Vision Driven System Engineering§

Matthias Jarke and Klaus Pohl

Informatik V, RWTH Aachen, Ahornstr. 55, 5100 Aachen, Germany
email: {jarke, pohl}@informatik.rwth-aachen.de

Abstract. Clearly defining, maintaining, and exploiting the system vision is a central prerequisite for successful system engineering. We address the question how the visions are concretized and maintained in information systems evolution. Visions are broken down into goals according to constraints imposed by context, and traded off against other goals or habits which exist in this context. Context information is organized according to four "worlds" and the context breakdown is viewed under a three-dimensional space of cognitive understanding, social agreement, and technical representation. Different uses and the evolution of goals in the system engineering process are supported by a quality and improvement oriented process model which distinguishes between product, control and improvement activities. Working with this model can be supported by a knowledge-based repository structure that is compatible with the IRDS standard.

Keyword Codes: D.2.1; D.4.1; H.4.0

Keyword: Requirement/Specification; Process Management; Information System Application

1. INTRODUCTION

The context in which software systems operate is increasingly recognized as a major determinant of system development and evolution. This is recognized in current CASE terminology by shifting the emphasis from *software* engineering to *system* engineering.

This trend has manifested itself in a number of concrete research activities. First, recent work in requirements engineering has emphasized the need for modeling the *system environment*, leading to activities such as enterprise modeling and composite systems modeling. Second, there has been a strong emphasis on measuring and improving *quality aspects* from the viewpoint of both software product and software process. Last not least,

---

§ This work was supported in part by ESPRIT Basic Research Project 6353 (NATURE) and in part by the Ministry of Science and Research of Nordrhein-Westfalen.

change management is no longer considered as a system-internal problem of version and configuration management but studied in its organizational context.

However, modeling the context independent of the actual project goals is an expensive and often poorly rewarded exercise. Practice in large system engineering efforts has shown that the clear definition and long-term perseverance of a *project vision* is important for getting and keeping a project going; nobody will sink money in a project for which the vision is unclear. A classical example of a successful vision is John F. Kennedy´s "send a man to the moon before the end of the decade". Counter examples abound among large ESPRIT projects where vision holders were too weak compared to the context.

It therefore seems that a careful balancing of vision and context management is a key to successful system engineering. Without vision, the best understanding of context remains passive. Without understanding and managing context, any vision is bound to fail due to naiveté. This implies two main questions:

- *where do the goals come from* — establishing the vision in context through a requirements engineering process
- *how are the goals used* — maintaining and evolving the vision and derived requirements through quality-oriented process management.

Sections 2 and 3 of this paper present a coherent set of models for these two closely interrelated tasks. These models — derived from earlier work by ourselves and various cooperation partners — are first and foremost intended to give structure to an evolutionary process that comprises both requirements engineering and the ongoing development and maintenance activities resulting from it. They can be used for analysing actual processes with the goal of understanding what is going on, or — at the meta level — for defining processes with the goal of controlling and improving them.

One of the difficulties we always face during such exercises is the data volume generated. Therefore, the models have been designed also with a second reading in mind. They can serve as the meta schema of a *repository* in which process observation and definition data are collected, and which is active in the sense that it can control the enacting of process definitions, maybe even enact some of them automatically. Section 4 of this paper therefore summarizes some technical considerations in the design of such a system. This gives an answer of sorts to the question: *how are the goals managed?*

2. WHERE DO THE GOALS COME FROM ?

Within the world in which the vision has to be realized, many habits exist. Some are based on formally stated goals, policies, or competing visions. Others are just regularly observable phenomena for which no structure or reasons are known a priori.

The task of the first phase of system engineering (requirements engineering) is therefore twofold. First, relevant habits must be analysed and the goals, policies and visions behind them must be made explicit. This is essentially a goal-directed abstraction process of existing practice (Dardenne et al. 1992). Second, new visions must be established in this context, i.e., their consequences are propagated into the existing context. Conflicts among the different viewpoints must be detected and resolved (Finkelstein et al. 1990). During this process, the vision often shifts. Many large projects appoint a "vision holder" to make sure that it does not get totally lost in the constraints of current practice.

Since context is complex and changing, some structure is needed by which context information can be organized. To bring vision and context together, nonfunctional goals, their trade-offs and implications for requirements change must be explicitly considered. To support the abstraction task, concrete and abstract, informal and formal requirements information must be related to each other.

Many requirements engineering methods emphasize one of these aspects at the expense of the other. Top-down decomposition methods such as Structured Analysis (McMenamin and Palmer 1984) do not say how context influences decomposition, whereas bottom-up approaches (ethnographics is an extreme example) have failed to provide guidance how analysis can be focused on the vision (Goguen 1993). The need for supporting both aspects simultaneously becomes more important as we are now in a situation where systems have become pervasive in the real world. Many such systems must evolve in multiple contexts and diverse configurations. Buzzwords such as composite systems (Fickas 1987) or Intelligent and Cooperative Information Systems (Brodie and Ceri 1992) accentuate this observation.

Our framework structures requirements engineering in two steps. First, it identifies four different "worlds" which impact evolution of the system and should be considered as a basis for analysing the "habits" in any requirements project (section 2.1). Second, it identifies a space of three dimensions in which the internal status and progress of the requirements process can be described (section 2.2).

2.1. How Can the Goals be Structured ?

What parts of the world are relevant and how are they related to the development process? This is in general quite unpredictable but requirements engineering simplifies the problem in various ways. First, it concentrates on identifying recurrent patterns (abstraction process) and tends to ignore spurious events. Second, certain habits can be predicted through a basic theory about the object of the requirements engineering process: the information system.

The theory we are going to follow is very simple (Jarke 1990, cf. figure 1). It sees an information system in analogy to "a telescope through which a user community observes a domain of interest" more efficiently than without the informaton system. The domain of interest may or may not overlap with the user community itself, it may or may not be changeable by the user community. But it makes sense to distinguish, from a cognitive as well as a social viewpoint, between the *usage world*, the *subject domain world*, and the *system* itself, and to describe their interrelationships. A fourth world, the *development environment*, has the basic task of assisting the vision holder in realizing the vision in the context of the other worlds (but also considering its internal development context of people, methods, experiences, and tools).

There are several advantages of organizing context information according to the four worlds. Each world is associated with certain *groups of stakeholders* who should be considered in requirements decision making. The model makes the social prediction that different areas of expertise, different languages, and different interests exist and need to be integrated. Moreover, it predicts certain *role-bound nonfunctional goals* which can be associated with the relationships between the worlds; dealing with such standard viewpoint resolution and negotiation tasks becomes a natural target for requirements engineering

methods and tools. Finally, we can expect that representatives in each of the worlds will have implicit or explicit *models about their own as well as the other worlds* which leads to further social, cognitive, and technical problems.

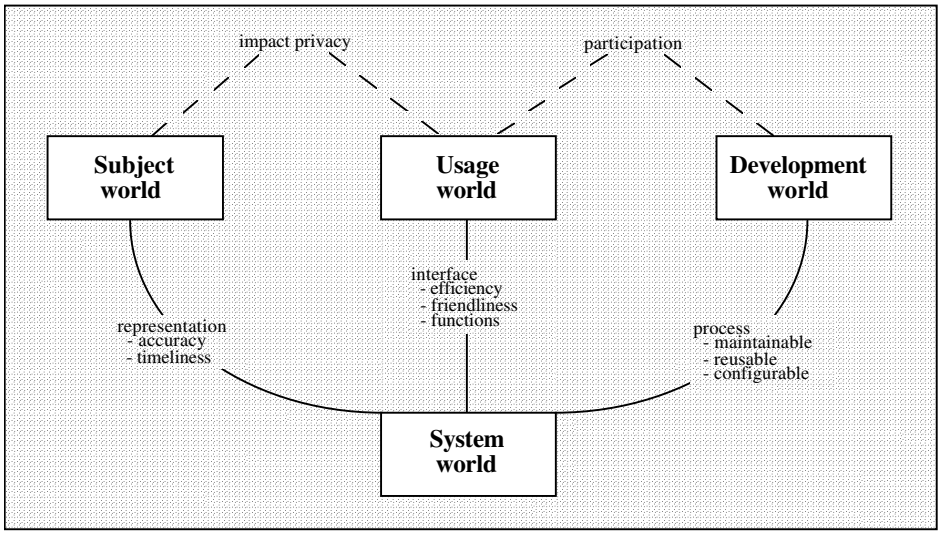

Figure 1**.** The "worlds" of information systems modeling.

Let us elaborate these general observations for each of the four worlds.

Subject World. The subject world is the domain which the system is intended to maintain information about. Stakeholders are directly the objects being represented, or people who have stakes in these objects but are not system users. Relationships to the usage world are often governed by legal concerns such as privacy or copyright. Relationships to the development world are difficult to establish and often very political -- quite frequently, subjects know nothing about the degree to which they are administered by systems. The relationship between system and subject world can be described by quality-of-information factors such as accuracy or timeliness. From a representational perspective, the subject world is the most general one: significant support can be offered only by domain specialization.

Usage World. The usage world comprises as stakeholders both the direct or indirect users, and the owners of the system. An organizational structure often describes their relationship, thus defining the (observed or intended) role of the information system in the work practice of the organization. The relationships between users and owners can vary widely but if both differ substantially, it is likely that the interests of the users are represented from an organizational perspective. With respect to the development world, the goal of participative design has often been cited. With respect to the system, quality-of-interaction factors such as response time, user-friendliness, and rich functionality are of interest as well as business goals such as reduced transaction costs, increased worker/organization effective–ness, worker (de-)qualification, and the like. From the representational perspective, business modeling as well as research in human-computer interaction are relevant for the usage world.

System World. The system world is often represented by technical gurus, or simply by the observed fact that the system is very hard to change, both due to its internal complexity and due to its established relationships to user and developer communities. Representation–wise, it is probably the best understood subworld with many available specification

languages. It contains a representation (function-oriented, data-oriented, object-oriented, ...) of the subject world, maybe also of its users, together with some mapping of these conceptual specifications to the design and implementation levels. The latter pose their own technological requirements. A system often exists in many different versions. It may be part of several different usage environments (in fact, it may be the main communications medium between these environments), and it may have lost much of its initial structure by changes that happened before the present "vision" came about.

Development World. All of this must be considered in the fourth world, the development world. It must ensure an adequate observation (cognitive aspect), participation (socio-organizational aspect), and representation (technical aspect) of itself and of the other worlds, and it must proceed under consideration of resource constraints and competing role-bound and individual goals.

The vision for change could come from any of the worlds. It could be driven by changing technology in the system world (moving from files to databases), changes in the subject domain (protests by privacy pressure groups, new theories about the subject domain), or long-term development concerns (improved maintainability, change of development responsibility from computer professionals to end users). Most frequently, visions come from the usage world, from users or their organizations.

2.2. How are the Goals Gained ?

Only a portion of the development process can be explained from the role expectations resulting from the four-worlds model. The situation and progress of a requirements process is equally influenced by individual differences and social behavior in a problem-solving team. This includes problems of cognition and social interaction as well as technical issues.

These problems cannot and need not be fully explicated in each individual case. However, their "essence" (McMenamin and Palmer 1984) with respect to the requirements process, their influence on the basic inputs and outputs of requirements engineering, should be captured. Briefly, the requirements engineering process starts with an opaque understanding of the problem, many different and conflicting viewpoints, and and informal representation. Ideally, it should end with full agreement on a well-understood requirements specification represented precisely enough to be transformable into a correct implementation. In other words, to be successful, the RE process has to make progress along three dimensions: the specification, the agreement and the representation dimension (Pohl 1993).

Specification Dimension. The specification dimension deals with the degree of requirements understanding at a given time. At the beginning of the RE process, only the vision is clearly defined. The specification of the system and its environment are more or less opaque. Focusing on this dimension, the aim of requirements engineering is to transform the vision into a complete specification through an iterative process of definition and validation. Standards and guidelines describe how the final requirements specification should look like (e.g., IEEE 830, British Standard 6719, ESA PSS-05-0). Empirical studies have shown that problems in achieving this aim include (Curtis et al. 1988, Goguen 1993):

- incomplete domain knowledge, poor and hazy system goals
- premature search reduction on a small set of sub-optimal solutions
- inadequate articulation of requirements

Agreement Dimension. The agreement dimension deals with the degree of agreement reached on a specification. Each person involved in the RE process has a personal view of integrating vision and habits, or may even have different visions. Few requirements may be initially shared among the team. It is important to recognize that the integration of different views at the representation level (e.g., integrating formally represented views into a comprehensive view) and the agreement on the integrated view among the people involved in the process are two separate things. A detected conflict or misunderstanding must be resolved through communication among people. A process of negotiation — comprising bargaining as well as mutual learning — and coordinated subgroup actvities may lead towards a setting of better understood personal views that are perceived to have at least sufficient commonality to start building a system. The process of gaining a common view is of course affected by the social and psychological aspects of requirements engineering. Its problems are often categorized according to three different levels (Ellis et al. 1991):

- communication, the technical ability to exchange messages
- interoperability, the technical and social ability of mutual understanding
- cooperation on a common task.

Representation Dimension. This dimension copes with the different representations used for expressing knowledge about the requirements process and its products. First, both abstract requirements specifications and concrete examples and scenarios are needed in requirements engineering. Second, the representation should enable the description of states and transitions within the considered worlds, as well as perceived intentions behind them. Third and most obviously, three categories of representation are distinguished with respect to formal semantics: informal (e.g., natural languages, graphics), semi-formal (e.g., SA / ER diagrams) and formal specification and knowledge representation languages (e.g., VDM (Jones 1990), Z [(Spivey 1989), ERAE (Hagelstein 1988), Telos (Mylopoulos et al. 1990)).

Due to the unique advantages of each category, people prefer different representations in different situations. Recent surveys confirm that people use informal representation at the beginning of the requirements engineering process, then move to semi-formal graphics, at the end also to formal representations (Lubars et al. 1993). Ideally, the final specification should be transformable to executable code. The different representations used must be kept consistent and the transformation between them must be supported and recorded.The representation dimension is orthogonal to the specification dimension. A vague imagination of the system can be expressed using natural language, but also using a formal representation language. Conversely, precise ideas can be represented using natural language (law makers try to do so).

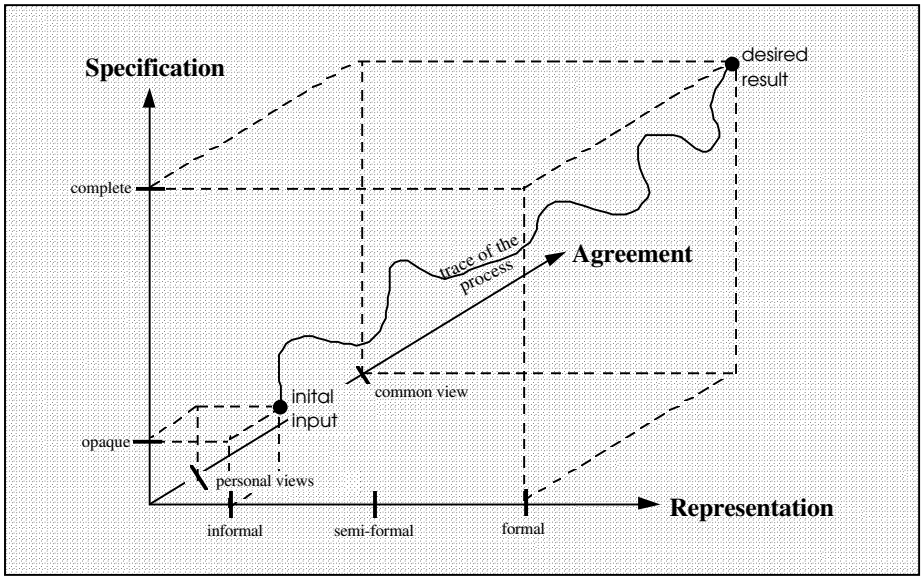

Figure 2. The three dimensions of requirements engineering.

The trace of the requirements engineering process is a curve within the three dimensions (cf. figure 2). The traditional requirements capture process would start somewhere near the origin of the framework with different personal views, little system understanding and a completely informal representation; it should end with agreement on a well-understood and formally described specification. Movement may sometimes seem erratic because the outside context of the four worlds changes.

3. HOW ARE THE GOALS USED ?

Quality orientation is one of the important aspects in product development. According to the TQM philosophy (Oakland 1989) the production process must assure that the product fullfills the customer requirements (the stated visions and goals); that quality is produced in the first place, rather than introduced through testing and rework (cf. Jarke and Pohl 1992). Therefore adequate production methods must be chosen, taking the goals under consideration. Quality control activities must check if the product satisfies the expectations. Hence, the choice of the control methods also depends on the goals. Achieving this requires that the production process is guided by a process model. This model itself must be quality oriented — the choice among alternatives must consider quality aspects. The task of system engineering involves, after transforming the vision into a requirements specification, to serve in the design and implementation tasks as a framework for the operationalization of the necessary changes in the real world. Of course, the goals change during these phases since new goals may be detected, or existing goals may be refined.

But, as the philosophy of TQM (Oakland 1989, Deming 1986) points out, quality-orientation is not enough. Continous improvement of process quality (which will automatically lead to better product quality) supported by selective process control is necessary. These process controls deliver raw data from which teamwork can derive concrete proposals for process improvement. Management commitment has to ensure that the results are actually instituted and not lost through bureaucratic hurdles.

The modeling framework we are going to propose in the sequel is not dependent on any particular formalism, although it does impose certain minimal requirements on formalisms in which it could be computer-supported. Briefly, such requirements include the need for representing multiple meta levels of typing, genericity hierarchies, complex descriptions, and some kind of logic for representing aspects such as policies, laws, or dependency structures. In our practical experiments, we are using the deductive object-oriented database language Telos (Mylopoulos et al. 1990) and its implementation ConceptBase (Jarke 1992). But most aspects of the model can be implemented in any object-oriented database environment. Since one of the syntax variants of Telos is a typed and directed graph, the graphics we use for the models can be understood as direct representations of a formal Telos knowledge base with a logical semantics (Jeusfeld 1992).

The basic concepts of our process model — situation, decision and actions — are adapted from the ALECSI model (section 3.1). Going further, we destinguish three different kinds of actions to emphasize their role with respect to goals: product, control and improvement actions, covering also the important role of teamwork in organizing and performing these actions (section 3.2).

3.1. The Basic Process Structure: Situated Decisions and Actions

Our model follows a *decision-oriented paradigm*. (Potts 1989, Jarke 1993). This gives the ability to more deeply integrate the semantics attached to the evolutionary aspects of the design. The notion of decision in the design process allows a better understanding of the designer's intention facilitating, in turn, better reuse of its results.

Even the decision-based approaches offer only limited hints when and how to decide on what. Following observations by Suchman (1987) and in the ALECSI project (Grosz and Rolland 1990), our model therefore makes the notion of situation (in which to decide) explicit and relates it to the broader question of context handling. The model associates the situation the system engineer faces to the decisions he can take to solve the local problem and to the actions to be performed to apply the decision (cf. figure 3).

A situation could be defined by itself (i.e., a subjective expressed perception of the user), by a view on a restricted set of object states (expressed using a constraint) and by a set of possible actions which could be applied in a particular situation of the defined kind. For predefined situation types, a ranking between the actions can be defined. Preconditions defined for each action make sure that it can only be applied within a given set of situations. The performance of the action can be controlled by the action constraint; e.g., assuring that changes are made to all relevant parts of the products. Postconditions test if, after executing an action, the state of the process (the actual situation) changes in a given manner. New situations are reached through actions, external events, or changing perceptions.

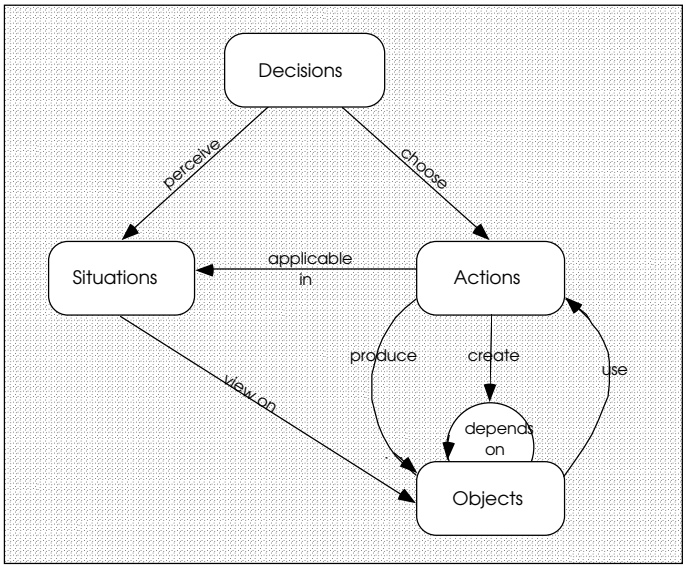

Figure 3. A process model based on situated decisions.

3.2. Goal-Oriented Differentiation of Action Types

By refining the interrelated lattices of product, situation, decision, and action types, we can model system engineering activities at macro and micro levels of detail. The decision context is structured by positioning the known situations and methods within the four worlds and the three dimensions discussed in section 2. However, we have no means yet to represent how the vision of the system can drive the process. The approach described below is intended to be carried through the full development cycle, justifying each design and implementation step by the quality goals it is intended to support. Thus, the whole process becomes effectively traceable to the interplay between vision and context.

A quality and improvement oriented process model must at least support three different kinds of actions:

- *product* process actions, which change product objects. They produce intermediate and final products and can be performed according to special product situations such as “baseline requirements model approved”. Action examples of this kind are “edit source file”, or “compile or map requirements into design”.
- *control actions*, which control the produced quality of the product. They may be applied in special process situations, e.g., “intermediate product completed”. Examples of such actions are measurements, reviews or audits.
- *improvement actions*, which change the process definition according to results of control actions, experience or general new insights; they therefore correspond to Madhavji´s meta level changes.

The term “action” refers to something that actually happens at the object level. In a process definition, we describe the kinds of product, control, and improvement actions by corresponding *product methods*, *control methods*, and *improvement methods.*

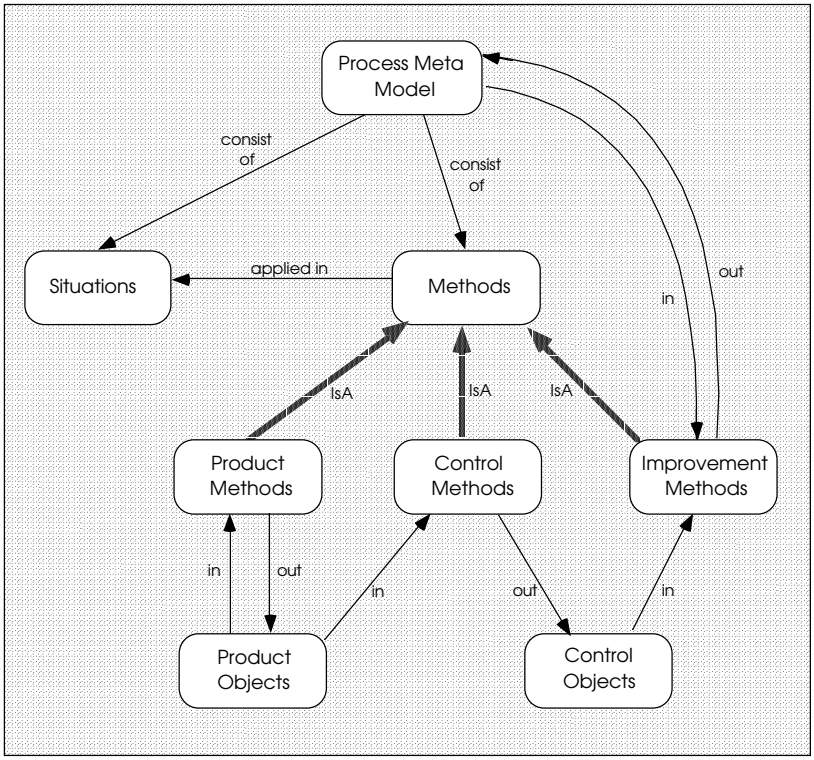

Figure 4. Three kinds of generic process actions and their input/output objects.

While all of these subtypes are basically "just actions", a careful analysis of their interrelationships and of the different situations in which they apply substantially increase the level of support offered. Formally, the three action types are gained by specializing the class *Methods*, as shown in figure 4. As part of defining the semantics of each kind of action, the objects used as input for the specialized actions as well as the output produced by them are specializations of the class *Object.* Similarly, the situations in which particular actions can be applied are defined by specializing the class *Situation* (not shown in figure 4).

Product Actions. Product actions change the state of the product objects. Methods for them are the ones normally considered by process modelling. For example, the software process is often divided in lifecycle phases such as requirements engineering, design in the large, design in the small, implementation and maintenance. Each of them is then modelled at a finer level of granularity. At the lowest level of abstraction, individual tool-supported actions are defined, e.g., for the situation "requirements model approved" the product method "map requirements to designs" is defined.

Executing a product action has several effects. First, it obviously changes the product. This change, and therefore the new state the product is in after the action is executed, does not only depend on the action used but also on the product states before the action. To make the process traceable, the dependencies between the different object states must be recorded.

The product produced by a product action serves as an input for the control actions. However, the usual product process models contain little in the way of facilitating this

relationship. Few of them include quality goals (nonfunctional requirements) directly in their framework of guiding process execution.

To produce quality in the first place, selection among applicable product methods must be based on the goals. It has to be defined when a certain method should be chosen, and which goals it is supposed to satisfy. But the relationship between goals and production method must not only be made explict during the process definition task, they must also be recorded. For example, if a goal changes, the affected production methods have to be localized and corresponding changes must be made.

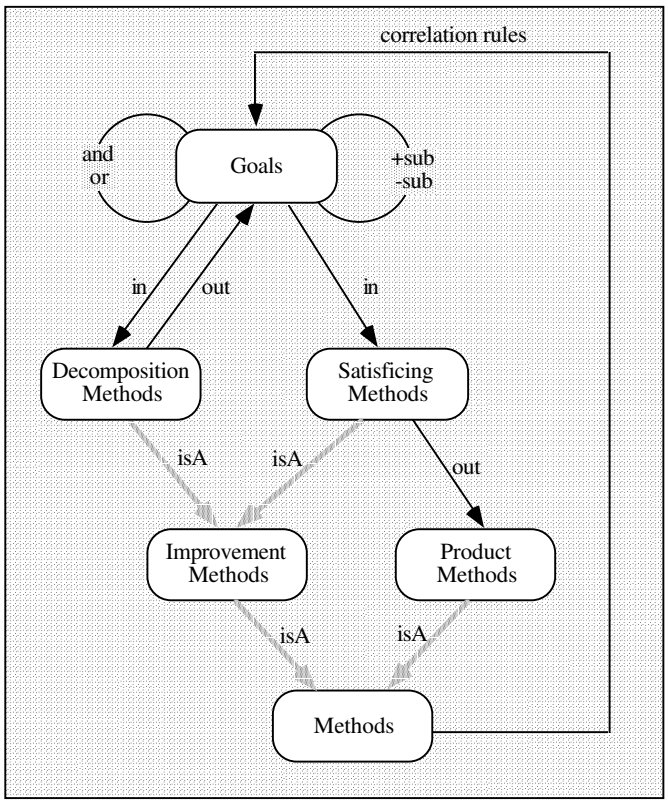

Figure 5. Metamodel for quality-driven design process.

Figure 5 is our semantic network re-interpretation of a model proposed in a search-tree formalism by Mylopoulos et al. (1992). It distinguishes three kinds of goals, and three kinds of methods which create these goals as well as links between them:

- *Decomposition methods* offer one possible scheme of subgoals that have to be achieved to achieve the overall goal.
- *Satisficing methods* describe options how to achieve these goals.
- *Argumentation methods* define the relationships between subgoals and methods — e.g., two methods exclude each other and that there are arguments for both of them.

In figure 5, the latter part of the model is omitted since we use a different approach to argumentation (cf. section 3.3). Links can be of type *and/or* (e.g., a conjunction of subgoals must be satisfied to satisfy a goal) or of type +/- (e.g., a method contributes positively or negatively to a goal). Correlation rules state unintended side-effects where the application of some method to achieve one goal also influences other goals. The approach is qualitative

in the sense that only positive and negative influences are recorded; trade-offs in conflict situations have to be made by humans. We are currently developing a method for selection of reusable process components under multiple quantitatively expressed quality criteria.

Control Actions. The definition of product action offers the technology to ensure that the production process can be defined. This definition is the basis for applying control actions which verify or validate intermediate products or the process execution itself. The result of a control action is recorded in a control object. The control object therefore contains information about the execution of the process; for instance, if a desired quality feature was produced in the first place. This information is essential for process improvement.

There exist different kinds of control actions within the software development process. In early stages such as requirements engineering, reviews, audits or walkthroughs are used. This is based on the fact that automatic control of product quality (the requirements engineering document) is nearly impossible. With growing formalization, some (but never all!) of the control activities can be automated. Many automated control actions during the implementation phase use *metrics* to obtain indirect software quality information. Which particular metrics is used depends on the quality goals which have to be reached.

The Goals-Question-Metric (GQM) model of TAME (Oivo and Basili 1992), shown in our reconstruction in figure 6, nicely complements figure 5. TAME relates the notions of goal and method to each other through the intermediate construct of questions that *need* to be answered in order to evaluate the contribution of a process execution towards a goal, and that *can* be partially answered by specific metrics. Thus, while the model of figure 5 describes the *planned* relationships between goals and methods through a goal-based product process model, the one in figure 6 measures the *actually achieved* relationships between the application of a given method and various goals (including but not only those it was originally designed to satisfy). This provides feedback on the quality of process definitions.

The activity of recording the process execution forms the basis for a second important class of control actions. The execution of the process might not follow the defined process model, due to decisions made by human developers or due to failure to follow the process rules by either humans or tools.

The fact that a process was not executed as defined is an important observation. By analysing the reasons, new insights for process improvement may be gained. But this is only possible if the process knowledge was recorded in a suitable way. So for this kind of control action, the differences between the defined process and the actually exectuted process must be made visible, together with rationales for these aberrations.

Another use of recorded process knowledge could also lead to process improvement. If, after the product is finished, the user observes that a certain quality goal was not achieved, changes to the product must be made. During the change, control actions must make sure that all actions which lead to the missing quality are localized, tracing back the process history to the action where the production of quality failed for the first time. The decision which lead to the execution of this action must then be revised. In retrospect, this makes the actual process execution to a certain degree undefined (we have changed the process definition post-mortem). The difference between the (new) defined process and the execution could serve as the basis for process improvement. Assuring that maintenance

activities (which are product actions) change the process execution and assuring that the violation of the process definition leads to a control object and builds the basis for process improvement.

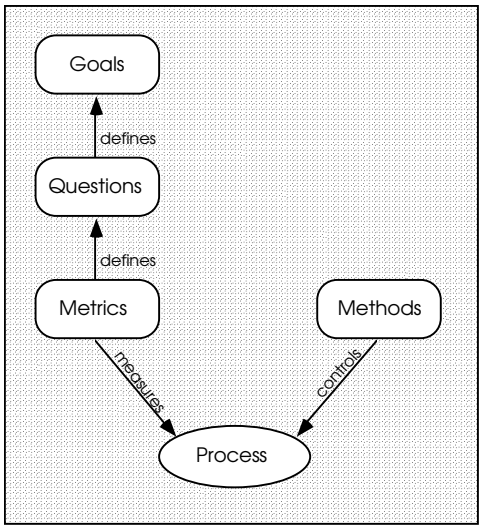

Figure 6. Process control according to the TAME meta model.

Summarizing both recording the difference between process execution and process definition (in the first place and due to changes afterwards) and the normal control activities applied on products (reviews, metrics, audits etc.) lead to control objects which serve as input for the improvement actions. Combined with the definitional facility of figure 5, the TAME framework of figure 6 could work in all three cases, provided that the notion of Goals in the figure also includes process goals.

Improvement Actions. "Improvement" is meant here similar to "change management" (Madhavji 1992), but with the added demand of explicitly considering quality factors in defining and recording the process definition change.

In contrast to product and control actions which deal with product and process objects at the execution level, improvement actions affect the definition of the process itself. Applying an improvement action either leads to an improved process model, or to a decison that no improvement is necessary. Since the latter could affect later decisions, it must also be recorded. The situation on which an improvement action is based does not only comprise the current process definition and control object information but also existing goals, policies, laws and resources. These are represented as special objects within the process model.

Our model distinguishes three possible kinds of improvement situations (fig. 7).

If *lack of quality* of an (intermediate) product was recognized by a control action, there exist control objects which characterize the improvement situation.

Where a *violation of the defined process* was recognized, not the missing quality of the product is the motivation for process improvement, but the unexpected performance of the process made to gain better quality in the first place.

In contrast to the first and second improvement situation which are recognized by performing the production process, the third kind of improvement situation is raised by

changes in the environment: changes of goals, policies etc. For instance, changing a policy must lead to improvement actions by which the new policy is integrated into the corresponding process model. Since the goals are represented within the process models, changes made to them (which are modelled as improvement actions) constitute improvement situations which assure that the process models can be adapted to these changes.

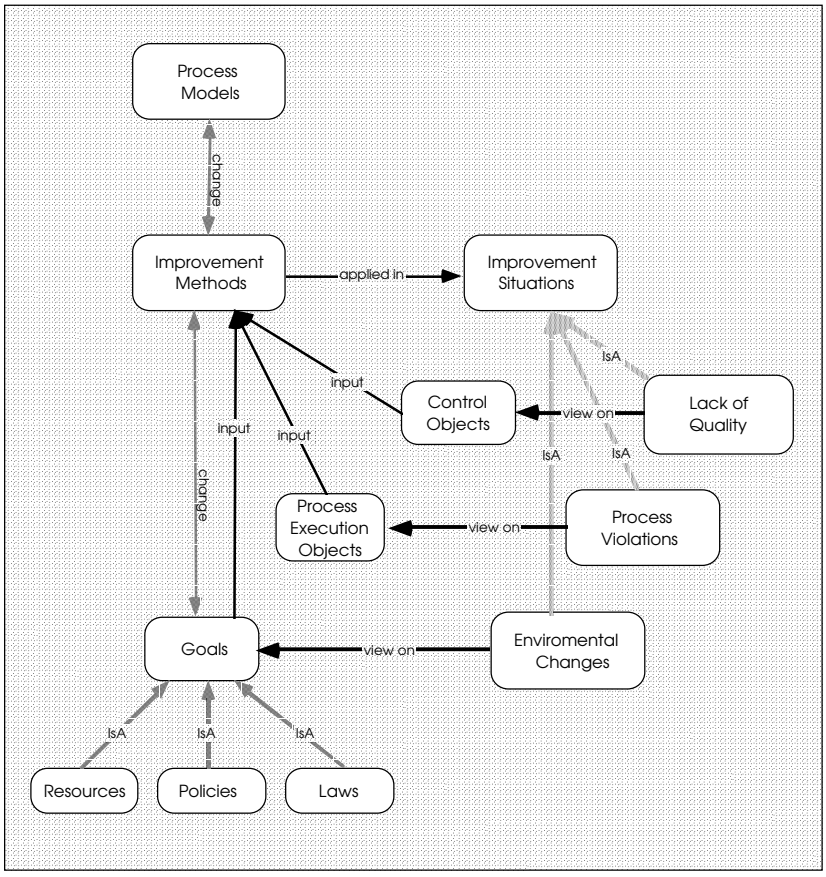

Figure 7. Improvement Methods with different improvement situations.

Improvement actions can have a different *scope with respect to organizational or product genericity* (cf. section 4.2). There are actions that affect the whole organization, but also for special product processes or even special project processes.

The different kinds of situations, the different kinds of improvement actions, and the improvement at different genericity levels make it necessary to differentiate between product process model and improvement process model. Specifically, to define the improvement process, it is necessary to investigate suitable metalevel operators that offer structure to process improvement and thus facilitate quality-oriented change management: has improve–ment been actually accomplished (repeatability)? how does a general improvement strategy look like (defineability)? how to manage change of improvement strategies (manageability)?.

The Integrated Model. Figure 8 shows how the three different kinds of methods for defining product, control and improvement actions are related to each other through the explicit management of quality goals.

Product processes are defined and executed using goal-oriented meta operators such as goal decomposition and satisficing. Experiences with these process definitions are captured through goal-driven control methods in the spirit of TAME. Improvement is carried out in a structured manner, again under explicit consideration of goals and with the use of structured meta operators; moreover, we recall that, similar to PRISM (Madhavji 1992), this model operates at multiple interacting levels of a genericity hierarchy of products and organizational units. Further semantics is provided to our model through the corresponding specialization of their associated situations, input and output objects, including the goals themselves.

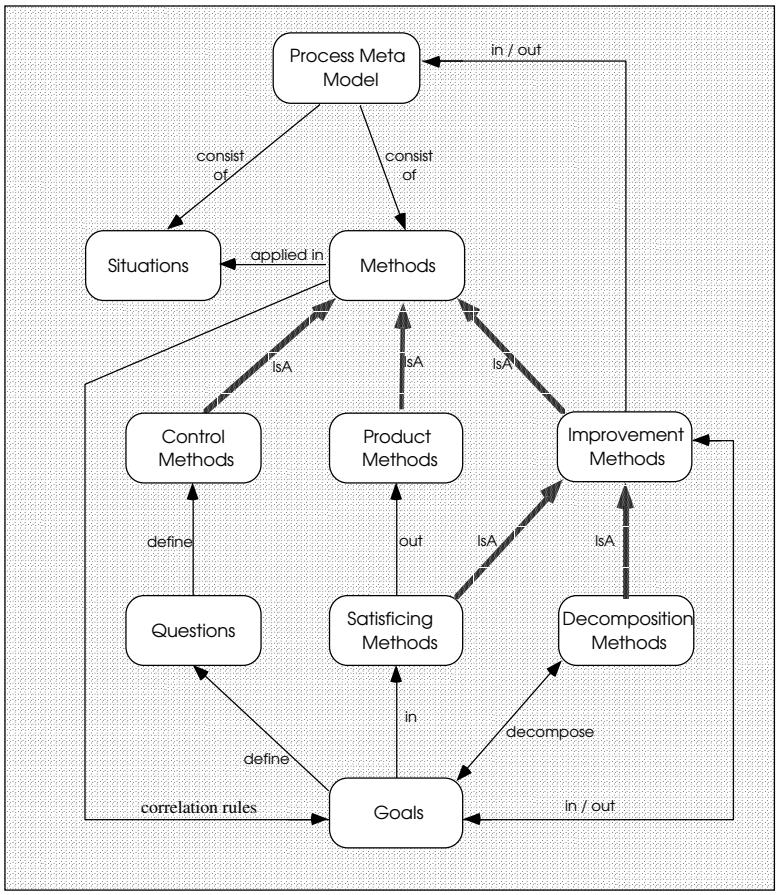

Figure 8. The integrated meta process model.

Teamwork Aspects. According to the TQM philosophy (Oakland 1989, Deming 1986), there is no process improvement without teamwork. Each person involved in a project must have the chance to understand the process definition, execution and evolution. Recording process knowledge could serve as a basis for later understanding of process execution and evolution. Teamwork is also the central means of acquiring knowledge about the quality requirements needed for our model.

We do not see process execution as an automatic transition from one state to the next one, but as a cooperation between agents. Taking a speech-act perspective (Winograd and Flores 1986), our model organizes this cooperation in conversation structures which lead to decisions that define and select actions. Since conversations may deal with knowledge not contained in the project-specific process definition (e.g., a person is ill), it is not possible to

fully predict a process execution. While the process definition may propose a ranking between the actions, the decision which of them is actually applied is the responsibility of the agents executing the process. As a result of the conversation, a decision could be made which chooses another action than the one with the highest a priori ranking. Even a decision for an action which was undefined for the given situation is conceivable.

The performance of an action may require special skills. Thus, decisions depend not only on the current process state but also on the availability of suitable agents. The conversation about the action to be performed in a given situation must therefore consider the skills needed for each action and the availability of suitably skilled agents. Structured conversations do not only play a role when executing process definitions but also when deciding on improvement actions. For structuring these conversations, we use Rittel´s Issue-based Information System model which has gained wide acceptance through its use for structuring the early phases of design deliberations in the gIBIS hypertext system (Conklin and Begeman 1988) and its more formal derivatives (Ramesh and Dhar 1992).

Figure 9 shows the integration of such conversation structures with our meta process model to achieve the required team-oriented quality and improvement oriented process meta model, extending the CAD^0 model used in the DAIDA project (Jarke and Rose 1992).

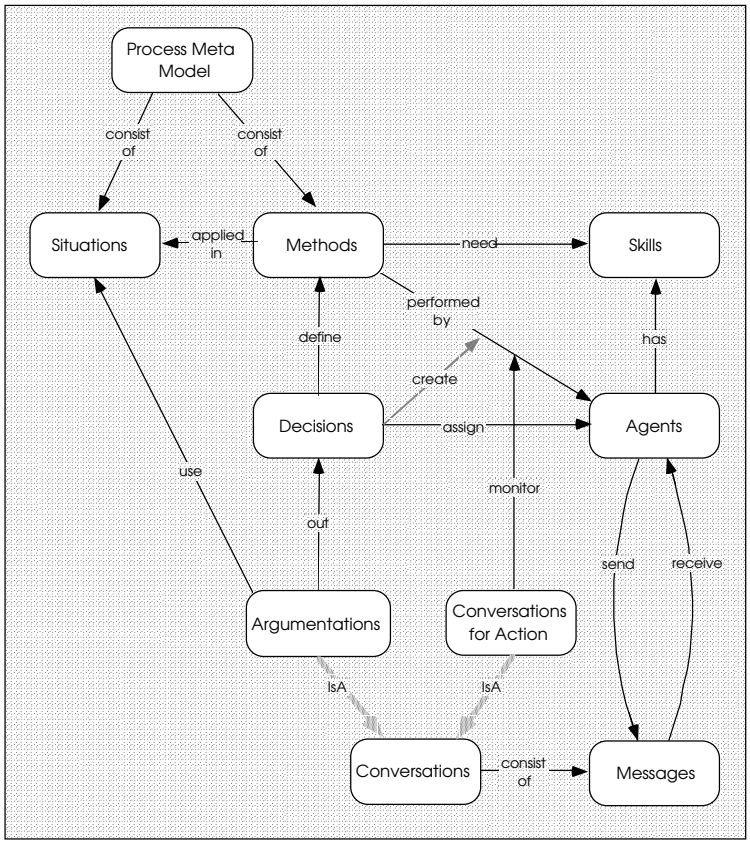

Figure 9. The meta process model including teamwork aspects.

4. HOW ARE THE GOALS MANAGED ?

The framework of goal acquisition and goal exploitation proposed in the previous sections implies a massive amount of recorded process information, even of process

definitions, when used to evaluate actual projects or general process models. Abstraction mechanisms and database technology are required to deal with this flood effectively.

In this section, we present a technical solution based on the repository technology standard IRDS (section 4.1) and two different abstraction hierarchies (section 4.2). Using this approach and our implementation of Telos, ConceptBase (Jarke 1992), we have imple–mented a prototypical system for supporting the usage of the information gained during system engineering and are using it in several major case studies.

4.1. Extending the Repository Standard IRDS

A repository is the central information system around which design environments are layered. We start from the four-level repository classification hierarchy defined in the Information Resource Dictionary Standard IRDS (1988). Level *n+1* formally constitutes a type system for level *n*. The lowest level, the *production data* layer, consists of actual databases and maybe software execution traces. The *data* layer describes and relates data and procedure schemata developed for each subsystem application. The *schema* layer represents, in a uniform framework, the individual data and application languages in which subsystems are defined — the schema of the repository. Finally, the uniform descriptive formalism used to define the schemata is defined in the fourth level, the *schema description* layer.

The first observation to be made when addressing quality improvement through repository technology is that all layers become process oriented rather than just product oriented. Figure 10 shows how, consequently, a process oriented repository hierarchy would look like.

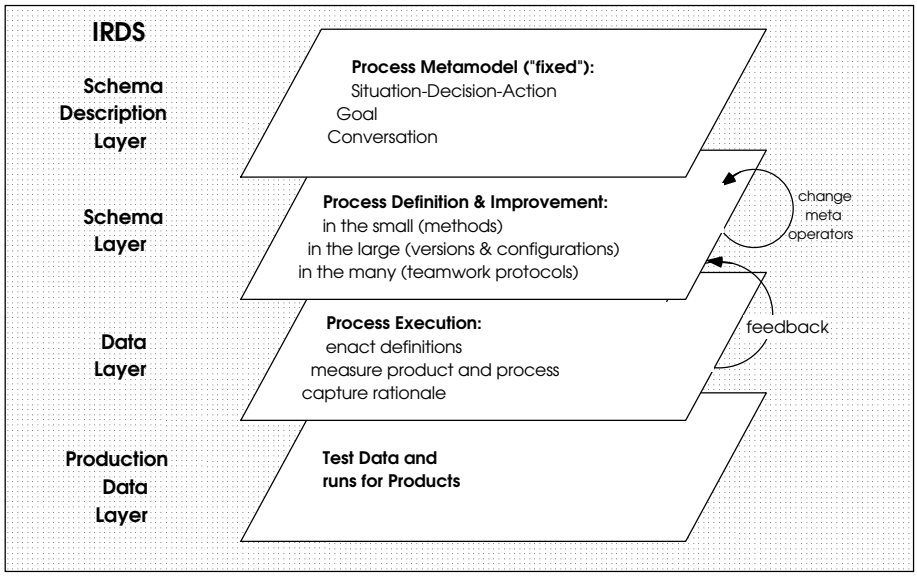

Figure 10. Repositories for process definition and evolution.

The *lowest* level contains application data and product execution traces which are interpreted as tests of product quality which can be used for feedback of process execution.

The *second* layer records process executions, e.g. capturing the rationale of design decisions for future reuse. If a formal process definition exists, recording its execution is a

basis for measurement of product and process, checking correct enacting, and feedback concerning the quality of the process definition.

The *third* level contains process definitions according to various viewpoints. We distinguish development in the small (content oriented methods for specific individual tasks), in the large (version and configuration management from object and task viewpoints), and in the many (team cooperation in the preparation, execution, and evaluation of processes).

The meta model at the *fourth* level provides the basic ontological structure according to which processes are defined. The model of section 3 forms our meta model.

These different levels of modelling enable us to interrelate process definitions, their specialization, their execution and execution data, and therefore gives the needed structure for the concepts of process improvement and change management as dicussed in section 3.

4.2 Classification vs. Genericity Hierarchies

Many researchers recognize the need of distinguishing different levels of abstraction when modeling processes (e.g., Perry 1990, Madhavji 1991, Basili & Rombach 1988). However, they usually focus on one dimension of abstraction only. This mixes two concerns:

- the *classification* hierarchy as in IRDS, where each level constitutes a type system (as in programming languages) for the level below;
- the *genericity* hierarchy that deals with different modeling scopes such as organizational dimension (e.g., individual, group, department, organization, inter-organizational, governmental, international) or product dimension (e.g., general software, information systems, knowledge-based information systems, ...).

Each level within the genericity hierarchy constitutes a specialization of a level above and therefore usually has to respect its rules, but can change those rules through some feedback formalism. The need of tailoring project independent models to project-specific models is widely accepted since (Osterweil 1987). They are the ones actually used during the project.

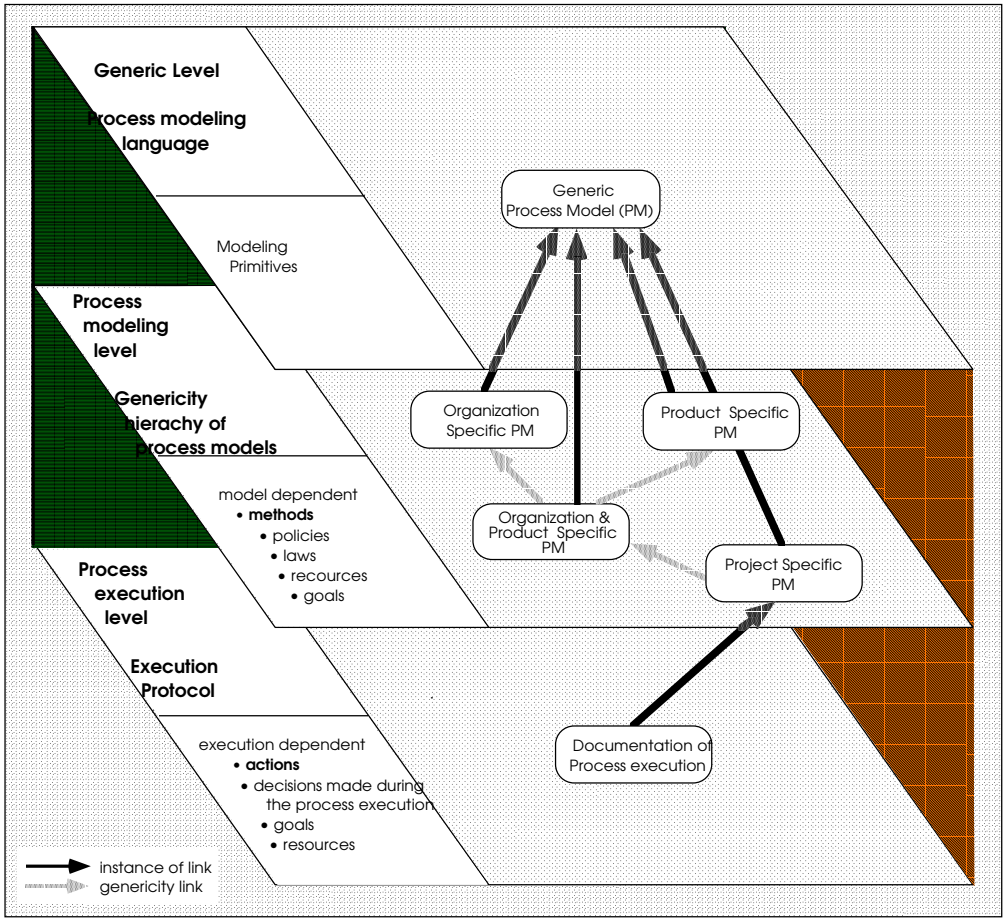

Figure 11. Classification and genericity hierarchies in process repositories.

The distinction between two abstraction hierarchies (classification and genericity) gives a richer picture of the problem-solving strategies that can be adopted in case of process breakdowns. For example, an organization may decide to use ISO 9000 as a policy. It therefore defines overall standard quality assurance methods at the organization-specific process definition level. In addition, product-specific quality assurance methods are defined. If a process model for a special product is built within an organization, both, the organization-specific and the product-specific quality assurance methods must be considered. But also additional methods could be defined for the product and organization specific process model. Further specializiation within the genericity hierarchy (e.g., for building a department model, group model or project model etc.) can be made.

In an object-oriented modeling formalism, the genericity hierarchy at the execution level need not be explicitly recorded but is automatically available through inheritance. We have *one* process execution which is indirectly an instance of each level in the genericity hierarchy to which it is attached. The different levels are views which define the process execution at their level of abstraction. Assume that there are four kinds of process models within the genericity hierarchy as shown in figure 11. Should it turn out (e.g., through failed process executions) that certain standard methods selected by the project model are infeasible, there are several choices of navigating within the space of process models to correct this:

- revise the process execution within the project process definition (e.g., choose an alternative method that satisfies the same goals).
- delegate the problem upwards to the people defining the project model, to have it solved by someone else within the organization & product specific definition.
- change the execution of the project model within the existing organization & process specific model.
- ask for changes of the organization & product specific model definition (here: the set of standard methods).

5. CONCLUSIONS

We first presented a framework for structuring the information gained during system engineering. Based on the concepts of "situation, decision and action", we then proposed a process model, whose definition and evolution is based on the goals to be reached. Finally we have shown, how the management of this information could be technically supported.

First empirical experience in analysing existing processes and formally defining some informally understood engineering methods shows, that software developers and users find it helpful to classify their knowledge according to the presented structure of

- the four worlds (development, usage, subject and system world),
- the three dimensions (agreement, representation and specification),
- the three different action types (control, improvement and product action)
- the three different improvement situations.

More work is needed, and done in the NATURE project, on elaborating detailed models within each of these four aspects. The question of how to map the goals down to production and control methods could by now only be answered for special kinds of goals, and our system experiments show that effective representation and visualization of the massive amounts of generated information is still an open issue, when we do not want to lose sight of the vision in the details.

Acknowledgments. Thanks are due to our partners in ESPRIT project NATURE, especially to the teams of Colette Rolland concerning the basic situation-decision-action model, of Janis Bubenko and Alastair Sutcliffe concerning domain and business modeling aspects, of John Mylopoulos concerning design with nonfunctional requirements, and of Yannis Vassiliou concerning Telos aspects. Also to the partners in BMFT project WibQus for providing excellent feedback on the practicability of many issues presented here.

REFERENCES

Basili, V. R. and Rombach, H. D. "The TAME Project: Towards Improvement-Oriented Software Environments" *IEEE Transactions on Software Engineering*, Volume 14, No. 6, 1988, pp. 758-773.

Brodie M.L. and Ceri S. "On Intelligent and Cooperative Information Systems: A Workshop Summary" *Intelligent & Cooperative Information Systems*, 1(2), June, 1992, 249-289.

Conklin, J. and Begeman, M. L. "gIBIS: A Hypertext Tool for Exploratory Policy Discussion" *ACM Transactions on Office Information Systems* Volume 6, No. 4, 1988, pp. 303 -331.

Curtis, B.; Krasner, H. and Iscoe, N. "Field Study of the Software Design Process for Large Systems" *Communications of the ACM*, 33 (11), 1988, 1268-1287.

Dardenne, A.; Fickas, S. and van Lamsweerde, A. "Goal-directed Concept Acquisition in Requirements Elicitation" *Proc. 6th IEEE Workshop System Specification and Design*, Como, Italy, 14-21.

Deming, W. E. *Out of the Crisis*. Massachusetts Institute of Technology, Center for Advanced Engineering Study, Cambridge, Massachusetts, 1986

Ellis, C.; Gibbs, S.; Rein G. "Groupware - Some Issues and Experiences" *Communications of the ACM* 34(1), 1991, 39-58.

Fickas S. "Automating analysis: An example" *Proc. 4th Intl.Workshop on Software Specification and Design*, Washington DC, 1987, 58-67.

Finkelstein A.; Kramer J. and Goedicke M. "Viewpoint-Oriented Software Development" P*roc. Conf Le Génie Logiciel et ses Applications,* Toulouse, 1990, 337-351.

Goguen J.A. "Social Issues in Requirements Engineering" *Proc. Int. Symposium on Requirements Engineering,* San Diego, Jan. 4-6, 1993, p. 194-195.

Grosz, G. and Rolland, C. "Using Artificial Intelligence Techniques to Formalize the Information System Design Process" *Proc. Intl. Conf. Databases and Expert Systems Applications*, 1990, 374-380.

Hagelstein, J. "Declarative Approach to Information Systems Requirements" *Knowledge Base Systems* 1(4), 1988, p. 211-220

IEEE *Software Engineering Standards.* Institute of Electrical and Electronics Engineers, Inc., 1987.

IRDS *Integrated Resource Dictionary Standard*, 1988, ANSI X3.138 und ISO WG3

Jarke, M. "DAIDA: Conceptual Modeling and Knowledge-based Support of Information Systems Development Processes" In *Technique et Science Informatiques*, Volume 9, No. 2, 1990, pp. 122-133.

Jarke, M. (ed.) *ConceptBase V. 3.1 User manual*, Technical Report, RWTH-Aachen, Germany, 1992.

Jarke, M. (ed.) *Database Application Engineering with DAIDA*. Springer Verlag, 1993.

Jarke, M and Pohl, K. "Information Systems Quality and Quality Information Systems" In *Proc. IFIP WG 8.2 Working Conf. Impact of Computer Supported Technologies on Information Systems Development*, Minneapolis, Minnesota, USA, 14-17 June, 1992, 345 -375.

Jarke, M. and Rose, T. "Specification Management with CAD^{o}" In Loucopoulos, P., Zicari, R. (eds.): *Conceptual Modeling, Databases, and CASE,* Chichester, UK: John Wiley, 1992.

Jeusfeld, M. A. "Änderungskontrolle in Deduktiven Objektbanken" DISKI Volume 17, Bad Honnef, Germany: INFIX Publ., 1992, (Diss. Univ. Passau, in German)

Jones, C.B. *Systematic Software Development Using VDM*, 2nd edition, Prentice Hall, 1990

Lubars, M.; Potts, C. and Richter C. "A Review on the State of the Practice in Requirements Modelling" *Proc. Int. Symposium on Requirements Engineering*, San Diego, Jan. 4-6, 1993, 2-14.

Madhavji, N. H. "The Process Cycle" In *IEE Software Engineering J.*, Volume 6, No. 5, 1991, pp. 234-242

Madhavji, N. H. "Environment Evolution: The Prism Model of Changes" *IEEE Transactions on Software Engineering*, Volume 18, No. 5, May, 1992, pp. 380-392.

McMenamin S.M. and Palmer J.F. *Essential System Analysis*. Yourdon Press, Englewood Cliffs, NJ, 1984.

Mylopoulos, J.; Borgida, A.; Jarke, M. and Koubarakis, M.; "Telos: Representing Knowledge About Information Systems" In *Transactions on Information Systems*, Volume 8, No. 4, 1990, pp. 325-362.

Mylopoulos, J.; Chung, L. and Nixon, B. "Representing and Using Non-Functional Requirements: A Process-Oriented Approach" *IEEE Trans. Software Engineering,* Volume 18, No. 6, 1992, pp. 483-497.

Oakland, J.S. "Total Quality Management" In *Proceedings 2nd Inernational Conference on Total Quality Management,* Oxford: Cotswold Press Ltd, 1989, pp. 3-17.

Oivo, M and Basili, V. R. "Representing Software Engineering Models: The TAME Goal Oriented Approach" *IEEE Transactions on Software Engineering,* Volume 18, No. 10, 1992.

Osterweil, L. "Software Processes are Software too" In *Procceedings 9th International Conference on Software Engineering*, Monterey, Ca, 1987, pp. 2-13.

Perry, D. E. "Policy and Product Directed Process Instantiation" In: *Proceedings of the 6th Int. Software Process Workshop*, Hokkaido, Japan, October, 1990.

Pohl, K. "The Three Dimensions of Requirements Engineering" *Proc. CAiSE Conf.,* Paris, June 8-11, 1993

Potts, C. "A Generic Model for Representing Design Methods" *Proc. 11th Intl. Conf. Software Engineering,*Pittsburgh, 1989.

Ramesh, B. and Dhar, V. “Process-Knowledge Based Group Support in Requirements Engineering” *IEEE Transactions on Software Engineering,* Volume 18, No. 6, 1992, 498-510.
Spivey, J.M. *The Z Notation*, Prentice Hall, 1989.
Suchman, L. A. *Plans and Situated Actions.* Cambridge Universiy Press, 1987
Winograd, T. and Flores F. *Understanding Computers and Cognitio*n, Ablex, Norwood, NJ, 1986.