tightly integrated with the workflow engine. The computed values are used for the general progress measurement.

The data tier of the three-tier architecture contains the Comos database in which all management data is stored. The data is accessed via the Comos API. In regular intervals, the management data is exported to a relational database in an instance of a Microsoft SQL Server. The data of this project data warehouse is processed and condensed by means of the Microsoft Analysis Services which are conceptually located on the logic tear but are, technically speaking, not part of the PROCEED system. The Project Status Analysis View of the PROCEED system accesses the project data warehouse via MDX queries to the Microsoft Analysis Services.

The project management system Microsoft Project has been coupled with PROCEED to use its Gantt chart view for the presentation of the project schedule. When MS Project is invoked from PROCEED, the current state of the dynamic task net is exported and a project plan is generated in Ms Project. Some changes to task properties can be directly made in MS Project which is why the coupling of the two systems is bidirectional.

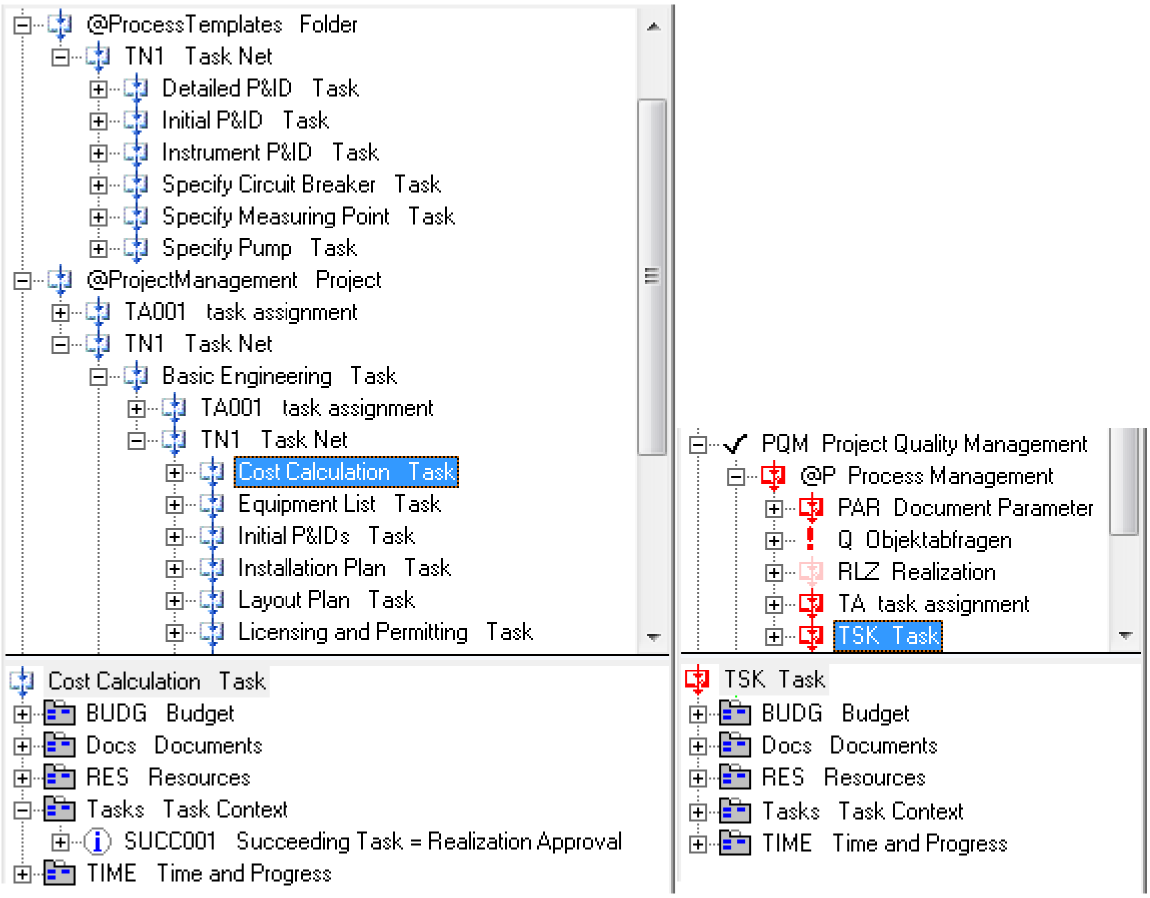
10.2 Design and Implementation

This section reviews some technical details regarding the design and implementation of the components of the PROCEED prototype. It is shown how the Process Engine accesses the objects in the Comos database. The relationships between the components which together provide the workflow management functionality in PROCEED are reviewed. With respect to task net scheduling, the data structures are described which enable a fast and side effect free computation of a project schedule. It is shown how multidimensional data sets are retrieved from the Project Data Warehouse. Finally, it is coarsely described how the coupling with the project management system MS Project has been realized.

10.2.1 Process Engine

The meta-model for the management data maintained in the Comos database which has been introduced in Chapter 5 forms the basis of the PROCEED *process engine*. Classes have been implemented for tasks, resources, documents, control flows, etc. On the one hand, these classes encapsulate the access to the data in the Comos database. In this sense, the process engine serves as a wrapper for the Comos API. On the other hand, additional functionality is implemented in the process engine classes including the constraint checks of structural, behavioral, and timing consistency constraints.

Engineering data of a plant design project is stored in the Comos database in the form of so-called planning objects. These planning objects may represent devices of the designed plant but also documents like flow diagrams. Tasks, which are managed by PROCEED, are stored as planning objects in the Comos database. Figure 10.1 shows the tree view which is used in Comos to browse the objects in the database.



- @ProcessTemplates Folder
  - TN1 Task Net
    - Detailed P&ID Task
    - Initial P&ID Task
    - Instrument P&ID Task
    - Specify Circuit Breaker Task
    - Specify Measuring Point Task
    - Specify Pump Task
- @ProjectManagement Project
  - TA001 task assignment
  - TN1 Task Net
    - Basic Engineering Task
      - TA001 task assignment
      - TN1 Task Net
        - Cost Calculation Task
        - Equipment List Task
        - Initial P&IDs Task
        - Installation Plan Task
        - Layout Plan Task
        - Licensing and Permitting Task

Cost Calculation Task
- BUDG Budget
- Docs Documents
- RES Resources
- Tasks Task Context
  - SUCC001 Succeeding Task = Realization Approval
- TIME Time and Progress

- PQM Project Quality Management
  - @P Process Management
    - PAR Document Parameter
    - Q Objektabfragen
    - RLZ Realization
    - TA task assignment
    - TSK Task

TSK Task
- BUDG Budget
- Docs Documents
- RES Resources
- Tasks Task Context
- TIME Time and Progress

Abbildung 10.1: Planning objects and base objects in Comos.

On the left side, planning objects are shown which represent process templates and tasks in a project. The realization of a task is represented by a separate object which is arranged below the task. The children of the realization object represent the subtasks. The bottom part of the tree view shows the specifications of the selected planning object. In this case, the specifications of the task Cost Calculation are displayed, and it can be seen that a specification refers to the succeeding task Realization Approval.

On the right side of Figure 10.1, the base objects are displayed which have been defined for PROCEED. Base objects serve as templates for planning objects. A base object can be customized by defining additional specifications. The base object TSK represents the most general task type. All planning objects which represent tasks are instantiated from this base object or from a specialization thereof. For the base object TSK several specifications have been defined which are therefore available for the task Cost Calculation.

The objects contained in a Comos database can be accessed via the Comos API which provides several interfaces for the different object types. In Figure 10.2 on the right, four different interfaces for Comos objects are depicted. The IComosBaseObject is the most general type. Planning objects are instances of the interface IComosDDevice. Specifications of planning objects are of the type IComosDSpeci-

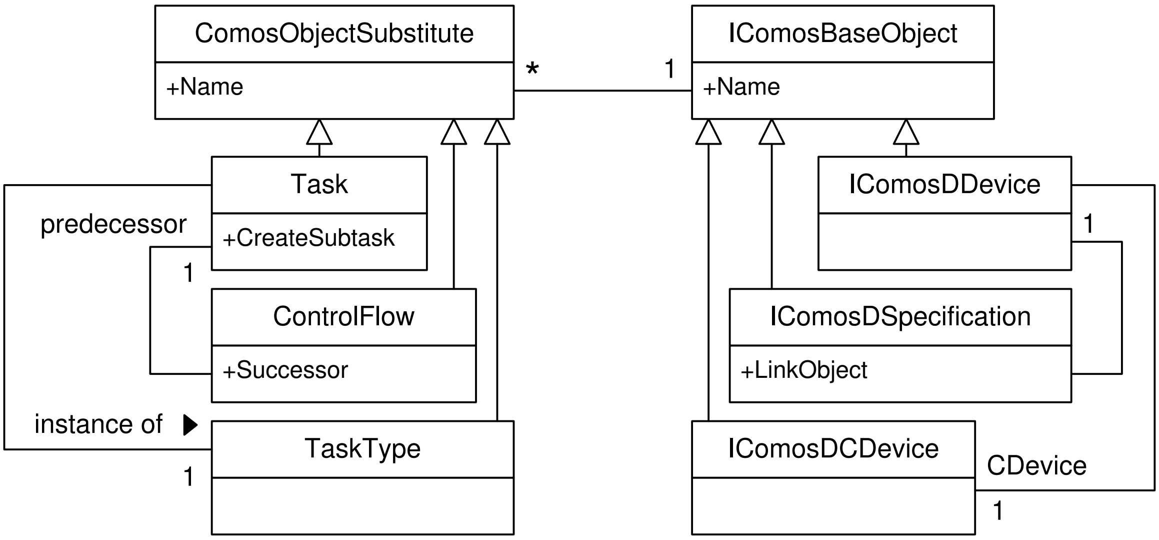


Abbildung 10.2: Relation of process engine classes to Comos interfaces.

fication. Finally, base objects, which are also called CDevices in Comos, have the interface IComosDCDevice. Every IComosDDevice object has a unique CDevice from which it is derived. Every specification belongs to exactly one planning object or base object where the latter association is not depicted in Figure 10.2.

The classes defined by the PROCEED process engine wrap the interfaces of the Comos API. They are all derived from the class ComosObjectSubstitute. Every object which is instantiated from a process engine class is associated with a unique object in the Comos database which has the same name. Tasks are represented by IComosDDevice objects and task types by IComosCDevice objects. A control flow which connects two tasks is realized as a specification of the predecessor which refers to the successor as the so-called LinkObject.

Whenever an object in the Comos database is accessed from PROCEED, an instance of the corresponding process engine class is created. As a consequence, there may be several instances of a process engine class referring to the same data object in the database at the same time. This does not pose a problem because no property values are cached in the management object of the process engine. Instead, the values are always retrieved from the Comos database.

The process engine classes do not merely wrap the Comos interfaces but also provide additional functionality. The constraint checks of structural, behavioral, and timing consistency constraints are implemented in the methods of these classes. Furthermore, the access control module is queried whenever a change operation is invoked on an object instantiated from a process engine class to determine whether the currently logged-in user is authorized to perform the operation. In Figure 10.2, the method CreateSubtask is depicted. In this method, the authorization of the user and the various constraints which apply to this change are checked before the subtask is actually created. The subtask is created as a new IComosDDevice object in the Comos database below the object which corresponds to the realization of the

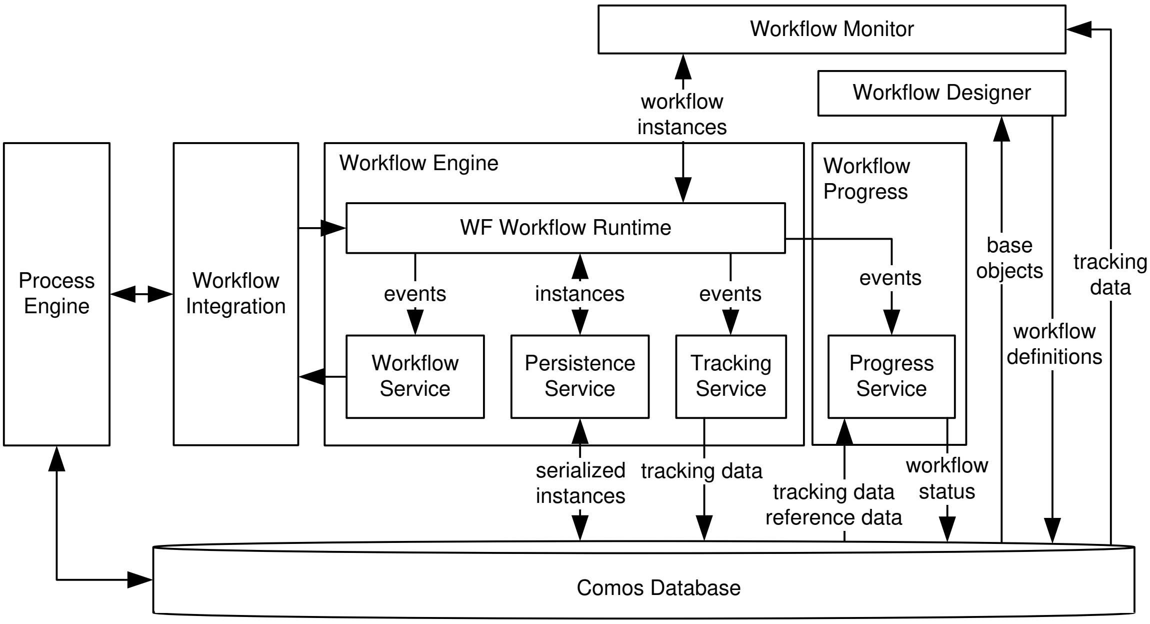


Abbildung 10.3: Workflow engine and related components and tools.

task.

10.2.2 Workflow Engine

The workflow engine of PROCEED has been implemented based on the Windows Workflow Foundation (WF) [Buk08]. As described in Section 3.4.4, the WF provides a runtime engine for the enactment of workflow instances and class libraries for activities which can be used to create workflow definitions. Services for workflow persistence and tracking are provided and it is possible to implement custom services. Finally, user controls are provided for the realization of design-time tools.

Figure 10.3 gives an overview over the components which together provide the workflow management functionality in PROCEED. The central component is the Workflow Engine. Workflow instances are enacted by the WF Workflow Runtime which is used for this purpose in all workflow-based applications which are implemented based on the Windows Workflow Foundation. Several custom services have been implemented. The Persistence Service serializes workflow instances in order to store their current state in the Comos database. The Tracking Services logs all workflow events like the start and completion of activities and stores tracking data for all running workflow instances in the Comos database. This tracking data is used together with reference data about all instances of a given workflow type to compute the degree of completion of a workflow instance of this type. The degree of completion is updated upon the completion of workflow activities. The computed workflow status is stored in the Comos database. The integration of the Workflow Engine and the Process Engine which is responsible for the enactment of dynamic task nets is realized by the Workflow Integration component which receives workflow events via the Workflow Service and invokes operations like starting a workflow instance via the Workflow Runtime. Finally, two tools have been implemented for the definition and monitoring of workflows. The Workflow Designer is used to create workflow definitions which are stored for the respective workflow templates in the Comos database. The Workflow Monitor displays the current status of a running workflow instance based on the tracking data in the Comos database. Furthermore, it can be used to apply dynamic changes to running workflow instances.

The workflow management functionality has been developed in close cooperation with Siemens Industry Software, at that time called innotec. The Workflow Engine, Workflow Progress component, Workflow Designer, and Workflow Monitor can also be used independently of PROCEED to define and enact engineering workflows in Comos. This functionality has already exceeded the prototype status and has been integrated in the current release of the Comos system.

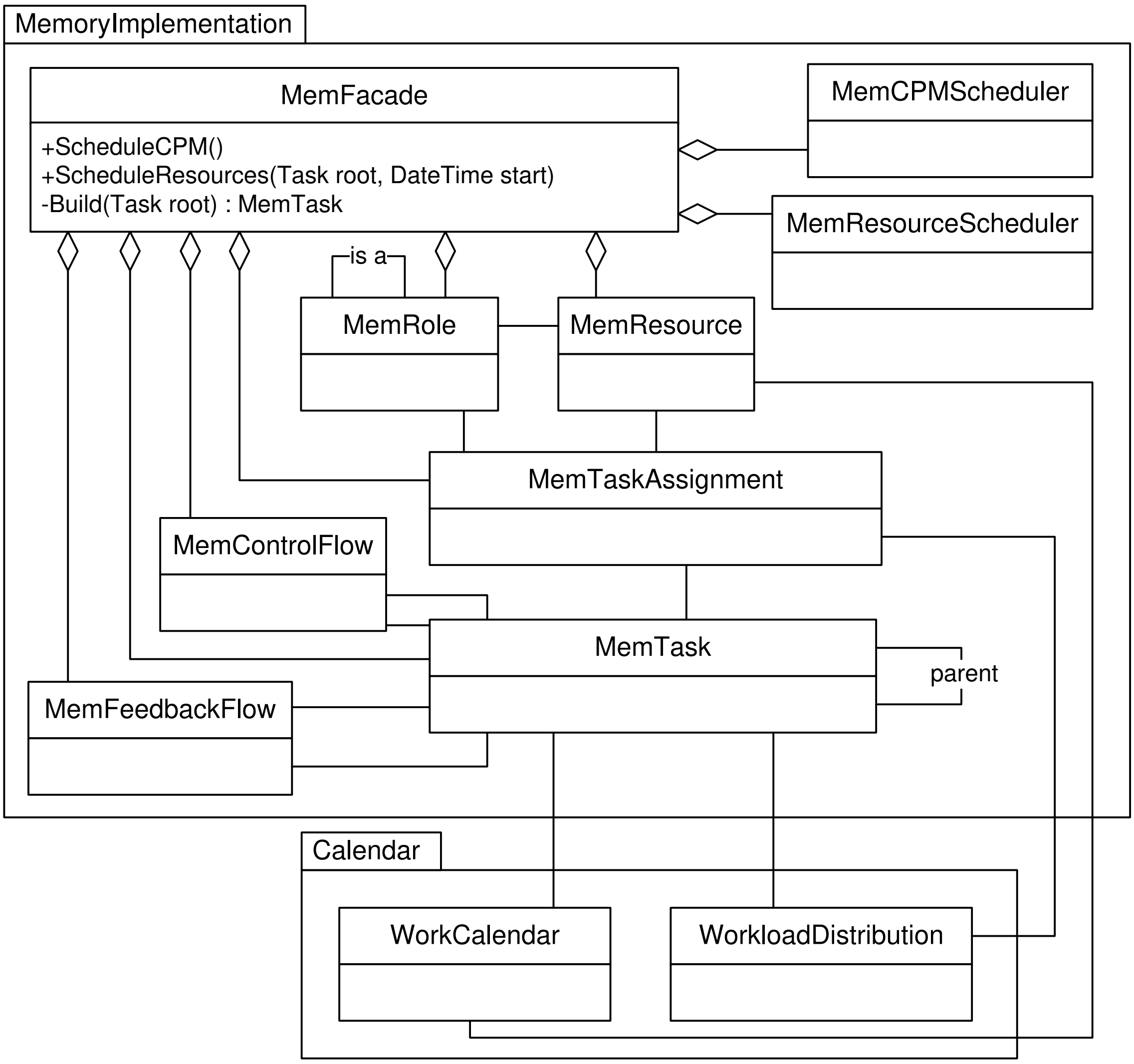
10.2.3 Scheduler

The management data of PROCEED is stored in the Comos database. This has several advantages. The Comos system can be used as the persistence layer of PROCEED and the management data is stored in the same database as the engineering data. When the user makes manual changes to a dynamic task net in PROCEED, these changes are directly written to the database.

For scheduling however, it is required to mirror the dynamic task net in the computer's main memory. The read and write access to the data in main memory is faster than the access to the Comos database which speeds up the computationally intensive scheduling algorithm. Furthermore, the computed timing property values should not be directly written to the database but only when the whole scheduling pass is successfully completed. This way, inconsistent states of the management data can be avoided in which some tasks have been rescheduled and others have not.

The classes required for the memory representation of a dynamic task net are contained in the namespace MemoryImplementation as depicted in Section 10.2.3. There are classes for tasks, task assignments, roles, resources, control flows, and feedback flows. The classes of the memory implementation which correspond to process engine classes have only the minimal set of properties which are required for scheduling. This includes for example the execution state but excludes the degree of completion of a task.

The class MemFacade provides a compact interface for scheduling dynamic task nets and implements the facade design pattern [GHJV94]. For example, the methods ScheduleCPM() and ScheduleResources(Task root, DateTime start) can be invoked to perform the respective scheduling algorithms on the dynamic task net of the current project. The respective algorithms are implemented in the classes MemCPMScheduler and MemResourceScheduler. Before scheduling, the memory implementation of the task net is internally build by the method Build(Task root) which returns the root of the task net hierarchy consisting of instances of the class MemTask. The classes WorkCalendar and WorkloadDistribution which are also used by the process engine



classes are directly used for the memory implementation because they do not access the Comos database after they have been instantiated.

After a successful scheduling pass, the computed planned start and end times are set at the MemTask objects. Furthermore, task assignments have been created by instantiating objects of the class MemTaskAssignment, and workload distributions have been generated for the task assignments. These scheduling results have to be written back to the Comos database. The export has the properties of a database transaction, in particular it is atomic, consistent and durable. Either the whole export of all property values and task assignments is successful, or the whole export is aborted and rolled back, so that the management data in the Comos database remains unchanged.

10.2.4 Project Data Warehouse

For the multidimensional visualization of project management data which has been described in Section 8.3, a data warehouse [JLVV00] has been realized to which the management data can be exported in regular intervals. This project data warehouse could not be realized using the Comos database. The Comos database is an object-oriented database which is based on an underlying relational database. It is not possible to add custom tables to this relational database.

Therefore, a separate relational database is used for the project data warehouse which is managed by an instance of the Microsoft SQL Server. In an ETL (Extract-Transform-Load) process, the management data is extracted from the Comos database, transformed to the schema of the relational database and loaded into the data warehouse. The Microsoft Analysis Services [Mic11] have been applied for the data processing. The measured values which are exported from Comos are arranged along the dimensions of a hypercube in the data warehouse. When this hypercube has been generated, multidimensional data records can be retrieved from the project data warehouse using the query language MDX. The following example of an MDX query retrieves the hypercube for the view configuration *Technical Crews* from the data warehouse.

```
Select [Dim Time].[Hierarchy].Members on axis(0),
       [Dim Roles].[Parent Role].Members on axis(1),
       [Dim Resources].[Parent Resource].Members on axis(2)
From [PM Cube]
Where [Dim Modus].[Dim Modus].&[planned], [Measures].[Workload]
```

The *time* dimension is mapped to the x-axis, the *roles* dimension to the y-axis, and the *resources* dimension to the third axis which is visualized by the stack layers of in the pivot table of the project status analysis view. The measure is the planned workload, aggregated over all tasks and plant parts .

Besides a full export of the management data, the export can also be performed *incrementally*. In this case, only changed measured values are exported. Whenever a change to a task, task assignment, resource or the like occurs, the changed values of the affected entities are immediately exported to the data warehouse and the hypercube is updated. This dynamic update functionality constituted a technical challenge, since the export and processing of data is time consuming even for an incremental export, but the user should not impeded in his work with PROCEED. Therefore, the incremental export of changed data sets is started in a separate thread, so that the user can work with the GUI of PROCEED while the export takes place. After the export has completed, a manual update of the pivot table of the project status analysis view will show the changed values. In this way, dynamic changes to the project management data are immediately reflected in the multidimensional analysis view.

For every full or incremental export of management data to the project data warehouse, a new *time stamp* is created. All exported measured values are associated with this time stamp. In this way, the history of plan changes is stored in the data warehouse and can be visualized in the project status analysis view. The subsequent values of the total workload of a task can be retrieved from the project data warehouse with the following MDX query.

```
Select [Dim Time Stamp].[Dim Time Stamp].Members on axis(0),
From [PM Cube]
Where ([Dim Modus].[Dim Modus].&[planned],
      [Dim Tasks].[Parent Task].&[A2CDOWFM78],
      [Measures].[Workload])
```

The time stamps for the subsequent states of planning are mapped to the x-axis. The modus is set to planned values thereby excluding the actual workload of the task. A slicing operation is performed for the *tasks* dimension, so that only the values for a particular task are returned. In this example, the unique id `A2CDOWFM78` identifies the task Basic Engineering. Finally, the measure workload is selected. The retrieved values are displayed in a line diagram instead of a pivot table as described in Section 8.3.

10.2.5 Coupling with External Project Management System

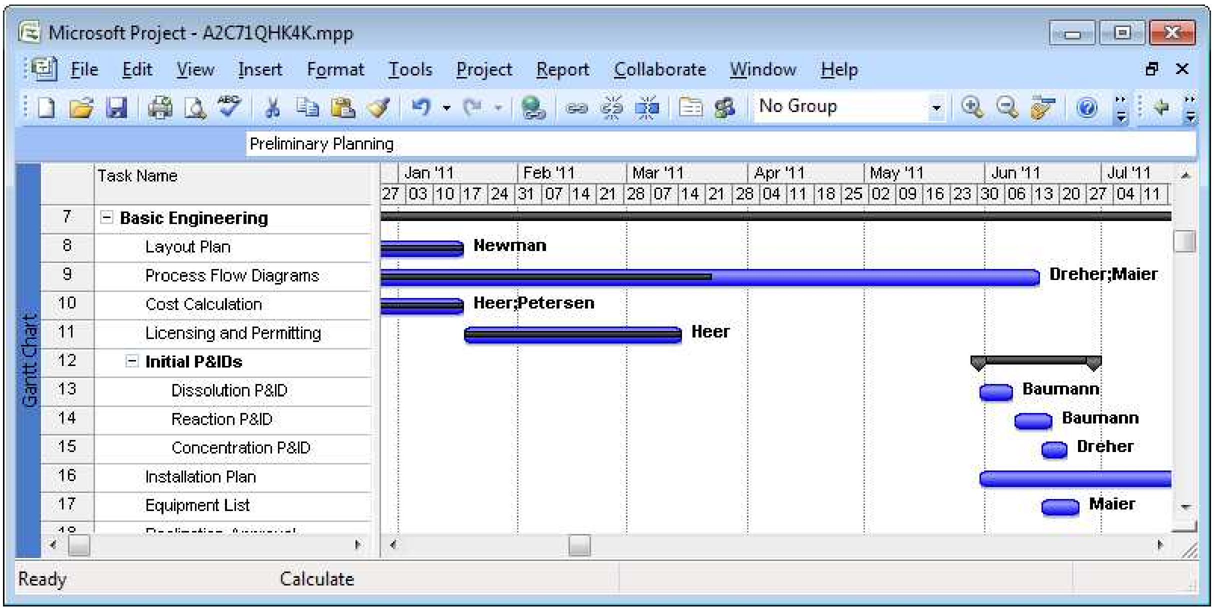
The project management system Microsoft Project has been coupled with PROCEED to use its Gantt chart view for the presentation of the project schedule. Figure 10.4 shows a screenshot of MS Project with the exported dynamic task net of the example scenario.

When MS Project is invoked from PROCEED, the current state of the dynamic task net is exported to a project plan in Ms Project. Only scheduled tasks are exported to MS Project, i.e. management tasks, work steps, and zero-duration tasks are not represented in the Gantt chart. For all other tasks the current and possibly existing older versions of the respective task are exported as individual tasks to MS Project.

The following properties of a task are exported from PROCEED to MS Project.

- Name
- Description
- Start and end time
- Planned start and end time
- Due date
- Assigned resources
- Degree of completion

A minor difference between tasks in PROCEED and MS Project is related to the planned end time of a task. While in PROCEED, the end time of a task is the last date on which working hours are scheduled, no working hours can be scheduled on the finish date of a task in MS Project. Therefore, the planned end time of a task had to be mapped to the next date in MS Project. Besides the functional properties of a task, the unique id of a task in the Comos database is also exported to establish the mapping between the tasks in PROCEED and MS Project.



	Task Name	
7	**Basic Engineering**	
8	Layout Plan	Newman
9	Process Flow Diagrams	Dreher;Maier
10	Cost Calculation	Heer;Petersen
11	Licensing and Permitting	Heer
12	**Initial P&IDs**	
13	Dissolution P&ID	Baumann
14	Reaction P&ID	Baumann
15	Concentration P&ID	Dreher
16	Installation Plan	
17	Equipment List	Maier

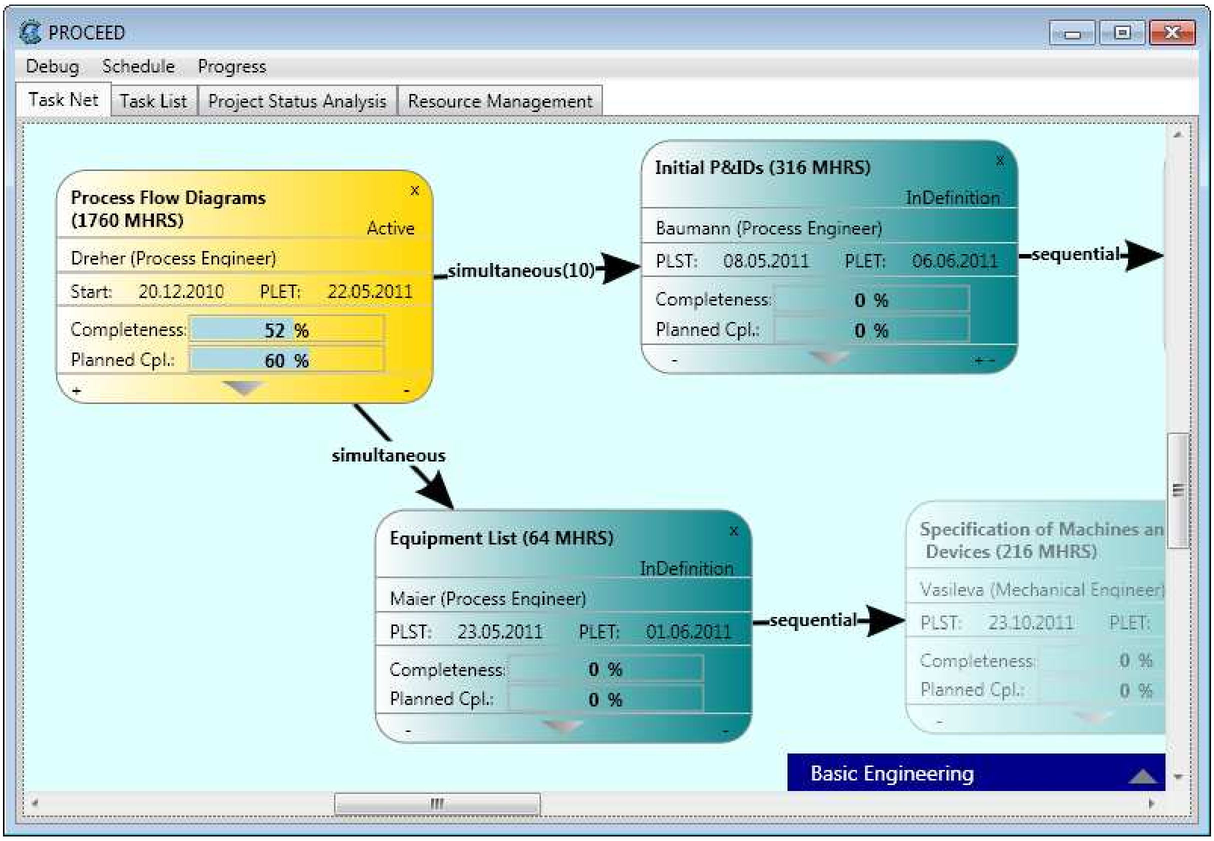
Abbildung 10.4: Exported dynamic task net in MS Project.

Control flows are not exported to MS Project. In general, control flows could be mapped to task dependencies in MS Project. However, several limitations rendered a semantics preserving mapping impossible. First, there are no simultaneous control flows in MS Project, and it is not possible to define two different task dependencies between two tasks, e.g. to simulate simultaneous control flows by a combination of start-start and end-end task dependencies. Second, the start-start and start-end task dependencies which can be defined in MS Project have no equivalent counterpart in dynamic task nets. Finally, complex tasks in MS Project cannot be the target of task dependencies which impose constraints on their end dates, i.e. start-end and end-end task dependencies cannot be defined for these tasks.

After all tasks have been exported to MS Project, the project plan reflects the corresponding part of the dynamic task net. An event mechanism has been realized which ensures that all subsequent changes to the dynamic task net in PROCEED are immediately reflected in MS Project by incrementally exporting the changed data. On the other hand, the user can change the planned start and end times of tasks in MS Project, and he can create new subtasks or delete existing tasks. If he is authorized to perform these changes in PROCEED, and the changes can be permanently or temporarily accepted with respect to behavioral and timing consistency constraints, then they are applied to the dynamic task net. Otherwise, the intended change operation is prohibited in MS Project and the project plan remains unchanged.

10.3 User Interface

In this section, the different management and monitoring views of the PROCEED prototype are presented. The presentation of the management data and the functionality



Process Flow Diagrams (1760 MHRS)
Active
Dreher (Process Engineer)
Start: 20.12.2010 PLET: 22.05.2011
Completeness: 52 %
Planned Cpl.: 60 %

Initial P&IDs (316 MHRS)
InDefinition
Baumann (Process Engineer)
PLST: 08.05.2011 PLET: 06.06.2011
Completeness: 0 %
Planned Cpl.: 0 %

Equipment List (64 MHRS)
InDefinition
Maier (Process Engineer)
PLST: 23.05.2011 PLET: 01.06.2011
Completeness: 0 %
Planned Cpl.: 0 %

Specification of Machines an
Devices (216 MHRS)
Vasileva (Mechanical Engineer)
PLST: 23.10.2011 PLET:
Completeness: 0 %
Planned Cpl.: 0 %

Abbildung 10.5: Screenshot of the Task Net View.

for modifying the data are described.

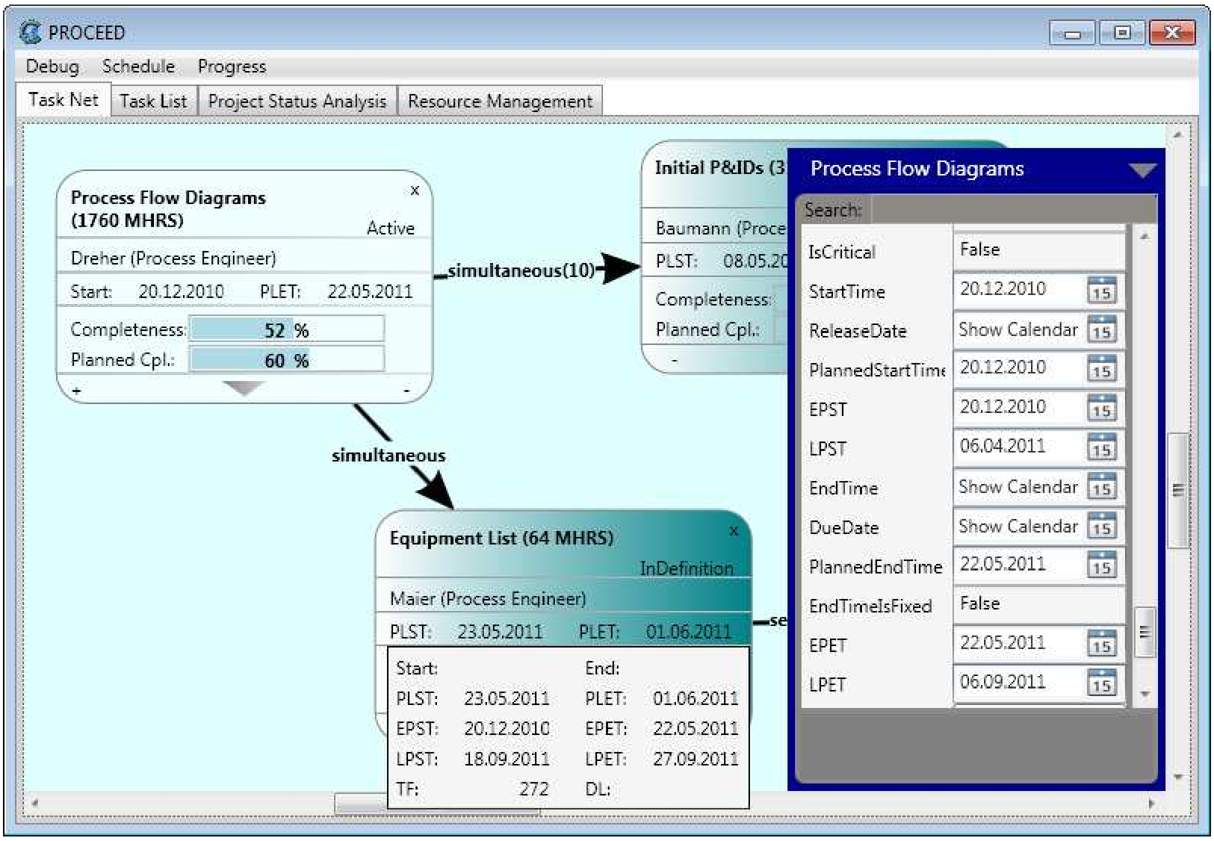
The user interface of the PROCEED prototype has been implemented using the Windows Presentation Foundation (WPF) framework by Microsoft [Nat06]. One of the advantages of this new graphics framework is that all controls are vector-based graphics and can be arbitrarily scaled, so that zooming functionality could be easily realized.

10.3.1 Project Management Views

PROCEED provides two different views for process management. In these views, tasks and task relationships of a process model instance can be defined and modified. Furthermore, the process model instance can be enacted, i.e. task execution states can be changed and document revisions can be produced.

Process model instances are internally represented as dynamic task nets in PROCEED. The *Task Net View* uses this internal data model also for the presentation of the management data to the user. Tasks and task dependencies are represented in a network diagram. In this way, all logical relationships between tasks are visualized, but not their extension and their relative position in time.

Figure 10.5 shows a screenshot of the PROCEED Task Net View. Tasks are repre-



Process Flow Diagrams (1760 MHRS) Active
Dreher (Process Engineer)
Start: 20.12.2010 PLET: 22.05.2011
Completeness: 52 %
Planned Cpl.: 60 %

Equipment List (64 MHRS) InDefinition
Maier (Process Engineer)
PLST: 23.05.2011 PLET: 01.06.2011

Start:		End:	
PLST:	23.05.2011	PLET:	01.06.2011
EPST:	20.12.2010	EPET:	22.05.2011
LPST:	18.09.2011	LPET:	27.09.2011
TF:	272	DL:	

Process Flow Diagrams

IsCritical	False
StartTime	20.12.2010
ReleaseDate	Show Calendar
PlannedStartTime	20.12.2010
EPST	20.12.2010
LPST	06.04.2011
EndTime	Show Calendar
DueDate	Show Calendar
PlannedEndTime	22.05.2011
EndTimeIsFixed	False
EPET	22.05.2011
LPET	06.09.2011

Abbildung 10.6: Task properties in the Task Net View.

sented by boxes and control flow by labeled edges between the task boxes. Besides the name and total workload of a task, its execution state, the responsible resource with the required role, and the start and end dates are displayed. For a preparing task, the planned start and end dates are displayed, while for running and terminated tasks, the respective actual dates are displayed. The actual and planned degrees of completion of a task are visualized by horizontal progress bars, so that they can be directly compared. The color of the task box indicates, whether the task has fallen below defined thresholds of the CPI or SPI. The setting whether the color markings in the management views represent the SPI or CPI can be changed by the user. In Figure 10.5, the yellow color of the task Process Flow Diagrams indicates, that the SPI value of the task is between 0.7 and 0.9.

The representation of a task in the task net does not allow to display all relevant property values at once. Therefore, the property values of a selected task can be inspected and modified in the property box which is depicted for the task Process Flow Diagrams in Figure 10.6. Furthermore, additional information about the task is displayed when the user holds the mouse over certain parts of the task box, e.g. in Figure 10.6 the mouse cursor is located above the planned dates of the task Equipment List whereupon a box with all actual and planned dates as well as computed constraint dates of the tasks is shown.

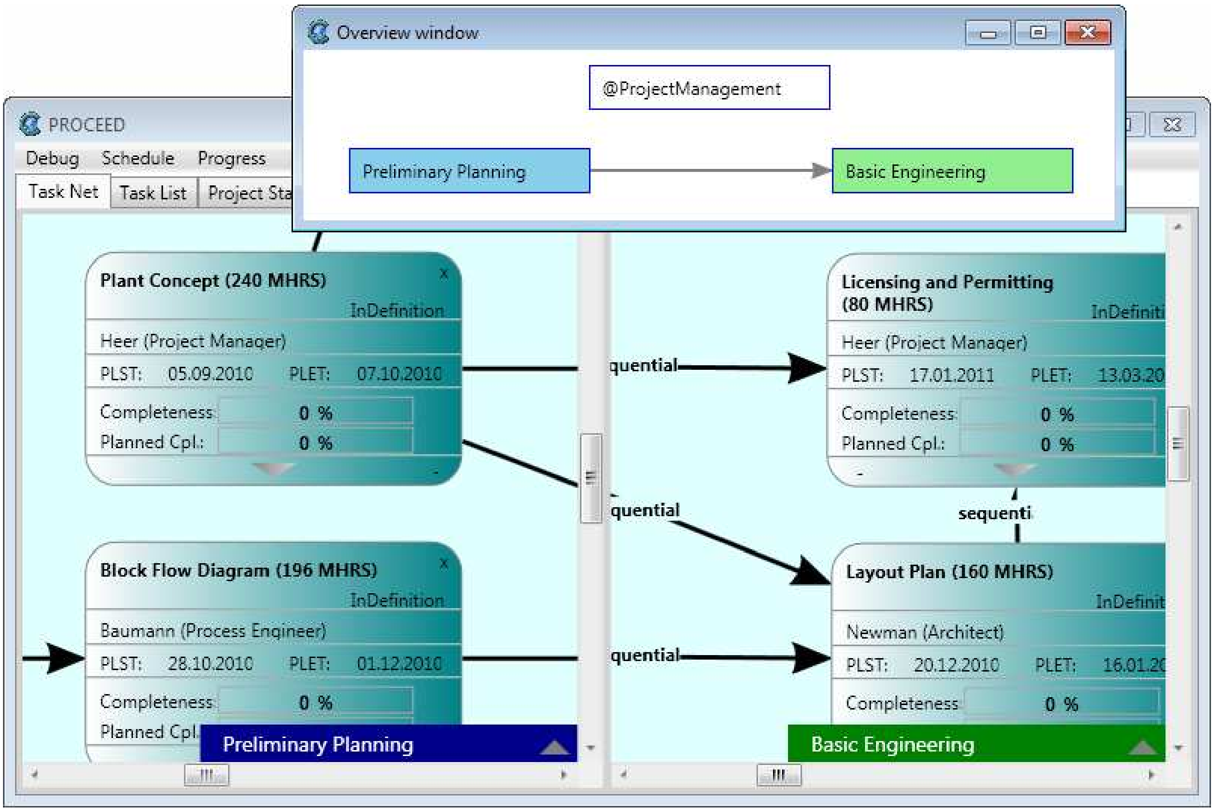
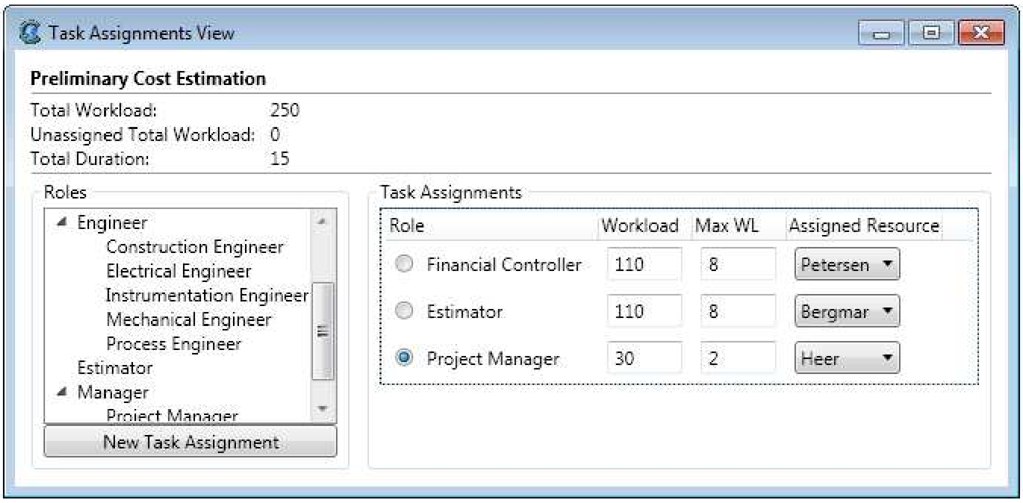


Abbildung 10.7: Split screen and overview window of Task Net View.

The *Task Net View* shows only one level of the task net hierarchy at a time. The user can navigate downwards and upwards in the task net hierarchy by clicking on the tasks. The tasks depicted in Figure 10.5 are all subtasks of the task Basic Engineering except for the task Specification of Machines and Devices. The latter is a subtask of the task Detail Engineering but is at the same time a successor of a task Equipment List. In dynamic task nets, it is possible to define control flows and data flows between tasks contained in different realizations. To display these dependencies, the dependent tasks from other realizations are depicted as transparent boxes in the Task Net View.

To define control flows between tasks of different realizations in the first place, it is required to display both at the same time. For this purpose, the Task Net View can be split into two parts as depicted in Figure 10.7. On the left hand, the subtasks of the task Preliminary Planning are displayed while on the right side the subtasks of Basic Engineering are visible. This *split screen* allows to define task dependencies between subtasks of different parent tasks. The *overview window* shows the user how the parent tasks are related to each other and thereby whether the subtasks may be connected by control flows.

Finally, an additional dialog can be opened for a task to define and modify its task assignments. This dialog is depicted in Figure 10.8. It shows the relevant properties of the task, namely its total workload, the unassigned total workload, and the total



Preliminary Cost Estimation

Total Workload: 250
Unassigned Total Workload: 0
Total Duration: 15

Role	Workload	Max WL	Assigned Resource
○ Financial Controller	110	8	Petersen ▾
○ Estimator	110	8	Bergmar ▾
● Project Manager	30	2	Heer ▾

Abbildung 10.8: Task assignment dialog.

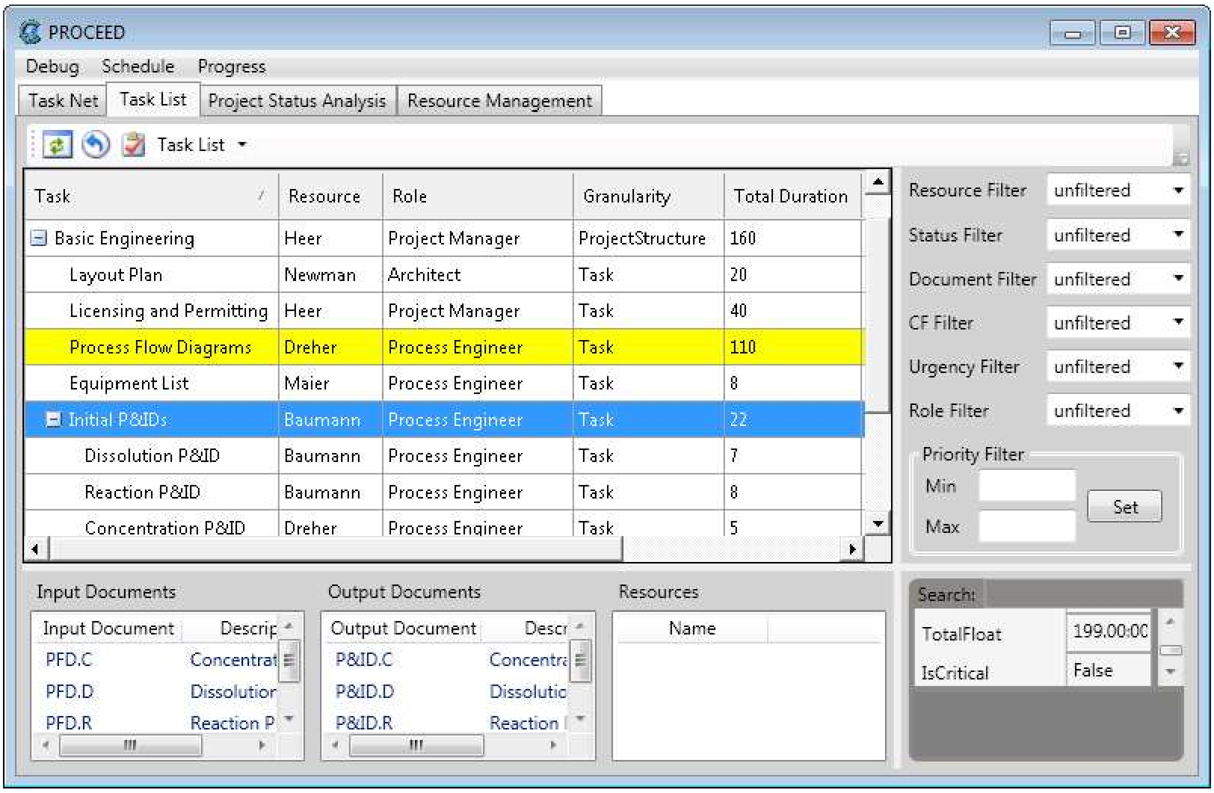
duration. On the right hand side the currently defined task assignments are listed, where the task assignment for the responsible resource is marked. A new task assignment can be defined by selecting a role from the list of project roles on the left side and clicking the button below.

Besides the Task Net View, a *Task List View* is provided by PROCEED, which can be used by individual project team members as a todo list with their assigned tasks, but also by the project manager to get an overview over all tasks in the project. The list reflects the hierarchical structure of the task net. Several task properties can be displayed in the list which can be selected by the user. When a task is selected in the list, its input and output parameters, additional resources, and all properties are displayed on the bottom of the Task List View. Tasks which do not perform as planned are marked in the Task List View just like in the Task Net View by colors indicating the degree of delay or budget overrun.

Several filters can be applied to the Task List View. An example setting is depicted in Figure 10.10 where only the preparing tasks to which the logged in user Heer is assigned are displayed.

Restricted Accessibility The authorization model described in Section 5.5 has been implemented in PROCEED. Authorization rules are evaluated to decide whether a user is allowed to perform a certain change operation and which management data is visible to the user. The functionalities of the user interface are enabled or disabled depending on the user's authorization so that the user is only permitted to perform authorized operations. In addition, the displayed data is filtered according to the user's authorization.

Figure 10.11 shows a screenshot of the Task Net View with restricted functionality. The user Baumann is currently logged in. He can see the task Initial P&IDs for which he is responsible as well as the tasks in the work context of this task. However, he cannot see all subtasks of Basic Engineering. The task Equipment List is not displayed.



Task	Resource	Role	Granularity	Total Duration
⊟ Basic Engineering	Heer	Project Manager	ProjectStructure	160
Layout Plan	Newman	Architect	Task	20
Licensing and Permitting	Heer	Project Manager	Task	40
Process Flow Diagrams	Dreher	Process Engineer	Task	110
Equipment List	Maier	Process Engineer	Task	8
⊟ Initial P&IDs	Baumann	Process Engineer	Task	22
Dissolution P&ID	Baumann	Process Engineer	Task	7
Reaction P&ID	Baumann	Process Engineer	Task	8
Concentration P&ID	Dreher	Process Engineer	Task	5

Resource Filter: unfiltered
Status Filter: unfiltered
Document Filter: unfiltered
CF Filter: unfiltered
Urgency Filter: unfiltered
Role Filter: unfiltered
Priority Filter: Min, Max

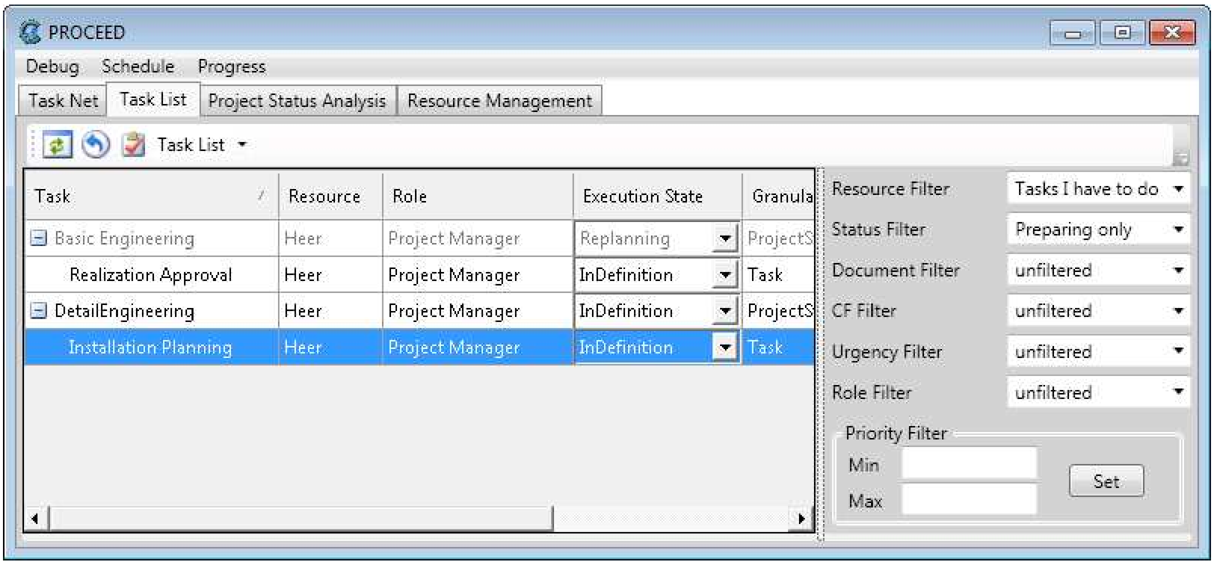
Input Document	Descrip
PFD.C	Concentrat
PFD.D	Dissolution
PFD.R	Reaction P

Output Document	Descr
P&ID.C	Concentra
P&ID.D	Dissolutio
P&ID.R	Reaction I

Name

TotalFloat	199.00:0C
IsCritical	False

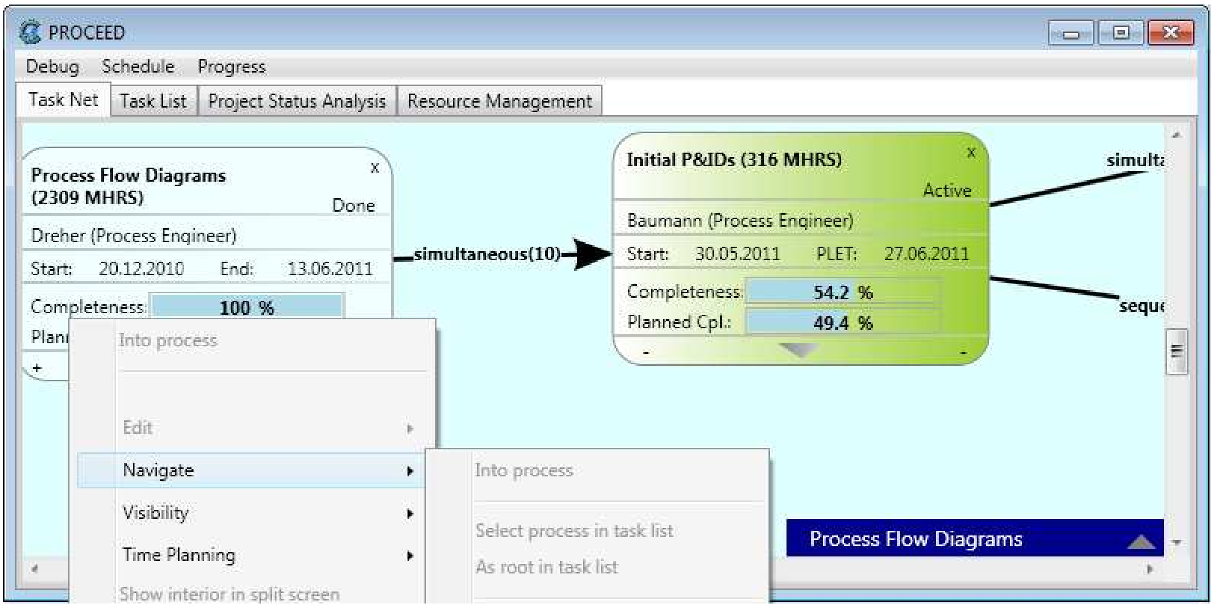
Abbildung 10.9: Screenshot of the Task List View.



Task	Resource	Role	Execution State	Granula
⊟ Basic Engineering	Heer	Project Manager	Replanning	ProjectS
Realization Approval	Heer	Project Manager	InDefinition	Task
⊟ DetailEngineering	Heer	Project Manager	InDefinition	ProjectS
Installation Planning	Heer	Project Manager	InDefinition	Task

Resource Filter: Tasks I have to do
Status Filter: Preparing only
Document Filter: unfiltered
CF Filter: unfiltered
Urgency Filter: unfiltered
Role Filter: unfiltered
Priority Filter: Min, Max

Abbildung 10.10: Filters applied to the Task List View.



Process Flow Diagrams (2309 MHRS)
Done
Dreher (Process Engineer)
Start: 20.12.2010 End: 13.06.2011
Completeness: 100 %

Initial P&IDs (316 MHRS)
Active
Baumann (Process Engineer)
Start: 30.05.2011 PLET: 27.06.2011
Completeness: 54.2 %
Planned Cpl.: 49.4 %

Abbildung 10.11: Limited visibility and accessibility of management data.

Furthermore, the user Baumann cannot navigate into the task Process Flow Diagrams to view its subtasks, and he may not perform any change operations on this task. This can be seen in Figure 10.11, where the user opened the context menu for the task Process Flow Diagrams in which certain entries are disabled.

In general, the management views are adapted according to the effective permissions of the logged in user. In contrast, other views for process definition, project monitoring, and resource management are completely disabled if the user does not have the required super permissions.

10.3.2 Process Definition Tools

While the management views are used in a project to manage and enact process model instances, there are dedicated views for the creation of process model definitions and the management of the process knowledge. The definition of task, process, and workflows templates for technical processes and management processes is usually performed by a process engineer who has to be one of the administrators of the Comos database. Task types are directly defined in Comos by creating according base objects. For the definition of process templates, the PROCEED *Template Editor* is used. This tool is similar to the Task Net View but provides only restricted functionality which is required to create a process template before project runtime.

Figure 10.12 shows a screenshot of the template editor, in which the task net part of the workflow template Specify Pump is displayed. It is not possible to change the execution states of the defined tasks and to set planned dates. The required roles of the responsible resources are defined but no actual resources can be assigned to the tasks.

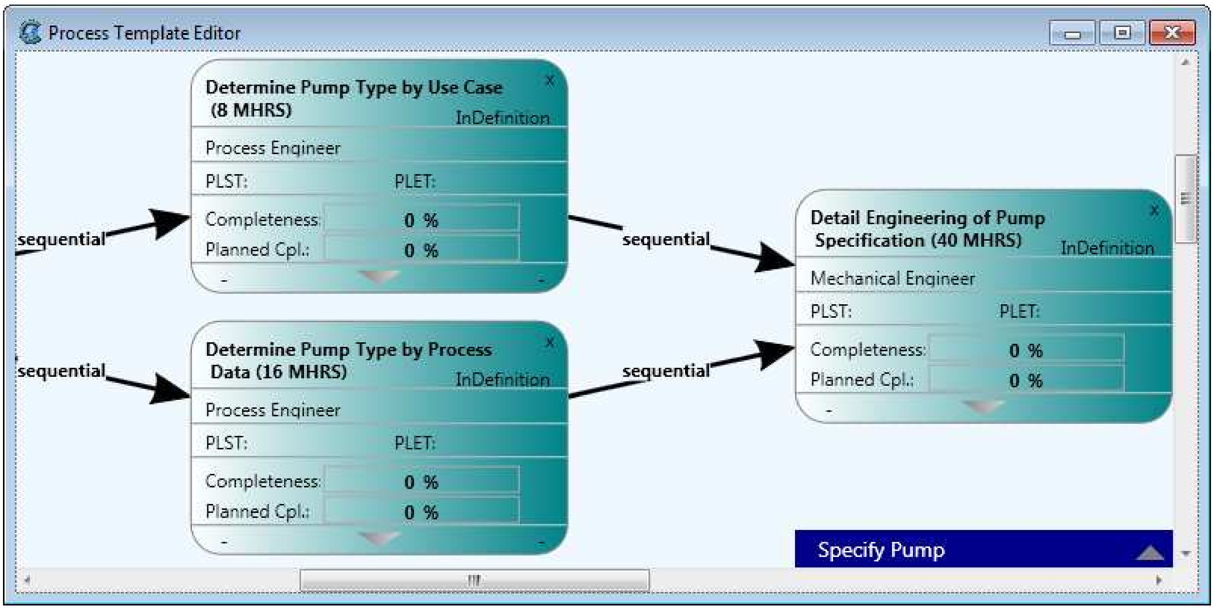


Abbildung 10.12: Screenshot of the PROCEED Process Template Editor.

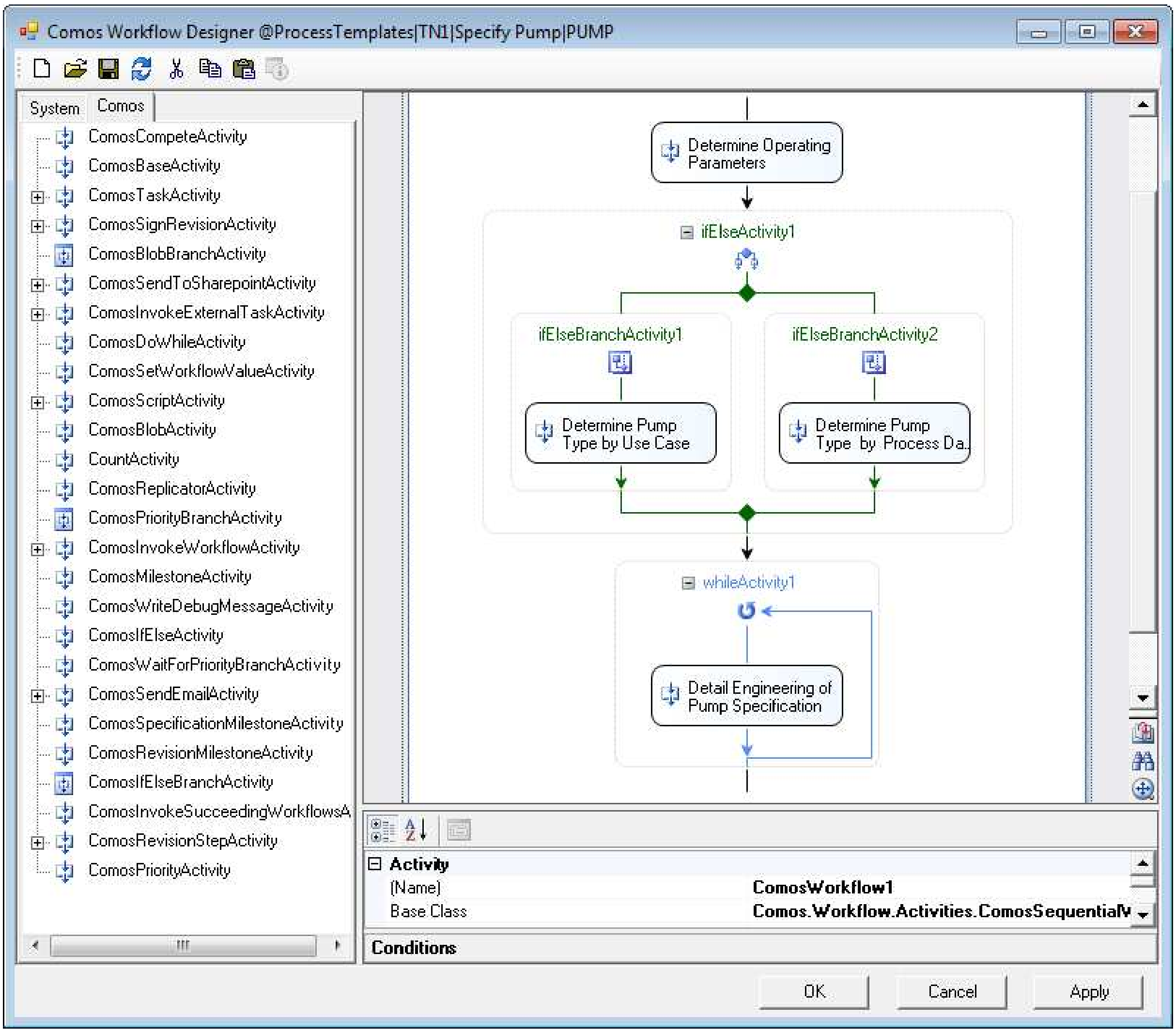
The workflow definition which is associated with such a workflow template is defined by means of the *Workflow Designer*. This tool allows to define control structures for alternative branching and loops. A screenshot of the Workflow Designer is depicted in Figure 10.13. On the right side, the workflow definition is visualized graphically. An IfElse activity and a While activity have been used to define the workflow Specify Pump. Below the workflow definition, the properties of the currently selected activity are displayed and can be modified. On the left side of the designer window, a list of available activity types is displayed most of which represent specific atomic and complex activities which have not been introduced in this thesis. An activity can be selected in the list, dragged onto the workflow definition, and dropped at the position where it shall be inserted.

The tools to define and monitor workflows have been implemented using the Windows Workflow Foundation (WF) [Buk08]. This framework provides several reusable controls, e.g. for the graphical representation of the workflow definition.

10.3.3 Monitoring Views

The management views already provide functionality for monitoring enacted process model instances. The execution states of tasks, their degree of completion, and the computed performance indices are displayed in the Task Net View and the Task List View.

An additional view can be used to inspect the current enactment state of workflow-managed task nets. This *Workflow Monitor* is an extended version of the Workflow Designer. The workflow definition is augmented by symbols which visualize the current enactment state of the workflow as depicted in Figure 10.14. Checkmarks



(Name) ComosWorkflow1
Base Class Comos.Workflow.Activities.ComosSequentialW

Abbildung 10.13: Screenshot of the PROCEED Workflow Designer.

indicate that an activity has been completed. Currently executing activities are marked with a play-symbol. In addition, the number of iterations is shown for every activity. In Figure 10.14, the While activity is in the second iteration. Furthermore, an event log is presented to the user in the Workflow Monitor which shows all execution state changes of the workflow instance and its activities.

The Workflow Monitor is not only used for monitoring running workflow instances, but also to apply dynamic structural changes to the workflow definition at runtime. Therefore, the Workflow Monitor provides the same functionality as the Workflow Designer. For dynamic changes at workflow runtime, certain restrictions apply, which have been described in Section 6.3.6. The WF provides the possibility to change running workflow instances programmatically via API calls, and it provides a user control for the graphical editing of workflow definitions. However, at the time of development of the PROCEED prototype, there was no solution available for performing dynamic changes to a running workflow instance using the controls

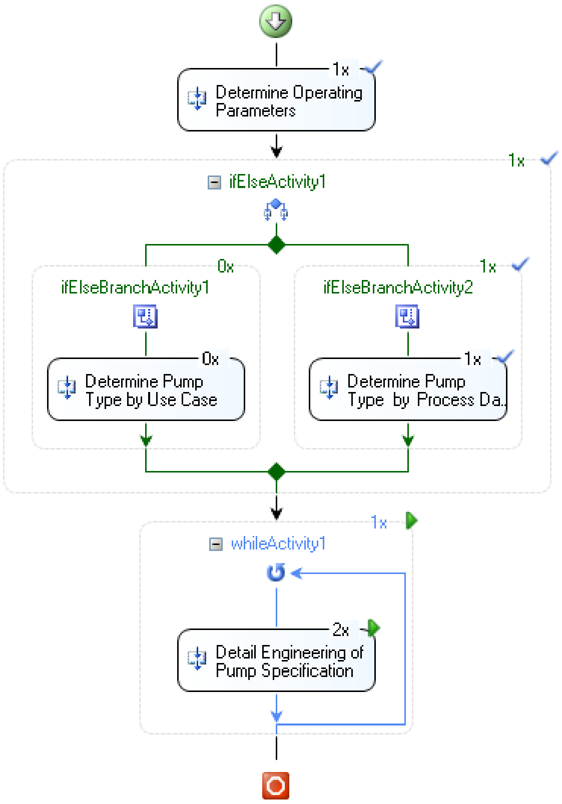
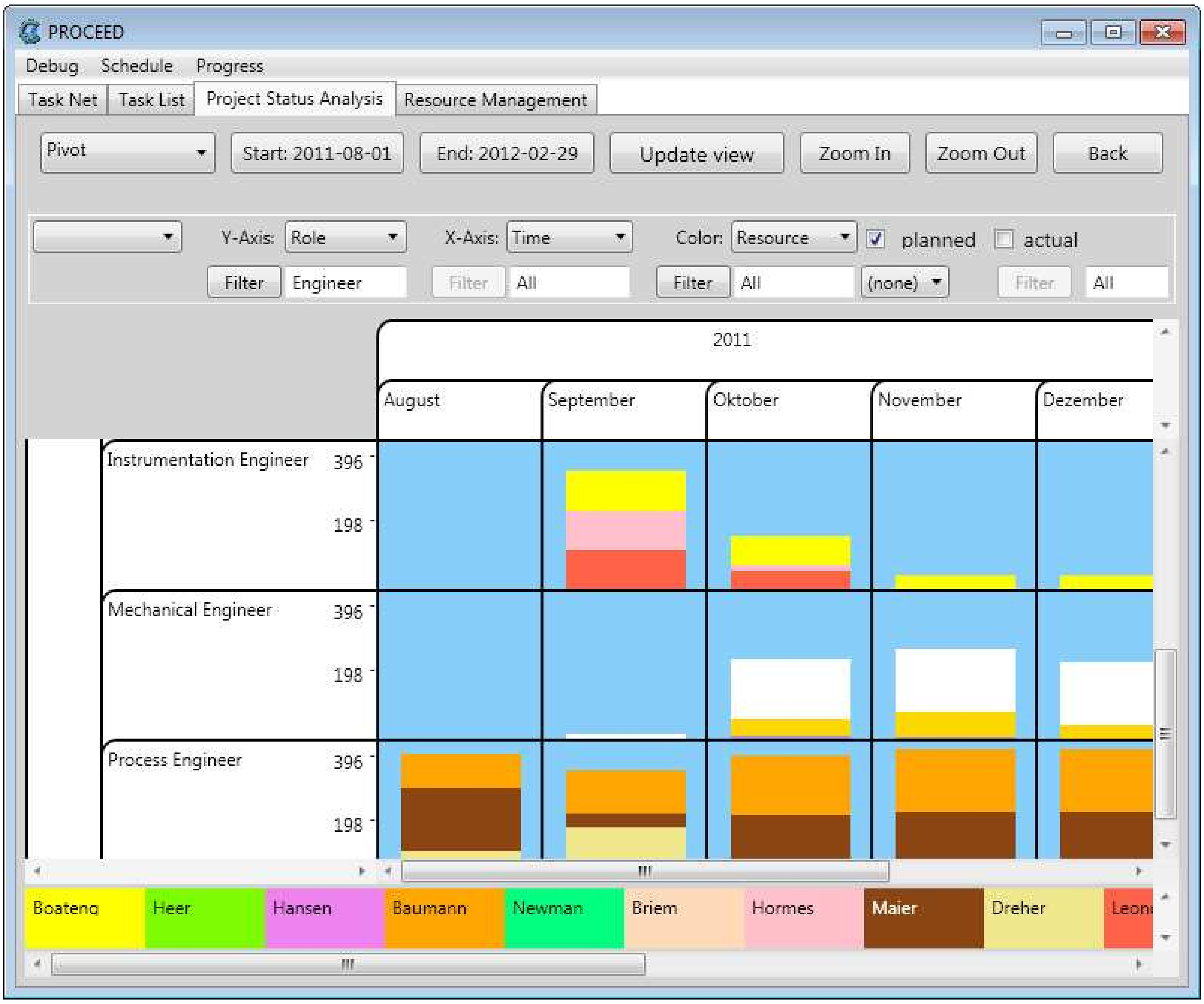


Abbildung 10.14: Workflow enactment state in PROCEED Workflow Monitor.

for the graphical representation of the workflow definition. This posed a technical challenge which could be successfully solved. The user can work with the Workflow Monitor like with the Workflow Designer by dragging activities into the workflow definition or moving activities inside the workflow definition. When he confirms his changes, they are applied to the running workflow instance.

The *Project Status Analysis View* which has been introduced in Section 8.3 enables project managers and controllers to visually analyze the current project status with respect to different key figures. Because this view provides an overview over all tasks in the project, it can only be used by authorized users who have the super permission to view all tasks in the project (cf. Section 5.5).

Figure 10.15 shows a screenshot of the pivot table which is used for the multi-dimensional visualization of the project management data. The user has selected the configuration *Technical Crews* which maps the *time* dimension to the x-axis, the *roles* dimension to the y-axis, and the *resources* dimension to the colors of the stacked bars. The user has furthermore defined a filter for the y-axis which only displays the role Engineer and specializations thereof. The height of the stacked bars shows how many working hours have been scheduled for the different roles and how they are distributed between the resources. This view can be used to analyze the current state of planning in order to optimize the assignment of resources to tasks. It can also help a project manager to find a suitable resource for a new task



Pivot

Start: 2011-08-01

End: 2012-02-29

Y-Axis: Role

X-Axis: Time

Color: Resource

☑ planned ☐ actual

Filter Engineer

Filter All

Filter All

(none)

Filter All

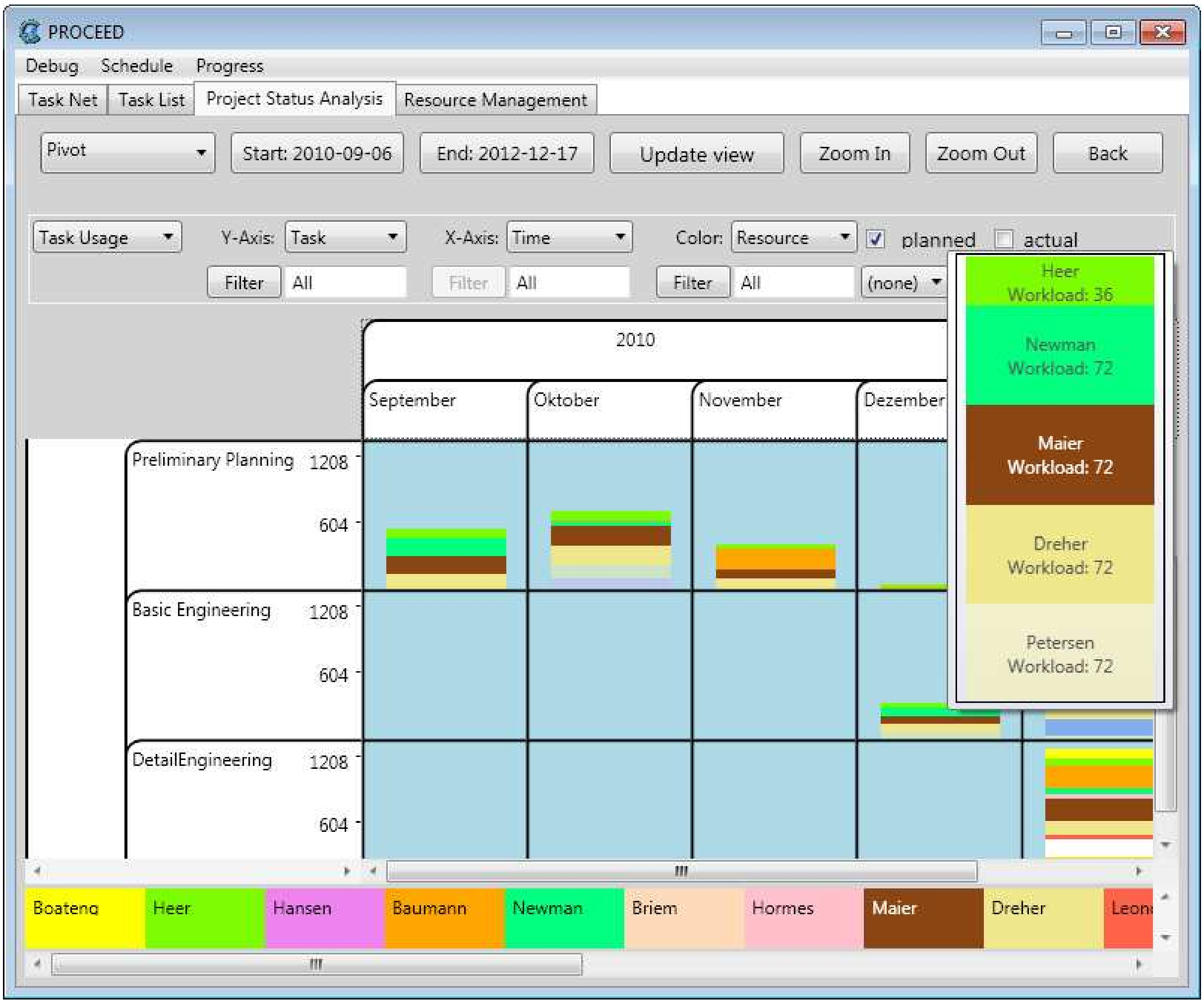
Abbildung 10.15: Pivot table configuration *Technical Crews*.

assignment which has not too many tasks assigned yet for the given timeframe.

In Figure 10.16, the view configuration *task usage* is depicted, in which the planned workload for different tasks in a selected timeframe is visualized, divided into the shares of the assigned users. The details of the stacked bars can be inspected by hovering of the coordinates of the pivot table with the mouse, so that data tips [Tid06] are displayed.

Further information graphics patterns have been applied in order to enable the user to grasp the complex multidimensional data. Data brushing [Tid06] is used to highlight data sets which refer to the same coordinate of a dimension, e.g. only the stack layers corresponding to a particular resource can be highlighted in Figure 10.15 while all other layers are displayed in the same color. The pivot table itself is an application of the small multiples [Tuf86] pattern, which allows for a very dense but clear presentation of large multidimensional datasets.

The project status analysis view allows the user to quickly *navigate* between different configurations of the pivot table. In addition to the manual view configuration



Pivot
Start: 2010-09-06
End: 2012-12-17
Task Usage
Y-Axis: Task
X-Axis: Time
Color: Resource
☑ planned ☐ actual
Filter: All
Filter: All
Filter: All
(none)

Heer Workload: 36
Newman Workload: 72
Maier Workload: 72
Dreher Workload: 72
Petersen Workload: 72

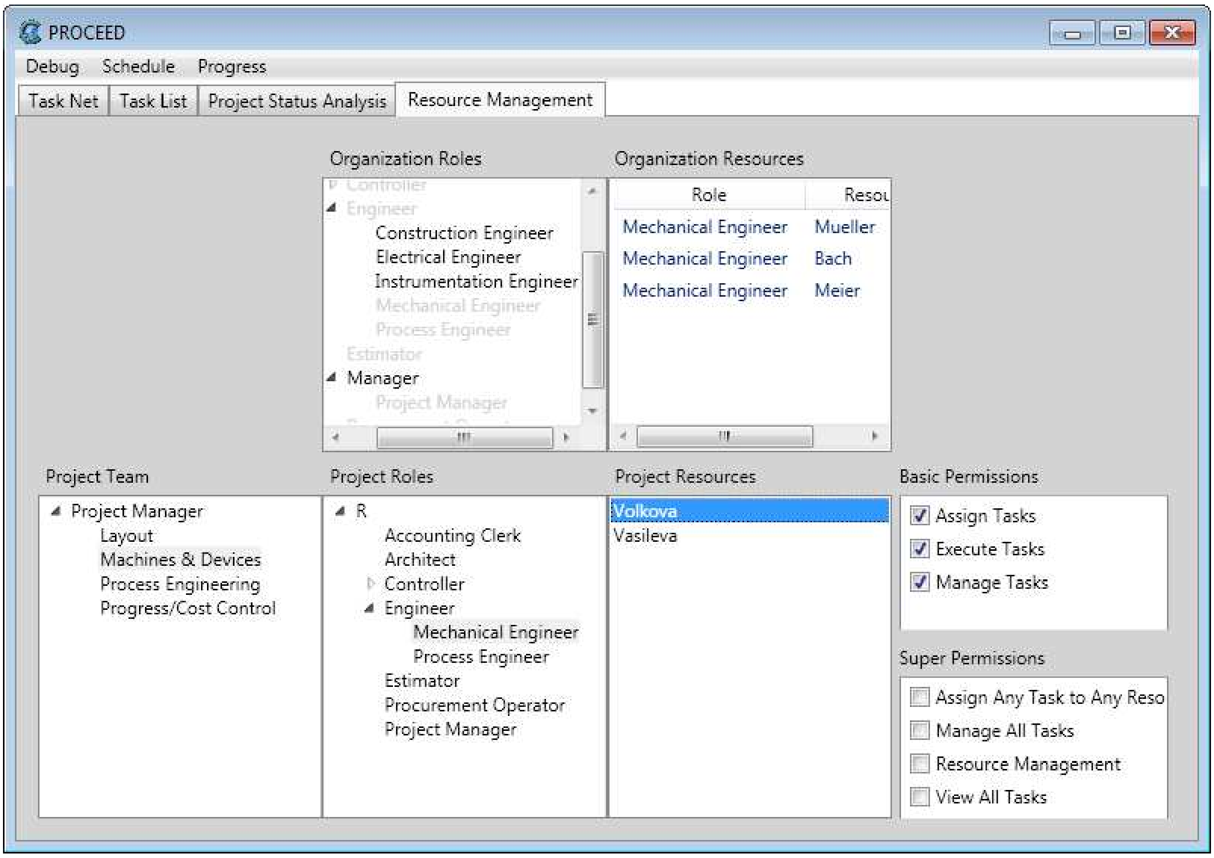
Abbildung 10.16: Pivot table configuration *Task Usage*.

and the selection of pre-defined configurations, several *navigation operations* are available which realize most of the operations on a multidimensional dataset which have been introduced in Section 8.3, e.g. pivoting by exchanging the dimensions of the axes. The operations can be invoked via context menus of the axes and the stacked-bar charts and ease the handling of the project status analysis view.

10.3.4 Resource Management View

A dedicated view has been implemented to support the management of roles, users, and permissions in a project. Figure 10.17 shows a screenshot of the *Resource Management View*. Only authorized users who have the super permission for resource management in the project (cf. Section 5.5) may use this view.

The top part of the view shows the roles which have been defined for the organization, and the users who can play these roles in the organization. The bottom part shows the structure of the project team, the available roles, the allocated resources,



Role	Resou
Mechanical Engineer	Mueller
Mechanical Engineer	Bach
Mechanical Engineer	Meier

Project Resources:
Volkova
Vasileva

Basic Permissions:
☑ Assign Tasks
☑ Execute Tasks
☑ Manage Tasks

Super Permissions:
☐ Assign Any Task to Any Reso
☐ Manage All Tasks
☐ Resource Management
☐ View All Tasks

Abbildung 10.17: PROCEED Resource Management View.

and their permissions in the project. When a subteam of the project team is selected in the tree view on the left, then the members of the subteam are displayed in the list Project Resources. If an available role is selected, then all project resources which can play this role in the project are displayed. When a project resource is selected, then the permission lists show the setting for this resource, e.g. the resource Volkova has only the basic permissions but no super permissions. A resource can be added to the project team by dragging it from the organization resources list onto one of the project subteams. When a new user is added to a project, the default permissions are automatically assigned to the user.

10.4 Implementation Size

The PROCEED prototype has been developed at the Department of Computer Science 3 at RWTH Aachen University in close cooperation with Siemens Industry Software, formerly known as innotec. A developer of innotec contributed to the workflow engine of the PROCEED system. Altogether, eleven developers including employees of the department and students contributed to the software prototype. The development took about three years with interruptions.

	C#		XAML	
Component	files	LOC	files	LOC
Process Engine, Access Control, Scheduling, Progress Measurement	66	21053	0	0
Project Management Views	146	27550	52	2090
Process Management Tools	19	4814	0	0
Project Status Analysis View	87	12869	24	479
Workflow Engine, Workflow Progress Measurement, Workflow Integration	106	14739	0	0
MS Project Integration	8	2283	0	0
Total	432	83308	76	2569

Tabelle 10.1: Lines of code of the PROCEED prototype.

Table 10.1 shows the size of the implementation in terms of source files and lines of code (LOC). The lines of code are counted excluding blank lines and comments. The PROCEED prototype has been implemented using the programming language C#. The main part of the user interface has been realized using the Windows Presentation Foundation (WPF). The source code of WPF-based user controls is divided into the layout part which is defined in the XAML format and the program logic which is written in C#. The XAML format is a WPF specific XML format. The Management Core excluding the workflow management functionality amounts to 66 source files and 21053 LOC. The views for task and resource management in a project add another 24550 lines of C# code and 2090 lines in XAML files. There are comparably few LOC for the process management tools because the Template Editor is merely a restricted version of the Task Net View, and the Workflow Monitor is an extended version of the Workflow Designer. The LOC for the Project Status Analysis View include the source code for the ETL export of the management data to the data warehouse. Altogether, about 85T lines of code have been written to realize the PROCEED prototype.

In contrast to the AHEAD prototype, no generative software development framework like the PROGRES system [SWZ99] could be used for the development of PROCEED. It has been a requirement of the industry partner to realize the prototype using the applied Microsoft technologies for which no tools exist which could generate the application logic from a formal specification. The Visual Studio development environment merely provides support for designing graphical user interfaces. User controls can be visually designed while the editor generates the source code which defines their layout.

10.5 Conclusion

In this chapter it has been shown that the concepts and algorithms presented in this thesis have been implemented in a working software prototype. The PROCEED prototype is based on an industrial platform. It is an extension to the widely used life cycle asset information system Comos. The workflow management functionality of PROCEED has already exceeded the prototype status and has been integrated into the current release of Comos. The concepts and algorithms for controlling development projects have been evaluated using the PROCEED prototype. Process model definitions have been defined, an example project has been created, the defined tasks have been scheduled, executed, and monitored in PROCEED. Furthermore, the dynamic replanning and rescheduling of enacted development processes has been simulated in PROCEED. The example scenario which has been described in various sections of this thesis has been used for this evaluation. For the execution of the three year-long example project, the progression of time has been simulated. By means of this virtual evaluation, the general applicability of the developed approach could be shown.

The next step would be an evaluation of the prototype in a real-world project in a plant engineering company. The circumstance that PROCEED is an extension to the Comos system which is widely used in the plant engineering industries, offers great possibilities for such an evaluation. In a first step, a plant engineering company would use PROCEED to define their design processes and the subprocesses thereof. This process knowledge would be stored in the central Comos database used for all design projects in the company. The second step would be the execution of a small design project using PROCEED for project planning and controlling. This evaluation would provide valuable insights in how project team members accept the new functionalities provided by PROCEED. The evaluation in a real-project could not be performed during the course of this research because plant design projects usually take several years.

Kapitel 11

Conclusion

This chapter summarizes and evaluates the contributions of this thesis. Furthermore, an outlook is provided on how the presented research could be continued.

11.1 Summary

In this thesis, a novel approach for controlling development processes by means of a process management system has been presented. The solution approach can be coarsely divided into two parts. First, the TNT meta-model has been defined to provide the necessary modeling capabilities for process model instances. Second, algorithms and tool functionalities have been developed to support the scheduling and monitoring of tasks in a development process as well as the controlled enactment of change management processes at project runtime.

The TNT meta-model has been presented in Chapter 5. It defines the entities, properties, and relationships required for modeling timed dynamic task nets. The TNT meta-model is based on the DYNAMITE and RESMOD meta-models which were defined for the AHEAD system. The main concepts for modeling dynamic task nets have been adopted in the structural and the behavioral model. Several adaptations have been made to address specific requirements which emerged in the industrial context of the research project. In particular, the resource management capabilities provided by PROCEED differ from those of the AHEAD system in that a project team with subteams and team leaders can be defined, and several resources can be assigned to a single task. The behavioral model of the TNT meta-model defines a new execution state and additional state transitions to enable the skipping of tasks, which is particularly required for the automatic enactment of subprocesses in a development project.

The TNT meta-model extends the DYNAMITE meta-model by three additional models: the timing model, the monitoring model, and the authorization model. These partial models enable the scheduling and monitoring of process model instances, and the access control to the management data, respectively.

The timing model, which has been presented in Section 5.3, introduces individual work calendars for tasks and resources which define the available and used working hours for every date. Timing properties have been defined for tasks, control flows, and resources. They are divided into planning data, time constraints, computed constraint dates, and planned dates. The introduced entities and properties cover all planning aspects of a process model instance with respect to time and resource management. Timing consistency constraints define the consistent states of the management data with respect to the timing property values. They result from a systematic analysis of all dependencies between the timing properties of different tasks, task relationships, and resources. Based on the timing consistency constraints, the consistent definition of a process model instance can be supported by the process management system.

The monitoring model introduces task properties for progress measurement and performance analysis. It has been presented in Section 5.4. First of all, actual dates have been introduced in addition to planned dates. The explicit distinction between planned and actual values enables the detection of deviations of the process performance from the plan. For performance analysis, the degree of completion of a task is defined as well as performance indices of the earned value analysis. A forecasted end time is defined in addition to the planned end time. Monitoring constraints define the enactment states in which the actual performance conforms to the plan. Based on the monitoring constraints, the enactment of process model instances in compliance with time restrictions can be supported the process management system.

The authorization model, which has been presented in Section 5.5, introduces the concept of a permission. Permissions for a project can be individually assigned to users of the system. Authorization rules specify which permissions are actually effective in a given situation. The rights of a user to change the management data of a project depend on his permissions, his task assignments, and his position in the project team. In contrast to access control mechanisms implemented in other academic or commercial process management systems, the authorization model also covers structural changes to a process model instance. The authorization model enables a project-specific tailoring of the access control policy. Depending on the assigned permissions, different management styles can be realized including the hierarchical delegation of tasks and the collaboration of project team members with equal rights. Furthermore, permissions can be assigned to users in a way that enables observation without the right to change the management data.

Process knowledge which can be reused in different development projects can be defined in several ways in PROCEED as described in Chapter 6. In Section 6.1, task types have been introduced which define default property values for their instances. In particular, default values of timing properties represent valuable process knowledge for planning a process model instance. But also the progress measure used to determine the degree of completion of a task instance can be predefined for a task type. Task types can be arranged in a generalization hierarchy where specialized task types inherit the property values of the more general types. There are two ways to define process model definitions in PROCEED. First, process templates can be defined which are copied to a process model instance during planning as described in Section 6.2. Process templates may contain several instances of a task type. They may define control flows and data flows between tasks. Second, workflow templates can be used which are extended process templates incorporating additional information for the automatic enactment of the defined subprocesses. The workflow management capabilities of PROCEED have been described in Section 6.3. A workflow definition which is associated with a workflow template defines control structures like alternative branching and loops. Decision variables, whose values can be set by process participants, are evaluated at process runtime to determine the actual flow of control. The process modeling capabilities provided by PROCEED support the planning and the enactment of process model instances. Process templates and procedural process model definitions, i.e. workflows, have been found suitable and sufficient for the application of PROCEED in industrial practice.

The timing model of the TNT meta-model enables the scheduling of tasks in a process model instance. Since manual scheduling is infeasible in large development projects, PROCEED provides support for automatic schedule generation. The implemented scheduling algorithms have been presented in Chapter 7. Not all tasks in a dynamic task net are necessarily scheduled. So-called zero-duration tasks are excluded from scheduling due to their granularity, duration, or purpose, as described in Section 7.1.

Scheduling is performed in two steps. First, a critical path analysis of the dynamic task net to be scheduled is performed. Afterwards, resource-constrained scheduling is performed to obtain a time- and resource-feasible schedule. The hierarchical critical path method for dynamic task nets has been described in Section 7.2. It computes the earliest and latest possible start and end times of tasks. Running and terminated tasks are treated differently compared to preparing tasks in that their planned dates determine the computed constraint dates. For resource-constrained scheduling, a heuristic algorithm has been developed based on a general parallel scheduling scheme. This heuristic has been described in Section 7.3. The hierarchical structure of dynamic task nets, the presence of simultaneous and standard control flows, and the dynamic computation of task durations together require backtracking during scheduling if planned end times are inconsistent. Despite backtracking, the time complexity of the scheduling algorithm is still polynomial. The constructive heuristic can be used to generate an initial baseline schedule but also to reschedule a dynamic task net at project runtime. In the latter case, the execution states of tasks are taken into account. Terminated tasks are not rescheduled at all and running tasks are not moved to a different start date. A dynamic task net can be scheduled locally. For this purpose, the root task of the subprocess to be scheduled has to be specified. All tasks which are not part of the subprocess defined by the root task remain unchanged during scheduling. It has been shown that the developed algorithms for critical path analysis and resource-constrained scheduling yield constraint dates and planned dates which fulfill all timing consistency constraints defined in the TNT meta-model.

Workflow-managed dynamic task nets are treated in a special way during scheduling. In Section 7.4, it has been described how critical path analysis and resource-constrained scheduling are performed in these cases. Critical path analysis uses the shortest paths through alternative branching and loop constructs to compute the constraint dates of the predecessors and successors of these constructs. Resource-constrained scheduling is initially performed for one alternative path of a branching construct and only one iteration of a loop. At runtime, a workflow-managed dynamic task net is automatically rescheduled upon decisions made by the workflow engine, i.e. when a different alternative is selected or a loop is iterated once again.

Altogether, the scheduling capabilities of PROCEED cover scheduling of hierarchically structured task nets, local rescheduling at process runtime whilst taking task execution states into account, and scheduling of workflow instances. Thereby, PROCEED stands out against related scheduling approaches from research and commercial software packages for project and workflow management.

PROCEED supports process controllers in determining the current project status as described in Chapter 8. An innovative approach has been developed for progress measurement, which integrates several different progress measures in one unified measuring framework. The available progress measures have been described and compared in Section 8.1. The degree of completion can be determined for individual tasks by means of the most appropriate progress measure which represents the best trade-off between measuring effort and accuracy. The common practice to determine the current status of a plant design project based on document states has been integrated with the concept of dynamic task nets in the form of a specific progress measure. The integrated management of tasks and products and the explicit representation of the actual data flow in dynamic task nets enable this way of progress measurement. In contrast, state of the art project management systems do not allow to connect the tasks in a project plan with the technical products. Another specific progress measure has been defined which relies on the timing information available for workflow templates. The degree of completion of a workflow-managed task is derived from the current enactment state of a workflow-managed task net and reference values for the expected durations of the remaining tasks. The progress measure which is used for a task can be predefined for a task type or a task instance in a process template. In this way, the measuring framework can be tailored to the process model definition of a specific development process. Finally, earned value analysis has been applied to compare the actual performance of a running process model instance with the plan as described in Section 8.2.

In addition to progress measurement based on the degrees of completion of tasks and earned value analysis, the project management data can be visualized in a project status analysis view, which has been described in Section 8.3. A flexibly configurable pivot table allows to analyze the inherently multidimensional management data from different perspectives. The history of changes to the planning data which is stored in a project data warehouse is visualized in the form of line diagrams. A user of PROCEED can navigate between the different project monitoring views and the management views to take immediate action when he has identified poor process performance.

All in all, the provided project monitoring functionality goes beyond the state of the art in academia and industrial practice. The integration of planning, enactment, and monitoring enables the comparison of the actual performance with the plan. Common project management systems generally do not support the execution of the defined tasks, and their monitoring capabilities are limited at best. While different progress measures have been promoted, no integrated approach can be found in research or practice which provides a flexible framework for progress measurement which can be tailored to a development process. The application of a data warehouse and visualization techniques for project status analysis has only recently been investigated. In particular, the integration of this functionality into a process management system distinguishes the presented approach from related work.

Controlling of development processes includes besides monitoring also steering the process by taking corrective measures and applying plan changes. As described in Chapter 9, these managerial activities are supported by PROCEED as well. In Section 9.1, it has been described how management processes are explicitly modeled and enacted in PROCEED. Workflow templates can be defined for all sorts of processes in a project including reporting, quality management, and change management processes. Task types for technical tasks can be parameterized, so that specific management workflows are enacted in certain predefined situations. In this way, the controlled enactment of management processes can be tailored to a development process. The explicit modeling and controlled enactment of management processes and their integration with the development process distinguishes PROCEED from related approaches in academia and practice. In related research approaches and commercial solutions, management processes in a development project are either not treated at all or without any relation to the enacted development process.

When disruptions occur at project runtime, plan changes may have to be performed. In Section 9.2, the possible disruptions have been analyzed with respect to whether they require manual plan changes and whether these changes require rescheduling of the project plan. The general change management procedure which has to be followed by authorized users when they perform changes to a dynamic task net has been presented in Section 9.3. It ensures, that the task net is eventually in a consistent state after replanning and rescheduling, in which the planned dates of the tasks represent a time- and resource-feasible schedule. Replanning and rescheduling are performed alternatively and iteratively in order to arrive at a feasible schedule which meets the requirements of the user. The change management procedure takes the execution states of tasks into account. Changes to the plan are ensured to be consistent with the current state of enactment. The support for plan changes at project runtime provided by PROCEED goes beyond the capabilities of common project and workflow management systems. Project management systems do not support the enactment of development processes which is why changes to the project plan may conflict with the process performance. On the other hand, workflow management systems which are commonly applied to enact predefined processes do not cover planning and scheduling and are therefore unsuitable for the management of development projects. Furthermore, the capabilities of commercial WfMS to perform changes to running workflow instances are limited.

The PROCEED prototype has been implemented as an extension to the commercial life cycle asset information system Comos. Several technical details of the implementation and the graphical user interface of PROCEED have been described in Chapter 10. PROCEED complements the functionalities of Comos for maintaining complex engineering data by process management functionality. The realization based on an industrial platform distinguishes PROCEED from other research prototypes of process management systems. In contrast to the AHEAD system, PROCEED has not been realized using a generative tool building framework, but it has been implemented in a conventional way using state-of-the-art technologies and frameworks which are commonly used for commercial applications. The functionalities provided by PROCEED can be used in plant design projects in the chemical industries. The workflow management functionality provided by PROCEED has already exceeded the prototype status and is integrated in the current release of the Comos system.

The approach to project controlling implemented in the PROCEED prototype has been evaluated by modeling a complex process model instance of a plant design project and simulating its enactment. The evaluation showed the applicability of the developed concepts, algorithms, and tool functionalities with respect to planning, scheduling, monitoring, and change management.

11.2 Outlook

The research results presented in this thesis offer some starting points for further research.

A memory representation of a dynamic task net is used for scheduling to avoid changes to the management data in the Comos database before scheduling has been successfully completed. This technique could also be applied to perform a what-if-analysis evaluating different planning scenarios. Using the memory representation, several different schedules of a dynamic task net could be computed for different manual plan changes. The resulting schedules could be used as decision support for choosing the best change operations.

Scheduling of multiple projects which share common resources is a well-known problem and a topic of ongoing research. Multi-project management has not been explicitly addressed in this thesis. The connection between several different projects could be established via the work calendars of resources. When a resource is used in one project, the working hours are not available for tasks in another project anymore. If the scheduling approach implemented in PROCEED would be transferred to the multi-project setting, then the main challenge would be to reschedule multiple parallel projects at runtime whilst taking the execution states of the defined tasks into account. Another challenge would be to adapt the authorization model and the general change management procedure to the multi-project setting.

A complementary problem is the scheduling of process model instances which span across multiple organizations. Solutions for the management of interorganizational cooperation in development processes by means of the AHEAD system have been presented in [Jäg02, Hel08a]. Subprocesses can be delegated to subcontractors who elaborate the details of the delegated tasks. If the overall development process would be rescheduled, then the delegated subprocesses would be affected as well.

However, the manager of the overall process could not see or modify the subtasks defined by the subcontractors for the delegated tasks. This scenario requires a scheduling approach which conforms to dynamic process views and cooperation protocols as they have been introduced in [Hel08a].

With respect to project monitoring, the progress measurement based on documents could be extended to the engineering data in the Comos database which is not contained in documents. For this purpose, the approach of Liefeldt et al. [LGB$^+$05] which has been reviewed in Section 8.4 could be adopted. Devices which are represented as objects in the Comos database could be connected with output parameters of tasks. The degree of completion of such a task could be determined by comparing the properties of defined values in the device specification with the properties required to complete the specification. In contrast to [LGB$^+$05], the aggregation of progress degrees would be performed in the work breakdown structure defined by a dynamic task net instead of the product breakdown structure defined for the chemical plant.

The application domain of this thesis has been the domain of plant engineering. Specific characteristics of plant design processes and the functionalities provided by the Comos system have influenced the developed solutions. The support of development processes in other engineering domains has not been explicitly addressed in this thesis. However, the presented approach could also be applied to software development processes. PROCEED would have to be decoupled from Comos and coupled with a version control system to be used in software development projects. User accounts and documents in the version control system would represent resources and products, respectively. PROCEED also represents an ideal platform for the integration of an issue tracking system as it is commonly used in software development projects. Tickets in an issue tracking system generally represent fine-grained tasks in a software development process. In a nutshell, the presented concepts and algorithms for controlling development processes are not confined to a specific engineering domain but are generally applicable to various types of development processes.

Literaturverzeichnis

[Aal96] W.M.P. van der Aalst. Three Good Reasons for Using a Petri-net-based Workflow Management System. In S. Navathe and T. Wakayama, editors, *Proceedings of the International Working Conference on Information and Process Integration in Enterprises (IPIC'96)*, pages 179–201, Camebridge, Massachusetts, 1996.

[Aal98] W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.

[AC04] Fahad T. Alotaiby and J. X. Chen. A Model for Team-based Access Control (TMAC 2004). In *ITCC '04: Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'04)*, volume 2, pages 450–454, Las Vegas, USA, 2004. IEEE Computer Society.

[AH96] Vijayalakshmi Atluri and Wei-Kuang Huang. An Authorization Model for Workflows. In *Proceedings of the 4th European Symposium on Research in Computer Security*, pages 44–64, Rome, Italy, 1996. Springer Verlag.

[ALM$^+$05] Haldun Aytug, Mark A. Lawley, Kenneth McKay, Shantha Mohan, and Reha Uzsoy. Executing Production Schedules in the Face of Uncertainties: A Review and some Future Directions. *European Journal of Operational Research*, 161(1):86–110, 2005.

[Anb03] Frank T. Anbari. Earned Value Project Management Method and Extensions. *Project Management Journal*, 34(4):12–23, 2003.

[AR00] Christian Artigues and Francois Roubellat. A Polynomial Activity Insertion Algorithm in a Multi-Resource Schedule with Cumulative Cosntraints and Multiple Modes. *European Journal of Operational Research*, 127(2):297–316, 2000.

[ASKP00] Gail-Joon Ahn, Ravi Sandhu, Myong Kang, and Joon Park. Injecting RBAC to Secure a Web-Based Workflow System. In *Proceedings of the 5th ACM Workshop on Role-based Access Control*, pages 1–10, Berlin, Germany, 2000. ACM.

[Auß09] Christoph Außem. Visualisierung multidimensionaler Projektstatusdaten im Anlagenbau. Diploma thesis, RWTH Aachen University, Aachen, Germany, January 2009.

[Bah05] Ali Bahrami. Integrated Process Management: From Planning to Work Execution. In *BSN '05: Proceedings of the IEEE EEE05 International Workshop on Business Services Networks*, pages 11–11, Piscataway, NJ, USA, 2005. IEEE Press.

[Bal00] Helmut Balzert. *Lehrbuch der Software-Technik*. Spektrum, 2000.

[Bau04] Thomas Bauer. Kooperation von Projekt- und Workflow-Management-Systemen. *Informatik Forschung und Entwicklung*, 19(2):74–86, 2004.

[BBMN91] James C. Bean, John R. Birge, John Mittenthal, and Charles E. Noon. Matchup Scheduling with Multiple Resources, Release Dates and Disruptions. *Operations Research*, 39(3):456–469, 1991.

[BGS07] Domenico Bianculli, Carlo Ghezzi, and Paola Spoletini. A Model Checking Approach to Verify BPEL4WS Workflows. In *Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications (SOCA) 2007*, pages 13–20, Newport Beach, USA, 2007.

[BJSW01] S. Becker, D. Jäger, A. Schleicher, and B. Westfechtel. A Delegation Based Model for Distributed Software Process Management. In A. Ambriola, editor, *Software Process Technology: 8th European Workshop, EWSPT 2001*, volume 2077 of *LNCS*, pages 130–144. Springer, 2001.

[BJSW02a] B. Böhlen, D. Jäger, A. Schleicher, and B. Westfechtel. UPGRADE: A Framework for Building Graph-Based Interactive Tools. *Electronic Notes in Theoretical Computer Science*, 72:2:113–123, 2002. Proceedings of the International Workshop on Graph-Based Tools (GraBaTs'02), Barcelona, Spain, October 7–8, 2002.

[BJSW02b] B. Böhlen, D. Jäger, A. Schleicher, and B. Westfechtel. UPGRADE: Building Interactive Tools for Visual Languages. In *Proceedings of the 6th World Multi-Conference On Systemics, Cybernetics and Informatics (SCI 2002), Orlando, Florida, USA*, pages 17–22, 2002.

[Bön99] Sabine Bönig. Erarbeitung sicherheitstechnischer Maßnahmenkataloge zur Entwicklung verfahrenstechnischer Prozesse, Maschinen und Anlagen. *IMW-Institutsmitteilungen Nr. 24*, pages 119–128, 1999.

[Bri08] Christoph Briem. Workflows in dynamischen Entwicklungsprozessen. Diploma thesis, RWTH Aachen University, 2008.

[Buk08] Bruce Bukovics. *Pro WF — Windows Workflow in .NET 3.5*. Apress, 2008.

[Bun06] *V-Modell XT.* Bundesministerium des Innern, 2006.

[Bur00] Manfred Burghardt. *Projektmanagement.* Publicis MCD Verlag, München, 5th edition, 2000.

[Bus98] Christoph Bussler. Workflow Instance Scheduling with Project Management Tools. In *DEXA '98: Proceedings of the 9th International Workshop on Database and Expert Systems Applications*, pages 753–758, Vienna, Austria, 1998. IEEE Computer Society.

[BWE04] Gregorio Baggio, Jacques Wainer, and Clarence Ellis. Applying Scheudling Techniques to Minimize the Number of Late Jobs in Workflow Systems. In *Proceedings of the 2004 ACM symposium on Applied computing*, pages 1396–1403, Nicosia, Cyprus, 2004.

[BWJ02] Claudio Bettini, X. Sean Wang, and Sushil Jajodia. Temporal Reasoning in Workflow Systems. *Distributed and Parallel Databases*, 11(3):269–306, 2002.

[CAD03] Ivica Crnkovic, Ulf Asklund, and Annita Persson Dahlqvist. *Implementing and Integrating Product Data Management and Software Configuration Management.* Artech House, 2003.

[Cas98] Fabio Casati. *Models, Semantics, and Formal Methods for the Design of Workflows and their Exceptions.* PhD thesis, Politecnico di Milano, 1998.

[CC02] Keith Chan and Lawrence Chung. Integrating Process and Project Management for Multi-Site Software Development. *Annals of Software Engineering*, 14(1–4):115–143, 2002.

[CCPP99] Fabio Casati, Stefano Ceri, Stefano Paraboschi, and Guiseppe Pozzi. Specification and Implementation of Exceptions in Workflow Management Systems. *ACM Transactions on Database Systems*, 24(3):405–451, 1999.

[CCS93] E.F Codd, S.B. Codd, and C.T. Salley. Providing OLAP to User-Analysts: An IT Mandate. Technical report, Codd & Date Inc., Michigan, 1993.

[CP06] Carlo Combi and Giuseppe Pozzi. Task Scheduling for a Temporal Workflow Management System. In *Proceedings of the Thriteenth International Symposium on Temporal Representation and Reasoning (TIME'06)*, Budapest, Hungary, 2006.

[CP09] Carlo Combi and Roberto Posenato. Controllability in Temporal Conceptual Workflow Schemata. In *Proceedings of the 7th International Conference on Business Process Management (BPM 2009)*, volume 5701 of *LNCS*, pages 64–79, Ulm, Germany, 2009.

[CSK02] Duk-Ho Chang, Jin Hyun Son, and Myoung Ho Kim. Critical Path Identification in the Context of a Workflow. *Information and Software Technology*, 44(7):405–417, 2002.

[Dau05] Bernhard Daubner. Empowering Software Development Environments by Automatic Software Measurement. In *Proceedings of the 11th IEEE International Symposium on Software Metrics (METRICS 2005)*, Como, Italy, 2005.

[Dau08] Bernhard Daubner. *Conceptual Design and Prototypical Implementation of a Framework for Automated Software Measurement.* PhD thesis, Universität Bayreuth, 2008. in German.

[DAV05] Boudewijn F. van Dongen, Wil M. P. van der Aalst, and Henricus M. W. Verbeek. Verification of EPCs: Using Reduction Rules and Petri Nets. In *Proceedings of the 17th International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 372–386, Porto, Portugal, 2005.

[DBC04] Songer A D., Hays B., and North C. Multidimensional Visualization of Project Control Data. *Construction Innovation: Information, Process, Management*, 4(3):173–190, 2004.

[DCA04] R. Dumke, I. Cote, and O. Andruschak. Statistical Process Control (SPC) - A Metric-Based Point of View of Software Processes Achieving the CMMI Level Four. Technical report, Fakultät für Informatik, Universität Magdeburg, 2004.

[DDHVdV06] Filip Deblaere, Erik Demeulemeester, Willy Herroelen, and Stijn Van de Vonder. Proactive Resource Allocation Heuristics for Robust Project Scheduling. *SSRN eLibrary*, 2006.

[DDO07] Remco M. Dijkman, Marlon Dumas, and Chun Ouyang. Formal Semantics and Analysis of BPMN Process Models. Technical report, Queensland University of Technology, 2007.

[DGB01] A. J. Davenport, C. Gefflot, and J. C. Beck. Slack-Based Techniques for Robust Schedules. In *Constraints and Uncertainty Workshop, Seventh International Conference on Principles and Practice of Constraint Programming*, Paphos, Cyprus, November 2001.

[DGe95] Klaus R. Dittrich, Stella Gatziu, and Andreas Geppert (eds.). The Active Database Management System Manifesto: A Rulebase of ADBMS Features. In *Proceedings of the Second International Workshop on Rules in Database Systems (RIDS 95)*, LNCS 985, pages 3–20, Athens, Greece, 1995. Springer.

[DH02] Erik L. Demeulemeester and Willy S. Herroelen. *Project Scheduling: a Research Handbook*, volume 49 of *International Series in Operations Research & Management Science*. Kluwer Academic Publishers, Boston, 2002.

[DHW06] Bernhard Daubner, Andreas Henrich, and Bernhard Westfechtel. Integrierte Softwaremessung durch Verankerung der Softwaremaße an Elementen des Vorgehensmodells. In Bettina Biel, Matthias Book, and Volker Gruhn, editors, *Software Engineering 2006, Proceedings der Fachtagung des GI-Fachbereichs Softwaretechnik*, pages 157–162, March 2006.

[Dib70] M. L. Dibon. *Ordonnancement et Potentiels - Méthode MPM*. Hermann, Paris, 1970.

[DIN06] *PAS 1059 Processing Plant Design — Procedural Model and Terminology*. DIN Deutsches Institut für Normung e.V., Berlin, 2006.

[DIN09] *Projektmanagement — Netzplantechnik und Projektmanagementsyteme*. Beuth Verlag, 2009.

[Dow91] Mark Dowson. Process and Project Management. In *Proceedings of the Seventh International Software Process Workshop (ISPW '91)*, Yountville, California, USA, 1991. IEEE Computer Society.

[Dre09] Michael Dreher. Terminplanung und Fortschrittskontrolle in dynamischen Entwicklungsprozessen. Diploma thesis, RWTH Aachen University, Aachen, 2009.

[DtH01] Marlon Dumas and Arthur H. M. ter Hofstede. UML Activity Diagrams as a Workflow Specification Language. *Lecture Notes in Computer Science*, 2185:76–90, 2001.

[dVBDH07] Stijn Van de Vonder, Francisco Ballestin, Erik Demeulemeester, and Willy Herroelen. Heuristic Procedures for Reactive Project Scheduling. *Computers and Industrial Engineering*, 52(1):11–28, February 2007.

[DvdAtH05] Marlon Dumas, Wil van der Aalst, and Arthur H. M. ter Hofstede, editors. *Process-Aware Information Systems*. John Wiley & Sons, 2005.

[dVDH08] Stijn Van de Vonder, Erik Demeulemeester, and Willy Herroelen. Proactive Heuristic Procedures for Robust Project Scheduling: An Experimental Analysis. *European Journal of Operational Research*, 189(3):723–733, 2008.

[DWH06] Bernhard Daubner, Bernhard Westfechtel, and Andreas Henrich. Towards Anchoring Software Measures on Elements of the Process

Model. In *Proceedings of the 1st International Conference on Software and Data Technologies (CSOFT 2006)*, pages 232–237, Setï¿½bal, Portugal, September 2006.

[EDL10] EDL Anlagenbau. Company Website. `http://www.edl.poerner.de/organisation1.0.html`, Janurary 2010.

[EGP00] Johann Eder, Wolfgang Gruber, and Euthimios Panagos. Temporal Modeling of Workflows with Conditional Execution Paths. In *Proceedings of the 11th International Conference on Databaes and Expert Systems Applications*, 2000.

[EK92] S. E. E. Elmaghraby and J. Kamburowski. The Analysis of Activity Networks under Generalized Precedence Relations. *Management Science*, 38(9):1245–1263, 1992.

[Ela08] Mohamed Elashri. Project Time Management. `http://www.slideshare.net/m_elashri/project-time-management`, Juni 2008.

[EOG02] Johann Eder, Georg E. Olivotto, and Wolfgang Gruber. A Data Warehouse for Workflow Logs. In *First International Conference on Engineering and Deployment of Cooperative Information Systems (EDCIS)*, pages 1–15, Beijing, China, September 2002.

[EP00] Johann Eder and Euthimios Panagos. *Workflow Handbook 2001*, chapter Managing Time in Workflow Systems, pages 109–132. Future Strategies Inc., 2000.

[EPPR99] Johann Eder, Euthimios Panagos, Heinz Pozewaunig, and Michael Rabinovich. Time Management in Worfklow Systems. In *Proceedings of the 3rd International Conference on Business Information Systems*, Poznan, Poland, 1999.

[EPR99] Johann Eder, Euthimios Panagos, and Michael Rabinovich. Time Constraints in Workflow Systems. In Matthias Jarke and Andreas Oberweis, editors, *Proc. of the 11th Intl. Conf. on Advanced Information Systems Engineering (CAiSE '99)*, volume 1626 of *LNCS*, pages 286–300, Berlin, 1999. Springer.

[FC99] William A. Florac and Anita D. Carleton. *Measuring the Software Process: Statistical Process Control for Software Process Improvement*. Addison-Wesley, Boston, USA, 1999.

[FLW03] C. Foltz, H. Luczak, and B. Westfechtel. Use-Centered Interface Design for an Adaptable Administration System for Chemical Process Design. In *Proceedings of the International Conference on Human-Computer Interaction (HCI International 2003), Crete, Greece*, pages 365–369, 2003.

[For94] Hans-Josef Forst, editor. *Projektmanagement im Anlagenbau*. VDE-Verlag, 1994.

[GB83] Sushil K. Gupta and M. P. Buddhdeo. Progress Measurement During Project Execution. *Engineering Management International*, 1(4):281–285, 1983.

[GDMR04] Michael Gnatz, Martin Deubler, Michael Meisinger, and Andreas Rausch. Towards an Integration of Process Modeling and Project Planning. In *5th International Workshop on Software Process Simulation and Modeling (ProSim 2004)*, Edinburgh, United Kingdom, 2004.

[GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley Longman, Amsterdam, 1994.

[Haj97] M. Hajdu. *Network Scheduling Techniques for Construction Project Management*. Kluwer Academic Publishers, 1997.

[Har05] Scott Harold. Enhanced PDM — Concepts and Benefits. Technical report, Planning Engineers Organisation, Mai 2005.

[Hau01] Gregory T. Haugan. *Effective Work Breakdown Structures*. Management Concepts, 2001.

[HAW10] Thomas Heer, Christoph Außem, and René Wörzberger. Flexible Multi-Dimensional Visualization of Process Enactment Data. In *Business Process Management Workshops of BPM 2009*, volume 43 of *LNBIP*, pages 104–115, Ulm, Germany, 2010. Springer.

[HBW09] Thomas Heer, Christoph Briem, and René Wörzberger. Workflows in Dynamic Development Processes. In *Business Process Management Workshops of BPM 2008*, volume 17 of *LNBIP*, pages 266–277, Milano, Italy, 2009. Springer.

[Hel08a] M. Heller. *Decentralized and View-based Management of Cross-company Development Processes (in German)*. PhD thesis, RWTH Aachen University, Aachen, 2008. 501 pp.

[Hel08b] Frank Peter Helmus. *Process Plant Design - Project Management from Inquiry to Acceptance*. Wiley-VCH, Weinheim, 2008.

[HHM$^+$06] R. Hai, M. Heller, W. Marquardt, M. Nagl, and R. Wörzberger. Workflow Support for Inter-organizational Design Processes. In *9th International Symposium on Process Systems Engineering*, pages 2027–2032, Garmisch-Partenkirchen, Germany, July 2006. Elsevier.

[HHWW10] T. Heer, M. Heller, B. Westfechtel, and R. Wörzberger. Tool Support for Dynamic Development Processes. In *Graph Transformations and Model-Driven Engineering*, volume 5765 of *LNCS*, pages 621–654. Springer, 2010.

[Hir99] Hans Günther Hirschberg. *Handbuch Verfahrenstechnik und Anlagenbau*. Springer, 1999.

[HJ04a] M. Heller and D. Jäger. Graph-Based Tools for Distributed Cooperation in Dynamic Development Processes. In Pfaltz et al. [PNB04], pages 352–368.

[HJ04b] M. Heller and D. Jäger. Interorganizational Management of Development Processes. In Pfaltz et al. [PNB04], pages 427–433.

[HJK$^+$08] M. Heller, D. Jäger, C.-A. Krapp, M. Nagl, A. Schleicher, B. Westfechtel, and R. Wörzberger. An Adaptive and Reactive Management System for Project Coordination. In Nagl and Marquardt [NM08], pages 307–373.

[HJS$^+$04] M. Heller, D. Jäger, M. Schlüter, R. Schneider, and B. Westfechtel. A Management System for Dynamic and Interorganizational Design Processes in Chemical Engineering. *Computers & Chemical Engineering*, 29(1):93–111, 2004.

[HKNW99] P. Heimann, C.-A. Krapp, M. Nagl, and B. Westfechtel. An Adaptable and Reactive Project Management Environment. In Nagl [Nag96], pages 504–534.

[HKW97] P. Heimann, C.-A. Krapp, and B. Westfechtel. An Environment for Managing Software Development Processes. In *Proceedings of the 8th Conference on Software Engineering Environments, Cottbus, Germany*, pages 101–109. IEEE Computer Society Press, 1997.

[HL04] Willy Herroelen and Roel Leus. The Construction of Stable Project Baseline Schedules. *European Journal of Operational Research*, 156(3):550 – 565, 2004.

[HL05] Willy Herroelen and Roel Leus. Project Scheduling under Uncertainty: Survey and Research Potentials. *European Journal of Operational Research*, 165(2):289–306, 2005.

[HN07] Thomas Froese Hao Nie, Sheryl Staub-French. OLAP-Integrated Project Cost Control and Manpower Analysis. *Journal of Computing in Civil Engineering*, 21(3):164–174, 2007.

[HNWH08] M. Heller, M. Nagl, R. Wörzberger, and T. Heer. Dynamic Process Management Based Upon Existing Systems. In Nagl and Marquardt [NM08], pages 733–748.

[HRD98] Willy Herroelen, Bert De Reyck, and Erik Demeulemeester. Resource-Constrained Project Scheduling: A Survey of Recent Developments. *Computers and Operations Research*, 25(4):279–302, 1998.

[HRK07] Thomas Heer, Daniel Retkowitz, and Bodo Kraft. Algorithm and Tool for Ontology Integration Based on Graph Rewriting. In Andy Schürr, Manfred Nagl, and Albert Zündorf, editors, *Applications of Graph Transformations with Industrial Relevance, Third International Symposium (AGTIVE)*, volume 5088 of *LNCS*, pages 484–490, Kassel, Germany, 2007. Springer.

[HRK08] Thomas Heer, Daniel Retkowitz, and Bodo Kraft. Incremental Ontology Integration. In José Cordeiro and Joaquim Filipe, editors, *Proceedings of the 10th International Conference on Enterprise Information Systems (ICEIS)*, page 8 pages, Barcelona, Spain, 2008.

[HRK09] Thomas Heer, Daniel Retkowitz, and Bodo Kraft. Tool Support for the Integration of Light-Weight Ontologies. In *Enterprise Information Systems, 10th International Conference, Revised Selected Papers*, volume 19 of *LNBIP*, pages 175–187. Springer, 2009.

[HSW04a] M. Heller, A. Schleicher, and B. Westfechtel. Graph-Based Specification of a Management System for Evolving Development Processes. In Pfaltz et al. [PNB04].

[HSW04b] M. Heller, A. Schleicher, and B. Westfechtel. Process Evolution Support in the AHEAD System. In Pfaltz et al. [PNB04], pages 454–460.

[HSXW10] Qi Hao, Weiming Shen, Yunjiao Xue, and Shuying Wang. Task Network-Based Project Dynamic Scheduling and Schedule Coordination. *Advanced Engineering Informatics*, 24(4):417–427, 2010.

[HTT09] Thomas Heer, Sven Tackenberg, and Manfred Theissen. Integrated Modeling, Simulation and Enactment of Design Processes in Chemical Engineering. In *8th World Congress of Chemical Engineering (WCCE8)*, Montréal, Canada, August 2009.

[HW06a] M. Heller and R. Wörzberger. Management Support of Interorganizational Cooperative Software Development Processes based on Dynamic Process Views. In *15th International Conference on Software Engineering and Data Engineering (SEDE 2006)*, pages 15–28, Los Angeles, USA, 2006.

[HW06b] Markus Heller and R. Wörzberger. A Management System Supporting Interorganizational Cooperative Development Processes in Chemical Engineering. In *9th World Conference on Integrated Design & Process Technology (IDPT 2006)*, pages 639–650, San Diego, USA, 2006. SDPS.

[HW07] M. Heller and R. Wörzberger. A Management System Supporting Interorganizational Cooperative Development Processes in Chemical Engineering. *Journal of Integrated Design and Process Science: Transactions of the SDPS*, 10(2):57–78, 2007.

[HW09] Thomas Heer and René Wörzberger. Support for Enactment and Monitoring of Engineering Design Processes. In *8th World Congress of Chemical Engineering (WCCE8)*, Montréal, Canada, August 2009.

[HW11] Thomas Heer and René Wörzberger. Support for Modeling and Monitoring of Engineering Design Processes. *Computers and Chemical Engineering*, 2011. accepted for publication.

[IGM04] Wendy K. Ivins, W. Alex Gray, and John C. Miles. Managing Changes to Engineering Products Through the Co-ordination of Human and Technical Activities. In *Proceedings of the Cooperative Information Systems (CoopIS) 2004 International Conference*, LNCS 3290, pages 442–459, Agia Napa, Cyprus, October 2004. Springer.

[ISO01] *Flow diagrams for Process Plants - General Rules (German version EN ISO 10628:2000).* Deutsches Institut für Normung e.V., 2001.

[ISO05] *Quality Management Systems - Fundamentals and Vocabulary (ISO 9000:2005).* Beuth Verlag, 2005.

[Jab95] Stefan Jablonski. *Workflow-Management-Systeme*. International Thomson Publishers, 1995.

[Jäg00] Dirk Jäger. Modeling Management and Coordination in Development Processes. In R. Conradi, editor, *Software Process Technology: 7th European Workshop*, volume 1780 of *LNCS*, pages 109–114, Kaprun, Austria, February 2000. Springer.

[Jäg02] Dirk Jäger. *Unterstützung übergreifender Kooperation in komplexen Entwicklungsprozessen*. PhD thesis, RWTH Aachen University, 2002.

[JB96] Stefan Jablonski and Christoph Bussler. *Workflow Management: Modeling Concepts, Architecture and Implementation*. International Thomson Computer Press, 1996.

[JBR99] Ivar Jacobson, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*. Addison-Wesley, 1999.

[JBS99] Stefan Jablonski, Markus Böhm, and Wolfgang Schulze. *Workflow-Management: Entwicklung von Anwendungen und Systemen*. dpunkt Verlag, 1999.

[JKN+99] D. Jäger, C.-A. Krapp, M. Nagl, A. Schleicher, and B. Westfechtel. Anpassbares Administrationssystem für die Projektkoordination. In M. Nagl and B. Westfechtel, editors, *Integration von Entwicklungssystemen in Ingenieuranwendungen – Substantielle Verbesserung der Entwicklungsprozesse*, pages 311–348. Springer, 1999.

[JLVV00] Matthias Jarke, Maurizio Lenzerini, Yannis Vassiliou, and Panos Vassiliadis. *Fundamentals of Data Warehouses*. Springer, 2000.

[Joe97] Gregor Joeris. Change Management Needs Integrated Process and Configuration Management. In *Proceedings of the 6th European Software Engineering Conference*, LNCS 1301, pages 125–141, Zurich, Switzerland, September 1997.

[JSW00] D. Jäger, A. Schleicher, and B. Westfechtel. AHEAD: A Graph-Based System for Modeling and Managing Development Processes. In M. Nagl, A. Schürr, and M. Münch, editors, *Applications of Graph Transformations with Industrial Relevance: International Workshop, AGTIVE'99, Kerkrade, The Netherlands, September 1999. Proceedings*, volume 1779 of *LNCS*, pages 325–340. Springer, 2000.

[KC04] Ralph Kimball and Joe Caserta. *The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data*. John Wiley & Sons, 2004.

[Kei02] Daniel A. Keim. Information Visualization and Visual Data Mining. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):1–8, 2002.

[Ker98] Harold Kerzner. *Project Management: A Systems Approach to Planning, Scheduling, and Controlling*. Wiley, 6th edition, 1998.

[KH98] R. Kolisch and S. Hartmann. *Project Scheduling: Recent Models, Algorithms and Applications*, chapter Heuristic Algorithms for Solving the Resource-Constrained Project Scheduling Problem: Classification and Computational Analysis, pages 147–178. Kluwer, 1998.

[KK99] Eleana Kafeza and Kamalakar Karlapalem. Temporally Constrained Workflows. In L. C.-K. Hui and D. L. Lee, editors, *Proceedings of the 5th International Computer Science Conference (ICSC'99)*, volume 1749 of *LNCS*, pages 246–255, Hong Kong, China, 1999. Springer.

[KKP+09] G. Karsai, H. Krahn, C. Pinkernell, B. Rumpe, M. Schindler, and S. Völkel. Design Guidelines for Domain Specific Languages. In *Proceedings of the 9th OOPSLA Workshop on Domain-Specific Modeling (DSM' 09)*, Orlando, Florida, USA, October 2009.

[KKSW00] C.-A. Krapp, S. Krüppel, A. Schleicher, and B. Westfechtel. Graph-Based Models for Managing Development Processes, Resources, and Products. In H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors, *Theory and Application of Graph Transformations: 6th International Workshop, TAGT'98, Paderborn, Germany, November 16–20, 1998. Selected Papers*, volume 1764 of *LNCS*, pages 455–474. Springer, 2000.

[Kle00] Robert Klein. *Scheduling of Resource-Constrained Projects*. Kluwer, Boston, 2000.

[KN04] B. Kraft and M. Nagl. Parameterized Specification of Conceptual Design Tools in Civil Engineering. In Pfaltz et al. [PNB04], pages 90–105.

[KNS92] G. Keller, M. Nüttgens, and A. W. Scheer. Semantische Prozessmodellierung auf der Grundlage Ereignisgesteuerter Processketten (EPK). Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft 89 (in German), University of Saarland, Saarbrücken, 1992.

[Kol99] Rainer Kolisch. Resource Allocation Capabilities of Commercial Project Management Software Packages. *Interfaces*, 29(4):19–31, 1999.

[KP01] Rainer Kolisch and Rema Padman. An Integrated Survey of Deterministic Project Scheduling. *OMEGA International Journal of Management Science*, 29(3):249–272, June 2001.

[KPF01] Myong H. Kang, Joon S. Park, and Judith N. Froscher. Access Control Mechanisms for Inter-Organizational Workflow. In *SACMAT '01: Proceedings of the sixth ACM Symposium on Access Control Models and Technologies*, pages 66–74. ACM, 2001.

[Kra98] C.-A. Krapp. *An Adaptable Environment for the Management of Development Processes*. PhD thesis, RWTH Aachen University, Aachen, 1998.

[Krü96] Sven Krüppel. Ein Ressourcenmodell zur Unterstützung von Software-Entwicklungsprozessen. Diploma thesis, RWTH Aachen University, February 1996.

[KS75] J.A.G.M Kerbosh and H.J. Shell. Network Planning by the Extended METRA Potential Method. Technical report, University of Technology Eindhoven, Department of Industrial Engineering, 1975.

[KSSL08] B. Kausch, N. Schneider, C. Schlick, and H. Luczak. Simulation-supported Workflow Optimization in Process Engineering. In Nagl and Marquardt [NM08], pages 666–674.

[KSW95] N. Kiesel, A. Schürr, and B. Westfechtel. GRAS: a Graph-Oriented Software Engineering Database System. *Information Systems*, 20(1):21–51, 1995.

[KW59] James E. Jr. Kelley and Morgan R. Walker. Critical Path Planning and Scheduling. In *Proceedings of the Eastern Joint IRE-AIEE-ACM Computer Conference*, pages 160–173, New York, 1959.

[KW00] S. Krüppel and B. Westfechtel. RESMOD: A Resource Management Model for Development Processes. In H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors, *Theory and Application of Graph Transformations: 6th International Workshop, (TAGT'98)*, volume 1764 of *LNCS*, pages 390–397. Springer, Paderborn, Germany, November 2000.

[Lan00] Wolfgang Langer. *Maschinen- und Anlagenbau*. Galileo Press, Bonn, 2000.

[Law97] Peter Lawrence, editor. *Workflow Handbook*. John Wiley & Sons, 1997.

[Leh03] Wolfgang Lehner. *Datenbanktechnologie für Data-Warehouse-Systeme: Konzepte und Methoden*. dpunkt Verlag, Heidelberg, 2003.

[Leo08] Suet Mooi Leong. Resource Allocation and Access Control in a Dynamic Process Management System. Master's thesis, RWTH Aachen University, 2008.

[LGB$^+$05] Andreas Liefeldt, Georg Gutermuth, Peter Beer, Stefan Basenach, and Richard Alznauer. Effizientes Engineering — Begleitende Fortschrittskontrolle großer Projekte der Automatisierungstechnik. *ATP — Automatisierungstechnische Praxis*, 47(7):60–64, 2005.

[Lib01] Vladimir Liberzon. Resource Critical Path Approach to Project Schedule Management. 4th European Project Management Conference, London, June 2001.

[Lic06] Horst Lichter. Skript: Software-Qualitätssicherung und Projektmanagement, RWTH Aachen University, 2006.

[Lip03] Walt Lipke. Schedule is different. Technical report, Oklahoma City Air Logistics Center, 2003.

[Lon93] Jacques Lonchamp. A Structured Conceptual and Terminological Framework for Software Process Engineering. In *Proceedings of the Second International Conference on the Software Process: Continuous Software Process Improvement*, Berlin, 1993.

[LY04] Hongchen Li and Yun Yang. Verification of Temporal Constraints for Concurrent Workflows. In *Proceedings of the 6th AsiaPacific Web Conference (APWeb 2004)*, volume 3007 of *LNCS*, pages 804–813. Springer, 2004.

[LY05] Hongchen Li and Yun Yang. Dynamic Checking of Temporal Constraints for Concurrent Workflows. *Electronic Commerce Research and Applications*, 4(2):124–142, 2005.

[LYC04] Hongchen Li, Yun Yang, and T.Y. Chen. Resource Constraints Analysis of Workflow Specifications. *The Journal of Systems and Software*, 73(2):271–285, 2004.

[MA07] Jan Mendling and Wil M. P. van der Aalst. Formalization and Verification of EPCs with OR-Joins Based on State and Context. In *Advanced Information Systems Engineering, 19th International Conference, CAiSE 2007*, volume 4495 of *Lecture Notes in Computer Science*, pages 439–453. Springer, 2007.

[Mad00] Bernd J. Madauss. *Handbuch Projektmanagement*. Schäffer-Poeschel Verlag, Stuttgart, 2000.

[Mar06] Wolfgang Marquardt. Prozessentwicklung in der Verfahrenstechnik. Lecture notes, AVT — Process Systems Engineering, RWTH Aachen University, 2006.

[MDB$^+$00] Frank Maurer, Barbara Dellen, Fawsy Bendeck, Sigird Goldmann, Harald Holz, Boris Kötting, and Martin Schaaf. Merging Project Planning and Web-Enabled Dynamic Workflow Technologies. *IEEE Internet Computing*, 4(3):65–74, 2000.

[MH04] Jürgen Münch and Jens Heidrich. Software Project Control Centers: Concepts and Approaches. *The Journal of Systems and Software*, 70(1-2):3–19, 2004.

[Mic10a] Microsoft Corporation. Microsoft Project. `www.microsoft.com/project`, October 2010.

[Mic10b] Microsoft Corporation. Windows Workflow Foundation. `http://msdn.microsoft.com/en-us/netframework/aa663328.aspx`, March 2010.

[Mic11] Microsoft Corporation. SQL Server 2008 Analysis Services. `http://www.microsoft.com/Sqlserver/2008/en/us/analysis-services.aspx`, January 2011.

[MJW08] M. Miatidis, M. Jarke, and K. Weidenhaupt. Using Developers' Experience in Cooperative Design Processes. In Nagl and Marquardt [NM08], pages 185–223.

[MO99] Olivera Marjanovic and Maria E. Orlowska. Time Management in Dynamic Workflows. In *The Proceedings of the Second International Symposium on Cooperative Database Systems for Advanced Applications (CODAS'99)*, pages 138–149, Wollongong, Australia, 1999. Springer.

[MRCF59] D. G. Malcolm, J. H. Rosenboom, C. E. Clark, and W. Fazar. Application of a Technique for Research and Development Program Evaluation. *Operations Research*, 4(5):646–669, 1959.

[MS03] Peter J. Mangan and Shazia W. Sadiq. A Constraint Specification Approach to Building Flexible Workflows. *Journal of Research and Practice in Information Technology*, 35(1):21–39, 2003.

[MT01] Christoph Mellentien and Norbert Trautmann. Resource Allocation with Project Management Software. *OR Spektrum*, 23(3):383–394, 2001.

[Nag90] Manfred Nagl. *Softwaretechnik: Methodisches Programmieren im Großen*. Springer Verlag, March 1990.

[Nag96] M. Nagl, editor. *Building Tightly Integrated Software Development Environments: The IPSEN Approach*, volume 1170 of *LNCS*. Springer, 1996.

[Nat06] Adam Nathan. *Windows Presentation Foundation Unleashed*. Sams Publishing, Indianapolis, USA, 2006.

[NM08] Manfred Nagl and Wolfgang Marquardt, editors. *Collaborative and Distributed Chemical Engineering: From Understanding to Substantial Design Process Support*, volume 4970 of *LNCS*. Springer, Berlin, 2008.

[NW94] Manfred Nagl and Bernhard Westfechtel. A Universal Component for the Administration in Distributed and Integrated Development Environments. Technical report, RWTH Aachen, 1994.

[NW03] M. Nagl and B. Westfechtel, editors. *Modelle, Werkzeuge und Infrastrukturen zur Unterstützung von Entwicklungsprozessen*. Wiley-VHC, 2003.

[NWS03] M. Nagl, B. Westfechtel, and R. Schneider. Tool Support for the Management of Design Processes in Chemical Engineering. *Computers and Chemical Engineering*, 27(2):175–197, 2003.

[ODtHvdA07] Chun Ouyang, Marlon Dumas, A.H.M. ter Hofstede, and W.M.P. van der Aalst. Pattern-Based Translation of BPMN Process Models to BPEL Web Services. *International Journal of Web Services Research*, 5(1):42–62, 2007.

[OP09] Djamila Ouelhadj and Sanja Petrovic. A Survey of Dynamic Scheduling in Manufacturing Systems. *Journal of Scheduling*, 12(4):417–431, August 2009.

[Pal87] Gabriel A. Pall. *Quality Process Management*. Prentice Hall, 1987.

[PEL97] Heinz Pozewaunig, Johann Eder, and Walter Liebhart. ePERT: Extending PERT for Worfklow Management Systems. In *First East-European Symposium on Advances in Database and Information Systems (ADBIS'97)*, pages 217–224, St. Petersburg, Russia, 1997.

[PMI04] PMI — Project Management Institute. *A Guide to the Project Management Body of Knowledge (PMBOK Guide)*. Project Management Institute, 3rd edition, 2004.

[PMI06] PMI — Project Management Institute. *Practice Standard for Work Breakdown Structures*. Project Management Institute, 2006.

[PNB04] John L. Pfaltz, M. Nagl, and B. Bï¿½hlen, editors. *Applications of Graph Transformations with Industrial Relevance: Second International Workshop, AGTIVE 2003, Charlottesville, VA, USA, September 27 – October 1, 2003, Revised Selected and Invited Papers*, volume 3062 of *LNCS*. Springer, 2004.

[Poh96] Klaus Pohl. *Process-Centered Requirements Engineering*. Research Studies Press, Taunton, 1996.

[Poh99] Klaus Pohl. *Continuous Documentation of Information Systems Requirements*. Habilitation, RWTH Aachen University, 1999.

[PR98] Gerold Patzak and Günter Rattay. *Projekt Management*. Linde, 1998.

[PR05] Gerold Patzak and Günter Rattay. *Projekt-Management : Leitfaden zum Management von Projekten, Projektportfolios und projektorientierten Unternehmen*. Linde, Wien, 5th edition, 2005.

[PTW03] Max S. Peters, Klaus D. Timmerhaus, and Ronald E. West. *Plant Design and Economics for Chemical Engineers*. McGraw-Hill, 5th edition, 2003.

[Pu09] Wen Pu. Semi-formal Process Models to Executable Workflows. Master's thesis, RWTH Aachen University, 2009.

[PW08] Lutz Priese and Harro Wimmel. *Theoretische Informatik - Petri Netze*. Springer, 2008.

[PWD$^+$98] K. Pohl, K. Weidenhaupt, R. Dömges, P. Haumer, and M. Jarke. Prozessintegration in PRIME: Modelle, Architektur, Vorgehensweise. In *Proceedings of Softwaretechnik '98*, pages 42–52, Paderborn, 1998.

[PWD$^+$99] Klaus Pohl, Klaus Weidenhaupt, Ralf Dömges, Peter Haumer, Matthias Jarke, and Ralf Klamma. PRIME - Towards Process-Integrated Modeling Environments. *ACM Transactions on Software Engineering and Methodology*, 8(4):343–410, October 1999.

[RD98] Manfred Reichert and Peter Dadam. ADEPTflex — Supporting Dynamic Changes of Workflows Without Losing Control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.

[Rei00] Manfred Reichert. *Dynamische Ablaufänderungen in Workflow-Management-Systemen*. PhD thesis, Universität Ulm, May 2000.

[RH96] Bert De Reyck and Willy Herroelen. A Branch-and-Bound Procedure for the Resource-Constrained Project Scheduling Problem with Generalized Precedence Constraints. Technical report, Operations Management Group, Department of Applied Economics, Katholieke Universiteit Leuven, 1996.

[Rin04] Stefanie Rinderle. *Schema Evolution in Process Management Systems*. PhD thesis, University of Ulm, 2004.

[Rup97] Walter Rupietta. *Organization and Role Models for Workflow Processes*, pages 165–172. In Lawrence [Law97], 1997.

[RvdAtHE05] Nick Russel, Wil M.P. van der Aalst, Arthur H.M. ter Hofstede, and David Edmond. Workflow Resource Patterns: Identification, Representation and Tool Support. In *Proceedings of the 17th Conference on Advanced Information Systems Engineering (CAiSE'05)*, volume 3520 of *LNCS*, pages 216–232, Porto, Portugal, 2005. Springer.

[SCFY96] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-Based Access Control Models. *IEEE Computer*, 29(2):38–47, 1996.

[Sch02] Ansgar Schleicher. *Management of Development Processes - An Evolutionary Approach*. PhD thesis, RWTH Aachen University, Aachen, 2002.

[Sie10] Siemens Industry Software GmbH & Co.KG. Comos Product Website. `http://www.comos.com`, December 2010.

[SKK05] Jin Hyun Son, Jung Sun Kim, and Myoung Ho Kim. Extracting the Workflow Critical Path from the Extended Well-Formed Workflow Schema. *Journal of Computer and System Sciences*, 70:86–106, 2005.

[SKW07] Andrew P. Snow, Mark Keil, and Linda Wallace. The Effects of Optimistic and Pessimistic Biasing on Software Project Status Reporting. *Information & Management*, 44(2):130–141, 2007.

[Smi94] Stephen F. Smith. *Intelligent Scheduling Systems*, chapter Reactive Scheduling Systems. Kluwer Publishing, 1994.

[Smi03] Stephen Smith. Is Scheduling a Solved Problem? In Graham Kendall, Edmund K. Burke, Sanja Petrovic, and Michel Gendreau, editors, *Multidisciplinary Scheduling: Theory and Applications*, pages 3–17. Springer US, 2003.

[SMO00] Shazia W. Sadiq, Olivera Marjanovic, and Maria E. Orlowska. Managing Change and Time in Dynamic Workflow Processes. *International Journal of Cooperative Information Systems*, 9(1-2):93–116, 2000.

[Sof10] Edgewall Software. Trac Project Website. `http://trac.edgewall.org/`, August 2010.

[SOS93] Norman M. Sadeh, Shinichi Otsuka, and Robert Schnelbach. Predictive and Reactive Scheduling with the Micro-Boss Production Scheduling and Control System. In *Proceedings of the IJCAI-93 Workshop on Knowledge-based Production Planning, Scheduling, and Control*, Chambery France, August 1993.

[SS06] Dharma Shukla and Bob Schmidt. *Essential Windows Workflow Foundation*. Microsoft .Net Development Series. Addison-Wesley Professional, 2006.

[STH03] Chris Stolte, Diane Tang, and Pat Hanrahan. Multiscale Visualization Using Data Cubes. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):176–187, 2003.

[SW00] Hani El Sakkout and Mark Wallace. Probe Backtrack for Minimal Perturbation in Dynamic Scheduling. *Constraints*, 5(4):359–388, October 2000.

[SW03] A. Schleicher and B. Westfechtel. Unterstützung von Entwicklungsprozessen durch Werkzeuge. In Nagl and Westfechtel [NW03], pages 329–332.

[SWZ99] A. Schürr, A. Winter, and A. Zündorf. The PROGRES Approach: Language and Environment. In H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors, *Handbook on Graph Grammars and Computing by Graph Transformation – Volume 2: Applications, Languages, and Tools*, pages 478–550. World Scientific, 1999.

[TB09a] N. Trautmann and P. Baumann. Resource-constrained Scheduling of a Real Project from the Construction Industry: A Comparison of Software Packages for Project Management. In *International Conference on Industrial Engineering and Engineering Management (IEEM)*, 2009.

[TB09b] Norbert Trautmann and Philipp Baumann. Resource-Allocation Capabilities of Commercial Project Management Software: an Experimental Analysis. In *Proceedings of the 39th International Conference on Computers & Industrial Engineering*, pages 1143–1148, Troyes, 2009.

[Tec10] Technip. Company Website. `http://www.technip.com`, April 2010.

[TFR05] Daniel E. Turk, Robert B. France, and Bernhard Rumpe. Assumptions Underlying Agile Software-Development Processes. *Journal of Database Management*, 16(4):62–87, 2005.

[THM08] M. Theißen, R. Hai, and W. Marquardt. Computer-Assisted Work Process Modeling in Chemical Engineering. In Nagl and Marquardt [NM08], pages 656–665.

[Tid06] Jenifer Tidwell. *Designing Interfaces*. O'Reilly, Beijing, 2006.

[TS97] Roshan K. Thomas and Ravi S. Sandhu. Task-Based Authorization Controls (TBAC): A Family of Models for Active and Enterprise-Oriented Authorization Management. In *Proceedings of the IFIP Eleventh International Conference on Database Security*, pages 166–181. Chapman & Hall, 1997.

[Tuf86] Edward R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, USA, 1986.

[Ull83] Hans Jürgen Ullrich. *Anlagenbau: Kommunikation, Planung, Management*. Georg Thieme Verlag, Stuttgart, 1983.

[VA00] Henricus M. W. Verbeek and Wil M. P. van der Aalst. Woflan 2.0: A Petri-Net-Based Workflow Diagnosis Tool. In *Proceedings of the 21st International Conference on Application and Theory of Petri Nets (ICATPN)*, volume 1825, pages 475–484, Aarhus, Denmark, June 2000. Springer.

[Vas10] Ventsislava Vasileva Vasileva. Workflow-Unterstützung für Managementprozesse in dynamischen Entwicklungsprozessen. Diploma thesis, RWTH Aachen University, Aachen, 2010.

[VBA01] Henricus M. W. Verbeek, Twan Basten, and Wil M. P. van der Aalst. Diagnosing Workflow Processes using Woflan. *Comput. J.*, 44(4):246–279, 2001.

[vdAtH05] Wil M. P. van der Aalst and Arthur H. M. ter Hofstede. YAWL: yet another workflow language. *Information Systems*, 30(4):245–275, 2005.

[vdAvH02] Wil van der Aalst and Kees van Hee. *Workflow Management. Models, Methods, and Systems*. MIT Press, 2002.

[VDH05] Stijn Van De Vonder, Erik Demeulemeester, and Willy Herroelen. Heuristic procedures for generating stable project baseline schedules. In *Proceedings of Third Euro Conference for Young OR researchers and practitioners (ORP3)*, pages 11–19, 2005.

[Ver04] Henricus M. W. Verbeek. *Verification of WF-Nets*. PhD thesis, Technische Universität Eindhoven, 2004.

[VW98] Gottfried Vossen and Matthias Weske. The WASA Approach to Workflow Management for Scientific Applications. In *Workflow Management Systems and Interoperability*, volume 164, pages 145–164. Springer, 1998.

[Wag03] Walter Wagner. *Planung im Anlagenbau*. Vogel Buchverlag, 2003.

[WAM$^+$07] Ueli Wahli, Vedavyas Avula, Hannah Macleod, Mohamed Saeed, and Anders Vinther. *Business Process Management: Modeling through Monitoring Using WebSphere V6.0.2 Products*. IBM, 1st edition, August 2007.

[Wan05] Juite Wang. Constraint-based Schedule Repair for Product Development Projects with Time-limited Constraints. *International Journal of Production Economics*, 95(3):399–414, 2005.

[WBK03] Jacques Wainer, Paulo Barthelmess, and Akhil Kumar. W-RBAC - A workflow security model incorporating controlled overriding of constraints. *International Journal of Cooperative Information Systems*, 12(3):455–485, 2003.

[WDGW08] Matthias Weidlich, Gero Decker, Alexander Großkopf, and Mathias Weske. BPEL to BPMN: The Myth of a Straight-Forward Mapping. In *On the Move to Meaningful Internet Systems, OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008*, volume 5331 of *Lecture Notes in Computer Science*, pages 265–282. Springer, 2008.

[WEH08] René Wörzberger, Nicolas Ehses, and Thomas Heer. Adding Support for Dynamics Patterns to Static Business Process Management Systems. In Cesare Pautasso and Éric Tanter, editors, *Software Composition*, volume 4954 of *Lecture Notes in Computer Science*, pages 84–91. Springer, 2008.

[Wei06] Ingo Weisemöller. Verteilte Ausführung dynamischer Entwicklungsprozesse in heterogenen Prozessmanagementsystemen. Diploma thesis, RWTH Aachen, 2006.

[Wes95] Bernhard Westfechtel. *Graph-Theoretic Concepts in Computer Science*, volume 903 of *LNCS*, chapter Using programmed graph rewriting for the formal specification of a configuration management system, pages 164–179. Springer, 1995.

[Wes96] B. Westfechtel. A Graph-Based System for Managing Configurations of Engineering Design Documents. *International Journal of Software Engineering & Knowledge Engineering*, 6:4:549–583, 1996.

[Wes99a] Mathias Weske. *Workflow Management Systems: Formal Foundation, Conceptual Design, Implementation Aspects*. Habilitationsschrift, Westfälische Wilhelms-Universität Münster, 1999.

[Wes99b] B. Westfechtel. Graph-Based Product and Process Management in Mechanical Engineering. In H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors, *Handbook on Graph Grammars and Computing by Graph Transformation – Volume 2: Applications, Languages, and Tools*, pages 321–368. World Scientific, 1999.

[Wes99c] B. Westfechtel. *Models and Tools for Managing Development Processes, Habilitation Thesis*, volume 1646 of *LNCS*. Springer, 1999. 418 pp.

[Wes01] B. Westfechtel. Ein graphbasiertes Managementsystem für dynamische Entwicklungsprozesse. *Informatik Forschung und Entwicklung*, 16(3):125–144, 2001.

[WH08] René Wörzberger and Thomas Heer. Process Model Editing Support Using Eclipse Modeling Project Tools. In Peter Friese, Simon Zambrovski, and Frank Zimmermann, editors, *Proceedings of the Second Workshop on MDSD Today*, Berichte aus der Softwaretechnik, pages 81–88. Shaker, 2008.

[WH11] René Wörzberger and Thomas Heer. DYPROTO - Tools for Dynamic Business Processes. *International Journal of Business Process Integration and Management (IJBPIM)*, 2011. accepted for publication.

[WHH07] R. Wörzberger, M. Heller, and F. Hässler. Evaluating Workflow Definition Language Revisions with Graph-Based Tools. *Electronic Communications of the EASST*, 2007. 6th International Workshop on Graph Transformations and Visual Modeling Techniques (GT-VMT'2007), Satellite Event of ETAPS'2007, Braga, Portugal.

[Whi05] Stephen A. White. Using BPMN to Model a BPEL Process. *BPTrends*, 3(3):1–18, 2005.

[Wie81] Jerome D. Wiest. Precedence Diagramming Method: Some Unusual Characteristics and their Implications for Project Managers. *Journal of Operations Management*, 1(3):121–130, February 1981.

[Wik10] Wikipedia. Change Management (Engineering). `http://en.wikipedia.org/wiki/Change_management_%28engineering%29`, November 2010.

[WKH08a] René Wörzberger, Thomas Kurpick, and Thomas Heer. Checking Correctness and Compliance of Integrated Process Models. In *Proceedings of the 10th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 576–583, Los Alamitos, CA, USA, 2008. IEEE Computer Society.

[WKH08b] René Wörzberger, Thomas Kurpick, and Thomas Heer. On Correctness, Compliance and Consistency of Process Models. In *WETICE*, pages 251–252. IEEE Computer Society, 2008.

[Wor95] Workflow Management Coalition. *The Workflow Reference Model*. `http://www.wfmc.org/`, January 1995. Document Number WFMC-TC00-1003, Issue 1.1.

[Wör10] René Wörzberger. *Management dynamischer Geschäftsprozesse auf Basis statischer Prozessmanagementsysteme*. PhD thesis, RWTH Aachen University, Aachen, Germany, 2010.

[WSJH03] B. Westfechtel, A. Schleicher, D. Jäger, and M. Heller. Ein Managementsystem für Entwicklungsprozesse. In Nagl and Westfechtel [NW03], pages 369–370.

[ZBY05] G Zhu, JF Bard, and G Yu. Disruption Management for Resource-constrained Project Scheduling. *Journal of the Operational Research Society*, 56(4):365–381, 2005.

[ZCP01] Hai Zhuge, To-Yat Cheung, and Hung-Keng Pung. A Timed Worfklow Process Model. *Journal of Systems and Software*, 55(3):231–243, 2001.

[ZDDD93] Monte Zweben, Eugene Davis, Brian Daun, and Michael J. Deale. Scheduling and Rescheduling with Iterative Repair. *IEEE Transactions on Systems, Man and Cybernetics*, 23(6):1588–1596, 1993.

[Zha92] J. Zhan. Calendarization of Time Planning in MPM Networks. *Mathematical Methods of Operations Research*, 36(5):423–438, 1992.

[zM99] Michael zur Muehlen. Resource Modeling in Workflow Applications. In *Proceedings of the 1999 Workflow Management Conference (WFM99)*, pages 137–153, 1999.

Lebenslauf

Thomas Heer

Geburtsdatum:	29. Juli 1980
Geburtsort:	Vechta
Geburtsname:	Heer
Staatsangehörigkeit:	deutsch
seit Oktober 2006	Wissenschaftlicher Angestellter am Lehrstuhl für Informatik 3 der RWTH Aachen; Beginn der Promotion
September 2006	Studienabschluss als Diplom-Informatiker (Dipl.-Inform.)
2000 – 2006	Informatikstudium an der RWTH Aachen
1999 – 2000	Zivildienst Malteser Hilfsdienst Vechta
1999	Abitur am St. Thomas Kolleg Vechta

Aachener Informatik-Berichte

This is the list of all technical reports since 1987. To obtain copies of reports please consult

http://aib.informatik.rwth-aachen.de/

or send your request to:

Informatik-Bibliothek, RWTH Aachen, Ahornstr. 55, 52056 Aachen,
Email: `biblio@informatik.rwth-aachen.de`

1987-01 * Fachgruppe Informatik: Jahresbericht 1986

1987-02 * David de Frutos Escrig, Klaus Indermark: Equivalence Relations of Non-Deterministic Ianov-Schemes

1987-03 * Manfred Nagl: A Software Development Environment based on Graph Technology

1987-04 * Claus Lewerentz, Manfred Nagl, Bernhard Westfechtel: On Integration Mechanisms within a Graph-Based Software Development Environment

1987-05 * Reinhard Rinn: Über Eingabeanomalien bei verschiedenen Inferenzmodellen

1987-06 * Werner Damm, Gert Döhmen: Specifying Distributed Computer Architectures in AADL*

1987-07 * Gregor Engels, Claus Lewerentz, Wilhelm Schäfer: Graph Grammar Engineering: A Software Specification Method

1987-08 * Manfred Nagl: Set Theoretic Approaches to Graph Grammars

1987-09 * Claus Lewerentz, Andreas Schürr: Experiences with a Database System for Software Documents

1987-10 * Herbert Klaeren, Klaus Indermark: A New Implementation Technique for Recursive Function Definitions

1987-11 * Rita Loogen: Design of a Parallel Programmable Graph Reduction Machine with Distributed Memory

1987-12 J. Börstler, U. Möncke, R. Wilhelm: Table compression for tree automata

1988-01 * Gabriele Esser, Johannes Rückert, Frank Wagner Gesellschaftliche Aspekte der Informatik

1988-02 * Peter Martini, Otto Spaniol: Token-Passing in High-Speed Backbone Networks for Campus-Wide Environments

1988-03 * Thomas Welzel: Simulation of a Multiple Token Ring Backbone

1988-04 * Peter Martini: Performance Comparison for HSLAN Media Access Protocols

1988-05 * Peter Martini: Performance Analysis of Multiple Token Rings

1988-06 * Andreas Mann, Johannes Rückert, Otto Spaniol: Datenfunknetze

1988-07 * Andreas Mann, Johannes Rückert: Packet Radio Networks for Data Exchange

1988-08 * Andreas Mann, Johannes Rückert: Concurrent Slot Assignment Protocol for Packet Radio Networks

1988-09 * W. Kremer, F. Reichert, J. Rückert, A. Mann: Entwurf einer Netzwerktopologie für ein Mobilfunknetz zur Unterstützung des öffentlichen Straßenverkehrs

1988-10 * Kai Jakobs: Towards User-Friendly Networking
1988-11 * Kai Jakobs: The Directory - Evolution of a Standard
1988-12 * Kai Jakobs: Directory Services in Distributed Systems - A Survey
1988-13 * Martine Schümmer: RS-511, a Protocol for the Plant Floor
1988-14 * U. Quernheim: Satellite Communication Protocols - A Performance Comparison Considering On-Board Processing
1988-15 * Peter Martini, Otto Spaniol, Thomas Welzel: File Transfer in High Speed Token Ring Networks: Performance Evaluation by Approximate Analysis and Simulation
1988-16 * Fachgruppe Informatik: Jahresbericht 1987
1988-17 * Wolfgang Thomas: Automata on Infinite Objects
1988-18 * Michael Sonnenschein: On Petri Nets and Data Flow Graphs
1988-19 * Heiko Vogler: Functional Distribution of the Contextual Analysis in Block-Structured Programming Languages: A Case Study of Tree Transducers
1988-20 * Thomas Welzel: Einsatz des Simulationswerkzeuges QNAP2 zur Leistungsbewertung von Kommunikationsprotokollen
1988-21 * Th. Janning, C. Lewerentz: Integrated Project Team Management in a Software Development Environment
1988-22 * Joost Engelfriet, Heiko Vogler: Modular Tree Transducers
1988-23 * Wolfgang Thomas: Automata and Quantifier Hierarchies
1988-24 * Uschi Heuter: Generalized Definite Tree Languages
1989-01 * Fachgruppe Informatik: Jahresbericht 1988
1989-02 * G. Esser, J. Rückert, F. Wagner (Hrsg.): Gesellschaftliche Aspekte der Informatik
1989-03 * Heiko Vogler: Bottom-Up Computation of Primitive Recursive Tree Functions
1989-04 * Andy Schürr: Introduction to PROGRESS, an Attribute Graph Grammar Based Specification Language
1989-05 J. Börstler: Reuse and Software Development - Problems, Solutions, and Bibliography (in German)
1989-06 * Kai Jakobs: OSI - An Appropriate Basis for Group Communication?
1989-07 * Kai Jakobs: ISO's Directory Proposal - Evolution, Current Status and Future Problems
1989-08 * Bernhard Westfechtel: Extension of a Graph Storage for Software Documents with Primitives for Undo/Redo and Revision Control
1989-09 * Peter Martini: High Speed Local Area Networks - A Tutorial
1989-10 * P. Davids, Th. Welzel: Performance Analysis of DQDB Based on Simulation
1989-11 * Manfred Nagl (Ed.): Abstracts of Talks presented at the WG '89 15th International Workshop on Graphtheoretic Concepts in Computer Science
1989-12 * Peter Martini: The DQDB Protocol - Is it Playing the Game?
1989-13 * Martine Schümmer: CNC/DNC Communication with MAP
1989-14 * Martine Schümmer: Local Area Networks for Manufactoring Environments with hard Real-Time Requirements

1989-15 * M. Schümmer, Th. Welzel, P. Martini: Integration of Field Bus and MAP Networks - Hierarchical Communication Systems in Production Environments

1989-16 * G. Vossen, K.-U. Witt: SUXESS: Towards a Sound Unification of Extensions of the Relational Data Model

1989-17 * J. Derissen, P. Hruschka, M.v.d. Beeck, Th. Janning, M. Nagl: Integrating Structured Analysis and Information Modelling

1989-18 A. Maassen: Programming with Higher Order Functions

1989-19 * Mario Rodriguez-Artalejo, Heiko Vogler: A Narrowing Machine for Syntax Directed BABEL

1989-20 H. Kuchen, R. Loogen, J.J. Moreno Navarro, M. Rodriguez Artalejo: Graph-based Implementation of a Functional Logic Language

1990-01 * Fachgruppe Informatik: Jahresbericht 1989

1990-02 * Vera Jansen, Andreas Potthoff, Wolfgang Thomas, Udo Wermuth: A Short Guide to the AMORE System (Computing Automata, MOnoids and Regular Expressions)

1990-03 * Jerzy Skurczynski: On Three Hierarchies of Weak SkS Formulas

1990-04 R. Loogen: Stack-based Implementation of Narrowing

1990-05 H. Kuchen, A. Wagener: Comparison of Dynamic Load Balancing Strategies

1990-06 * Kai Jakobs, Frank Reichert: Directory Services for Mobile Communication

1990-07 * Kai Jakobs: What's Beyond the Interface - OSI Networks to Support Cooperative Work

1990-08 * Kai Jakobs: Directory Names and Schema - An Evaluation

1990-09 * Ulrich Quernheim, Dieter Kreuer: Das CCITT - Signalisierungssystem Nr. 7 auf Satellitenstrecken; Simulation der Zeichengabestrecke

1990-11 H. Kuchen, R. Loogen, J.J. Moreno Navarro, M. Rodriguez Artalejo: Lazy Narrowing in a Graph Machine

1990-12 * Kai Jakobs, Josef Kaltwasser, Frank Reichert, Otto Spaniol: Der Computer fährt mit

1990-13 * Rudolf Mathar, Andreas Mann: Analyzing a Distributed Slot Assignment Protocol by Markov Chains

1990-14 A. Maassen: Compilerentwicklung in Miranda - ein Praktikum in funktionaler Programmierung (written in german)

1990-15 * Manfred Nagl, Andreas Schürr: A Specification Environment for Graph Grammars

1990-16 A. Schürr: PROGRESS: A VHL-Language Based on Graph Grammars

1990-17 * Marita Möller: Ein Ebenenmodell wissensbasierter Konsultationen - Unterstützung für Wissensakquisition und Erklärungsfähigkeit

1990-18 * Eric Kowalewski: Entwurf und Interpretation einer Sprache zur Beschreibung von Konsultationsphasen in Expertensystemen

1990-20 Y. Ortega Mallen, D. de Frutos Escrig: A Complete Proof System for Timed Observations

1990-21 * Manfred Nagl: Modelling of Software Architectures: Importance, Notions, Experiences

1990-22 H. Fassbender, H. Vogler: A Call-by-need Implementation of Syntax Directed Functional Programming

1991-01	Guenther Geiler (ed.), Fachgruppe Informatik: Jahresbericht 1990
1991-03	B. Steffen, A. Ingolfsdottir: Characteristic Formulae for Processes with Divergence
1991-04	M. Portz: A new class of cryptosystems based on interconnection networks
1991-05	H. Kuchen, G. Geiler: Distributed Applicative Arrays
1991-06 *	Ludwig Staiger: Kolmogorov Complexity and Hausdorff Dimension
1991-07 *	Ludwig Staiger: Syntactic Congruences for w-languages
1991-09 *	Eila Kuikka: A Proposal for a Syntax-Directed Text Processing System
1991-10	K. Gladitz, H. Fassbender, H. Vogler: Compiler-based Implementation of Syntax-Directed Functional Programming
1991-11	R. Loogen, St. Winkler: Dynamic Detection of Determinism in Functional Logic Languages
1991-12 *	K. Indermark, M. Rodriguez Artalejo (Eds.): Granada Workshop on the Integration of Functional and Logic Programming
1991-13 *	Rolf Hager, Wolfgang Kremer: The Adaptive Priority Scheduler: A More Fair Priority Service Discipline
1991-14 *	Andreas Fasbender, Wolfgang Kremer: A New Approximation Algorithm for Tandem Networks with Priority Nodes
1991-15	J. Börstler, A. Zündorf: Revisiting extensions to Modula-2 to support reusability
1991-16	J. Börstler, Th. Janning: Bridging the gap between Requirements Analysis and Design
1991-17	A. Zündorf, A. Schürr: Nondeterministic Control Structures for Graph Rewriting Systems
1991-18 *	Matthias Jarke, John Mylopoulos, Joachim W. Schmidt, Yannis Vassiliou: DAIDA: An Environment for Evolving Information Systems
1991-19	M. Jeusfeld, M. Jarke: From Relational to Object-Oriented Integrity Simplification
1991-20	G. Hogen, A. Kindler, R. Loogen: Automatic Parallelization of Lazy Functional Programs
1991-21 *	Prof. Dr. rer. nat. Otto Spaniol: ODP (Open Distributed Processing): Yet another Viewpoint
1991-22	H. Kuchen, F. Lücking, H. Stoltze: The Topology Description Language TDL
1991-23	S. Graf, B. Steffen: Compositional Minimization of Finite State Systems
1991-24	R. Cleaveland, J. Parrow, B. Steffen: The Concurrency Workbench: A Semantics Based Tool for the Verification of Concurrent Systems
1991-25 *	Rudolf Mathar, Jürgen Mattfeldt: Optimal Transmission Ranges for Mobile Communication in Linear Multihop Packet Radio Networks
1991-26	M. Jeusfeld, M. Staudt: Query Optimization in Deductive Object Bases
1991-27	J. Knoop, B. Steffen: The Interprocedural Coincidence Theorem
1991-28	J. Knoop, B. Steffen: Unifying Strength Reduction and Semantic Code Motion
1991-30	T. Margaria: First-Order theories for the verification of complex FSMs
1991-31	B. Steffen: Generating Data Flow Analysis Algorithms from Modal Specifications
1992-01	Stefan Eherer (ed.), Fachgruppe Informatik: Jahresbericht 1991

1992-02 * Bernhard Westfechtel: Basismechanismen zur Datenverwaltung in strukturbezogenen Hypertextsystemen

1992-04 S. A. Smolka, B. Steffen: Priority as Extremal Probability

1992-05 * Matthias Jarke, Carlos Maltzahn, Thomas Rose: Sharing Processes: Team Coordination in Design Repositories

1992-06 O. Burkart, B. Steffen: Model Checking for Context-Free Processes

1992-07 * Matthias Jarke, Klaus Pohl: Information Systems Quality and Quality Information Systems

1992-08 * Rudolf Mathar, Jürgen Mattfeldt: Analyzing Routing Strategy NFP in Multihop Packet Radio Networks on a Line

1992-09 * Alfons Kemper, Guido Moerkotte: Grundlagen objektorientierter Datenbanksysteme

1992-10 Matthias Jarke, Manfred Jeusfeld, Andreas Miethsam, Michael Gocek: Towards a logic-based reconstruction of software configuration management

1992-11 Werner Hans: A Complete Indexing Scheme for WAM-based Abstract Machines

1992-12 W. Hans, R. Loogen, St. Winkler: On the Interaction of Lazy Evaluation and Backtracking

1992-13 * Matthias Jarke, Thomas Rose: Specification Management with CAD

1992-14 Th. Noll, H. Vogler: Top-down Parsing with Simultaneous Evaluation on Noncircular Attribute Grammars

1992-15 A. Schuerr, B. Westfechtel: Graphgrammatiken und Graphersetzungssysteme(written in german)

1992-16 * Graduiertenkolleg Informatik und Technik (Hrsg.): Forschungsprojekte des Graduiertenkollegs Informatik und Technik

1992-17 M. Jarke (ed.): ConceptBase V3.1 User Manual

1992-18 * Clarence A. Ellis, Matthias Jarke (Eds.): Distributed Cooperation in Integrated Information Systems - Proceedings of the Third International Workshop on Intelligent and Cooperative Information Systems

1992-19-00 H. Kuchen, R. Loogen (eds.): Proceedings of the 4th Int. Workshop on the Parallel Implementation of Functional Languages

1992-19-01 G. Hogen, R. Loogen: PASTEL - A Parallel Stack-Based Implementation of Eager Functional Programs with Lazy Data Structures (Extended Abstract)

1992-19-02 H. Kuchen, K. Gladitz: Implementing Bags on a Shared Memory MIMD-Machine

1992-19-03 C. Rathsack, S.B. Scholz: LISA - A Lazy Interpreter for a Full-Fledged Lambda-Calculus

1992-19-04 T.A. Bratvold: Determining Useful Parallelism in Higher Order Functions

1992-19-05 S. Kahrs: Polymorphic Type Checking by Interpretation of Code

1992-19-06 M. Chakravarty, M. Köhler: Equational Constraints, Residuation, and the Parallel JUMP-Machine

1992-19-07 J. Seward: Polymorphic Strictness Analysis using Frontiers (Draft Version)

1992-19-08 D. Gärtner, A. Kimms, W. Kluge: pi-Redˆ+ - A Compiling Graph-Reduction System for a Full Fledged Lambda-Calculus

1992-19-09 D. Howe, G. Burn: Experiments with strict STG code

1992-19-10 J. Glauert: Parallel Implementation of Functional Languages Using Small Processes

1992-19-11 M. Joy, T. Axford: A Parallel Graph Reduction Machine

1992-19-12 A. Bennett, P. Kelly: Simulation of Multicache Parallel Reduction

1992-19-13 K. Langendoen, D.J. Agterkamp: Cache Behaviour of Lazy Functional Programs (Working Paper)

1992-19-14 K. Hammond, S. Peyton Jones: Profiling scheduling strategies on the GRIP parallel reducer

1992-19-15 S. Mintchev: Using Strictness Information in the STG-machine

1992-19-16 D. Rushall: An Attribute Grammar Evaluator in Haskell

1992-19-17 J. Wild, H. Glaser, P. Hartel: Statistics on storage management in a lazy functional language implementation

1992-19-18 W.S. Martins: Parallel Implementations of Functional Languages

1992-19-19 D. Lester: Distributed Garbage Collection of Cyclic Structures (Draft version)

1992-19-20 J.C. Glas, R.F.H. Hofman, W.G. Vree: Parallelization of Branch-and-Bound Algorithms in a Functional Programming Environment

1992-19-21 S. Hwang, D. Rushall: The nu-STG machine: a parallelized Spineless Tagless Graph Reduction Machine in a distributed memory architecture (Draft version)

1992-19-22 G. Burn, D. Le Metayer: Cps-Translation and the Correctness of Optimising Compilers

1992-19-23 S.L. Peyton Jones, P. Wadler: Imperative functional programming (Brief summary)

1992-19-24 W. Damm, F. Liu, Th. Peikenkamp: Evaluation and Parallelization of Functions in Functional + Logic Languages (abstract)

1992-19-25 M. Kesseler: Communication Issues Regarding Parallel Functional Graph Rewriting

1992-19-26 Th. Peikenkamp: Charakterizing and representing neededness in functional loginc languages (abstract)

1992-19-27 H. Doerr: Monitoring with Graph-Grammars as formal operational Models

1992-19-28 J. van Groningen: Some implementation aspects of Concurrent Clean on distributed memory architectures

1992-19-29 G. Ostheimer: Load Bounding for Implicit Parallelism (abstract)

1992-20 H. Kuchen, F.J. Lopez Fraguas, J.J. Moreno Navarro, M. Rodriguez Artalejo: Implementing Disequality in a Lazy Functional Logic Language

1992-21 H. Kuchen, F.J. Lopez Fraguas: Result Directed Computing in a Functional Logic Language

1992-22 H. Kuchen, J.J. Moreno Navarro, M.V. Hermenegildo: Independent AND-Parallel Narrowing

1992-23 T. Margaria, B. Steffen: Distinguishing Formulas for Free

1992-24 K. Pohl: The Three Dimensions of Requirements Engineering

1992-25 * R. Stainov: A Dynamic Configuration Facility for Multimedia Communications

1992-26 * Michael von der Beeck: Integration of Structured Analysis and Timed Statecharts for Real-Time and Concurrency Specification

1992-27	W. Hans, St. Winkler: Aliasing and Groundness Analysis of Logic Programs through Abstract Interpretation and its Safety
1992-28 *	Gerhard Steinke, Matthias Jarke: Support for Security Modeling in Information Systems Design
1992-29	B. Schinzel: Warum Frauenforschung in Naturwissenschaft und Technik
1992-30	A. Kemper, G. Moerkotte, K. Peithner: Object-Orientation Axiomatised by Dynamic Logic
1992-32 *	Bernd Heinrichs, Kai Jakobs: Timer Handling in High-Performance Transport Systems
1992-33 *	B. Heinrichs, K. Jakobs, K. Lenßen, W. Reinhardt, A. Spinner: Euro-Bridge: Communication Services for Multimedia Applications
1992-34	C. Gerlhof, A. Kemper, Ch. Kilger, G. Moerkotte: Partition-Based Clustering in Object Bases: From Theory to Practice
1992-35	J. Börstler: Feature-Oriented Classification and Reuse in IPSEN
1992-36	M. Jarke, J. Bubenko, C. Rolland, A. Sutcliffe, Y. Vassiliou: Theories Underlying Requirements Engineering: An Overview of NATURE at Genesis
1992-37 *	K. Pohl, M. Jarke: Quality Information Systems: Repository Support for Evolving Process Models
1992-38	A. Zuendorf: Implementation of the imperative / rule based language PROGRES
1992-39	P. Koch: Intelligentes Backtracking bei der Auswertung funktional-logischer Programme
1992-40 *	Rudolf Mathar, Jürgen Mattfeldt: Channel Assignment in Cellular Radio Networks
1992-41 *	Gerhard Friedrich, Wolfgang Neidl: Constructive Utility in Model-Based Diagnosis Repair Systems
1992-42 *	P. S. Chen, R. Hennicker, M. Jarke: On the Retrieval of Reusable Software Components
1992-43	W. Hans, St.Winkler: Abstract Interpretation of Functional Logic Languages
1992-44	N. Kiesel, A. Schuerr, B. Westfechtel: Design and Evaluation of GRAS, a Graph-Oriented Database System for Engineering Applications
1993-01 *	Fachgruppe Informatik: Jahresbericht 1992
1993-02 *	Patrick Shicheng Chen: On Inference Rules of Logic-Based Information Retrieval Systems
1993-03	G. Hogen, R. Loogen: A New Stack Technique for the Management of Runtime Structures in Distributed Environments
1993-05	A. Zündorf: A Heuristic for the Subgraph Isomorphism Problem in Executing PROGRES
1993-06	A. Kemper, D. Kossmann: Adaptable Pointer Swizzling Strategies in Object Bases: Design, Realization, and Quantitative Analysis
1993-07 *	Graduiertenkolleg Informatik und Technik (Hrsg.): Graduiertenkolleg Informatik und Technik
1993-08 *	Matthias Berger: k-Coloring Vertices using a Neural Network with Convergence to Valid Solutions
1993-09	M. Buchheit, M. Jeusfeld, W. Nutt, M. Staudt: Subsumption between Queries to Object-Oriented Databases

1993-10 O. Burkart, B. Steffen: Pushdown Processes: Parallel Composition and Model Checking
1993-11 * R. Große-Wienker, O. Hermanns, D. Menzenbach, A. Pollacks, S. Repetzki, J. Schwartz, K. Sonnenschein, B. Westfechtel: Das SUKITS-Projekt: A-posteriori-Integration heterogener CIM-Anwendungssysteme
1993-12 * Rudolf Mathar, Jürgen Mattfeldt: On the Distribution of Cumulated Interference Power in Rayleigh Fading Channels
1993-13 O. Maler, L. Staiger: On Syntactic Congruences for omega-languages
1993-14 M. Jarke, St. Eherer, R. Gallersdoerfer, M. Jeusfeld, M. Staudt: ConceptBase - A Deductive Object Base Manager
1993-15 M. Staudt, H.W. Nissen, M.A. Jeusfeld: Query by Class, Rule and Concept
1993-16 * M. Jarke, K. Pohl, St. Jacobs et al.: Requirements Engineering: An Integrated View of Representation Process and Domain
1993-17 * M. Jarke, K. Pohl: Establishing Vision in Context: Towards a Model of Requirements Processes
1993-18 W. Hans, H. Kuchen, St. Winkler: Full Indexing for Lazy Narrowing
1993-19 W. Hans, J.J. Ruz, F. Saenz, St. Winkler: A VHDL Specification of a Shared Memory Parallel Machine for Babel
1993-20 * K. Finke, M. Jarke, P. Szczurko, R. Soltysiak: Quality Management for Expert Systems in Process Control
1993-21 M. Jarke, M.A. Jeusfeld, P. Szczurko: Three Aspects of Intelligent Cooperation in the Quality Cycle
1994-01 Margit Generet, Sven Martin (eds.), Fachgruppe Informatik: Jahresbericht 1993
1994-02 M. Lefering: Development of Incremental Integration Tools Using Formal Specifications
1994-03 * P. Constantopoulos, M. Jarke, J. Mylopoulos, Y. Vassiliou: The Software Information Base: A Server for Reuse
1994-04 * Rolf Hager, Rudolf Mathar, Jürgen Mattfeldt: Intelligent Cruise Control and Reliable Communication of Mobile Stations
1994-05 * Rolf Hager, Peter Hermesmann, Michael Portz: Feasibility of Authentication Procedures within Advanced Transport Telematics
1994-06 * Claudia Popien, Bernd Meyer, Axel Kuepper: A Formal Approach to Service Import in ODP Trader Federations
1994-07 P. Peters, P. Szczurko: Integrating Models of Quality Management Methods by an Object-Oriented Repository
1994-08 * Manfred Nagl, Bernhard Westfechtel: A Universal Component for the Administration in Distributed and Integrated Development Environments
1994-09 * Patrick Horster, Holger Petersen: Signatur- und Authentifikationsverfahren auf der Basis des diskreten Logarithmusproblems
1994-11 A. Schürr: PROGRES, A Visual Language and Environment for PROgramming with Graph REwrite Systems
1994-12 A. Schürr: Specification of Graph Translators with Triple Graph Grammars
1994-13 A. Schürr: Logic Based Programmed Structure Rewriting Systems
1994-14 L. Staiger: Codes, Simplifying Words, and Open Set Condition

1994-15 *	Bernhard Westfechtel: A Graph-Based System for Managing Configurations of Engineering Design Documents
1994-16	P. Klein: Designing Software with Modula-3
1994-17	I. Litovsky, L. Staiger: Finite acceptance of infinite words
1994-18	G. Hogen, R. Loogen: Parallel Functional Implementations: Graphbased vs. Stackbased Reduction
1994-19	M. Jeusfeld, U. Johnen: An Executable Meta Model for Re-Engineering of Database Schemas
1994-20 *	R. Gallersdörfer, M. Jarke, K. Klabunde: Intelligent Networks as a Data Intensive Application (INDIA)
1994-21	M. Mohnen: Proving the Correctness of the Static Link Technique Using Evolving Algebras
1994-22	H. Fernau, L. Staiger: Valuations and Unambiguity of Languages, with Applications to Fractal Geometry
1994-24 *	M. Jarke, K. Pohl, R. Dömges, St. Jacobs, H. W. Nissen: Requirements Information Management: The NATURE Approach
1994-25 *	M. Jarke, K. Pohl, C. Rolland, J.-R. Schmitt: Experience-Based Method Evaluation and Improvement: A Process Modeling Approach
1994-26 *	St. Jacobs, St. Kethers: Improving Communication and Decision Making within Quality Function Deployment
1994-27 *	M. Jarke, H. W. Nissen, K. Pohl: Tool Integration in Evolving Information Systems Environments
1994-28	O. Burkart, D. Caucal, B. Steffen: An Elementary Bisimulation Decision Procedure for Arbitrary Context-Free Processes
1995-01 *	Fachgruppe Informatik: Jahresbericht 1994
1995-02	Andy Schürr, Andreas J. Winter, Albert Zündorf: Graph Grammar Engineering with PROGRES
1995-03	Ludwig Staiger: A Tight Upper Bound on Kolmogorov Complexity by Hausdorff Dimension and Uniformly Optimal Prediction
1995-04	Birgitta König-Ries, Sven Helmer, Guido Moerkotte: An experimental study on the complexity of left-deep join ordering problems for cyclic queries
1995-05	Sophie Cluet, Guido Moerkotte: Efficient Evaluation of Aggregates on Bulk Types
1995-06	Sophie Cluet, Guido Moerkotte: Nested Queries in Object Bases
1995-07	Sophie Cluet, Guido Moerkotte: Query Optimization Techniques Exploiting Class Hierarchies
1995-08	Markus Mohnen: Efficient Compile-Time Garbage Collection for Arbitrary Data Structures
1995-09	Markus Mohnen: Functional Specification of Imperative Programs: An Alternative Point of View of Functional Languages
1995-10	Rainer Gallersdörfer, Matthias Nicola: Improving Performance in Replicated Databases through Relaxed Coherency
1995-11 *	M.Staudt, K.von Thadden: Subsumption Checking in Knowledge Bases
1995-12 *	G.V.Zemanek, H.W.Nissen, H.Hubert, M.Jarke: Requirements Analysis from Multiple Perspectives: Experiences with Conceptual Modeling Technology

1995-13 * M.Staudt, M.Jarke: Incremental Maintenance of Externally Materialized Views

1995-14 * P.Peters, P.Szczurko, M.Jeusfeld: Oriented Information Management: Conceptual Models at Work

1995-15 * Matthias Jarke, Sudha Ram (Hrsg.): WITS 95 Proceedings of the 5th Annual Workshop on Information Technologies and Systems

1995-16 * W.Hans, St.Winkler, F.Saenz: Distributed Execution in Functional Logic Programming

1996-01 * Jahresbericht 1995

1996-02 Michael Hanus, Christian Prehofer: Higher-Order Narrowing with Definitional Trees

1996-03 * W.Scheufele, G.Moerkotte: Optimal Ordering of Selections and Joins in Acyclic Queries with Expensive Predicates

1996-04 Klaus Pohl: PRO-ART: Enabling Requirements Pre-Traceability

1996-05 Klaus Pohl: Requirements Engineering: An Overview

1996-06 * M.Jarke, W.Marquardt: Design and Evaluation of Computer–Aided Process Modelling Tools

1996-07 Olaf Chitil: The Sigma-Semantics: A Comprehensive Semantics for Functional Programs

1996-08 * S.Sripada: On Entropy and the Limitations of the Second Law of Thermodynamics

1996-09 Michael Hanus (Ed.): Proceedings of the Poster Session of ALP96 - Fifth International Conference on Algebraic and Logic Programming

1996-09-0 Michael Hanus (Ed.): Proceedings of the Poster Session of ALP 96 - Fifth International Conference on Algebraic and Logic Programming: Introduction and table of contents

1996-09-1 Ilies Alouini: An Implementation of Conditional Concurrent Rewriting on Distributed Memory Machines

1996-09-2 Olivier Danvy, Karoline Malmkjær: On the Idempotence of the CPS Transformation

1996-09-3 Víctor M. Gulías, José L. Freire: Concurrent Programming in Haskell

1996-09-4 Sébastien Limet, Pierre Réty: On Decidability of Unifiability Modulo Rewrite Systems

1996-09-5 Alexandre Tessier: Declarative Debugging in Constraint Logic Programming

1996-10 Reidar Conradi, Bernhard Westfechtel: Version Models for Software Configuration Management

1996-11 * C.Weise, D.Lenzkes: A Fast Decision Algorithm for Timed Refinement

1996-12 * R.Dömges, K.Pohl, M.Jarke, B.Lohmann, W.Marquardt: PRO-ART/CE* — An Environment for Managing the Evolution of Chemical Process Simulation Models

1996-13 * K.Pohl, R.Klamma, K.Weidenhaupt, R.Dömges, P.Haumer, M.Jarke: A Framework for Process-Integrated Tools

1996-14 * R.Gallersdörfer, K.Klabunde, A.Stolz, M.Eßmajor: INDIA — Intelligent Networks as a Data Intensive Application, Final Project Report, June 1996

1996-15 * H.Schimpe, M.Staudt: VAREX: An Environment for Validating and Refining Rule Bases

1996-16 * M.Jarke, M.Gebhardt, S.Jacobs, H.Nissen: Conflict Analysis Across Heterogeneous Viewpoints: Formalization and Visualization
1996-17 Manfred A. Jeusfeld, Tung X. Bui: Decision Support Components on the Internet
1996-18 Manfred A. Jeusfeld, Mike Papazoglou: Information Brokering: Design, Search and Transformation
1996-19 * P.Peters, M.Jarke: Simulating the impact of information flows in networked organizations
1996-20 Matthias Jarke, Peter Peters, Manfred A. Jeusfeld: Model-driven planning and design of cooperative information systems
1996-21 * G.de Michelis, E.Dubois, M.Jarke, F.Matthes, J.Mylopoulos, K.Pohl, J.Schmidt, C.Woo, E.Yu: Cooperative information systems: a manifesto
1996-22 * S.Jacobs, M.Gebhardt, S.Kethers, W.Rzasa: Filling HTML forms simultaneously: CoWeb architecture and functionality
1996-23 * M.Gebhardt, S.Jacobs: Conflict Management in Design
1997-01 Michael Hanus, Frank Zartmann (eds.): Jahresbericht 1996
1997-02 Johannes Faassen: Using full parallel Boltzmann Machines for Optimization
1997-03 Andreas Winter, Andy Schürr: Modules and Updatable Graph Views for PROgrammed Graph REwriting Systems
1997-04 Markus Mohnen, Stefan Tobies: Implementing Context Patterns in the Glasgow Haskell Compiler
1997-05 * S.Gruner: Schemakorrespondenzaxiome unterstützen die paargrammatische Spezifikation inkrementeller Integrationswerkzeuge
1997-06 Matthias Nicola, Matthias Jarke: Design and Evaluation of Wireless Health Care Information Systems in Developing Countries
1997-07 Petra Hofstedt: Taskparallele Skelette für irregulär strukturierte Probleme in deklarativen Sprachen
1997-08 Dorothea Blostein, Andy Schürr: Computing with Graphs and Graph Rewriting
1997-09 Carl-Arndt Krapp, Bernhard Westfechtel: Feedback Handling in Dynamic Task Nets
1997-10 Matthias Nicola, Matthias Jarke: Integrating Replication and Communication in Performance Models of Distributed Databases
1997-11 * R. Klamma, P. Peters, M. Jarke: Workflow Support for Failure Management in Federated Organizations
1997-13 Markus Mohnen: Optimising the Memory Management of Higher-Order Functional Programs
1997-14 Roland Baumann: Client/Server Distribution in a Structure-Oriented Database Management System
1997-15 George Botorog: High-Level Parallel Programming and the Efficient Implementation of Numerical Algorithms
1998-01 * Fachgruppe Informatik: Jahresbericht 1997
1998-02 Stefan Gruner, Manfred Nagel, Andy Schürr: Fine-grained and Structure-Oriented Document Integration Tools are Needed for Development Processes

1998-03	Stefan Gruner: Einige Anmerkungen zur graphgrammatischen Spezifikation von Integrationswerkzeugen nach Westfechtel, Janning, Lefering und Schürr
1998-04 *	O. Kubitz: Mobile Robots in Dynamic Environments
1998-05	Martin Leucker, Stephan Tobies: Truth - A Verification Platform for Distributed Systems
1998-06 *	Matthias Oliver Berger: DECT in the Factory of the Future
1998-07	M. Arnold, M. Erdmann, M. Glinz, P. Haumer, R. Knoll, B. Paech, K. Pohl, J. Ryser, R. Studer, K. Weidenhaupt: Survey on the Scenario Use in Twelve Selected Industrial Projects
1998-09 *	Th. Lehmann: Geometrische Ausrichtung medizinischer Bilder am Beispiel intraoraler Radiographien
1998-10 *	M. Nicola, M. Jarke: Performance Modeling of Distributed and Replicated Databases
1998-11 *	Ansgar Schleicher, Bernhard Westfechtel, Dirk Jäger: Modeling Dynamic Software Processes in UML
1998-12 *	W. Appelt, M. Jarke: Interoperable Tools for Cooperation Support using the World Wide Web
1998-13	Klaus Indermark: Semantik rekursiver Funktionsdefinitionen mit Striktheitsinformation
1999-01 *	Jahresbericht 1998
1999-02 *	F. Huch: Verifcation of Erlang Programs using Abstract Interpretation and Model Checking — Extended Version
1999-03 *	R. Gallersdörfer, M. Jarke, M. Nicola: The ADR Replication Manager
1999-04	María Alpuente, Michael Hanus, Salvador Lucas, Germán Vidal: Specialization of Functional Logic Programs Based on Needed Narrowing
1999-05 *	W. Thomas (Ed.): DLT 99 - Developments in Language Theory Fourth International Conference
1999-06 *	Kai Jakobs, Klaus-Dieter Kleefeld: Informationssysteme für die angewandte historische Geographie
1999-07	Thomas Wilke: CTL+ is exponentially more succinct than CTL
1999-08	Oliver Matz: Dot-Depth and Monadic Quantifier Alternation over Pictures
2000-01 *	Jahresbericht 1999
2000-02	Jens Vöge, Marcin Jurdzinski A Discrete Strategy Improvement Algorithm for Solving Parity Games
2000-03	D. Jäger, A. Schleicher, B. Westfechtel: UPGRADE: A Framework for Building Graph-Based Software Engineering Tools
2000-04	Andreas Becks, Stefan Sklorz, Matthias Jarke: Exploring the Semantic Structure of Technical Document Collections: A Cooperative Systems Approach
2000-05	Mareike Schoop: Cooperative Document Management
2000-06	Mareike Schoop, Christoph Quix (eds.): Proceedings of the Fifth International Workshop on the Language-Action Perspective on Communication Modelling
2000-07 *	Markus Mohnen, Pieter Koopman (Eds.): Proceedings of the 12th International Workshop of Functional Languages

2000-08	Thomas Arts, Thomas Noll: Verifying Generic Erlang Client-Server Implementations
2001-01 *	Jahresbericht 2000
2001-02	Benedikt Bollig, Martin Leucker: Deciding LTL over Mazurkiewicz Traces
2001-03	Thierry Cachat: The power of one-letter rational languages
2001-04	Benedikt Bollig, Martin Leucker, Michael Weber: Local Parallel Model Checking for the Alternation Free mu-Calculus
2001-05	Benedikt Bollig, Martin Leucker, Thomas Noll: Regular MSC Languages
2001-06	Achim Blumensath: Prefix-Recognisable Graphs and Monadic Second-Order Logic
2001-07	Martin Grohe, Stefan Wöhrle: An Existential Locality Theorem
2001-08	Mareike Schoop, James Taylor (eds.): Proceedings of the Sixth International Workshop on the Language-Action Perspective on Communication Modelling
2001-09	Thomas Arts, Jürgen Giesl: A collection of examples for termination of term rewriting using dependency pairs
2001-10	Achim Blumensath: Axiomatising Tree-interpretable Structures
2001-11	Klaus Indermark, Thomas Noll (eds.): Kolloquium Programmiersprachen und Grundlagen der Programmierung
2002-01 *	Jahresbericht 2001
2002-02	Jürgen Giesl, Aart Middeldorp: Transformation Techniques for Context-Sensitive Rewrite Systems
2002-03	Benedikt Bollig, Martin Leucker, Thomas Noll: Generalised Regular MSC Languages
2002-04	Jürgen Giesl, Aart Middeldorp: Innermost Termination of Context-Sensitive Rewriting
2002-05	Horst Lichter, Thomas von der Maßen, Thomas Weiler: Modelling Requirements and Architectures for Software Product Lines
2002-06	Henry N. Adorna: 3-Party Message Complexity is Better than 2-Party Ones for Proving Lower Bounds on the Size of Minimal Nondeterministic Finite Automata
2002-07	Jörg Dahmen: Invariant Image Object Recognition using Gaussian Mixture Densities
2002-08	Markus Mohnen: An Open Framework for Data-Flow Analysis in Java
2002-09	Markus Mohnen: Interfaces with Default Implementations in Java
2002-10	Martin Leucker: Logics for Mazurkiewicz traces
2002-11	Jürgen Giesl, Hans Zantema: Liveness in Rewriting
2003-01 *	Jahresbericht 2002
2003-02	Jürgen Giesl, René Thiemann: Size-Change Termination for Term Rewriting
2003-03	Jürgen Giesl, Deepak Kapur: Deciding Inductive Validity of Equations
2003-04	Jürgen Giesl, René Thiemann, Peter Schneider-Kamp, Stephan Falke: Improving Dependency Pairs
2003-05	Christof Löding, Philipp Rohde: Solving the Sabotage Game is PSPACE-hard
2003-06	Franz Josef Och: Statistical Machine Translation: From Single-Word Models to Alignment Templates

2003-07	Horst Lichter, Thomas von der Maßen, Alexander Nyßen, Thomas Weiler: Vergleich von Ansätzen zur Feature Modellierung bei der Softwareproduktlinienentwicklung
2003-08	Jürgen Giesl, René Thiemann, Peter Schneider-Kamp, Stephan Falke: Mechanizing Dependency Pairs
2004-01 *	Fachgruppe Informatik: Jahresbericht 2003
2004-02	Benedikt Bollig, Martin Leucker: Message-Passing Automata are expressively equivalent to EMSO logic
2004-03	Delia Kesner, Femke van Raamsdonk, Joe Wells (eds.): HOR 2004 – 2nd International Workshop on Higher-Order Rewriting
2004-04	Slim Abdennadher, Christophe Ringeissen (eds.): RULE 04 – Fifth International Workshop on Rule-Based Programming
2004-05	Herbert Kuchen (ed.): WFLP 04 – 13th International Workshop on Functional and (Constraint) Logic Programming
2004-06	Sergio Antoy, Yoshihito Toyama (eds.): WRS 04 – 4th International Workshop on Reduction Strategies in Rewriting and Programming
2004-07	Michael Codish, Aart Middeldorp (eds.): WST 04 – 7th International Workshop on Termination
2004-08	Klaus Indermark, Thomas Noll: Algebraic Correctness Proofs for Compiling Recursive Function Definitions with Strictness Information
2004-09	Joachim Kneis, Daniel Mölle, Stefan Richter, Peter Rossmanith: Parameterized Power Domination Complexity
2004-10	Zinaida Benenson, Felix C. Gärtner, Dogan Kesdogan: Secure Multi-Party Computation with Security Modules
2005-01 *	Fachgruppe Informatik: Jahresbericht 2004
2005-02	Maximillian Dornseif, Felix C. Gärtner, Thorsten Holz, Martin Mink: An Offensive Approach to Teaching Information Security: “Aachen Summer School Applied IT Security”
2005-03	Jürgen Giesl, René Thiemann, Peter Schneider-Kamp: Proving and Disproving Termination of Higher-Order Functions
2005-04	Daniel Mölle, Stefan Richter, Peter Rossmanith: A Faster Algorithm for the Steiner Tree Problem
2005-05	Fabien Pouget, Thorsten Holz: A Pointillist Approach for Comparing Honeypots
2005-06	Simon Fischer, Berthold Vöcking: Adaptive Routing with Stale Information
2005-07	Felix C. Freiling, Thorsten Holz, Georg Wicherski: Botnet Tracking: Exploring a Root-Cause Methodology to Prevent Distributed Denial-of-Service Attacks
2005-08	Joachim Kneis, Peter Rossmanith: A New Satisfiability Algorithm With Applications To Max-Cut
2005-09	Klaus Kursawe, Felix C. Freiling: Byzantine Fault Tolerance on General Hybrid Adversary Structures
2005-10	Benedikt Bollig: Automata and Logics for Message Sequence Charts
2005-11	Simon Fischer, Berthold Vöcking: A Counterexample to the Fully Mixed Nash Equilibrium Conjecture

2005-12	Neeraj Mittal, Felix Freiling, S. Venkatesan, Lucia Draque Penso: Efficient Reductions for Wait-Free Termination Detection in Faulty Distributed Systems
2005-13	Carole Delporte-Gallet, Hugues Fauconnier, Felix C. Freiling: Revisiting Failure Detection and Consensus in Omission Failure Environments
2005-14	Felix C. Freiling, Sukumar Ghosh: Code Stabilization
2005-15	Uwe Naumann: The Complexity of Derivative Computation
2005-16	Uwe Naumann: Syntax-Directed Derivative Code (Part I: Tangent-Linear Code)
2005-17	Uwe Naumann: Syntax-directed Derivative Code (Part II: Intraprocedural Adjoint Code)
2005-18	Thomas von der Maßen, Klaus Müller, John MacGregor, Eva Geisberger, Jörg Dörr, Frank Houdek, Harbhajan Singh, Holger Wußmann, Hans-Veit Bacher, Barbara Paech: Einsatz von Features im Software-Entwicklungsprozess - Abschlußbericht des GI-Arbeitskreises “Features”
2005-19	Uwe Naumann, Andre Vehreschild: Tangent-Linear Code by Augmented LL-Parsers
2005-20	Felix C. Freiling, Martin Mink: Bericht über den Workshop zur Ausbildung im Bereich IT-Sicherheit Hochschulausbildung, berufliche Weiterbildung, Zertifizierung von Ausbildungsangeboten am 11. und 12. August 2005 in Köln organisiert von RWTH Aachen in Kooperation mit BITKOM, BSI, DLR und Gesellschaft fuer Informatik (GI) e.V.
2005-21	Thomas Noll, Stefan Rieger: Optimization of Straight-Line Code Revisited
2005-22	Felix Freiling, Maurice Herlihy, Lucia Draque Penso: Optimal Randomized Fair Exchange with Secret Shared Coins
2005-23	Heiner Ackermann, Alantha Newman, Heiko Röglin, Berthold Vöcking: Decision Making Based on Approximate and Smoothed Pareto Curves
2005-24	Alexander Becher, Zinaida Benenson, Maximillian Dornseif: Tampering with Motes: Real-World Physical Attacks on Wireless Sensor Networks
2006-01 *	Fachgruppe Informatik: Jahresbericht 2005
2006-02	Michael Weber: Parallel Algorithms for Verification of Large Systems
2006-03	Michael Maier, Uwe Naumann: Intraprocedural Adjoint Code Generated by the Differentiation-Enabled NAGWare Fortran Compiler
2006-04	Ebadollah Varnik, Uwe Naumann, Andrew Lyons: Toward Low Static Memory Jacobian Accumulation
2006-05	Uwe Naumann, Jean Utke, Patrick Heimbach, Chris Hill, Derya Ozyurt, Carl Wunsch, Mike Fagan, Nathan Tallent, Michelle Strout: Adjoint Code by Source Transformation with OpenAD/F
2006-06	Joachim Kneis, Daniel Mölle, Stefan Richter, Peter Rossmanith: Divide-and-Color
2006-07	Thomas Colcombet, Christof Löding: Transforming structures by set interpretations
2006-08	Uwe Naumann, Yuxiao Hu: Optimal Vertex Elimination in Single-Expression-Use Graphs
2006-09	Tingting Han, Joost-Pieter Katoen: Counterexamples in Probabilistic Model Checking

2006-10 Mesut Günes, Alexander Zimmermann, Martin Wenig, Jan Ritzerfeld, Ulrich Meis: From Simulations to Testbeds - Architecture of the Hybrid MCG-Mesh Testbed

2006-11 Bastian Schlich, Michael Rohrbach, Michael Weber, Stefan Kowalewski: Model Checking Software for Microcontrollers

2006-12 Benedikt Bollig, Joost-Pieter Katoen, Carsten Kern, Martin Leucker: Replaying Play in and Play out: Synthesis of Design Models from Scenarios by Learning

2006-13 Wong Karianto, Christof Löding: Unranked Tree Automata with Sibling Equalities and Disequalities

2006-14 Danilo Beuche, Andreas Birk, Heinrich Dreier, Andreas Fleischmann, Heidi Galle, Gerald Heller, Dirk Janzen, Isabel John, Ramin Tavakoli Kolagari, Thomas von der Maßen, Andreas Wolfram: Report of the GI Work Group “Requirements Management Tools for Product Line Engineering”

2006-15 Sebastian Ullrich, Jakob T. Valvoda, Torsten Kuhlen: Utilizing optical sensors from mice for new input devices

2006-16 Rafael Ballagas, Jan Borchers: Selexels: a Conceptual Framework for Pointing Devices with Low Expressiveness

2006-17 Eric Lee, Henning Kiel, Jan Borchers: Scrolling Through Time: Improving Interfaces for Searching and Navigating Continuous Audio Timelines

2007-01 * Fachgruppe Informatik: Jahresbericht 2006

2007-02 Carsten Fuhs, Jürgen Giesl, Aart Middeldorp, Peter Schneider-Kamp, René Thiemann, and Harald Zankl: SAT Solving for Termination Analysis with Polynomial Interpretations

2007-03 Jürgen Giesl, René Thiemann, Stephan Swiderski, and Peter Schneider-Kamp: Proving Termination by Bounded Increase

2007-04 Jan Buchholz, Eric Lee, Jonathan Klein, and Jan Borchers: coJIVE: A System to Support Collaborative Jazz Improvisation

2007-05 Uwe Naumann: On Optimal DAG Reversal

2007-06 Joost-Pieter Katoen, Thomas Noll, and Stefan Rieger: Verifying Concurrent List-Manipulating Programs by LTL Model Checking

2007-07 Alexander Nyßen, Horst Lichter: MeDUSA - MethoD for UML2-based Design of Embedded Software Applications

2007-08 Falk Salewski and Stefan Kowalewski: Achieving Highly Reliable Embedded Software: An empirical evaluation of different approaches

2007-09 Tina Kraußer, Heiko Mantel, and Henning Sudbrock: A Probabilistic Justification of the Combining Calculus under the Uniform Scheduler Assumption

2007-10 Martin Neuhäußer, Joost-Pieter Katoen: Bisimulation and Logical Preservation for Continuous-Time Markov Decision Processes

2007-11 Klaus Wehrle (editor): 6. Fachgespräch Sensornetzwerke

2007-12 Uwe Naumann: An L-Attributed Grammar for Adjoint Code

2007-13 Uwe Naumann, Michael Maier, Jan Riehme, and Bruce Christianson: Second-Order Adjoints by Source Code Manipulation of Numerical Programs

2007-14	Jean Utke, Uwe Naumann, Mike Fagan, Nathan Tallent, Michelle Strout, Patrick Heimbach, Chris Hill, and Carl Wunsch: OpenAD/F: A Modular, Open-Source Tool for Automatic Differentiation of Fortran Codes
2007-15	Volker Stolz: Temporal assertions for sequential and concurrent programs
2007-16	Sadeq Ali Makram, Mesut Güneç, Martin Wenig, Alexander Zimmermann: Adaptive Channel Assignment to Support QoS and Load Balancing for Wireless Mesh Networks
2007-17	René Thiemann: The DP Framework for Proving Termination of Term Rewriting
2007-18	Uwe Naumann: Call Tree Reversal is NP-Complete
2007-19	Jan Riehme, Andrea Walther, Jörg Stiller, Uwe Naumann: Adjoints for Time-Dependent Optimal Control
2007-20	Joost-Pieter Katoen, Daniel Klink, Martin Leucker, and Verena Wolf: Three-Valued Abstraction for Probabilistic Systems
2007-21	Tingting Han, Joost-Pieter Katoen, and Alexandru Mereacre: Compositional Modeling and Minimization of Time-Inhomogeneous Markov Chains
2007-22	Heiner Ackermann, Paul W. Goldberg, Vahab S. Mirrokni, Heiko Röglin, and Berthold Vöcking: Uncoordinated Two-Sided Markets
2008-01 *	Fachgruppe Informatik: Jahresbericht 2007/2008
2008-02	Henrik Bohnenkamp, Marielle Stoelinga: Quantitative Testing
2008-03	Carsten Fuhs, Jürgen Giesl, Aart Middeldorp, Peter Schneider-Kamp, René Thiemann, Harald Zankl: Maximal Termination
2008-04	Uwe Naumann, Jan Riehme: Sensitivity Analysis in Sisyphe with the AD-Enabled NAGWare Fortran Compiler
2008-05	Frank G. Radmacher: An Automata Theoretic Approach to the Theory of Rational Tree Relations
2008-06	Uwe Naumann, Laurent Hascoet, Chris Hill, Paul Hovland, Jan Riehme, Jean Utke: A Framework for Proving Correctness of Adjoint Message Passing Programs
2008-07	Alexander Nyßen, Horst Lichter: The MeDUSA Reference Manual, Second Edition
2008-08	George B. Mertzios, Stavros D. Nikolopoulos: The λ-cluster Problem on Parameterized Interval Graphs
2008-09	George B. Mertzios, Walter Unger: An optimal algorithm for the k-fixed-endpoint path cover on proper interval graphs
2008-10	George B. Mertzios, Walter Unger: Preemptive Scheduling of Equal-Length Jobs in Polynomial Time
2008-11	George B. Mertzios: Fast Convergence of Routing Games with Splittable Flows
2008-12	Joost-Pieter Katoen, Daniel Klink, Martin Leucker, Verena Wolf: Abstraction for stochastic systems by Erlang's method of stages
2008-13	Beatriz Alarcón, Fabian Emmes, Carsten Fuhs, Jürgen Giesl, Raúl Gutiérrez, Salvador Lucas, Peter Schneider-Kamp, René Thiemann: Improving Context-Sensitive Dependency Pairs
2008-14	Bastian Schlich: Model Checking of Software for Microcontrollers
2008-15	Joachim Kneis, Alexander Langer, Peter Rossmanith: A New Algorithm for Finding Trees with Many Leaves

2008-16	Hendrik vom Lehn, Elias Weingärtner and Klaus Wehrle: Comparing recent network simulators: A performance evaluation study
2008-17	Peter Schneider-Kamp: Static Termination Analysis for Prolog using Term Rewriting and SAT Solving
2008-18	Falk Salewski: Empirical Evaluations of Safety-Critical Embedded Systems
2008-19	Dirk Wilking: Empirical Studies for the Application of Agile Methods to Embedded Systems
2009-01 *	Fachgruppe Informatik: Jahresbericht 2009
2009-02	Taolue Chen, Tingting Han, Joost-Pieter Katoen, Alexandru Mereacre: Quantitative Model Checking of Continuous-Time Markov Chains Against Timed Automata Specifications
2009-03	Alexander Nyßen: Model-Based Construction of Embedded Real-Time Software - A Methodology for Small Devices
2009-05	George B. Mertzios, Ignasi Sau, Shmuel Zaks: A New Intersection Model and Improved Algorithms for Tolerance Graphs
2009-06	George B. Mertzios, Ignasi Sau, Shmuel Zaks: The Recognition of Tolerance and Bounded Tolerance Graphs is NP-complete
2009-07	Joachim Kneis, Alexander Langer, Peter Rossmanith: Derandomizing Non-uniform Color-Coding I
2009-08	Joachim Kneis, Alexander Langer: Satellites and Mirrors for Solving Independent Set on Sparse Graphs
2009-09	Michael Nett: Implementation of an Automated Proof for an Algorithm Solving the Maximum Independent Set Problem
2009-10	Felix Reidl, Fernando Sánchez Villaamil: Automatic Verification of the Correctness of the Upper Bound of a Maximum Independent Set Algorithm
2009-11	Kyriaki Ioannidou, George B. Mertzios, Stavros D. Nikolopoulos: The Longest Path Problem is Polynomial on Interval Graphs
2009-12	Martin Neuhäußer, Lijun Zhang: Time-Bounded Reachability in Continuous-Time Markov Decision Processes
2009-13	Martin Zimmermann: Time-optimal Winning Strategies for Poset Games
2009-14	Ralf Huuck, Gerwin Klein, Bastian Schlich (eds.): Doctoral Symposium on Systems Software Verification (DS SSV'09)
2009-15	Joost-Pieter Katoen, Daniel Klink, Martin Neuhäußer: Compositional Abstraction for Stochastic Systems
2009-16	George B. Mertzios, Derek G. Corneil: Vertex Splitting and the Recognition of Trapezoid Graphs
2009-17	Carsten Kern: Learning Communicating and Nondeterministic Automata
2009-18	Paul Hänsch, Michaela Slaats, Wolfgang Thomas: Parametrized Regular Infinite Games and Higher-Order Pushdown Strategies
2010-01 *	Fachgruppe Informatik: Jahresbericht 2010
2010-02	Daniel Neider, Christof Löding: Learning Visibly One-Counter Automata in Polynomial Time
2010-03	Holger Krahn: MontiCore: Agile Entwicklung von domänenspezifischen Sprachen im Software-Engineering

2010-04	René Wörzberger: Management dynamischer Geschäftsprozesse auf Basis statischer Prozessmanagementsysteme
2010-05	Daniel Retkowitz: Softwareunterstützung für adaptive eHome-Systeme
2010-06	Taolue Chen, Tingting Han, Joost-Pieter Katoen, Alexandru Mereacre: Computing maximum reachability probabilities in Markovian timed automata
2010-07	George B. Mertzios: A New Intersection Model for Multitolerance Graphs, Hierarchy, and Efficient Algorithms
2010-08	Carsten Otto, Marc Brockschmidt, Christian von Essen, Jürgen Giesl: Automated Termination Analysis of Java Bytecode by Term Rewriting
2010-09	George B. Mertzios, Shmuel Zaks: The Structure of the Intersection of Tolerance and Cocomparability Graphs
2010-10	Peter Schneider-Kamp, Jürgen Giesl, Thomas Ströder, Alexander Serebrenik, René Thiemann: Automated Termination Analysis for Logic Programs with Cut
2010-11	Martin Zimmermann: Parametric LTL Games
2010-12	Thomas Ströder, Peter Schneider-Kamp, Jürgen Giesl: Dependency Triples for Improving Termination Analysis of Logic Programs with Cut
2010-13	Ashraf Armoush: Design Patterns for Safety-Critical Embedded Systems
2010-14	Michael Codish, Carsten Fuhs, Jürgen Giesl, Peter Schneider-Kamp: Lazy Abstraction for Size-Change Termination
2010-15	Marc Brockschmidt, Carsten Otto, Christian von Essen, Jürgen Giesl: Termination Graphs for Java Bytecode
2010-16	Christian Berger: Automating Acceptance Tests for Sensor- and Actuator-based Systems on the Example of Autonomous Vehicles
2010-17	Hans Grönniger: Systemmodell-basierte Definition objektbasierter Modellierungssprachen mit semantischen Variationspunkten
2010-18	Ibrahim Armaç: Personalisierte eHomes: Mobilität, Privatsphäre und Sicherheit
2010-19	Felix Reidl: Experimental Evaluation of an Independent Set Algorithm
2010-20	Wladimir Fridman, Christof Löding, Martin Zimmermann: Degrees of Lookahead in Context-free Infinite Games
2011-01 *	Fachgruppe Informatik: Jahresbericht 2011
2011-02	Marc Brockschmidt, Carsten Otto, Jürgen Giesl: Modular Termination Proofs of Recursive Java Bytecode Programs by Term Rewriting
2011-03	Lars Noschinski, Fabian Emmes, Jürgen Giesl: A Dependency Pair Framework for Innermost Complexity Analysis of Term Rewrite Systems
2011-04	Christina Jansen, Jonathan Heinen, Joost-Pieter Katoen, Thomas Noll: A Local Greibach Normal Form for Hyperedge Replacement Grammars
2011-07	Shahar Maoz, Jan Oliver Ringert, Bernhard Rumpe: An Operational Semantics for Activity Diagrams using SMV
2011-08	Thomas Ströder, Fabian Emmes, Peter Schneider-Kamp, Jürgen Giesl, Carsten Fuhs: A Linear Operational Semantics for Termination and Complexity Analysis of ISO Prolog
2011-09	Markus Beckers, Johannes Lotz, Viktor Mosenkis, Uwe Naumann (Editors): Fifth SIAM Workshop on Combinatorial Scientific Computing
2011-10	Markus Beckers, Viktor Mosenkis, Michael Maier, Uwe Naumann: Adjoint Subgradient Calculation for McCormick Relaxations

2011-11	Nils Jansen, Erika Ábrahám, Jens Katelaan, Ralf Wimmer, Joost-Pieter Katoen, Bernd Becker: Hierarchical Counterexamples for Discrete-Time Markov Chains
2011-12	Ingo Felscher, Wolfgang Thomas: On Compositional Failure Detection in Structured Transition Systems
2011-13	Michael Förster, Uwe Naumann, Jean Utke: Toward Adjoint OpenMP
2011-14	Daniel Neider, Roman Rabinovich, Martin Zimmermann: Solving Muller Games via Safety Games
2011-16	Niloofar Safiran, Uwe Naumann: Toward Adjoint OpenFOAM
2011-18	Kamal Barakat: Introducing Timers to pi-Calculus
2011-19	Marc Brockschmidt, Thomas Ströder, Carsten Otto, Jürgen Giesl: Automated Detection of Non-Termination and NullPointerExceptions for Java Bytecode
2011-24	Callum Corbett, Uwe Naumann, Alexander Mitsos: Demonstration of a Branch-and-Bound Algorithm for Global Optimization using McCormick Relaxations
2011-25	Callum Corbett, Michael Maier, Markus Beckers, Uwe Naumann, Amin Ghobeity, Alexander Mitsos: Compiler-Generated Subgradient Code for McCormick Relaxations
2011-26	Hongfei Fu: The Complexity of Deciding a Behavioural Pseudometric on Probabilistic Automata
2012-01 *	Fachgruppe Informatik: Annual Report 2012

* These reports are only available as a printed version.

Please contact `biblio@informatik.rwth-aachen.de` to obtain copies.