Teamwork Support in a Knowledge-Based Information Systems Environment

Udo Hahn, Matthias Jarke, *Senior Member, IEEE*, and Thomas Rose

***Abstract*—DAIDA is an experimental environment for the knowledge-assisted development and maintenance of database-intensive information systems from object-oriented requirements and specifications. Within the DAIDA framework, the CoNeX project has developed an approach to integrate different tasks encountered in software projects via a conceptual modeling strategy. Emphasis is put on integrating the semantics of the software development domain with aspects of group work, on social strategies to negotiate problems by argumentation, and on assigning responsibilities for task fulfillment by way of contracting. The implementation of a CoNeX prototype is demonstrated with a sample session.**

***Index Terms*—Computer-supported cooperative work, group problem-solving, information systems environments, knowledge-based software engineering, multiagent conversations.**

I. Introduction

THE emerging paradigm of computer-supported cooperative work [17] seems to imply a shift of attention in the area of computer support for software development teams. First, software development by teams, although constrained by technical requirements, is recognized as a social process comprised by the (in)formal interactions of the members of a team (*group work*) within an organizational setting. Second, social processes (*work procedures*) in task-oriented groups underlie particular conditions for negotiations, commitments, and responsibilities that are essential for smoothly accommodating to dynamic changes of the project environment, planning faults, etc.

From an engineering point of view, many tools are already available to support software development processes [42]. These include traditional project management software as well as tools for resource management (TAME [22]), contract control (IStar [12]), or hypertext facilities for project documentation (DIF [14]). However, they tend to have a rather centralized view of project planning and also lack of a sound coverage of the knowledge of the software domain. Procedures are documented or changes are displayed on a mostly syntactic level while semantic issues (reasons for changes, constraint violations, and conflicts) are beyond the scope of these devices [31]. Often, they provide island solutions incapable of being integrated above the low level of language facilities. Some recent proposals to improve this situation are still at the design stage (e.g., ISHYS [15]) or focus on support for the project manager rather than for the whole project team (e.g., PIMS [26] or DesignNet [27]).

Only relatively recently, theories and tools are appearing that take a less centralized viewpoint of software projects and aim at accounting for the human factor inherent in their work procedures. Starting with [19], people began to explicitly model software development as a collaborative activity, and to see the software development environment as a forum of communication. The role of participative project management, creating win–win situations for all stakeholders, has been also emphasized by recent work on "Theory-W" project management [3], but this discussion covers management issues, not tools.

Formal foundations for this approach consider notions from speech act theory [38] and more *ad hoc* conversational models as the basis of typed messaging or conferencing systems [29]. Especially in the requirements phase, the usefulness of such tools is now generally acknowledged [10], [34]. However, a pure conversation perspective has been criticized as being too one-sided. Content aspects are neglected and the communication protocol can take the same dictatorship-like role accorded to traditional centralized project management software. Also, the style of conversation to be supported (strict coordination [44], [11] versus open argumentation [28], [9]) is subject to much debate.

Based on an analysis of the above approaches, we defined four requirements:

1) Activities of members of a software development team require support beyond basic multiuser facilities (normally used to partition teams with standard schemes of concurrency control)—support of group interactions must account for human collaboration techniques such as negotiations, commitments, and responsibility contracts.
2) Interactions of groups require support beyond the formal level of technical communication lines (e.g., e-mail, electronic conferencing systems)—the social protocols that underlie group communication have to be accounted for in terms of human strategies and policies for argument exchange, contract assignment, decision making, etc.

Manuscript received August 8, 1990; revised January 7, 1991. Recommended by P. A. Ng. This work was supported in part by the Deutsche Forschungsgemeinschaft under Grant Ja-445/1-2 and in part by the Commission of the European Communities under Esprit Contract 892 (DAIDA). This paper is a revised and expanded version of a contribution to the IFIP 8.4 Working Conference on Multiuser Interfaces and Applications, held in Iraklion, Greece, September 24–26, 1990.

U. Hahn was with the University of Passau, P.O. Box 2540, 8390 Passau, Germany. He is now with the University of Freiburg, Werthmannplatz, 7800 Freiburg, Germany.

M. Jarke was with the University of Passau, P.O. Box 2540, 8390 Passau, Germany. He is now with Lehrstuhl Informatik V, RWTH Aachen, Ahornstr. 55, 5100 Aachen, Germany.

T. Rose was with the University of Passau, P.O. Box 2540, 8390 Passau, Germany. He is now with the Department of Computer Science, University of Toronto, Toronto, ON M5S 1A4, Canada.

IEEE Log Number 9143151.

3) Tools require content-oriented specification of knowledge beyond language facilities—proper tool support and properly controlled tool integration mechanisms must consider domain knowledge of the underlying software project, work procedures and languages used for specification, design, and implementation.
4) Finally, all these single modeling efforts have to be combined. The whole process of software development is stratified at several layers (requirements analysis, workpackage planning, programming, etc.), each one covered by a specialized model. These submodels have to be integrated within a composite formal model of software project management to guarantee that transitions between these submodels are also under formal control. This is important for later replay, discussion, or refutation of reasons for actions on which any rational account of human group work is based.

These requirements seem to be in accord with the demands of [3] for separate consideration of people and problems, for the documentation of objective criteria, and for the expression of interests rather than just positions or decisions. The term IPSE (integrated project support environment), although overused, clearly points out the perceived need for integrating these different aspects. However, realizing this idea has proven difficult [6]. Based on previous work both in group decision support and in knowledge-based software engineering, the CoNeX project at the University of Passau has pursued the development of integrated models and multiuser tools for software projects; CoNeX stands for "*Co*ordination and *Ne*gotiation support for e*X*perts in design applications."

A CoNeX prototype has been implemented as an extension to the ConceptBase software information system [13] which serves as the central knowedge manager for the DAIDA information systems development environment [20]. This integration with an advanced software environment gave us the opportunity for extensive experimentation. We have distributed the prototype to a number of industrial and research sites and have begun to use it for the organization of software development within our own group and for the management of programming classes in our university. Although these experiences are by no means representative, they do seem to indicate that the system has made it easier to integrate new people into the team, and that it simplifies substantially the integration of heterogeneously developed software through a better understanding of design rationales.

In this paper, CoNeX is mostly presented from a user perspective; a companion paper describes technical details of configuration process management in the same system context [36]. After a brief overview of the DAIDA environment (Section II), a CoNeX sample session (Section III) illustrates the diversity of group support requirements. Section IV presents the conceptual models used in CoNeX to satisfy these requirements. From these models and additional practical demands, Section IV derives a general implementation concept and describes specific multiuser tools. Section V presents applications and a comparison with other work, while Section VI summarizes some conclusions.

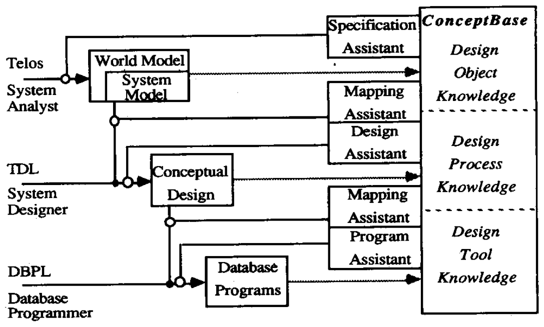


Fig. 1. Architecture of the DAIDA environment.

II. The DAIDA Environment

In this section we give a brief overview of the DAIDA environment in which the CoNeX subsystem is embedded and introduce an example we are going to use throughout the paper.

A. DAIDA Overview

DAIDA is an experimental software environment for database-intensive information systems (DAIDA = *D*evelopment *A*ssistance for *I*nteractive *D*atabase *A*pplications). It was developed over the past four years by a consortium of software houses and research partners within the ESPRIT program of the European Community.

In DAIDA, software is represented as a layered knowledge base where the layers correspond to different interrelated views of a system (Fig. 1) [20]:

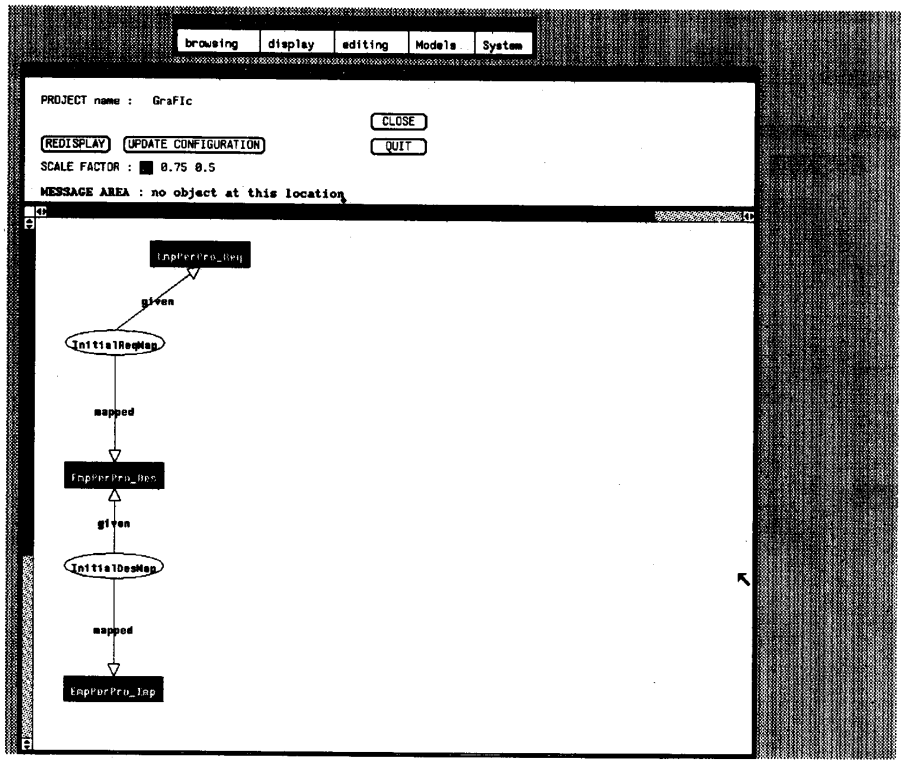
- Various application perspectives are captured in a requirements model which represents the different system roles as part of an evolving world model in the knowledge representation language Telos
- An overall systems perspective is represented as a conceptual design in the object-oriented specification language TDL
- Subsystems are implemented in the Modula-based database programming language DBPL.

These layers are related by knowledge-based mapping assistants that help the developer to:

- Embed system functions in a world model and integrate the different system roles in a conceptual design, according to functional and nonfunctional requirements [8]
- Derive formally verified database programs from conceptual designs [5].

Obviously, such a multilayered and highly interrelated structure needs strong process support not only in the initial development of software but also for maintenance with the reuse of existing experiences. The DAIDA architecture therefore contains a central knowledge-based manager, *ConceptBase*, which is responsible for the coordination, control, and documentation of work in the DAIDA environment.

The name ConceptBase is programmatic: our main goal was to raise process management from a file or document level to an application-oriented conceptual organization. This orientation toward concepts enables the formal and technical integration of several process management tasks:



PROJECT name : GraFIc

SCALE FACTOR : ■ 0.75 0.5

MESSAGE AREA : no object at this location

Fig. 2. Three-level representation of example information system.

- Distribution of development process information over heterogeneous object managers
- Scaling up from detail tasks such as documentation of refinement proofs to version and configuration management and group coordination within the same formal framework
- Management of evolving environments and their process models, as well as control, enacting, and documentation of individual software processes.

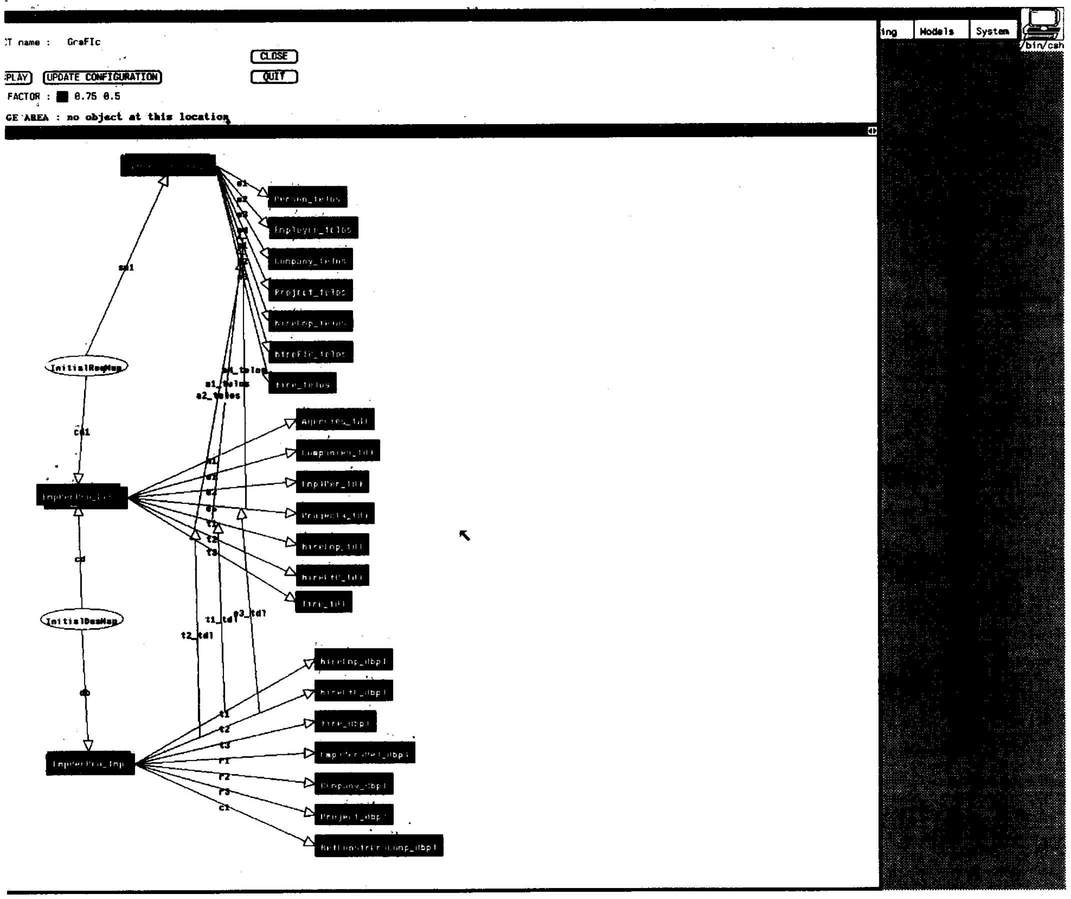
In the CoNeX project we have focused on the group coordination aspects of these tasks, as well as on their integration with the other aspects.

B. A DAIDA Development Example

Consider a group of programmers, designers, system analysts, and prospective users collaborating on the development of an information system. The system under development is a conventional personnel management system involving projects and people, assigning people to tasks, etc. According to the DAIDA architecture, it is described from three different viewpoints which are related by development decisions (Fig. 2). The requirements analysis object `EmpPerPro_Req` is a model of the personnel management domain into which the system will be integrated after completion. These requirements are transformed into a corresponding conceptual design `EmpPerPro_Des` which provides a system perspective and is the result of a mapping decision `InitialReqMap`. This conceptual design forms the basis for a database program implementation in terms of three different relations (`EmpPer-Pro_Imp`). This implementation has been created through the choice and subsequent execution of a decision documented as `InitialDesMap`.

Two features should be pointed out here. First, requirements, designs, and implementations are formal objects of a software development knowledge base. Second, the transformations from requirements to designs and from designs to implementations are formally controlled by mapping decisions covered by a conceptual software development model [21].

To explore the semantics of design histories like the one shown in Fig. 2, the DAIDA environment provides additional models and tools. For instance, a zooming operation (Fig. 3) shows semantic descriptions of the objects in Fig. 2. The requirements model is composed of four entity classes (`Person`, `Employee`, `Company`, `Project`) and three transactions (`hireEmployee`, `hireEmployeeforProject`, `fireEmp`). `Person` and `Employee`, and `hireEmp` and `hireEfP` are related by specialization (isA) relationships (not shown). In the design and implementation, these two isA hierarchies have been treated differently. While designers have merged `Person` and `Employee` into a single `EmplPers` data class of the design, each activity object of the requirements model (`hireEmp` and `hireEfP`) has been mapped to an individual transaction specification in TDL, and each of these specifications has been implemented as a separate DBPL transaction procedure. This



T name : GraFIc

FACTOR : ■ 0.75 0.5

GE AREA : no object at this location

Fig. 3. Zooming into the design history using conceptual descriptions and dependencies.

relationship is documented in Fig. 3 by showing dependency links created by the design decisions in Fig. 2 (the vertical arrows in Fig. 3). Such dependencies allow users to trace details of the design history for reuse or correction though the rationale behind the documented decisions remains unclear.

III. Introducing Teamwork Support: A Sample Session with CoNeX

The role of people in the development history shown in Figs. 2 and 3 is so far unclear. In this section, we extend the example by group aspects, working with the CoNeX prototype. In the scenario, we shall take the role of an Analyst who has been called into an ongoing project discussion in order to help resolve a conflict among other project participants.

A. Project Auditing

Consider some later point in the evolution of the example introduced in Figs. 2 and 3. Now the Analyst enters the scene. Upon demand she gets a snapshot of the current state of the project. Fig. 4 displays that the requirements model of Fig. 2 still persists, but that the initial design has undergone a kind of schema evolution. The new variant of the design object `EmplPers_Des` results from a different requirements mapping decision. This versioning decision collapses the `isA` hierarchy of transactions (`hireEfP isA hireEmp`) and considers only two significant transactions for the system, `hireEfP` and `fireEmp`. This change in design may cause a different implementation of transactions (`EmplPers_Imp`); i.e., require additional efforts of the project team.

This refinement covers the vital aspect of variant and version management on a conceptual level [35]. An explicit representation of the transformation steps among versions on the design and implementation level is provided in terms of decisions made and computerized tools used for that step. This kind of information enables the `Analyst` to perceive the logical dependencies (here: of programming-in-the-large) that underlie the changes. Still, the `Analyst` does not get any impression (except on the basis of intuitive guesses) of the reasons behind this development process.

B. Screening Task Debates (Project Documentation)

The Analyst now reviews the argumentation round that documents the change in design (Fig. 5). It started with the `Designer` applying (`poseing`) the `InitialReqMapping`

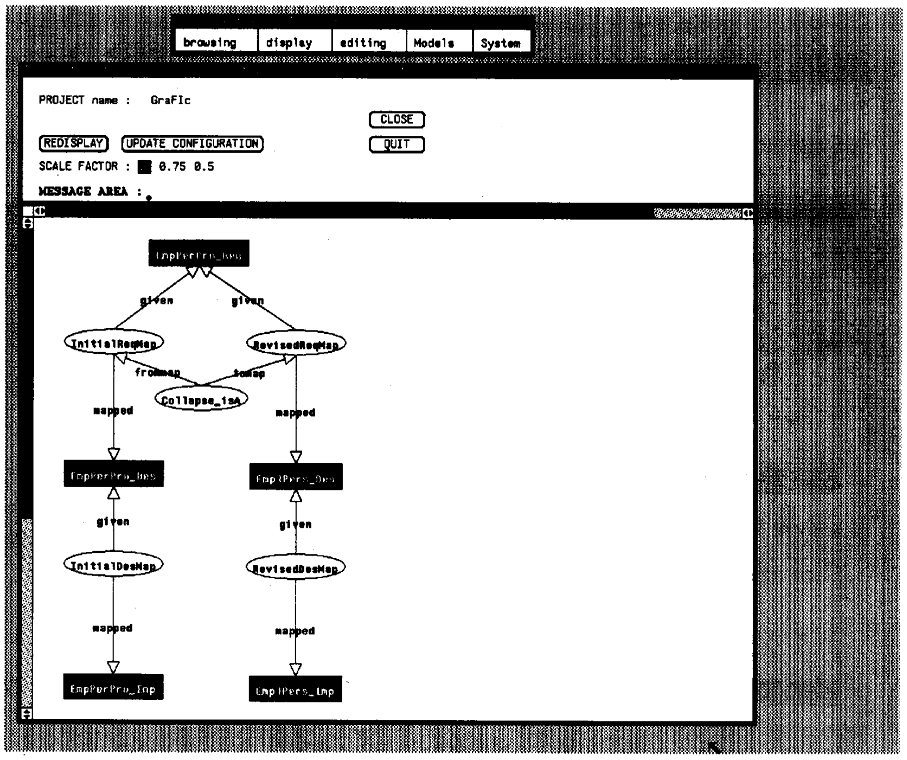


Fig. 4. Two alternative versions of the example information system.

to create the design object EmpPerPro_Des from initial requirements EmpPerPro_Req. After seeing the result through a DAIDA prototyping tool (not shown), the prospective User had argued against it. The objection was based on the reason that the consideration of two different transactions is too complicated (oppose argument, issue: UserFriendly). The User thus proposes the above alternative scheme where both transactions are merged (pose argument for RevisedReqMap, issue: UserFriendly).

The Designer poses to collapse the isA hierarchy of transactions in the initial design object into one compound transaction, i.e., to map both activity objects of the requirements model (cf. Fig. 3) to the same transaction specification, using inheritance (pose argument for the versioning decision Collapse_isA, issue: RevisionLogic). As a consequence, a new implementation object in terms of EmplPers_Imp (when applying RevisedDesMapping to EmplPers_Des) is required. Although the Designer is satisfied on the conceptual level, he recognizes the potential WorkOverhead that solution might imply (oppose argument for Collapse_isA, issue: WorkOverhead). This indirect objection to the conceptual solution requires a decision how to proceed. This is why the Analyst was called in.

Here, we have introduced one particular application of our group work model. It deals with qualitative means of negotiation in terms of argument exchange among multiple agents. The argumentation history lays open the reasons which caused system evolution in a coherent and discourse pragmatically plausible way. In practice, only crucial arguments that have actually influenced the design decision process would remain documented for long. Moreover, the screening capability, so far, is a passive device. A complementary component for active participation in the discussion is described next.

C. Decision Tool Integration: Taking the Argumentation and Decision Initiative

The Analyst doubts the feasibility statement of the Designer. Her idea can be phrased like this: is there a reasonable chance to meet the demands of the User *and* to reuse existing code from EmpPerPro_Imp to minimize the efforts of the new EmplPers_Implementation? To study this idea the Analyst activates a ConfigurationTool (see Fig. 6). It evaluates a query that analyzes the implementation dependencies between the existing EmpPerPro_ Implementation and the envisaged EmplPers_Implementation; incidentally, the dependencies to be followed by the query include those shown in Fig. 3. As in the case of Collapse_isA, a MappingDesignRevision leads from InitialDesMapping to a RevisedDesMapping within reasonable bounds of effort for the system development team. Given these data by the decision aid tool kit, the Analyst argues in favor of the MapDesRevision, agrees on the RevisedDesMapping, and therefore attacks the Designer's argument of unacceptable WorkOverhead

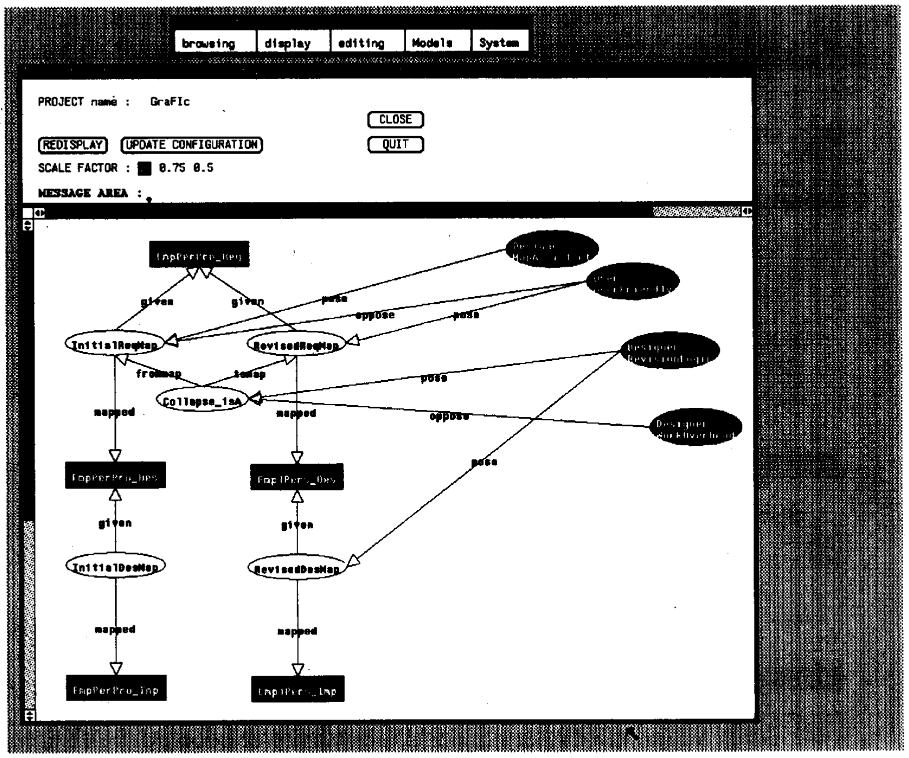


Fig. 5. Reviewing the argumentation about design decisions.

when realizing `Collapse_isA` via `RevisedReqMap`.

This is a sketch of the integration of automatic decision support techniques into the interactive design of a project work plan. It also illustrates the effects they have on generating rational evidences to push wicked, controversial argumentation lines on plans toward actual decision making (required to realize plans). We have also experimented with more sophisticated tools such as optimization procedures, simulation programs, etc. However, at least in our application settings to date, they appeared to be too complicated to be practical.

In any case, the results generated by these decision devices are conceptually embedded into the rational pursuit of a task debate; i.e., they are not isolated from their immediate task environment. Having settled what has to be done, the next step requires the assignment of tasks to people (who does it) in order to realize the design of the information system. This usually consists of two steps: task decomposition planning for a stable division of labor with limited need for coordination, and contracting of these task components to developers. A solution for the former problem is presented in [40]; below, we assume that our example programming task is small enough to be carried out by one person.

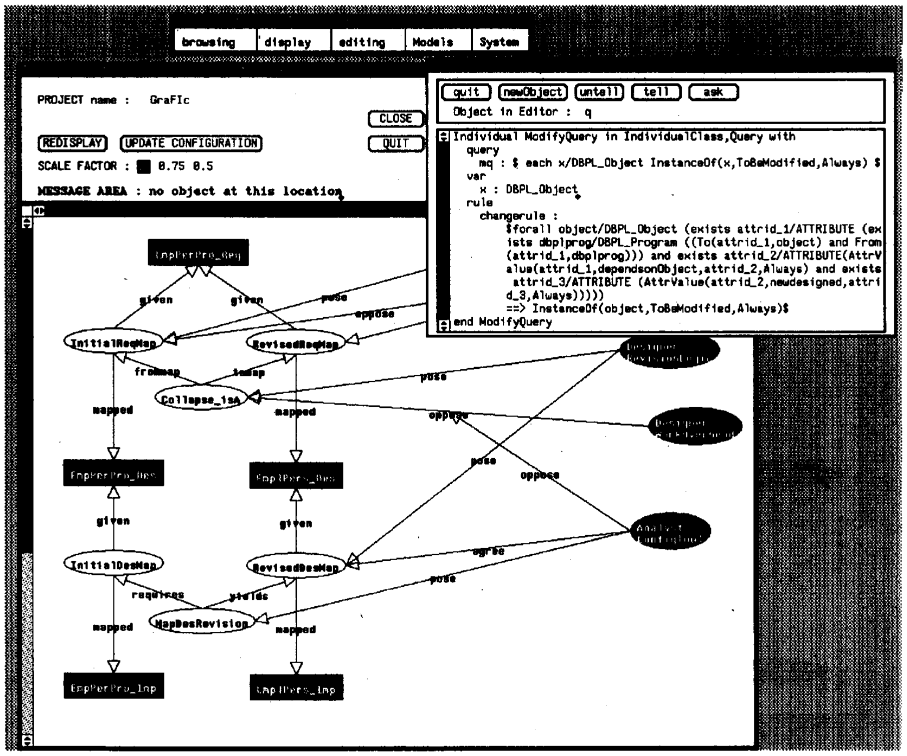
D. Task Contracting

Based on the foregoing `VariantConversation`, an appropriate `VariantContract` must be settled to realize `EmplPers_Imp` as an `EmpPerPro_Variant`. The `Analyst` starts a contracting dialog `Requesting` the `Programmer` to realize `EmpPerPro_Variant` (`s` stands for `sender`, `r` for `receiver`, and `c` for `contents`). The `Programmer` replies with a `Counter` (say, time budget too low) followed by a `Reply` of the `Analyst` (granting a sufficient time budget). The `Programmer` then `Promises` to take over the job (signing a `Promise` protocol, see below) and starts to work on `EmpPerPro_Variant`. After some period of time the contract dialog is resumed. This time, the `Programmer` issues a message to the `Analyst` that signals `ReportComplete` for the task accepted in the `Promise` stage of that dialog.

Note that the contract dialog (as well as the argumentation dialog outlined in Section III-B and C) is connected to the formal specifications of the task to be done. Furthermore, as with argumentation above, there is a strict formal protocol which has to be followed when members of the project team are engaged in argumentation or contract dialogs. Both kinds of dialogs are multiagent interactions among several members of the project team.

E. Decision Tool Integration: Contract Supervision

A final problem relates to the evaluation of the quality of



```
PROJECT name :   GraFIc
SCALE FACTOR : ■ 0.75 0.5
MESSAGE AREA : no object at this location
Object in Editor :  q
```

```
Individual ModifyQuery in IndividualClass,Query with
  query
    mq : $ each x/DBPL_Object InstanceOf(x,ToBeModified,Always) $
  var
    x : DBPL_Object
  rule
    changerule :
        $forall object/DBPL_Object (exists attrid_1/ATTRIBUTE (ex
        ists dbplprog/DBPL_Program ((To(attrid_1,object) and From
        (attrid_1,dbplprog))) and exists attrid_2/ATTRIBUTE(AttrV
        alue(attrid_1,dependsonObject,attrid_2,Always) and exists
         attrid_3/ATTRIBUTE (AttrValue(attrid_2,newdesigned,attri
        d_3,Always)))))
        ==> InstanceOf(object,ToBeModified,Always)$
end ModifyQuery
```

Fig. 6. Application of a decision support tool to trace consequences and qualify arguments.

the product the `Programmer` has delivered to the `Analyst`. Obviously, the resulting object must meet the initial requirements expressed on the design level (`EmplPers_Des`). This leads to the last step of product assessment. Fig. 7 contains a window which informs the `Analyst` of the physical location of the file the `Programmer` has created in the course of task completion. The `Analyst` might then use a standardized test bed for program evaluation, to test whether the program satisfies the requirements. In Fig. 8, the detailed dependency browser of Fig. 3 is refreshed to show the resulting situation after the integration of the new program. The content features we have discussed in the previous subsections are clearly visible in this picture: the collapsing of the isA hierarchy of transactions in the new design version, the reuse of most existing components at the design and program levels, and the availability of the new transactions. Depending on the result of this integration step, the deliverable is either `Declare-Completed` or `Rejected` by adding another message to the conversation structure in Fig. 7 (not shown).

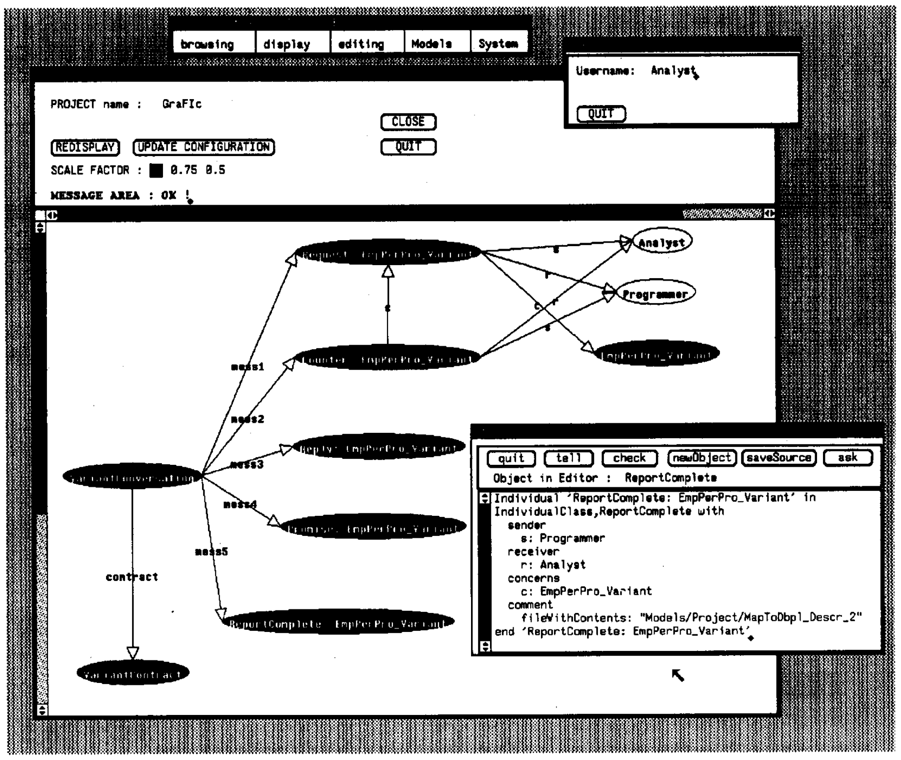
Note that the decision to accept or reject the piece of software is based on its semantic specification, as well as its conformity with constraints of the contract protocol. Although the deliverable might realize some specification perfectly, it might be useless if it is not available in time. This idea might lead to a formal notion of overall product integrity as distinguished from currently prevailing local views of integrity constraints and trigger mechnisms in software databases. In our prototype, it has also led to techniques for the integration of a knowledge-based management system, a typed message system, and commercial software development tools (see Section IV-C, below).

F. Summary of Modeling Requirements

The various steps of the sample session were intended to demonstrate requirements for group work in software projects, and to illustrate the solutions CoNeX provides:

- a conceptual model of software development processes that makes explicit logical dependencies between requirements, design, and implementation decisions;
- a conceptual model of group work that has been specialized to the needs and particularities of (software) project work;
- a conceptual model of task-oriented debates with multiple agents that makes explicit the reasons behind logical dependencies related to requirements, designs, and implementation decisions, their versions and configurations;
- a conceptual model of task contracting and contract supervision that characterizes long lasting transactions in the project team in terms of a semantic model of project coordination.

As will be shown below, these submodels stand united in a uniform theoretical framework and a coherent model of the semantics of its application domain. Of course, the associated



```
PROJECT name :   GraFIc
SCALE FACTOR : ■ 0.75 0.5
MESSAGE AREA : OK !
Object in Editor :  ReportComplete
Individual 'ReportComplete: EmpPerPro_Variant' in
IndividualClass,ReportComplete with
  sender
    s: Programmer
  receiver
    r: Analyst
  concerns
    c: EmpPerPro_Variant
  comment
    fileWithContents: "Models/Project/MapToDbpl_Descr_2"
end 'ReportComplete: EmpPerPro_Variant'
```

Fig. 7. Status overview of a contract conversation.

As will be shown below, these submodels stand united in a uniform theoretical framework and a coherent model of the semantics of its application domain. Of course, the associated group tools must be complemented by a software decision support tool kit (illustrated above by a configuration checker and a program test bed) that generates evidence for project management decisions and actions.

IV. A Conceptual Model

Abstracting from our project management example, this section sketches a formal model of (software) project management. We focus on the overall picture; the full formalization and technical analysis of individual components is found in related papers [18], [21], [36], [40].

In order to offer comprehensive support, a conceptual model has to represent several levels of abstraction within a cooperation structure. Specifically, it has to allow:

- *documentation* of the actual objects, design decisions, and communications exchanged. Most of our sample session is at this level: objects such as `EmplPers_Des`, decisions such as `InitialReqMap` or `Collapse_IsA`, arguments such as `oppose` (issue: `WorkOverhead`) or contract messages such as `DeclareComplete`. We give one example of a documented message in Telos syntax (cf. also Fig. 7):

```
Individual
  DeclareComplete_EmpPerProVariant
    in DeclareComplete
with
sender
   s : Analyst
receiver
   r : Programmer
reference
   d : ReportComplete_EmpPerProVariant
concerns
   c : EmpPerPro_Variant
end
```

- *specification* of object and agent classes, decision categories, and conversation protocols. These classes serve as schemata for the instance level. They control operations on instances through structural, predicative, and temporal integrity constraints and deduction rules. The specification level also precisely defines the interaction of group-related and domain-related submodels. For instance, completion of a contract as documented in the above example is specified by a message class whose instances can only reply to a corresponding `ReportComplete` message on the same topic. A `ReportComplete` message is a specialized `DAIDAResponse`; this, in turn, is a specialization of a general `DAIDAMessage` class in software teams working in the DAIDA environment:

```
IndividualClass DAIDAMessage in Message
with
```

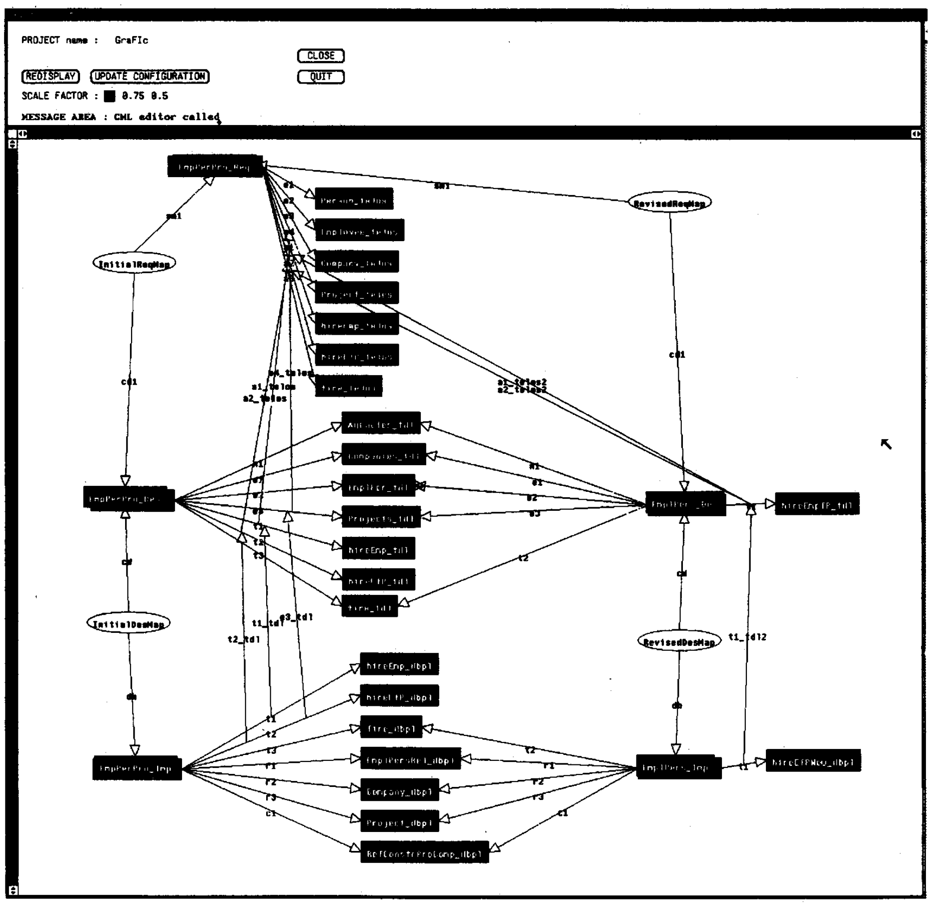


Fig. 8. Revised system structure after integration of new design and program versions.

```
IndividualClass DAIDAResponse isA DAIDAMessage
with sender
   : SoftwareAgent
receiver
   : SoftwareAgent
attribute
   reference : DAIDAMessage
integrityConstraint
   correctParticipants : $ THIS.sender = THIS.reference.receiver and
                           THIS.receiver = THIS.reference.sender $
deductiveRule
   correctTopic : $ forall X/DAIDATask
                    AttrValue(THIS.reference, concerns, X) ==>
                    AttrValue(THIS, concerns, X) $
end
```

```
IndividualClass DeclareComplete isA
  DAIDAResponse
with
attribute
   reference : ReportComplete
end
```

• *metalevel organization* of the basic interrelationships between group structure models, conversation models, and domain models. This defines what kinds of class-level specifications can be defined and how they interact. In CoNeX, it also forms the basis for the kinds of tools to be adopted, as well as graphical standards for the interface presentation (e.g., rectangles for design objects, gray ovals for design decisions, black ones for conversational objects, etc.). For example, the following metaclass `Message` restricts `DAIDATask` above to be an instance of `DesignDecision`:

```
IndividualClass Message in MetaClass
with
attribute
   sender : Agent;
   receiver : Agent;
   concerns : DesignDecision
end
```

To keep the discussion manageable, the rest of this section mostly investigates the metalevel of cooperation support, and only gives a few examples of specification-level informations. We consider it crucial to observe that this metalevel hierarchy approach leads to full extensibility and adaptability of the protocols to be followed. In previous work on the group decision support system Co-oP [7], we have even found it useful to give user groups the opportunity to dynamically tailor protocols to their current needs. The CoNeX metalevel approach represents a much more flexible solution to this problem than the fixed menus of Co-oP.

Of course, a powerful knowledge representation language is required to support the above features. We have been using the language Telos [30] in the version implemented in the ConceptBase knowledge-based management system [13]. Telos integrates a unique kind of structurally object-oriented data model with predicative assertions and an interval-based time calculus. It lends itself naturally to the kind of hypertext-like intermixing of frame-based and network-oriented components shown in the examples. Rather than elaborating the language in detail, we shall hope for understandability of the examples and otherwise refer the reader to the cited papers.

The remainder of this section is organized as follows. We first describe separately the modeling of group work structures and conversations and our domain metamodel for software engineering. Then the integration of these metamodels is presented.

A. A Group Model for Task Cooperation

From the group work viewpoint, software project management is a particular case of general task-oriented multiagent collaboration. We consider the following requirements essential for any such general model:

- Teams of experts are dynamically assembled depending on the type of problem to be solved, and the actual availability of individuals or material resources
- One individual may be associated with several teams at the same time according to different issues, obligations, levels of competence, etc.
- Solutions to (sub)problems are worked out and delivered in parallel
- Problem and task definitions, feasibility constraints, and approvals underlie the rationale of the task to be solved; nevertheless, they are all subject to social negotiations
- Definition and execution of tasks is plan-
- guided
- Facilities have to be devised for dealing with conflicting and incomplete solutions coming from individual team members.

Given these requirements, the group model is composed of the following basic conceptual entities and relations which are summarized in the semantic network of Telos metaclasses shown in Fig. 9:

Agents (Organizational Perspective): Groups are constituted by a number of `Agents`. From the viewpoint of problem solving capability, an individual `Agent` or of a group is characterized by a specific kind of competence (`qualification: AKBObject`), the records of which are available as an item in the Application Knowledge Base. In order to solve problems an `Agent` is assigned a certain amount of `Resources` (e.g., technical equipment, money, time).

Two basic types of `Agents` are distinguished for task-oriented collaboration using the `affiliation` links: `HumanAgents` and `TechnicalAgents`. Each one of them may be considered on the level of individuals and on the level of group aggregation. For example, single `Persons` are dynamically forming a `HumanGroup`, or various `Tools` (workstations, net servers, net software) can be configured in terms of a compound `TechnicalGroup` (a computer network). In addition, the modeling of `Agents` has to account for relevant dependencies between both subclasses: a `TechnicalAgent` can only be handled by a `HumanAgent` with appropriate qualification (`operated_by: HumanAgent`); conversely, the physical availability of a `TechnicalAgent` for a `HumanAgent` has to be represented adequately (`equipped_with: TechnicalAgent`).

Resources, Actions, Plans, Commitments (Project Perspective): In order to execute a project task, various `Resources` have to be supplied. Besides common economic criteria (time, money, physical and technical resources) we also count `Agents` among the resources to be managed.

Projects serve to execute a sequence of mutually reconciled actions. The goal perspective according to which single actions are ordered is achieved by creating plans. The group model allows various layers of granularity to specify, to agree upon, and realize plans by activities. On the technical level we conceive plans to be represented by `PlanTransitions` whose results are manifest in terms of `PlanDeliverables`. Plans that are created and modified in the course of negotiations we shall refer to as `Conversation` for possibilities. Our focus has not been on the planning issue as such: e.g., plan generation, plan hierarchies related to project scheduling, cf. [37], [40]. Instead, we concentrate on various social policies to agree upon plans (by contracting in terms of commitments), to modify already settled plans in the light of new evidences (by debating in terms of argument exchanges), and to monitor the success of plan execution relative to the contracts finally agreed upon.

Coupling `Actions` with plans as well as executing an `Action` according to the logic of a particular plan is based upon a social `contract`. The agents involved (`manager`, `contractor`) agree upon the `issue` of negotiation in terms of what has to be done, how it should be done, and when it has to be done. Task-oriented negotiations of this kind are

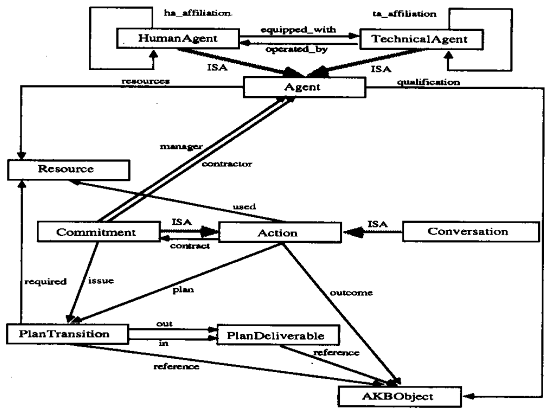


Fig. 9. Group work model—organizational and project perspective.

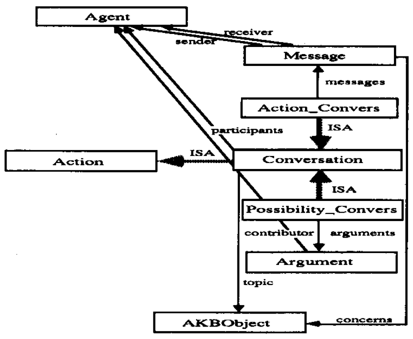


Fig. 10. Multiagent conversation model.

subject to `Conversation` about actions (see Section III-D). Specific applications of the group model (e.g., software project management) require the domain knowledge to be plugged in at the level of the application specific knowledge base (cf. Section III-C for the software engineering scenario).

B. A Multiagent Conversation Model for Task-Oriented Negotiations

The coordination of plans and actions realizing these plans is based on communication among agents. Analogous to the proposal by Winograd [44], we focus on two qualitative techniques to control the interactions that task-oriented groups often use to achieve or modify agreements:

- *Conversation for action* aims at the direction of people. Messages are passed in order to assign plans to people, to make binding commitments in order to achieve a project goal, to implement a plan in terms of activities, and guarantee proper termination and acknowledgment of the task oriented activities.
- *Conversation for negotiation* (sometimes also referred to as conversation for possibilities) aims at negotiation among people. In this communication mode opinions are exchanged in terms of debates in order to coordinate goals, and agree upon some plan or activity to be done through argument exchange and final decision making.

In Fig. 10, `Action_Conversation` is a specialization of a `Conversation`. It is characterized by the exchange of `messages` and several `participants` (that property is inherited from the `Conversation` object). `Messages` themselves are composed of a `sender` and `receiver` component stating the immediate `Agents` involved in the communication process, and a characterization of the issue dealt with (`concerns: AKBObject`). `Action_Conversations` being a special kind of `Action` have a particular specification for admitted sequences of conversation steps (`plan: PlanTransition`). This way, plans do not only cover the literal aspects of realizing work plans by task-oriented activities, but also incorporate the speech act notion of discourses for plan modification and task assignment as an integral part of project activities. Such a conversation plan is composed of several primitives of the basic `Message` type, some of which appear in the sample session: `Request`, `Counter`, `Reply`, `Promise`, `ReportComplete` (cf. Fig. 7).

As a second major conversation type, we have been considering loosely structured, argumentation-based debates (`Possibility_Conversations`). In this case `messages` are replaced by `arguments`, the issue is the `topic` of a `Possibility_Conversation`, while `participants` of a debate are inherited from the general `Conversation` object. Since `Possibility_Conversation ISA Action` we inherit the concept of an argumentation plan consisting of a sequence of `Argument` primitives, each one characterized by its `contributor`.

The specific argumentation model we have been developing [18] is based upon Toulmin's argumentation theory [43]. Examples of its primitives and their proper combination according to a particular argumentation topic are illustrated in the sample session (Section II). The introductory act of `posing` an argument (`ArgumentProper`), the qualification of previous arguments in terms of `opposing` them (`Qualification`), and the final decision making primitive in terms of `agreeing` upon a particular decision (cf. Figs. 5 and 6).

C. Software Process Data Model

The content-oriented part of our model introduces the software engineering domain into the overall picture. According to our experience in the DAIDA project and to the substantial literature in the field, the following requirements should be satisfied by such a model:

- recording of administrative aspects of software objects (requirements analyses, designs, implementations, documentation, etc.) and design decisions, including source management;
- recording of semantic aspects of software objects and design decisions, including the semantic dependencies created by design decisions (classified as refinement decisions, mapping decisions, versioning decisions, configuration decisions, etc.) for reuse in maintenance;
- integrity control and rule-based partial automation of administrative and content-oriented actions;
- administrative, semantic, and technical integration of externally developed design tools;

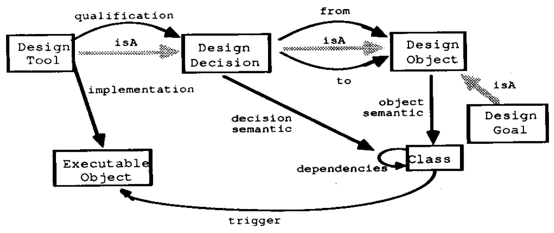


Fig. 11. Software process data model.

• extensibility in the definition of supported languages, design methodologies, and tools.

There are also other requirements of less interest to this paper. In [21] we have developed a software process data model, the so-called Decision-Object-Tool or D.O.T. model, that seems to address these requirements and was used for tool integration, process control, and documentation throughout the various DAIDA subtasks.

Fig. 11 gives the basic idea. Software development and maintenance is viewed as a process of tool-aided decisions that transform software objects into other software objects. The derived objects are then said to be justified by the underlying design decision; in turn, design decisions can have nonfunctional input objects, called goals.

As in other systems [1], [15], design objects each come with a reference to an uninterpreted information container which can be worked on by certain tools. However, they also come with a semantic description of the content of this container; it was this semantic description we have used for zooming in the examples in Figs. 3 and 8. Moreover, objects can be associated with triggers that activate directly the tools that support some decision applicable to the object.

The model gains much of its power by its combination with the abstraction principles of classification and generalization offered by Telos. An instance of the metamodel would define a certain software environment with its languages (object classes), methodologies (decision classes), and tools. The execution of an actual software project, as in Section II, would be modeled as an instantiation of this software environment model.

A limitation of the decision-object-tool model in its above form is that, like most similar models, it does not cover the group work aspects of software engineering, although the concept of design decision provides a good handle. Simply speaking, the integration is achieved by making design decisions the topic of argumentations and contracts; in particular, arguments can become goal objects of the decisions. Some more details are given in the next subsection.

D. Model Integration

The various submodels for project management introduced above have been designed for conceptual integration. Fig. 12 shows that the group model and the conversation models intersect in the `Agent`, the `Action`, and the `AKBObject` areas.

Obviously, multiple `Agents` converse, and the linking of `Conversations` to `Actions` provides a general protocol scheme for `Actions`. However, the integration does not provide any contents, i.e., what people converse about and act upon. The `AKBObject` is entirely open with respect to a particular interpretation of its domain. This changes when the application domain is plugged in. Since we are dealing with the software project management world the application dependent component has to be interpreted this way; i.e., the D.O.T. model becomes a major part of the `AKBObject`. Therefore that component is the natural and only intersection area where group processes and project management issues converge.

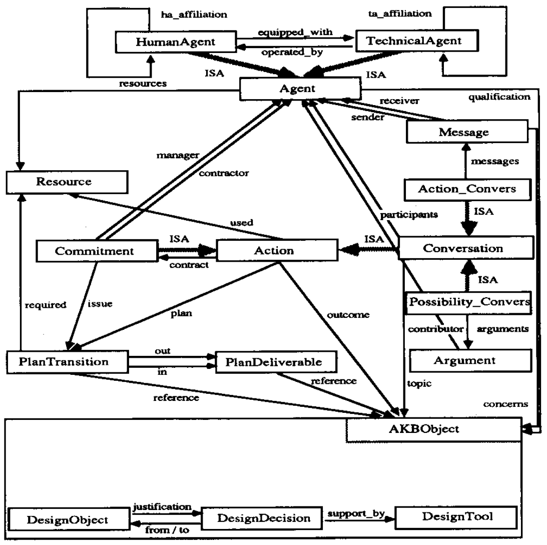


Fig. 12. Model integration—group work in software projects.

V. CoNeX: An Experimental Multiagent Project Support Environment

The CoNeX prototype has been implemented as an extension of DAIDA's knowledge-based software information system ConceptBase [13] by group collaboration facilities. Since ConceptBase supports the knowledge representation language Telos we have used for formalizing the models sketched in Section III, that model can be directly used as the basic internal schema of a knowledge base for CoNeX.

This section is devoted to the technical and architectural issues of CoNeX. It points out the concept followed in the implementation of CoNeX and describes the implementation of available tools.

A. Implementation Concept

Satisfactory support for collaborating agents in a multiagent setting does not only require an internal model of collaboration, but also an operational working environment. In addition to the conceptual modeling aspects, multiagent environments require at least facilities to exchange information among team members in real-time and asynchronous debates; assemble teams and configure communication channels dynamically with respect to competence and tasks; support working environments distributed on computer networks; allow the flexible construction of working environments tailored to agent's tasks; and provide access to information resources for decision support.

Based on this specification of operational requirements, we propose five concepts:

1) declarative specification of partitions of the knowledge base to be handled by a certain working environment;
2) an algorithm which propagates modifications from one working environment to another with respect to the assembled team;
3) a communication protocol which allows the network transparent transfer of information;
4) a tool-kit designed to build hypertext-like usage environments supporting interactive browsing and editing of (views of) the knowledge base;
5) a user-defined formal mapping between graphical presentation and the knowledge base.

ConceptBase provides implementational facilities for 1) to 5) in a client–server architecture. The server provides the central repository of information about the project. Each client constitutes a view on this central repository and provides a graphic-oriented working environment.

The first concept allows the specification of knowledge-based views which are available inside a working environment. The second concept concerns view updates and update propagation; modifications inside working environments are interpreted as updates on views. Third, updates inside working environments cause the notification of the server responsible for propagating updates. Technically, notifications are executed by a communication protocol which transfers the data from the server to the working environments.

The fourth and fifth concepts concern the graphical presentation of data, derived and controlled by the view concept. Each view presents a substructure of the knowledge base and thus requires sophisticated browsing and editing facilities. For this purpose we employed the standard usage environment of ConceptBase, which offers a hypertext-like switching among graphical and textual representations. In addition, the toolkit for the interface has to support the specification of the mapping from knowledge-based objects to graphical icons and admissible user interactions, and vice versa.

B. The CoNeX Tools

The client–server architecture of ConceptBase allows the collaboration of human and technical agents via a local area network. It allows the construction of dedicated and distributed working places and the dynamic configuration of task groups by communication channels. In our sample setting, we have four agents participating in the conversation (Fig. 13).

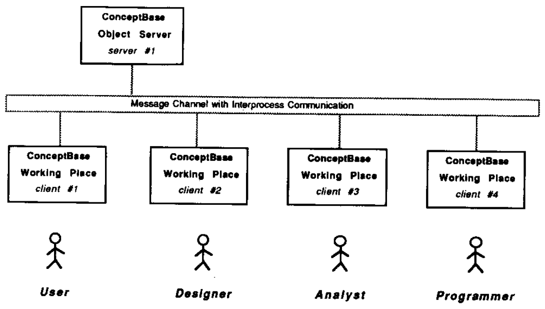

ConceptBase Object Server server #1
Message Channel with Interprocess Communication
ConceptBase Working Place client #1
ConceptBase Working Place client #2
ConceptBase Working Place client #3
ConceptBase Working Place client #4
User
Designer
Analyst
Programmer

Fig. 13. CoNeX client–server architecture for collaborating agents.

Each agent can be provided with a working environment tailored to her tasks. For instance, the analyst first contributed to a conversation for negotiation, then she directed a programmer. Similarly, programmer and user require distinct working environments, especially with respect to information resources they intend to apply for decision support. In this manner, the CoNeX prototype not only comprises a set of tools for conversation support but also integrates domain-specific tools.

Tools dedicated to conversation support include:

- an *Argument Editor* to support conversations for negotiation;
- a *Contract Monitor* to document conversations for actions;
- a *Conference System* to exchange informal messages.

The Argument Editor and Contract Monitor can be multiplexed in order to operate in real-time distributed software development settings. Interactive updates inside one working environment are propagated to other working environments which share the same focus of attention. For instance, an argument or activity posed by the analyst can be seen by the Argument Editor of the other participants if such real-time setting is desired. In addition to the synchronization of formal conversations, CoNeX supports informal information exchange by a Conference System. One major application concerns change notifications.

The second major topic of the CoNeX prototype concerns the application of information resources related to the development process. These resources are represented by the software process data model. Hypertext-like browsing and editing tools combined with predicative filters enable agents to trace the history of a software project, to discover alternative solutions tried in earlier phases, to determine products available for possible reuse or to analyze dependencies among specifications, designs, and implementations, and to introduce and propagate changes [21].

More active tools concern conceptual version and configuration management and multicriteria decision support. A concept-based configuration process subenvironment [36] provides the specification of versions and compatible configurations structured in terms of the application domain. It frees the user from physical representations such as documents or files. The document-based implementation of conceptual versioning and configuration decisions is handled semiautomatically. In some of these models, the predicative specification of conversation protocols is compiled into nondeterministic finite automata or similar internal models which implement the sequencing constraints more efficiently. Additionally, a multicriteria decision-support assistant helps individuals or groups in evaluating the trade-offs among conflicting goals.

VI. Discussion and Conclusion

Starting from the application of project management models originally developed for economic domains, software engineers soon recognized the need for automated tool support in software development processes (for an overview, cf. [6], [42]). These tools cover the specific aspects of software production as distinguished, for example, from manufacturing. They supply, for example, dedicated language tools (syntax-sensitive editors, debugging facilities) or support formal group structures (check-out/ check-in facilities) as, for example, in SUN's Network Software Environment NSE [1]. They do not consider the social dimension of software group work. This idea appeared in the IStar environment using a contractual approach [12]. IStar allows formal task fulfillment relations to be created among a contractor and a client, contracts to be kept as formal objects in a contract database and work results (program code) to be evaluated by semiautomatic devices. The various roles people play in project management and their relationships in terms of formal communication protocols were investigated in MONSTR [19], that later became the XCP protocol for general cooperative procedures [39]. Similar ideas apply to the interaction of technical agents [34].

However, these approaches provide only syntactic mechanisms to manage team work. The semantic deficit has then led to the incorporation of conceptual modeling languages, of which Telos is an advanced representative. Although being more explicit about basic semantic relationships of software project management, these approaches (cf., for example, [2], [16], [27], [37]) tend to concentrate on traditional notions of project management (such as deliverables, milestones, etc.), neglecting social factors such as conflict negotiation, work plan revision, etc. [3]. A few of these aspects are introduced in the decision recording model of [31] which, however, does not model active components or tools.

The importance of social criteria for the communication processes in project teams has been considered in Kedzierski [25], who integrates formal speech act notions into the protocol mechanisms of a project management support system. Our contracting submodel is close in spirit to Coordinator-style modeling of messages [44]. Like CHAOS [11], it additionally incorporates technical aspects. Unlike CHAOS, it includes group-specific social factors (contracting, argument exchange, multiagent problem solving) in a comprehensive formal model that is based on the semantics of its application domain. This strict formal control does not imply that the style of conversations needs to be dictatorship-like. Within the formalism, it is perfectly possible to devise very flexible kinds of protocols (such as [24]) which are adaptable dynamically, but still controlled and documented formally.

The exchange of arguments within an information system has first been considered in SYNVIEW [28], and was also adopted by ArgNoter [41]. These approaches are completely informal (based on the display of arguments and simple ranking procedures that use voting input) and thus do not allow any formal control of the reasoning processes and the subject matter on which arguments are exchanged. Formal deductive control of argument exchange and the repercussions' opinion changes, on the currently held belief set of an arguing agent as explored in a TMS environment by [4], [23], concentrate on single-agent argumentation. The gIBIS protocol [9] allows a limited formal control through a state transition network which specifies legal patterns of argument exchange. However, the semantics of the issues (application domain) is outside the modeling scope of gIBIS. Actually, a corresponding extension of the IBIS model is underway [32].

The emphasis of our work has been on the proper integration of conceptual components in terms of a comprehensive model of software projects. In particular, it contains:

- a semantic specification of knowledge about the software engineering world;
- a group model for the interactions within a task-oriented problem-solving team;
- a model of social activities that groups perform, emphasizing debates and contracts.

The power of our approach comes from the integration of these submodels into a composite conceptual model of group work specialized according to the needs of software development processes. Having established this integrated model, we have implemented a set of collaboration tools which are formally related to, and controlled by, the model. Technically, this meant the provision of:

- point-to-point as well as multiagent communication protocols;
- integrated conversation structuring and domain handling as in the argument editor;
- interleaving of synchronous and asynchronous collaboration modes.

The collaboration environment needs further refinement based on ongoing experiences with the prototype. Our experiments are not limited to software project cooperation as described here, but also include cooperative requirements engineering, coauthoring systems for system documentation, and support for certain kinds of business negotiations related to the acquisition of complex products.

Another current interest concerns the recognition, management, and resolution of conflicts that occur in task fulfillment. From a conceptual point of view, an elaborate classification scheme for conflicts and conflict resolution may improve the optimistic coordination of system development. The classification scheme relates to concept hierarchies of work tasks and possible resolution procedures like primitive document merging, regressive integration, locking, etc. From a technical point of view, one requires a technical facility supporting physical workspaces. We apply NSE as a technical means for providing workspaces and controlling activities and control it through a coupling with our conceptual model.

Acknowledgment

The authors gratefully acknowledge the contributions of R. Gallersdörfer, M. Gocek, A. Klemann, C. Maltzahn, and H. W. Nissen to the prototype implementation of the approach proposed here.

References

[1] E. W. Adams, M. Honda, and T. C. Miller, "Object management in a CASE environment," in *Proc. 11th Int. Conf. Software Engineering,* Pittsburgh, PA, 1989, pp. 154–163.

[2] K. D. Bimson and L. Boehm Burris, "Conceptual model-based reasoning for knowledge-based software project management" in *Proc. 21st Hawaii Int. Conf. System Sciences,* Kona, HI, 1988, vol. 3, pp. 255–265.

[3] B. W. Boehm and R. Ross, "Theory-W software project management: Principles and examples," *IEEE Trans. Software Eng.,* vol 15, no. 7, pp. 902–916, 1989.

[4] A. Borgida and T. Imielinski, "Decision making in committees: A framework for dealing with inconsistency and non-monotonicity," in *Proc. Workshop Non-Monotonic Reasoning,* New Paltz, NY, 1984, pp. 21–32.

[5] A. Borgida, J. Mylopoulos, J. Schmidt, and I. Wetzel, "Support for data-intensive applications: Conceptual design and software development," in *Proc. 2nd Workshop Database Programming Languages,* Gleneden Beach, OR, 1989, pp. 258–280.

[6] A. W. Brown, "Integrated project support environments," *Inform. Management,* vol. 15, no. 2, pp. 125–134, 1988.

[7] X. T. Bui and M. Jarke, "Communications design for Co-oP—A group decision support system," *ACM Trans. Office Inform. Syst.,* vol. 4, no. 2, pp. 81–103, 1986.

[8] L. Chung, P. Katalagarianos, M. Marakakis, M. Mertikas, J. Mylopoulos, and Y. Vassiliou, "From information systems requirements to designs: A mapping framework," *Inform. Syst.,* to be published.

[9] J. Conklin and M. L. Begeman, "A hypertext tool for exploratory policy discussion," *ACM Trans. Office Inform. Syst.,* vol. 6, no. 4, pp. 303–331, 1988.

[10] B. Curtis, "Models of iteration in software development," in *Proc. 3rd Int. Software Process Workshop,* Breckenridge, CO, 1986, pp. 53–56.

[11] F. De Cindio, G. De Michelis, C. Simone, R. Vassallo, and A. Zanaboni, "CHAOS as a coordination technology," in *Proc. CSCW 86,* Austin, TX, 1986, pp. 325–342.

[12] M. Dowson, "Integrated project support with IStar," *IEEE Software,* vol. 4, no. 4, pp. 6–15, 1987.

[13] S. Eherer, M. Jarke, M. Jeusfeld, A. Miethsam, and T. Rose, *ConceptBase V2.1 User Manual,* Univ. Passau, Germany, Rep. MIP-8936, 1989.

[14] P. K. Garg and W. Scacchi, "A software hypertext environment for configured software descriptions," in *Proc. Int. Workshop Software Version and Configuration Control,* J. Winkler, Ed., Grassau, Germany, 1988.

[15] ——, "ISHYS: Designing an intelligent software hypertext system," *IEEE Expert,* vol. 4, no .3, 1989.

[16] L.-M. Gilham, R. Jüllig, P. Ladkin, and W. Polak, *Knowledge-Based Software Project Management.* Palo Alto, CA: Kestrel Institute, 1986.

[17] I. Greif, Ed., *Computer-Supported Cooperative Work: A Book of Readings.* San Mateo, CA: Morgan Kaufmann, 1988.

[18] U. Hahn, "Dialogstrukturen in Gruppendiskussionen—Ein Modell für argumentative Verhandlungen mehrerer Agenten," in *Proc. 13th German Workshop Artificial Intelligence,* Eringerfeld, Germany, 1989, pp. 409–420.

[19] A. W. Holt and P. M. Cashman, "Designing systems to support cooperative activity: An example from software maintenance management," in *Proc. COMPSAC '81,* Los Alamitos, CA, 1981, pp. 184–191.

[20] M. Jarke, Ed., *Development Assistance for Interactive Database Applications.* Heidelberg, Germany: Springer-Verlag, to be published.

[21] M. Jarke, M. Jeusfeld, and T. Rose, "A software process data model for knowledge engineering in information systems," *Inform. Syst.,* vol. 15, no. 1, pp. 85–116, 1990.

[22] D. R. Jeffrey and V. R. Basili, "Validating the TAME resource data model," in *Proc. 10th Int. Conf. Software Engineering,* Singapore, 1988, pp. 187–201.

[23] K. Kandt, "A tool to support competitive argumentation," *J. Management Inform. Syst.,* vol. 3, no. 4, pp. 54–64, 1987.

[24] B. Karbe and N. Ramsberger, "Support of cooperative work by electronic circulation folders," in *Proc. Conf. Office Information Systems,* Cambridge, MA, 1990.

[25] B. I. Kedzierski, "Knowledge-based project management and communication support in a system development environment," in *Proc. 4th Jerusalem Conf. Information Technology,* Jerusalem, Israel, 1984, pp. 444–451.

[26] A. Leclerc, J. Paris, and D. Ribot, "PIMS: An integrated environment for supporting project managers," *Technique et Science Informatiques,* vol. 9, no. 2, pp. 113–120, 1990.

[27] L.-C. Liu and E. Horowitz, "A formal model for software project management," *IEEE Trans. Software Eng.,* vol. 15, no. 10, pp. 1280–1293, 1989.

[28] D. Lowe, "Cooperative structuring of information: The representation of reasoning and debate," *Int. J. Man–Machine Studies,* vol. 23, pp. 97–111, 1985.

[29] T. W. Malone, K. R. Grant, K.-Y. Lai, R. Rao, and D. Rosenblitt, "Semistructured messages are surprisingly useful for computer-supported coordination," *ACM Trans. Office Inform. Syst.,* vol. 5, no. 2, pp. 115–131, 1987.

[30] J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis, "Telos: Representing knowledge about information systems," *ACM Trans. Inform. Syst.,* vol. 8, no. 4, 1990.

[31] C. Potts and G. Bruns, "Recording the reasons for design decisions," in *Proc. 10th Int. Conf. Software Engineering,* Singapore, 1988, pp. 418–427.

[32] B. Ramesh, "Knowledge-based support for systems development and maintenance," Ph.D. dissertation, Stern School, New York Univ., New York, NY, 1991.

[33] T. Rodden, P. Sawyer, and I. Sommerville, "Cooperation and communication within an active IPSE," *Knowledge-Based Syst.,* vol. 1, no. 4, pp. 240–248, 1988.

[34] W. N. Robinson, "Negotiation behavior during requirements specification," in *Proc. 12th Int. Conf. Software Engineering,* Nice, France, 1990, pp. 268–276.

[35] T. Rose and M. Jarke, "A decision-based configuration process model," in *Proc. 12th Int. Conf. Software Engineering,* Nice, France, 1990, pp. 316–325.

[36] T. Rose, M. Jarke, M. Gocek, C. Maltzahn, and H. W. Nissen, "A decision-based configuration process environment," *Software Eng. J.* (*Special Issue on Software Environments and Factories*), to be published.

[37] A. Sathi, T. E. Morton, and S. F. Roth, "Callisto: An intelligent project management system," *AI Mag.,* vol. 7, no. 5, pp. 34–52, 1986.

[38] J. R. Searle, *Speech Acts.* London: Cambridge University Press, 1969.

[39] S. Sluizer and P. M. Cashman, "XCP: An experimental tool for sporting office procedures," in *Proc. IEEE Conf. Office Automation,* New Orleans, LA, 1984, pp. 73–80.

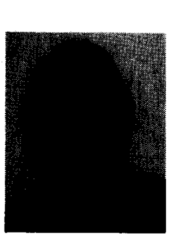
[40] R. Srikanth and M. Jarke, "The design of knowledge-based systems for managing ill-structured software projects," *Decision Support Syst.,* vol. 5, no. 4, pp. 425–447, 1989.

[41] M. Stefik, G. Foster, D. G. Bobrow, K. Kahn, S. Lanning, and L. Suchman, "Beyond the chalkboard—Computer support for collaboration and problem solving in meetings," *Commun. ACM,* vol. 30, no. 1, pp. 32–47, 1987.

[42] R. H. Thayer, Ed., *Tutorial Software Engineering Project Management.* Washington, DC: IEEE Computer Society Press, 1988.

[43] S. Toulmin, *The Uses of Argument.* London: Cambridge University Press, 1958.

[44] T. Winograd, "A language/ action perspective on the design of cooperative work," *Human–Comput. Interaction,* vol. 3, pp. 3–30, 1988.



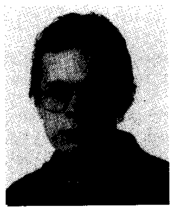
Udo Hahn received the Master's degree in linguistics from the University of Mainz in 1980 and the Doctoral degree in information science from the University of Constance in 1987.

After three years at the University of Passau, he is currently affiliated with the University of Freiburg, Germany, as an Associate Professor of Computational Linguistics. His interests include natural language processing, distributed artificial intelligence, object-oriented parallel programming, and knowledge-based information systems.

Matthias Jarke (SM'91) received Diplomas in computer science and business administration and the Doctorate from the University of Hamburg, Germany.

He holds a chair for Information Systems at the Rheinisch-Westflische Technische Hochschule (RWTH) in Aachen, Germany. Prior to joining RWTH, he served on the faculties of New York University and the University of Passau, Germany. In his research, he has investigated the integration of database, artificial intelligence, decision support, and software engineering methods. From 1986 to 1990 he was Technical Manager of ESPRIT project DAIDA in which a knowledge-based information systems environment was developed. He is currently involved in several other ESPRIT and national projects which further investigate issues such as logical foundations, hypermedia interfaces, and teamwork support within the approach.

Professor Jarke serves on the Editorial Board of IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, *Decision Support Systems*, and *Information Systems*, among others. He is a member of the Association for Computing Machinery and the IEEE Computer Society.



Thomas Rose received the Diploma in computer science from the University of Dortmund, Germany, in 1985.

He is currently a Research Associate of Computer Science at the University of Toronto, Toronto, ON, Canada. After receiving his Diploma he continued his work on knowledge-based management systems and logic programming in ESPRIT project EPSILON. From 1986 to 1990 he worked at the University of Passau for ESPRIT project DAIDA on a prototype environment for information systems development and maintenance. His interests include conceptual modeling, version and configuration management, and project support. Since 1990 he has been involved in the IRIS project at the University of Toronto, which is concerned with information systems reusability.