# Specification Management with CADo

**Matthias Jarke,** Informatik V, RWTH Aachen, Ahornstr. 55, 51 Aachen, Germany
**Thomas Rose,** Dept. Computer Sc., University of Toronto, Ont M5S 1A4, Canada

*Specification management has many facets which are tackled by several different disciplines. Conceptual modeling -- supported by extended database technology -- turns out to be a suitable basis for the unification of these contributions in integrated software information systems. Due to the central importance of process aspects in specification management, such a conceptual model has to include both a structured and versioned object submodel and a strongly teamwork-oriented execution model.*

## 1 Introduction

The set of specifications of an information system includes requirements documents, formal models, designs, source code, even object code and test data, as well as project plans, progress reports, communication traces, and similar material. Specifications can be informal (text or multimedia), formal, or even computer-executable; they can address product, project, or process aspects; they can be contributed by developers, users, managers, or other stakeholders.

Although the need for specification and documentation has been stressed in traditional software lifecycles, this advice was usually followed at best during the initial development of systems. Already quite early in the usage and maintenance period of systems, the relationship between the different kinds of specifications and the actual product was lost. This lack of integration, in particular the lack of recorded process knowledge [DJ85], has seriously impeded the productivity gains promised by CASE. Most CASE environments today offer tools for each individual task in software development but their integration at both a semantic and a technical level is insufficient.

The use of software database technology has been proposed as one possible solution to this dilemma. Since available standard DBMS technology turned out to be inadequate to this task [BERN87], new kinds of software databases are being developed under the label of *repository*. Two important emerging standards, the Portable Common Tool Environment (PCTE) promoted by the European Community and IBM´s AD/Cycle methodology, place repository technology and software development models in key positions of their proposed environments.

The usage of repository-based specification management is not restricted to the initial development or maintenance of individual information systems.

Another way to view a specification database is that it provides a rich set of access paths to large software libraries. The many facets of specification management can then be used to inform the user about the structure, the possible application domains, the functional and nonfunctional properties of software objects, and the quality of the software process that created them.

A third usage of specification management takes an even more grandiose picture: it can be employed to provide meta information about all information sources available inside and outside an organization, to describe the birth, lifecycle and death of individual information subsystems together with methods to make optimal use of them in furthering an organization´s goal. Examples of such "Information Server" attempts include the domain of Computer-Integrated Manufacturing as well as interorgani- zational systems in banking or government [HEDB91]. The modeling of agents and conversations in addition to the specifications and their evolution is crucial in order to cover issues such as responsibility and security in such integrated information systems.

---

This work was supported in part by the Deutsche Forschungsgemeinschaft in its program "Object banks for experts" (no. Ja 445/1-3) and by the US National Science Foundation under grant no. IRI-8946799. Thanks are due to our collaborators, in particular Vasant Dhar of New York University.

In all of these applications, we observe a close relationship between the problems of specification management and those of requirements engineering in general. Requirements engineering often does not only prescribe changes in an information system but also changes in the user organization. Organizational processes are not that different from software development processes and can be modeled roughly with the same means. This brings us to a final emerging application area of specification management: the *re-engineering of organizational processes*. Here, specification management tools directly support high-level executives in analyzing and changing the way an orgnaization achieves its goals. Obviously, this requires the representation of non-functional goals as well as the representation of the ways how process alternatives contribute to these goals [CKM*91]. Such non-functional requirements can emerge from informal or semi-formal discussions [HJR90], or they can be "axiomatically" derived from economic theories, such as the theory of value-added chains [PM85]. The following figure, expanded from one shown in the chapter by John Mylopoulos (cf. also [JARK90, MBJK90]), demonstrates how different kinds of non-functional requirements can be associated with the relationships between a model of the system under study and different aspects of its environment. In the figure, the subject and usage "worlds" would be traditionally associated with requirements modeling, whereas the development and system worlds would be associated with specification management. In the future organizational systems we are talking about, it will be necessary to integrate these aspects.

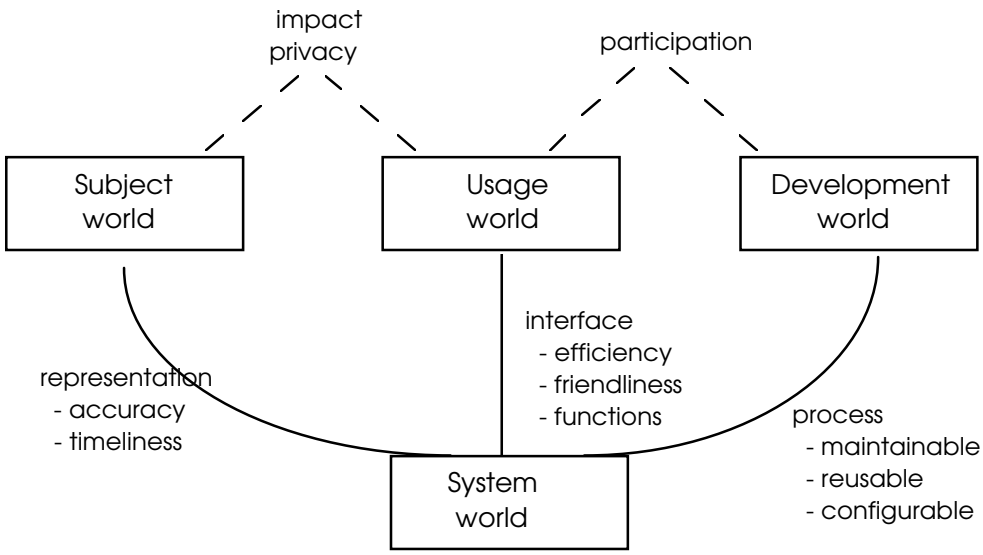


*Figure 1:* Requirements engineering and specification management

In the following sections, we review some proposed solutions to the specification management problem from a database and conceptual modeling perspective, and then proceed to show how they can be integrated via a formal conceptual meta model. We also show how such a conceptual meta model can be fruitfully applied to different tasks in a software environment organized around a concept-based software information system. The use of computer-supported conceptual modeling achieves independence of specific languages, tools, or physical environments, and helps with the integration of heterogeneous and evolving environments. Several practical experiences, though limited in scope and size, indicate the feasility of this approach.

## 2 Specification Management as a Database Task

The idea of CASE repositories is that the set of specifications constitutes a database about the information system´s data structures, operations, constraints, and development history -- in other words, *specification management is concerned with meta databases*.

The representation and implementation issues of meta databases have recently received increasing attention [JR88, STM88, CN89, SLTM91]. The meaning of a meta database is defined by its schema (the meta-meta database). This meta schema formally defines the *software engineering environment* in which specifications are created and maintained. In turn, such a meta schema has to follow some data model -- the meta$^3$ database. Like any DBMS, this "software process" data model can be considered an Abstract Data Type which consists of an object and an execution submodel. Both are influenced by the context and

organization in which their instances are used -- large communities of software developers, users, and managers.

Change, complexity, and heterogeneity of representation are ubiquitous in specification management. They must be addressed in the object submodel, e.g., by representations for versioning, configuration, and mapping between different object views. The execution model has to cope not only with the problems of multi-version concurrency control but also with explicit communication and coordination in large development teams, possibly distributed in space and time.

The meta data model that supports such a setting could in principle be very loose (e.g., hypertext), or it could require full formalization of the available language concepts, methods, and tools, even of the roles of human developers.

Probably, neither extreme should be followed. Simple hypertext structures have the advantage of being completely open with respect to the multimedia documents they represent; but they lack control of the process and easily leave the user "lost in hyperspace". Full formalization, though nice from a correctness point of view, assumes that the data model of the specification manager subsumes those of all languages used in the software environment. If possible at all, this would cause terrible performance problems for the corresponding reasoners. Moreover, the cost of knowledge acquisition may be unacceptably high; its automation is almost impossible for arbitrary combinations of languages and tools.

Therefore, a specification manager should offer what we call "controlled openness": a certain freedom to introduce new concepts, to employ different choices of software development processes, and to choose among many degrees of quality assurance; in short, a high degree of flexibility. But this freedom should be controlled by standards, methodologies, coded "good engineering practice", and the like. Moreover, it is restricted by the available languages, tools, and experiences.

The important role of both conceptual modeling and database technology in specification management is acknowledged by the fact that both, PCTE´s Object Management System [THOM88] and AD/Cycle´s repository manager [SAGA90, MM90], rely on forms of conceptual modeling (specifically, extended Entity-Relation- ship models) and place a repository at the center of proposed standard environments.

Despite these and other standardization efforts, it is by no means clear what should go into the repository and how it should be represented. Representations of software information have been offered by several disciplines: software databases, software process models, project management systems, and groupware.We briefly review each of these contributions.

## 2.1 Object Management

There are several ways to represent the objects created in a software process. Research in software object has seen a progression through three stages. Early models, e.g., those in version and configuration management [TICH88], have *files*

as their objects, at best characterized by certain standard attributes (e.g., version number, creation date, access rights). A second generation of systems considers specifications as typed and possibly structured *documents* which may be connected by various types of relationships. Still, this does not fully address the conceptual content of documents. Therefore, a third generation of systems attempts to manage an information system´s specifications and their interrelationships as an evolving *knowledge base* about the system [BJM*87, PLA89], thus providing a basis for detailed formal analysis of individual specifications, their development history, and their composition.

Regardless of these differences in conceptualization and precision, the basic services provided by object managers are roughly the same: all of them have to support the creation, analysis, composition, and evolution of the objects they understand. Of course, as the object model supports more semantics, these services become more differentiated. For example, advanced versioning models may not only distinguish between revisions (historical evolution) and variants (explored alternatives) but offer a rich set of subtypes with associated semantic controls, as well as versions not only of individual components but also of complex objects. Similarly, sophisticated mappings between different representations of specifications can be supported provided the model provides a sufficient formal foundation (e.g., algebra [EM90] or logic [MJJG91]). Let us briefly consider some concrete prototype systems.

Software databases [RW89] are concerned with the efficient management of software documents and their interrelationships. Supported by analyses such as [BERN87], many software engineers feel that databases are still too weak for their requirements and prefer to work on file directories. Object-oriented databases have been proposed as a good starting point for better database support [LK86, GPT89]. However, until they have actually lived up to their promise, a mix of file-based object storage and a relatively small database or managing relationships seems to be the compromise of choice. Even very recent contributions from the software engineering community such as the Marvel environment [KBFS88] and the hypertext system DIF [GS90] follow this hybrid approach.

The most frequently quoted limitations that render current databases unsuitable for software environments include the lack of complex objects, the need for integrating external tools, the extensibility in terms of available languages, methods, and tools, problems of consistency control, and insufficient multi-user coordination by traditional transacton concepts.

DAMOKLES [DGL86] and ORION [BCG*87] pioneered the use of *complex objects* in an extended entity-relationship approach. Specifically, the DAMOKLES object model DODM extends the entity relationship model by object structure (part-of relationships) and versions. Software entities are usually understood as representing documents. They are described by attributes, including long fields as containers for unstructured pieces, and by their composition from

sub-entity relationship diagrams. Versioning is modeled like a simple form of structural inheritance: the set of versions of an object inherits its attributes and structure but any version may have its additional attributes and structure. A large set of software-specific relationship types have been proposed, including call relationships, export/import relationships among modules, and the like. Each of these relationship types implies a particular semantics which, however, is usually not formally specified in the model (and in some cases, not enforced by the implementation either). Subsequent work in the DAMOKLES team [KDM88] has investigated a trigger concept of pattern-invoked procedures, for constraint violation detection and user-defined exception handling.

A second theme is the need for *creating and maintaining derived data*, such as compiled code, cross reference lists, dependencies among specifications and their implementation, etc. ENCORE [ZW86] and IRIS [LK86] enhanced the structural object orientation by two other salient features of object-oriented languages: the encapsulation of user-defined operations and more general inheritance than offered by the simple versioning mechanism mentioned above. IRIS also emphasized the role of user-defined functions in addition to data structure, whereas Arcadia´s CACTIS [HK87] offers selective re-computation of data derived by such functions. Tool integration is sometimes addressed by so-called "envelopes" that represent and activate the functionality of tools by literals in a rule language [KBFS88].

The many object-oriented or next-generation database manifestos indicate that there is no obvious consensus on a common data model. This lack of consensus -- and the quest for open environments -- has led many to believe that *extensibility* is an essential ingredient for specification management. To a certain degree, extensibility can be offered by the standard inheritance mechanisms of object-oriented or semantic data models. Alternatively, a very loose approach can be chosen where the user can do whatever he likes (e.g., a simple E-R model). Control over the extensions combined with a great freedom to define your own sublanguages, however, is only introduced through the use of *metalevels*.

Several models and tools exist for the definition and handling of such meta models. For example, the MetaEdit tool [SLTM91] is a graphics-based environment with a fixed meta-meta model that allows the interactive definition of meta models and their graphical representations. A limited semantics for these meta models may be provided by the usage of the meta-meta model structure (distinction between Objects, Properties, Roles, and Relationships) and by cardinality information. Fully extensible knowledge representation languages like Telos [MBJK90] allow the association of arbitrary predicative rules and constraints with meta levels at arbitrary degree of abstraction. This gives a precise semantics but yields also implementation and user presentation problems.

## 2.2 Process Management

In contrast to the more document-oriented software database models, software process models are concerned with descriptions and prescriptions how software objects are created and maintained. The sentiment expressed in [OSTE87] of running software processes like programs is an extreme example; more modest proposals include, for example, the use of Petri nets as in office systems that guide but do not fully program the process [MGDS90]. Processes must be managed at varying levels of granularity and with the consideration of the relevant man-machine interaction in a large development team. For example, IStar [DOWS87] is concerned with organizing the software process via contracts between human developers and tools.

Process management is intended to cover a wide spectrum of tasks and it is fair to say that neither an "ideal" process model nor a fully satisfactory process management technology exist today. Frequently quoted tasks include the specification of processes, the enacting and control of process specifications, the documentation and tracing of actual process histories, and the evaluation of process quality with the aim of improving the process model of an organization. One way to view some of these activities is the automation of organization handbooks -- a bureaucrat´s dream but a creativist´s nightmare.

In the ALF project [GJLO89], a comprehensive wish list for useful process modeling concepts has been derived from an analysis of the literature. It includes capturing the objects and activities, their decomposition and control structures, the synchronisation of activities through messages or object sharing, the handling of incomplete and informal knowledge, and the adaptability of the process model to changing organizational demands. From this, a comprehensive model of assisted software processes (MASPs) is derived. It uses both explicit operators and inferencing to handle activities, object sharing for cooperation, operator orderings for control, and the mutual monitoring of MASPs as a basis for adaptability.

It is interesting to observe the similarities of this model to a process model for the development of knowledge-based systems proposed by ESPRIT project KADS [AC89]. Like many software engineering methodologies, KADS observes that software development comprises a phase of increasing abstraction (requirements analysis) followed by one of decreasing abstraction (design and implementation). Domain-specific models of these phases are organized according to the four levels of domain (basic terminology and structure), inference (relationships and rules), task (goals to be achieved), and strategy (control knowledge). In comparison to KADS, the ALF approach is somewhat broader in scope but gives less specific advice on how to proceed in software development. Here, we find again the trade-offs of "controlled openness" mentioned in the introduction to this paper.

## 2.3 Project Management

The project perspective to specification management emphasizes resource and work distribution aspects such as decomposition or scheduling of tasks from the viewpoint of a project manager. Software objects are abstracted away as deliverables or milestones in a temporally arranged pattern of activities; activities are more characterized by their time and resource requirements than by their contents.

Traditional project management techniques try to fully plan ahead and then stick to the plan. Due to the high degree of innovation inherent in software development, these traditional methods usually fail in software engineering. *Proactive planning* explicitly accepts that there will be changes and organizes the project in a way that they can be accomodated as locally as possible. Algorithms for robust decomposition have been derived from relational database theory [SJ89], based on a conceptual classification of dependencies among project activities. These algorithms include the identification of necessary communication patterns when breakdowns do occur.

A prerequisite for such extensions is that the traditional predefined mathematical formalisms for project management are augmented by an underlying conceptual model that explains the numbers and thus makes them changeable in an understandable way. Work in the KBSA project [GJL*86] and the PIMS ESPRIT project [PV88] has made significant progress towards representing the relevant objects, agents, and activities in software projects. For example, the PIMS model includes concepts for Project Management Activities from the perspectives of organizational project environment, the socio-technical project work system in which activities are accomplished, and the project management system in which they are planned and monitored.

However, all project management environments of this kind see specification management from a very centralized viewpoint, that of the project manager, and do not directly support collaboration in a software project. So-called IPSEs (Integrated Project Support Environments) [BROW88] integrate project and process management aspects (and part of the object management aspects) but are only recently being extended to cover multi-user collaboration aspects as well.

## 2.4 Groupware

The above concepts for specification management are in a sense "objective". They concern observable outcomes or work activities and could be criticized as converting developers into parts of some global software productivity machinery. Even many of the so-called design transaction concepts maintain the traditional goal of transaction management to isolate user actions from each other. They do so by decomposing transactions into sub-transactions, either at design time (nested transactions [BKK85]) or when conflicts actually occur (split transactions [PKH88]). These approaches increase conflict-free parallelism but do not address the question how to deal with serious conflicts. A few systems concepts (e.g.,

SUN´s Network Software Environment [AHM89]) show that conflicts can in fact be tolerated and resolved explicitly.

In strong contrast to the centralized views of specification management, some recent approaches -- largely influenced by ideas from linguistics [WF86] and distributed problem solving [SD81] -- consider specification evolution as a side effect of communicative actions pursued by interacting agents. There is evidence that in large projects, communication is the major source of project costs and thus the major bottleneck to be addressed by specification management. Consequently, systems influenced by these ideas tend to manage conversations rather than the resulting actions and objects. Typical examples include the gIBIS discussion tool for very early phases of software development [CB88], and the Coordinator [WF86] resp. CHAOS [DDS*86] systems for contract-based work assignment and monitoring.

At first sight, it may sound strange to classify these "office communication" tools under specification management. However, firstly, the records of conversations can become so large that their administration requires database technology; and secondly, conversation records give valuable information for specification management, such as the competence and experience of developers, the expressed rationales for certain design decisions, etc. Moreover, we shall show in the next section that conversation concepts can be combined with object and process concepts to offer novel kinds of multi-user support in specification management.

## 3 Conceptually Based Software Information Management

Each of the approaches discussed in the previous section only covers certain aspects of the overall specification management problem. This section sketches an attempt to unify these aspects in a software process data model based on conceptual modelling ideas. The goal is to manage the objects and relationships in a software process and to control and partially automate the software process itself. In addition, predicative deduction rules and integrity constraints are used for the specification of the environment. This does not only yield a formal semantics for data model extensions but also gives the specification manager some of the functionality of active databases in the sense that derived data (such as configurations, compilations, etc.) can be generated automatically and checked for formal consistency.

The scope of requirements on such an integrated software process data model become clear when we consider the different granularities it has to cover:

- *Programming-in-the-small:* The discussion in section 2 showed that we need to represent not only the software objects of different languages, but also their interdependencies and, moreover, the reasons for these dependencies (non- functional requirements or goals). These reasons lead to choices among applicable generation rules or refinement alternatives, also to the application of specific formal rules and technical tools. For maintenance and reusability, it is necessary to preserve much of this information.
- *Programming-in-the-large:* To scale up, software object descriptions have to be organizable in a modular way. Modules can be seen as complex objects configured according to certain constraints from simpler components. The components, the configuration specification, and the desired properties of the module can change over time; to reduce the cost of reconfigurations, we need incremental support for the versioned management of modular objects.
- *Programming-in-the-many:* Specifications are created and maintained cooperatively by members of the development team. Cooperation can be viewed as the interaction of communicating sharing processes which have to be modeled and managed. The sharing of tasks can be handled by negotiation and contract modeling tools. The sharing of results causes problems such as workspace management and result integration in case of conflicts. Only result sharing has been addressed by database transaction management, and in a limited fashion.

As a unifying concept of which all these problems can be viewed as facets, we have chosen the concept of *design decision*: something that is based on goals, created by conversations, leads to derived objects and dependencies, can be classified into different kinds, and is executed via contracts in a distributed fashion. The versatility of the design decision concept has already been observed by other researchers [MOST85, CURT86] and was exploited by some specific tools.

### 3.1 The CAD$^O$ Model

*CAD$^O$* stands for *Conversations among Agents on Decisions over Objects*. *CAD$^O$* evolved from a simpler decision-object-tool model which did not explicitly consider the modeling of human developers [JJR90]. This model, in turn, was based on an object-dependency model gained from a large case study in the oil industry [DJ85] which resembles the "artifact-deliberation" model proposed in [PB88]. In the sequel, we formally define the four main components of the CAD$^O$ model, plus some auxiliary metaclasses needed to organize the semantical description of these components.

All models are formally expressed in the conceptual modeling language Telos [MBJK90]. For CAD$^O$, Telos´ most general built-in system class `Proposition` is extended to enable the representation of derived data and of action triggering. The semantics of these additional built-in attribute categories is defined through Telos´ temporal sublanguage; the implementation guarantees that these integrity constraints are always satisfied:

```
Proposition in Class with
   attribute
        attribute : Proposition;
        trigger : Action;
        dependsOn : Proposition
   integrityConstraint
        triggerActivation : $ this costarts this.trigger $;
        objectLifetime : $ this during this.dependsOn $
end
```

***Objects.*** The scaling from programming-in-the-small to programming-in-the-large requires a model of versioned modular objects. Due to the possible diversity and size of a software object base, CAD$^O$ only considers limited semantic descriptions of the actual software objects and provides pointers to containers where the actual objects can be accessed by tools that understand them. Thus, we have the following formal definition:

```
Individual Object in M2_Class with
  attribute
        objsemantics : Proposition;
        objstore : Directory;
        part : Object
  integrityConstraint
        partDef :
          $ exists cd/Decision forall p/Object (p in this.part
             and this in cd.output ==> p in cd.functionalInput)
$
end
```

As defined, a semantic description can be any Telos proposition, i.e., an arbitrarily simple or complex object. It can belong to certain classes, have

attributes, and contain rules, constraints, and temporal information. The semantic description is utilized to retrieve software objects in the knowledge base. As expressed in the above integrity constraint, it also takes the role of an interface specification of a possibly complex module. Consistent configurations of component objects must satisfy this specification.

Conversely, the interface specification of an object is used to reason about possible configurations of which this object is a component. The configuration relationship itself is represented as being created by a specific type of design decision (see below).

Instances of the metaclass `Object` specify what the specification manager knows about the language concepts available in a particular development environment. For example, a meta model for a DAMOKLES-like Entity-Relationship approach where entities may themselves be ER-diagrams (without versioning) might look as follows:

```
    Individual ER-Model in Object with
      objstore
           : ER-Directory
      objsemantics
           entities : Entity;
           relationships : Relationship
      deductiveRule
           : $ forall r/this.relationships e/ this.entities
                (e in r.roleObject ==> r in e.roles)
    end

    Individual Entity in Class with
      attributes
           Eattributes : Built-in-Token;
           keyAttr : Built-in-Token;
           roles : Relationship
      integrityConstraint
           : $ forall k/keyAttr (k in Eattributes) $
    end
    Individual Relationship in Class with
      attributes
           Rattributes : Builtin-Token;
      objsemantics, necessary
           roleObjects : Object
    end

    Individual One-to-Many isA Relationship with
      integrityConstraint
           : $ exists r/this.roleObjects (r in Class!singleValued)
$
    end

    Individual One-to-One isA One-to-Many with
      integrityConstraint
           : $ forall r/this.roleObjects (r in Class!singleValued)
$
    end
```

`necessary` and `singleValued` are attribute categories that have here been defined as attributes of Telos´ system class `Class` through appropriate metalevel integrity constraints [MBJK90]. The use of `necessary` ensures the referential

integrity inherent in the ER model, whereas the use of `singleValued` distinguishes the different cardinality constraints (`Relationship` is intended to denote the class of many-to-many relationships). The special variable `this` denotes any instance of the class being defined. `Builtin-Token` is the class that contains unstructured built-in attribute values (with subclasses String, Integer, etc.) but no classes or user defined token objects (which might themselves have attributes and are therefore not attributes according to the ER model). With a meta model graphics editor such as MetaEdit [SLTM91] or GraFIc [ROUG91], it would now be quite easy to associate presentation symbols to the above classes, thus automatically generating a structure browser or editor for ER models.

***Decisions*.** Knowledge about the software development process, strategies, tactics and constraints, constitutes an essential ingredient of specification management. CAD$^{O}$ defines these as design decision classes. A design decision is considered a special design object which can be subject to other design decisions. Its semantic description relates the descriptions of its input and output objects by dependencies. A special kind of design object, called `goal`, governs development from non-functional requirements:

```
Individual Decision isA Object with
   attribute
        from, to : Object;
        nonfunctionalInput : Goal;
        objsemantics : Proposition!dependencies
   integrityConstraint
        rightDependencies :
         $ objsemantics in to!dependsOn and
           (to(objsemantics) in from or
            to(objsemantics) in nonfunctionalInput) $
end

Individual Goal isA Object end
```

Instances of the metaclass `Decision` define a particular methodology for working on the object classes they refer to. Decision classes for programming-in-the-small define methods for refinement within a language and for mapping between languages; their semantic descriptions are dependency classes. For example, the following decision classes represent a methodology that maps simple entity-relationship models without partOf structure to corresponding relational database schemas. We hope that the underlying metamodel of relational databases is understood by the reader without explicit definition.

```
Individual ERMap in Decision with
  from
       : ER-Model
  to
       : Rel-Model
  deductiveRule
       entityMap : $ forall e/Entity
                        (e in this.from.entities
                         ==> e in this.to.relations
                         and e.relkey = e.keyAttr
                         and e.relattributes = e.Eattributes) $
```

```
        relMap : $ forall r/Relationship
                        (r in this.from.relationships and
                         not (r in One-to-Many)
                         ==> r in this.to.relations
                         and r.relkey = r.roleObjects.keyAttr
                         and r.relattributes = r.Rattributes) $
end
```

An optimized version of this mapping might be represented as a specialization of `ERMap` with rules or constraints that merge the relations derived from a binary one-to-many relationship with those for the entity on the "many" side. The rationale behind such a specialized methodology could be represented by instantiating the attribute category `nonfunctionalInput` by the class `avoidjoins` which may be a specialization of the `Goal` class `efficiency`.

Similarly, one may introduce hierarchies of decision classes for programming in-the-large: a specialization hierarchy of versioning decisions and a specialization hierarchy of configuration decision classes (including vertical composition by mapping decisions, horizontal composition within a representation, etc.). Both are treated orthogonally; that is, we can build versions of configurations from versions of components. For example, the following definition clarifies the distinction between the concepts of revision and variant in version management: a revision decision changes an object to a new version, whereas a variant decision revises a previous revision decision ( `oldObject` is an instance of from defined with the Class `Revision`).

```
Individual Variant in Decision with
  from
       oldRevision : Revision
  to
       alternativeRevision : Revision
  integrityConstraint
       : $ this.alternativeRevision.oldObject =
            this.initialRevision.oldObject $
end
```

***Agents.*** Functional as well as nonfunctional specifications do not come out of nowhere but result from ideas surfaced in conversations among the agents involved in the process. In CAD$^{o}$, agent objects represent the organizational principles of the active components in a development environment. The model distinguishes two kinds of agents: human (groups of) developers, and technical agents (tools). Following the idea of multi-layered representation of software knowledge, we define an agent as a mapping from its qualification to certain actions that implement this qualification; a similar approach to tool modeling -- the use of rules as "envelopes" for tools -- is followed in Marvel [KBFS88] which, however, does not model its human users.

Human agents and their actions are represented in much the same way. However, as discussed below, they are the only ones that can participate in certain kinds of conversations, their qualification is generally broader but less postconditions of their actions can be guaranteed.

The qualification of an agent specifies preconditions (`functionalInput`), post- conditions (`Output`), and non-functional qualities (e.g., efficiency, user friendliness, reliability) of agents. Formally, we have (constraints omitted for simplicity):

```
Individual Agent in Decision with
   from
        qualification : Decision
   nonfunctionalInput
        quality : Goal
   to
        realization : Action
end
```

Note that the class `Agent` is defined one meta-level below that of `Object` or `Decision`. Tool descriptions can thus be instantiated by the documentation of a particular tool application. If correct, the tool application leads to the instantiation of the corresponding decision class; otherwise, the knowledge base does not accept the decision as executed but leaves the results intact as separate objects. This way, the agent has a chance to correct minor errors without redoing large amounts of work.

***Conversations*.** So far, decisions have been characterized as having effects on objects and being executed by agents. But where do the decisions come from and how do we get agents committed to execute them? Also, can we use the decision concept to ensure the correctness of parallel work?

We answer these questions by saying that decisions are also the subject of conversations among agents. Conversations are considered as special kinds of objects which evolve due to special kinds of decisions called messages. Instances of this metaclass specify protocols for message exchange among agents and thus define cooperation structures in a software environment:

```
Individual Conversation isA Object with
   attribute
        participants : Agent;
        subject : Decision;
        objsemantics : ConversationState
end

Individual ConversationState in M2_Class with
   attribute
        transition : ConversationState
end

Individual Message isA Decision with
   attribute
        from, to : Conversation;
        decsemantics : ConversationState!transition;
        next : Message;
        talksabout : Decision;
        sender, receiver : Agent;
   integrityConstraint
```

```
        rightTopic :
              $ from = to and talksabout = to.subject $;
        rightPeople :
              $ sender in to.participants and
                receiver in to.participants $
end
```

Messages cause state transitions in conversation protocols among a specified but time-varying set of agents, the participants. Conversations and messages can be considered as a means for *sharing*. As indicated above, several concepts have to be shared in a software project; we therefore distinguish process descriptions for *idea sharing* (conversations for possibilities), *task sharing* (conversations for action), and *result sharing* (design transactions) [JMR92].

*Idea sharing* is used to prepare design decisions and to document their rationales. Its associated message classes define a multi-lateral argumentation protocol according to one of the well-known theories of argumentation put forth by [TOUL58] (used, e.g., in SYNVIEW [LOWE85]) or [KR70] (used in gIBIS [CB88]). For example, the IBIS model includes message types such as *Issue*, *Position*, and *Argument* which may relate to each other in several different way specified by the protocol. As demonstrated by systems such as SYNVIEW, gIBIS, or Object Lens [LMY88], such models can be used to control a structured electronic mail tool or a special graphical editor; inclusion of multimedia (recorded video or speech) is possible and sometimes useful.

The advantage of our integrated approach over purely conversation-based ones is its formal relationship to the design decisions and thus the object model that is the subject of the discussion; only this integration yields the possibilities to use the tools for capturing the rationale for design decisions, and for rationally changing plans midway through their execution. CAD$^0$-based argumentation tools have been built for both Toulmin´s and Rittel´s models [HJR90, RD91].

*Task sharing* and *result sharing* are closely intertwined processes that concern the distributed execution of decisions. Task sharing concerns the distribution of work and the monitoring of work contracts in a development team. In our implementation, the Telos representation of a contract negotiation and monitoring protocol as in [WF86] is related to a specialized email tool as well as to the result sharing submodel. The creation of a contract does not only regulate the distribution of tasks but also assigns them to agent workspaces (represented as specific kinds of design objects).

Result sharing defines the distribution of access rights within the development team, i.e., the transfer of objects among workspaces. This has been the traditional topic of concurrency control in design databases. Two solutions have been proposed: pessimistic multi-version methods such as checkout/checkin, and optimistic methods which pursue a copy-merge paradigm in which parallel work on the same objects is allowed and controlled through explicit communication [AHM89].

In our integrated approach, task sharing and result sharing protocols are tightly coupled through rules and notification servcies. Acceptance of a contract is

associated with check-out of the objects required for the task. Upon completion of the contract, results are merged into a higher-level workspace, or directly into the project database. This integration of check-out/ check-in with the contracting protocol leads to an optimistic concurrency control procedure with conflict tolerance and forward conflict resolution without costly undo procedures. Agents may also take the initiative and offer the results of unscheduled work without prior contract.

Figure 2 summarizes the CAD$^o$ model in a semantic network structure.

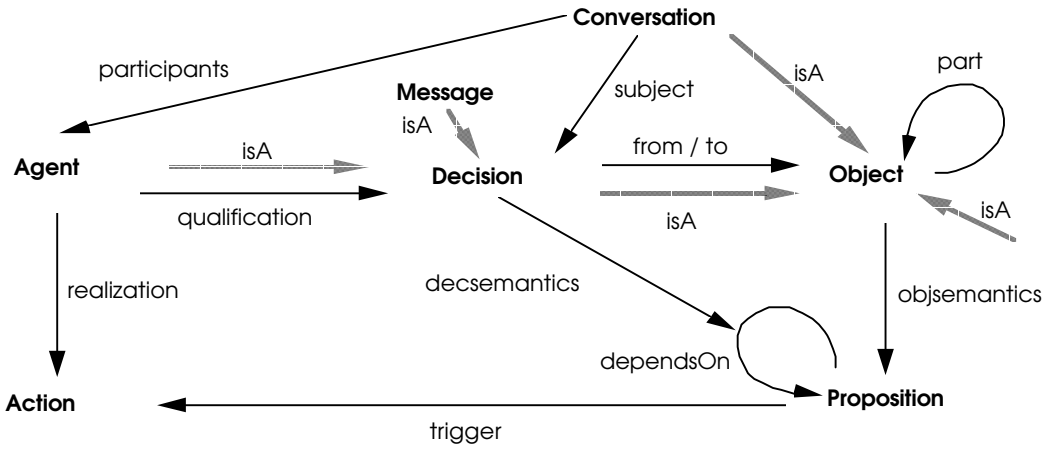


*Figure 2*: The concepts of CAD$^o$

## 3.2 ConceptBase: A Specification Management Environment

This section complements the description of our example meta model by briefly sketching the functionalities of a specification management built in our group. *ConceptBase* [JARK91] employs a client-server architecture which works on both local and wide area networks under the internet protocol. The ConceptBase server manages a Telos knowledge base of software environment descriptions (represented as Telos metaclasses), software process documentation (simple classes) and example objects (tokens) according to the CAD$^o$ model. All "official" activities in a software project must be documented in this knowledge base and are therefore controlled by the model.

The knowledge base contains only abstract descriptions of design objects, decisions and tools. The actual objects appear elsewhere. Design objects, which may be expressed in any language defined as an instance of `Object`, are managed in document libraries, implemented using commercial file-based version and configuration management tools. Design decisions are carried out in workspaces assigned to software developers. Design tools are integrated as clients in the network and described to the knowledge base by their qualification as stated in the definition of `Agent`.

Two major groups of client processes have to be considered in a CASE environment, one concerned with programming-in-the-small, the other supporting more general decision classes associated with programming-in-the-large and programming-in-the- many issues.

***Programming-in-the-small.*** Tools for programming-in-the-small activities are modeled as so-called versioning assistants. They support specific kinds of versioning decisions in-the-small, e.g., the refinement of a particular object, the mapping between different levels of specification, and local corrections resp. enhancements to existing models. In CASE environments, such tools are usually graphics-oriented or rely on language-sensitive editors to reduce errors, increase productivity, and ensure proper documentation with minimal effort on the part of the developers. Tools are activated from a workspace with respect to a particular design decision. They are provided with relevant data from the document libraries and the server. The server also informs the assistants where to store results and checks whether tool usage corresponds to correct instantiation of the decision class. Briefly, the usage cycle for programming-in-the- small operates as follows:

- A tool is selected from a decision class via its qualification attribute; the `container` attributes of input objects for that decision class determine the input data for the tool activation.
- The tool is run, creating a tool instance in the knowledge which documents its presence.
- When the tool completes its execution, it documents its activities by instantiating the decision class that caused its execution, its results, and a Telos description of these results; ConceptBase checks the integrity of the attempted instantiations.
- Control is returned to ConceptBase and the tool object instance is terminated.

***Programming-in-the-large/ Programming-in-the-many.*** In order to create and maintain the information about version and configuration decisions and their relationships to agents and conversations, a second set of clients has been constructed. These clients are intended to support cooperative development-in-the-large at a conceptual level. The following brief scenario describes the usage of the five assistants belonging to this sub-environment [RJG*91]:

- Decisions to configure a system or to alter an existing system are prepared using an *argument editor* [HJR90, RD91]. For subsequent maintenance, selected crucial recorded arguments are retained to document the rationale for design decisions.
- A *proactive planning assistant* supports decision makers in the robust decomposition of work tasks into "islands of control". Each island should be assigned to an appropriate agent such that this agent can locally absorb as much unexpected change as possible and the need for reorganization is minimized. The algorithm for this is a normalization procedure in the spirit

of relational database design theory, using specific dependency classes between work modules [SJ89].

- A *contracting assistant* supports the assignment of tasks to agents and monitors accomplishment of contracted tasks through a conversation-for-action protocol. Through integration with a result sharing protocol, this assistant is also responsible for notifications and similar coordination mechanisms in case of conflict caused by parallel work on related objects. It offers an optimistic concurrency control scheme with merge-based conflict resolution implemented on top of SUN's Network Software Environment [AHM88].
- A graphics-based *(re-)configuration assistant* supports the initial creation and incremental versioning of consistent complex objects with visualized constraint propagation and integrity control. The structure of configurations (the so-called configuration thread) is specified by a configuration decision class. Consistency of configuration decisions is defined by consistency constraints between interface and component specifications, by compatibility constraints among different components, and by derivation rules for automatic completion of components.

All of the above assistants work at the conceptual level only. An *implementation assistant* is responsible for the mapping of conceptual object structures and decision classes to physical data structures and operations of commercial version and configuration managers or software databases. These object stores maintain the document containers and derived objects, such as compiled programs or configurations of linked object code.

## 4 Conclusions

Specification management models requires specific execution submodels as well as object submodel but also an execution submodel. Our brief survey showed that most existing systems emphasize one at the expense of the other, some offering sophisticated object models with relatively simple design transaction concepts, others supporting very advanced process specifications without due considerations of the underlying object model.

CAD$^{O}$ combines a model of consistently configured versioned modular objects with a conversation-based teamwork model for cooperative execution; the notions of design decision and of communicating sharing processes is the basis for an apparently happy marriage between object and execution model. A family of prototypical tools has been produced in the context of the ConceptBase software information manager to validate our ideas. However, there is little reason to be overly optimistic about a short- term solution to the specification management problem.

The problems that the AD/Cycle and PCTE approaches are facing in reaching agreement on the specific concepts to be offered for specification management

indicate the difficulties of this topic, especially in an era of constantly evolving CASE languages, methods, and tools. How can one evaluate something as abstract as a metamodel of specification management?

The evaluation approach we have followed for CAD$^o$ was a constant trial and error process with various versions of the model, ranging from a simple object-dependency model [DJ85] via DAIDA´s decision-object-tool model [JJR90] to the present concept. Each of these steps was triggered by the results of substantial case studies and formal analyses. For example, a complete meta schema of the DAIDA environment has been constructed [JMSV91], using a subset of CAD$^o$, and a number of medium-sized application projects were modeled as instances of this meta schema. These experiences pointed out the knowledge acquisition problems faced by such environments and resulted in the investigation of using extended syntax editors for this purpose. Using the full CAD$^o$ model, the DAIDA environment model has been extended to deal with programming-in-the-large and programming-in-the-many. Furthermore, a major case study was conducted concerning the portation of several medium-to-large software systems from one environment to another, and parts of the model are being used as a basis for project management in several research groups that use ConceptBase, including our own.

A lot has been learned from these experiments but by their very nature they cannot prove the "correctness" of our approach. In particular, reasonable instantiations of the abstract meta meta model are an active area of research. Adequate object descriptions, dedicated software process and project management structures, as well as more effective group tools in both realtime and asynchronous settings are needed, and CAD$^o$ is just one possible approach to integrate these perspectives conceptually. Nevertheless, our experiments seem to allow at least the general conclusion that conceptual modeling supported by extended database technology is a strong candidate for future specification management environments.

## References

[AHM89] Adams, E.W., Honda, M., Miller, T.C. (1989). Object management in a CASE environment. *Proc. 11th Intl. Conf. Software Eng.,* Pittsburgh, Pa, 154-163.

[BKK85] Bancilhon, F., Kim, W.,. Korth, H.F. (1985). A model of CAD transactions. *Proc. 11th OIntl. Conf. Very Large Data Bases,* Stockholm, Sweden, 25-33.

[BCG*87] Banerjee, J., Chou, H.-T., Garza, J.F., Kim, W., Woelk, D., Ballou, N., Kim, H.-J. (1987). Data model issues for object-oriented applications. *ACM Trans. Office Information Systems 5,* 1, 3-26.

[BE87] Belkhatir, N., Estublier, J. (1987). Software management constraints and action triggering in the Adele program database. *Proc. First Europ. Conf. Software Eng.*, Strasbourg, France, 44-54.

[BERN87] Bernstein, P.A. (1987). Database support for software engineering. *Proc. 9th Intl. Conf. Software Eng.,* San Francisco, Ca.

[BJM*87] Borgida, A., Jarke, M., Mylopoulos, J., Schmidt, J.W., Vassiliou, Y. (1987). The software development environment as a knowledge base management system. In Schmidt, J.W., Thanos, C. (eds.): *Foundations of Knowledge Base Management,* New York: Springer-Verlag.

[BROW88] Brown, A.W. (1988). Integrated project support environments. *Information & Management 15,* 2, 125-134.

[CB88] Conklin, J., Begeman, M.L. (1988). A hypertext tool for exploratory policy discussion. *ACM Trans. Information Systems 6,* 4, 303-331.

[CGG90] Chroust, G., Goldmann, H., Gschwandtner, O. (1990). The role of work management in application development. *IBM Systems J. 29,* 2, 189-208.

[CKM*91] Chung, L., Katalagarianos, P., Mertikas, M., Mylopoulos, J., Vassiliou, Y. (1991). From information systems requirements to designs: a mapping framework. *Information Systems 16,* 4.

[CN89] Chen, M., Nunamaker, J.F. (1989). MetaPlex: an integrated environments for organization and information systems development. *Proc. 10th Intl. Conf. Information Systems,*Boston, Mass, 141-151.

[CURT86] Curtis, B. (1986). Models of iteration in software development. *Proc. 3rd Intl. Software Process Workshop,* Breckenridge, Co, 53-56.

[DDS*86] DeCindio, F., De Michelis, G., Simone, R., Vassallo, R., Zanaboni, A. (1986). CHAOS as a coordination technology. *Proc. CSCW ´86,* Austin, Tx, 325-342.

[DGL86] Dittrich, K., Gotthard, W., Lockemann, P.C. (1986). Damokles - a database system for software engineering environments. *Proc. Intl. Workshop Advanced Programming Environments,* Trondheim, Norway.

[DJ85] Dhar, V., Jarke, M. (1985). Learning from prototypes. *Proc. 6th Intl. Conf. Information Systems*, Indianapolis, Ind, 114-133.

[DOWS87] Dowson, M. (1987). Integrated project support with IStar. *IEEE Software 4,* 4, 6-15.

[EM90] Ehrig, H., Mahr, B. (1990). *Fundamentals of Algebraic Specifications -- Module Specifications and Constraints,* Heidelberg: Springer-Verlag.

[ES88] Endres, R., Schneider, M. (1988). The GRASPIN software engineering environments. *Proc. ESPRIT ´88 Conf.,* Brussels, Belgium, 349-364.

[GJLO89] Griffiths, P., Jamart, P., Legait, A., Oldfield, D.E. (1989). The ALF approach to process modeling. *Proc. ESPRIT Conf.,* Brussels, 543-553.

[GS90] Garg, P.K., Scacchi, W. (1988). A software hypertext environment for configured software descriptions. In Winkler, J. (ed.): *Intl Workshop on Software Version and Configuration Control,* Grassau, Germany.

[HEDB91] Hedberg, B. (1991). The role of information systems in imaginary organizations. *Proc. IFIP TC8 Conf. Collaborative Work, Social Communications, and Information Systems,* Helsinki, Finland, 1-8.

[HJR90] Hahn, U., Jarke, M., Rose, T. (1999). Group work in software projects: integrated conceptual models and collaboration tools. *Proc. IFIP WG8.4 Conf. Multi-User Applications and Interfaces,* Iraklion, Greece, 83-101.

[HK87] Hudson, S.E., King, R. (1987). Object-oriented database support for software environments. *Proc. ACM-SIGMOD Conf.,* San Francisco, Ca.

[JARK90] Jarke, M. (1990). DAIDA -- conceptual modeling and knowledge-based support for information systems development processes. *Theorie et Science Informatique 9,* 2, 121-133.

[JARK91] Jarke, M., ed. (1991). ConceptBase V3.0 user manual. Report MIP-9106, Universität Passau.

[JJR90] Jarke, M., Jeusfeld, M., Rose, T. (1990). A software process data model for knowledge engineering in information systems. *Information Systems 15,*1, 85-116.

[JMR92] Jarke, M., Maltzahn. C., Rose, T. (1992). Sharing processes: integrating agent and object worlds. to appear in *Intl. J. Intelligent and Cooperative Information Systems 1,* 1.

[JR88] Jarke, M., Rose, T. (1988). Managing knowledge about information system evolution. *Proc. ACM SIGMOD Conf.,* Chicago, Il.

[KBFS88] Kaiser, G.E., Bhargouti, N.S., Feiler, P., Schwanke, R.W. (1988). Database support for knowledge-based engineering environments. *IEEE Expert 4,* 2, 18-32.

[KBSA88] *Proc. 3rd RADC Knowledge-Based Software Assistant Conference*, Utica, N.Y.

[KDM88] Kotz, A.M., Dittrich, K.R., Mülle, J.A. (1988). Supporting semantic rules by a generalized event/trigger mechanism. *Proc. EDBT ´88,* Venice, Italy, 76-91.

[LK86] Lyngbaek, Kent, W. (1986). A data modeling methodology for the design and implementation of information systems. *Proc. Intl. Workshop Object- Oriented Database Systems,* Pacific Grove, Ca.

[LMY88] Lai, K.-Y., Malone, T.W., Yu, K.-C. (1988). Object lens: a "spreadsheet" for cooperative work. *ACM Trans. Office Information Syst. 6,* 4, 332-353

[LOWE85] Lowe, D. (1985). Cooperative structuring of information: the representation of reasoning and debate. *Intl. J. Man-Machine Studies 23,* 1, 97-111.

[MBJK90] Mylopoulos, J., Borgida, A., Jarke, M., Koubarakis, M. (1990). Telos: a language for representing knowledge about information systems. *ACM Trans. Information Systems 8,* 4.

[MBW80] Mylopoulos, J., Bernstein, P., Wong, H. (1980). A language for designing interactive data-intensive applications. *ACM Trans. Datab. Syst. 5*, 2, 185-207.

[MGDS90] Madhavji, N., Gruhn, V., Deiters, W., Schäfer, W. (1990). PRISM = methodology + process-oriented environment. *Proc. 12th Intl. Conf. Software Eng.,* Nice, France.

[MJJG91] Miethsam, A., Jeusfeld, Jarke, M., Gocek, M. (1991). Constrained abduction as a basis for configuration management and reusability. Report, ESPRIT Basic Research Action 3012 (COMPULOG), RWTH Aachen.

[MM90] Matthews, R.W., McGee, W.C. (1990). Data modeling for software development. *IBM Systems J. 29,* 2, 228-235.

[MOST85] Mostow, J. (1985). Towards better models of the design process. *AI Magazine 6,* 1, 44-57.

[OSTE87] Osterweil, L. (1987). Software processes are software too. *Proc. 9th Intl. Conf. Software Eng.,* San Francsico, Ca, 2-13.

[PB88] Potts, C., Bruns, G. (1988). Recording the reasons for design decisions. *Proc. 10th Intl. Conf. Software Eng.,* Singapore, 418-427.

[PK88] Perry, D.E., Kaiser, G.E. (1988). Models of software development environments. *Proc. 10th Intl. Conf. Software Eng.,* Singapore, 60-68.

[PKH88] Pu, C., Kaiser, G.E., Hutchinson, N. (1988). Split transactioons for open-ended activities. *Proc. 14th Intl. Conf. Very Large Data Bases,* Los Angeles, Ca, 26-37.

[PLA89] Palaskas, Z., Loucopoulos, P., van Assche, F. (1989). AMORE -- object- oriented extensions to Prolog for the RUBRIC implementation environment. *Proc. ESPRIT ´89 Conf.,* 475-489.

[PM85] Porter, M.E., Millar, V.E. (1985). How information gives you competitive advantage. *Harvard Business Review,* July-August, 149-160.

[PV89] Paris, J., Vauquois, P. (1989). PIMS: a project-integrated management system. *Proc. ESPRIT ´89 Conf.,* Brussels, Belgium, 389-402.

[RD91] Ramesh, B., Dhar, V. (1991). Group support and change propagation in requirements engineering. In Jarke M. (ed.): *Development Assistance for Database Applications*, Springer-Verlag, to appear.

[RJG*91] Rose, T., Jarke, M., Gocek, M., Maltzahn, C., Nissen, H.W. (1991). A decision-based configuration process environment. Special Issue on Software Environments and Factories, *Software Eng. J. 6,* 3.

[ROUG91] Rouge, A. (1991). GraFIc: a graphical editor for knowledge bases. In Jarke M. (ed.): *Development Assistance for Database Applications*, Springer-Verlag, to appear.

[RW89] Rowe, L.A., Wensel, S., eds. (1989). *Proc. ACM SIGMOD Workshop on Software CAD Databases,* Napa, Ca.

[SAGA90] Sagawa, J.W. (1990). Repository manager technology. *IBM Systems J. 29,* 2, 209-227.

[SLTM91] Smolander, K., Lyytinen, K., Tahvanainen, V.-P., Marttiin, P. (1991). MetaEdit -- a flexible graphical environment for methodology modeling. *Proc. CAiSE 91*, Trondheim, Norway.

[SD81] Smith, R.G., Davis, R. (1981). Frameworks for cooperation in distributed problem solving. *IEEE Trans. Systems, Man, and Cybernetics 11,* 1, 61-70.

[SJ89] Srikanth, R., Jarke, M. (1989). The design of knowledge-based systems for managing ill-structured software projects. *Decision Support Systems 5,* 4, 425-447.

[STM88] Sorenson, P.G., Tremblay, J.-P., McAllister, A.J. (1988). The Metaview system for specification environments. *IEEE Software (March), 30-38.*

[THOM88] Thomas, I. (1988). Writing tools for PCTE and PACT. *Proc. ESPRIT ´88 Conf.,* Brussels, Belgium, 453-459.

[TICH88] Tichy, W. (1988). Tools for software configuration management. *Proc. Workshop Software Version and Configuration Control,* Grassau, Germany, 1-20.

[WF86] Winograd, T., Flores, F. (1986). *Understanding Computers and Cognition,* Norwood, NJ: Ablex.

[ZW86] Zdonik, S.H., Wegner, P. (1986). A database approach to languages, libraries, and environments. *Proc. Intl. Workshop Object-Oriented Databases,* Pacific Grove, Ca.