Managing Knowledge about Information System Evolution

Matthias Jarke, Thomas Rose

University of Passau, P.O. Box 2540
D-8390 Passau, W. Germany

Abstract. This paper describes the design and initial prototype implementation of a knowledge base management system (KBMS) for controlling database software development and maintenance. The KBMS employs a version of the conceptual modelling language CML to represent knowledge about the tool-aided development process of an information system from requirements analysis to conceptual design to implementation, together with the relationship of these system components to the real-world environment in which the information system is intended to function. A decision-centered documentation methodology facilitates communication across time and among multiple developers (and possibly users), thus enabling improved maintenance support.

1 INTRODUCTION

ESPRIT project DAIDA [JV87] investigates a new strategy for KBMS development in the context of information systems environments. Strongly based on knowledge about a particular domain of application, this strategy derives specialized knowledge representations and tools from a formalized requirements analysis. Similar to the coupling approach to integrating knowledge and data bases, portions of these representations are then realized in database programs; unlike in the coupling approach, however, the database structures and programs may change whenever the higher-level knowledge representation changes.

Managing the relationships between requirements analyses, specifications, designs, and implementations in a knowledge base requires strong *evolution support* to be effective. In the growing literature on databases for software engineering environments [BERN87], several different approaches have been reported to provide this kind of support. Early software database efforts placed the emphasis on a consistent and semantically supported modelling of software *objects*. A good example is the object-oriented IRIS data model [LK86] which offers sophisticated facilities for modelling objects together with a hard-wired set of operators; these operators define a software engineering methodology but are not themselves IRIS objects.

In contrast, recent work in the software engineering community emphasizes the need for a direct and explicit representation of the *software process* itself: modelling not only the relationships among design objects but also the intended and the actual behaviour of the designers. Several current models and systems, including DAMOKLES [ABRA86], already consider special kinds of process relationships (such as versions). The CACTIS semantic database model offers functionally derived attributes as an additional tool for evolution [HK87]. Some AI-based software engineering environments define whole classes of transformation strategies whose application, when documented in a software database, leads to typed dependencies among software objects [SKW85]; if sufficiently powerful, the corresponding type structures can also describe complete transformation *methodologies*. In a more special environment, a complete set of evolution rules is described for the object-oriented database systems ORION in [BKKK87].

In summary, two trends can be observed here: the use of structurally or behaviourally object-oriented representation mechanisms, and the explicit representation of transformational operations as objects. DAIDA tries to take both of these approaches one step further to support consistent maintenance (error corrections, enhancements, and extensions), reusability, and configuration of multi-layered software descriptions. In contrast to the systems mentioned above, the CML knowledge representation language [CML87] underlying the DAIDA KBMS allows an arbitrary instantiation hierarchy of *metaclasses* to be defined not only for objects but also for their attributes (which are themselves considered objects); this yields a lot of flexibility in adapting the descriptions of objects and transformations to different environments by creating language dialects. Additionally, we view transformations as performed *design decisions*. This means that a clear distinction is made between the task (or decision *class*) to be solved, the execution of this task in a particular situation (the decision *instance*), and the description of the *tools* that could be used, or are actually used to support the execution. Moreover, the concept of design decision suggests future extensions that would include the representation of goals and group decision processes (e.g., argumentation structures, project management) in the knowledge base.

Acknowledgments. This work was supported in part by the Commission of the European Communities under Esprit contract 892 (DAIDA) which includes: BIM, Belgium; BP Research Center, UK; Cretan Computer Research Center, Greece; GFI, Paris, France; SCS Technische Automation und Systeme, Hamburg, FRG; University of Frankfurt, FRG; University of Passau, FRG. The CML language was developed by a team under the direction of John Mylopoulos, Alex Borgida, and Yannis Vassiliou. John Gallagher guided a first partial implementation while Manfred Jeusfeld has significantly contributed to ConceptBase itself. Members of the Frankfurt group around Joachim Schmidt helped with the discussion of possible mapping strategies.

Developing an efficient and usable knowledge base management system for the above concepts is a difficult task. Therefore, the DAIDA team decided to exploit the additional power gained by concentrating on a particular subdomain of software engineering, namely the development and maintenance of database-intensive information systems. Within this framework, the DAIDA architecture, summarized in figure 1-1, is based on the following concepts and observations:

(1) Life-cycle oriented levels of representation: DAIDA views an information system as a multi-layered description of requirements analyses, designs, and implementations [BORG88]. The layers are represented in similar but distinct languages: the conceptual modelling language CML [CML87], evolved from the requirements modelling language RML [GBM86], for requirements analysis; a purely declarative version of the language Taxis [MBW80], called TaxisDL [TDL87], for conceptual design and predicative specification; and the database programming language DBPL [ECKH85], a successor to Pascal/R [SCHM77], for implementation design and programming.

For example, in a project meeting organization scenario [BORG88, JJR87], a *world model* represented in CML would give a general account of meetings as an activity in a real world with time; a *system model*, also described by CML (system) objects and activities, would be embedded in the world model in several functional parts corresponding to various user views. The combined world and system model is mapped to a TaxisDL *conceptual design* which would integrate these views into user interaction scripts as well as data object and transaction specifications organized in generalization hierarchies, for example, hierarchies of documents generated during a meeting. In a last step, this semantic data and transaction model is mapped to efficient and modular database programs in DBPL.

(2) Extensible set of interrelated transformation assistants: The literature has developed a rich set of transformation rules for refining and implementing specifications. For example, the CIP [BAUE85], Z [SPIV87], and REFINE [SKW85] projects propose user-guided formal transformation strategies, whereas the Programmer's Apprentice [WATE85] views a program as a puzzle of adapted clichées which must be maintained in a consistent state in case of changes, using dependency-directed backtracking strategies. Most of these tools have been successful only for programming-in-the-small, whereas information systems are often quite large. Therefore, DAIDA provides a flexible "open" environment which can support a range of development situations from (almost) manual to (almost) automatic, depending on the currently available set of transformation tools. To achieve this, transformation tools are embedded in a fairly large number of small "expert systems" called assistants which communicate via the common knowledge base to be described below; due to the multi-layered structure of DAIDA, *language assistants* for each level must interact with *mapping assistants* between the levels. As a consequence of restricting the application domain of DAIDA to data-intensive information systems, the special representations, theoretical results, and methods of database design research can be exploited.

(3) Formalization of information systems requirements: Most formal software development methodologies start with a formal specification of system functionality. Formalizing the requirements analysis which leads to these specifications, has been traditionally considered very difficult or even impossible. Again, the concentration on data-intensive information systems improves the situation. Database schemata naturally represent a system model of the relevant world domain; therefore, the analysis underlying the development of the initial database schema can be reused as a starting point for the requirements analysis of new applications. However, a

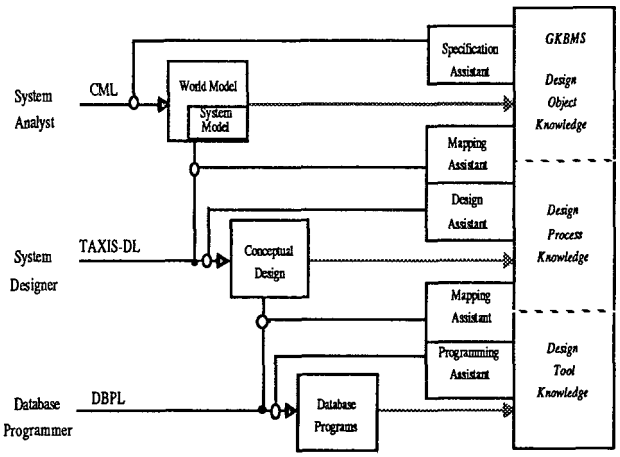


Fig. 1-1: DAIDA architecture

knowledge representation language more powerful than traditional data definition languages, even for semantic data models, is required to describe the relationship of the system model (as in the database schema) to the world model, and the development of this relationship over time. The conceptual modelling language CML [CML87] offers an object-oriented model with generalized instantiation hierarchies and embedded time calculus to support this task.

(4) Decision-based documentation knowledge base: Representing multiple layers of system description as well as their relationship to a description of the underlying real world can offer powerful development and maintenance support for information systems but requires itself a knowledge base management system for maintaining the different descriptions consistent over time. Rather than just modelling (versions of) development *objects*, the DAIDA global KBMS (GKBMS) views the software development and maintenance process as a history of *tool-supported decisions*. These decisions are directly represented, they can be planned for, reasoned about, and selectively backtracked in case of errors or requirements changes. *Ex ante*, the GKBMS can be seen as an integrative tool server which helps users in selecting tasks and tools within a large development project; *ex post*, it plays the role of a documentation service in which development objects are related to the decisions and tools that created or changed them (i.e., justify their current status). Many recent ideas from design database research apply to the implementation of such a system; applying the DAIDA philosophy to the GKBMS (viewed as a data-intensive information system about the history of "software worlds"), a dialect of CML is chosen as the knowledge representation language.

This paper presents a fairly broad overview of a first GKBMS prototype; a number of special questions are treated in more detail elsewhere. In section 2, the decision-centered approach and support requirements of the GKBMS are illustrated by a simple example. Section 3 presents a bottom-up description of the design and prototype implementation, following a conceptual model base management approach. Section 4 summarizes the system status and mentions some open questions.

2 GKBMS REQUIREMENTS OVERVIEW

2.1 A Support Scenario

In this subsection, we explore a simple scenario using some of the decisions involved in mapping a TaxisDL generalization hierarchy of data classes, and the corresponding hierarchy of transactions, to a set of relations, views, integrity constraints, and database transactions in DBPL. The example is intentionally simplified since our goal is not to discuss the complex problem of mapping semantic data models; rather, we want to show the knowledge structures and tools needed for a GKBMS.

In Fig. 2-1 (*), the developer has employed a *hierarchical text browser* tool to determine unmapped TaxisDL objects. He has further decided to *focus* on the mapping of entity structures in a document data model, in particular, invitations and their generalization, papers. This selection causes the display of a *menu* with *applicable decision classes and tools*. There are several possible mapping strategies [BGM85, WEDD87]; *distribute* would generate one relation per TaxisDL entity class, whereas *move-down* only generates relations for leaves of the hierarchy and represents the other ones by views (called *constructors* in DBPL).

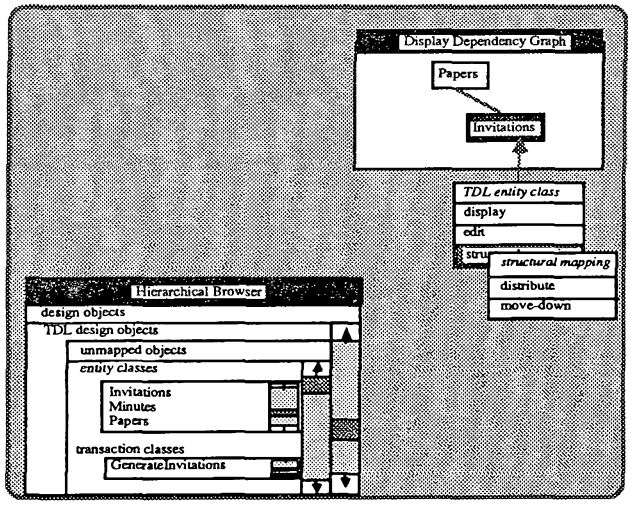


Fig. 2-1: Browsing design objects and focusing on an IsA hierarchy of the conceptual design

The graph in fig. 2-2 shows *dependencies* created by the decision for *move-down*, relating the new objects to existing ones and to a representation of the applied tool. Furthermore, selection of the *InvitationRel* node causes access to, and display of, the corresponding source code in an editor.

InvitationType contains a set-valued attribute; a *normalization decision* is therefore offered in the menu, leading to the extended dependency graph in fig. 2-3. The new *selector* expresses the referential integrity constraint among the two relations, whereas the new *constructor* allows the reconstruction of the initial, unnormalized invitation relation. Additionally, fig. 2-3 demonstrates how *automatic and manual execution of decisions* could interact. Observing that the system contains only invitations and no other subclasses of papers, the developer decides to "make the system more user-friendly", by replacing the artificial *paperkey* attribute (initially required to map the object-oriented TaxisDL model which does not have keys) with *date, author*.. This change also implies adaption of the corresponding constructor, selector, and possibly transaction definitions.

(*) The graphical tools shown in the figures have been implemented in a SUN™ environment (see section 3.3) but are simulated here with a Macintosh for clarity of exposition.

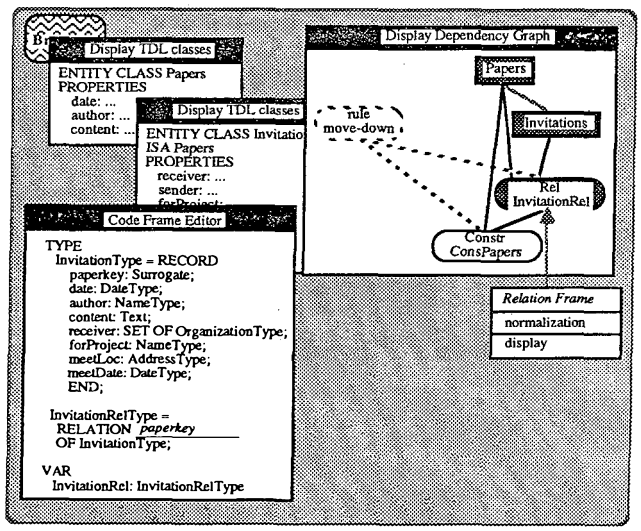


Fig. 2-2: Graphical display of dependencies and code frames generated by mapping rules

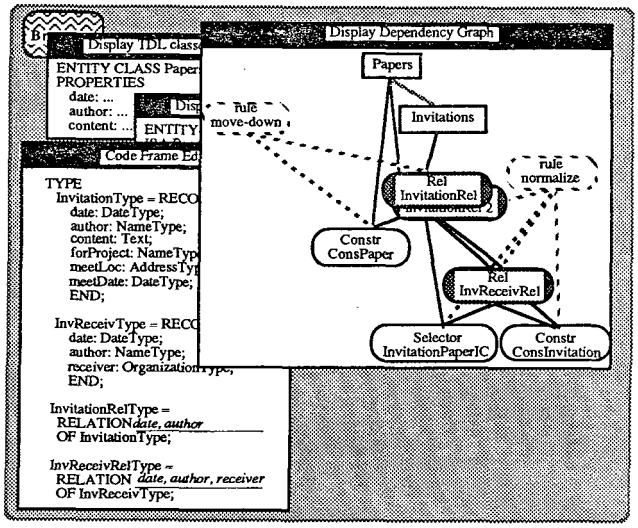


Fig. 2-3: Dependency graph and code frames after normalization and key substitution

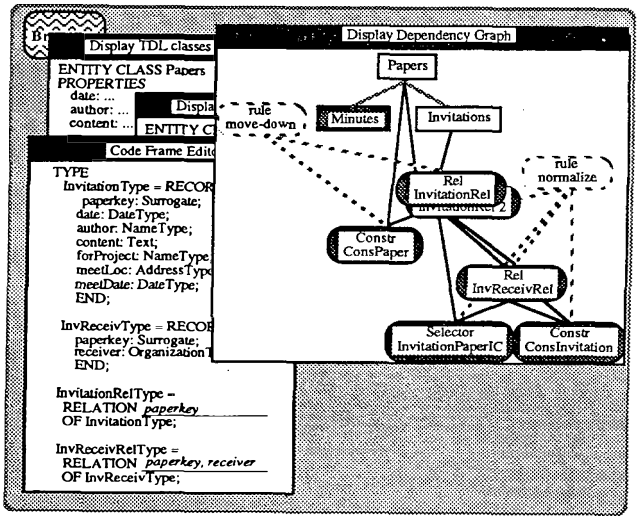


Fig. 2-4: Code frames and dependency graph after backtracking the decision on key substitution

Unfortunately, the assumption that *Invitations* are the only kind of *Papers* leads to an inconsistency as soon as the mapping of *Minutes*, the second subclass of *Papers*, is considered (fig. 2-4). Therefore, the decision to choose associative keys must be *retracted*, together with all its consequent changes, without redoing all the rest of the design; supporting this consistent, selective backtracking is the main purpose of introducing the explicit documentation of design decisions and dependencies. In the current example, the inconsistency can be resolved by selectively backtracking to the state before the introduction of associative keys; in other cases, or if the granularity of representation in the dependency graph is insufficient, additional manual or tool-aided corrections may become necessary. Note, that the graph in fig. 2-4 only highlights the objects to be changed when introducing *Minutes*; the actual correction would need a more detailed representation -- the GKBMS must have some kind of *zooming* facility for both design objects and design decisions.

2.2 Decision-Centered Representation

The above example should have illustrated how the interaction between *design objects* and *design tools* is mediated by human or computerized *design decisions*. In the following, we show that the semantic representation of this interrelationship in a knowledge base can serve as the basis for a wide variety of supporting roles provided by the GKBMS. Specifically, the example illustrated selective exploration of a design status, system-guided tool selection, decision documentation and selective maintenance (leading, among other things, to versions). This section presents an informal overview of a conceptual semantic model to support these features.

The term **design object** denotes any software object and document involved in world/system modelling, system design and database programming. Figure 2-5 shows the GKBMS view of design objects. As in object-oriented databases, design objects are classified by a hierarchy of *design object classes* which form a model of the information system. To enable a uniform representation for all stages in the software life cycle, the representation must be abstract. Thus, tokens of the GKBMS only represent characteristic features of sources recorded outside the GKB in the DAIDA sub-environments. In the first GKBMS prototype, these design object classes are based on the syntactic structures of the DAIDA languages, CML, TaxisDL, and DBPL.

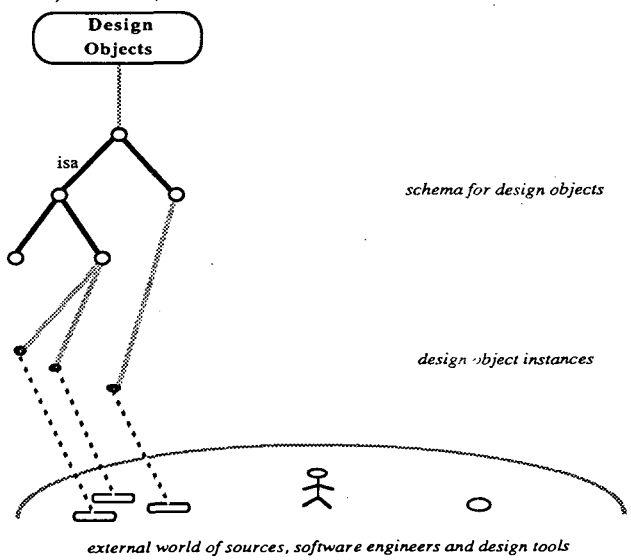


Fig. 2-5: Levels of design object knowledge base

The GKBMS represents information system development as a process of tool-aided execution of **design decisions** (fig. 2-6). Design decision *classes* specify how to transform an existing set of design objects into another set of objects. Design decision *instances* reflect history and rationales of the development.

Design decision classes are closely related to design **tool specifications**. Design tools assist the user in executing design decisions. Therefore, each *design decision class* is linked to a set of tool specifications. A decision class may be fully supported by a tool; or the tool may just aid manual decision execution. In the latter case, verification obligations are defined by the decision class for those constraints not guaranteed by the tool.

Figure 2-6 illustrates the selection of applicable tools for an active object and the documentation of actual decisions. Input and output interrelationships are denoted by *FROM* and *TO* links (not shown are predicative specifications of the I/O relationships). Tool associations are represented by *BY* links. The class of a selected object is matched against the input classes of decision classes; by testing the other input objects and preconditions of these classes, possible decisions applicable to this object are determined. A tool is now applicable to the initial object if it can execute (i.e., is associated with) one of these decision classes, normally the most specific one. For example, mapping a TaxisDL entity class to the corresponding DBPL relations and auxiliary structures could be executed semi-automatically by a specialized mapping tool, or manually by an editor (associated with the most general DBPL mapping decision).

At the level of GKBMS instances, each performed design decision is associated with a set of design objects and tools. By convention, links labeled with small letters are instances of those denoted by capitals. Due to this instantiation principle, all links among GKBMS instances must be interpreted as specified at the level of classes and tool specifications. For instance, each design object is associated with a set of design objects and a decision reflecting its development. In turn, each design decision and tool application is justified by a set of design objects (i.e., status of system development).

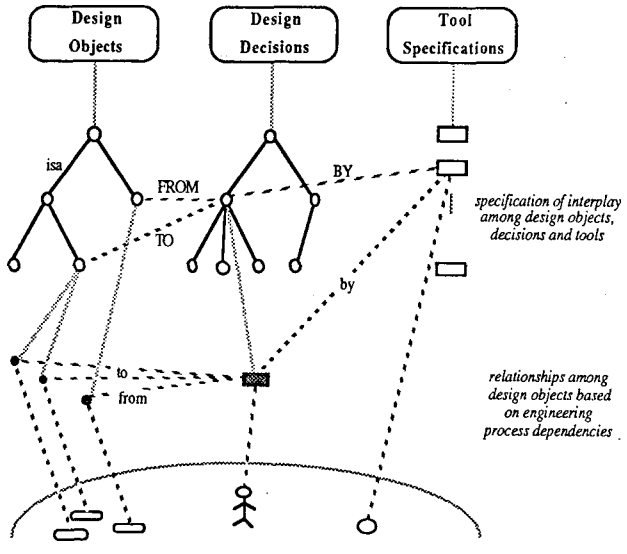


Fig. 2-7: Decision instance created after selection and tool-aided execution of an applicable decision class

As shown in figure 2-6, the GKBMS consists of three *levels of representation*:

- A *conceptual process model* at its top layer reflects information system development and maintenance as a process of tool-supported decisions; this layer also provides concepts (i.e. metaclasses) to express knowledge about design objects, tools and their interrelationships.
- At its middle layer, the GKBMS comprises knowledge about design objects, decisions and tools; in contrast to recent proposals (cf. section 1), this *development knowledge is extensible* to capture additionally evolved knowledge about languages, design decisions and tools. In its first prototype, the GKBMS provides a preliminary set of rather general design decision classes such as mapping / refinement. This kernel knowledge will then be extended based on improved tool assistants and experience gained during the DAIDA project. In parallel, sophisticated design object classes will be determined to cover the requirements of decision classes and tool specifications. As a starting point, design object classes follow an abstract syntax of applied languages.
- At the bottom layer, the GKBMS provides *documentation service*, i.e. recording of executed decisions, applied tools and involved design objects as GKBMS instances, which are themselves abstractions of objects external to the GKBMS.

This approach differs in at least two ways from typical software databases or object-oriented systems. First, the GKBMS representation starts at a higher level of abstraction; its lowest-level instances (the actual software documents, human decisions, and tools) are outside the system. On the other hand, adding the highest metalevel allows for extensibility at the language level. Taken together, these properties should make it relatively easy to integrate heterogeneous sub-environment under the same GKBMS.

A second point is that, in contrast to typical object-oriented style, methods/tools are *not* directly associated with object classes but only indirectly via the mediating concept of decision class. This should, among other things, make it easier to enforce *methodology* in design processes since a methodology can be viewed as a global decision class.

3 GKBMS IMPLEMENTATION DESIGN

It appears reasonable to base the implementation of an extensible knowledge representation language such as CML on an extensible KBMS architecture. In this architecture, the three kinds of knowledge identified in the last section (object, decision, and tool knowledge), and the specific tools sketched in section 2.1 are embedded in a conceptual model base management system, **ConceptBase**. ConceptBase which reorganizes and extends an earlier CML support system [GALL85] implements CML based on the definitions in [CML87], augmented with features to describe system behaviours, complex object configurations, and display facilities. This section describes ConceptBase and relates its features to the GKBMS requirements and tools.

3.1 ConceptBase Kernel System

ConceptBase is organized in three levels according to three different interpretations of CML language objects [CML87]; see figure 3-1. The lowest level, the *proposition processor*, represents the knowledge base as a semantic network with time and a logic-based assertion language. This level is useful not only for the formal definition of semantics but also as a basis for graphical, hypertext-style presentation (cf. fig. 2-1 to 2-4).

At the second level, the *object processor* understands the knowledge base as a deductive relational database; in this way, large sets of similarly structured objects can be managed more efficiently. The highest level, the *conceptual model processor*, offers complex object manipulation and presentation. We now discuss each of the levels in turn.

A CML **proposition** is a quadruple

$p = \langle x, l, y, t \rangle$

where

p is the *identifier* of the proposition,
x is the name of the *source* proposition,
l is the *label* of the proposition,
y is the name of the *destination* proposition and
t is the *time* associated with p.

One can interpret such a proposition as a directed link in a network: the node x has a link labelled l to node y at time t and this link has the name p. Note that nodes are also represented by propositions. For example, p can appear as the source component of another proposition p'.

Axioms of CML restrict the set of well-formed networks and help define their semantics. They reflect the existence of propositions with predefined interpretation. **Classification** allows grouping of propositions to *classes* which are again propositions. This is done by inserting *instanceof* links from the propositions (so-called *instances*) to their class. **Specialization** is done analogously by *isa* propositions. If two propositions *c1*,*c2* are connected by a directed path of *isa* links then every instance of *c1* must be an instance of *c2*. **Aggregation** employs *attribute* propositions for composing simple objects to complex ones. **Deduction** (*rule* propositions) allows the definition of Horn clauses which assert a proposition in their conclusion. Thus, there are explicit propositions, inherited propositions (through specialization) and deduced propositions. **Constraints** (*constraint* propositions) place restrictions on the instances of a class. They are connected to the class by *constraint* propositions which point to objects representing first-order logic expressions. Certain axioms define how these expressions have to be applied to the instances of a class. **Behaviours** (*behaviour* propositions) are much like methods of classes in SMALLTALK [GR83]. They associate operations such as *create* or *display* to the instances of a class by appropriate *behaviour* links.

Thus, the interpretation of each proposition depends on the class(es) it belongs to. For example, there is a predefined class

IsA_1 = <SimpleClass, isa, SimpleClass, Always>

whose instances, e.g.

p37 = <Invitation, isa, Paper, Always>,

relate specialized simple classes to their generalizations. If we want to know how to interpret *p37* we have to look for a proposition like

p37a = <p37, instanceof, IsA_1, t37a>,

where *t37a* is the time interval during which we want *Invitation* to be a specialization of *Paper*, presumably *Always*.

The **Proposition Processor** enables the manipulation of propositions according to the axioms of CML. The interface of the proposition processor is defined by the behaviour links but mainly consists of the two operations *retrieve_proposition(p)* and *create_proposition(p)* which allow the insertion of new propositions and the querying of the propositions in the **Proposition Base** subject to the content of the **CML Axiom Base**. Several physical representations (e.g. Prolog workspaces, external databases) of propositions can be managed by the proposition base. In its interface it exports operations for retrieving and creating *stored* propositions (as opposed to the proposition processor as a whole which deals with stored, inherited and deduced propositions). The CML axiom base maintains the semantics of the six predefined links listed above by interpreting the *rule* and *constraint* propositions attached to them. Thus, the axioms of CML are represented by propositions themselves, enabling very flexible modification and extension of the language. Similarly, the time components, and the relationships (e.g. *during*, *before*) between them, are again viewed as propositions.

The next layer of ConceptBase, the **Object Processor**, groups propositions around a common source: the object identifier. Consider, for example, a class *TDL_EntityClass* called *Invitation*, which relates invitations to persons by an attribute *sender*. The **Object Transformer** transforms this class into a set of propositions as shown in Fig. 3-2. Links without label stand for *instanceof* propositions. The time components of the propositions are not shown in the figure; the following propositions show a possible configuration of two of them:

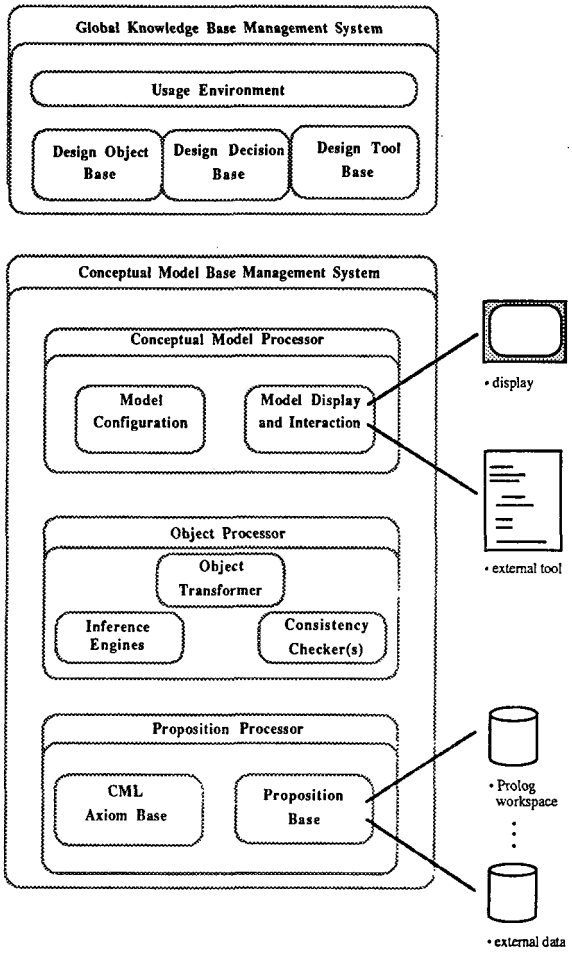


Fig. 3-1: Overall architecture of ConceptBase with GKBMS

P1 = <Invitation, instanceof, CLASS, version17>
P1' = <P1, instanceof, InstanceOf_omega, 21-Sep-1987+>

where

InstanceOf_omega = <PROPOSITION, instanceof, CLASS, Always>.

The time component of *P1*, *version17*, stands for the time interval during which version 17 of the design is regarded as valid [CML 87]. On the other hand, *P1'* asserts that *P1* is known since *21-Sep-1987*, i.e., the programmer told the KB about *P1* on September 21, 1987.

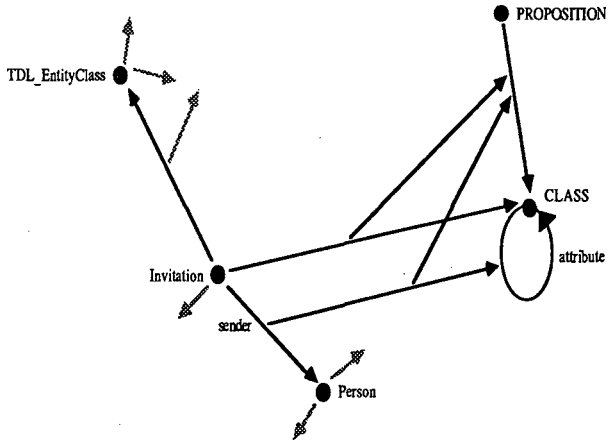


Fig. 3-2: Propositional representation of *Invitation*

After transformation, the object processor passes the generated propositions to the proposition processor. After executing a decision, the knowledge base must be in an consistent state (satisfying all the axioms of CML and the constraints imposed on certain objects in the knowledge base). This is verified by a **Consistency Checker** [GALL86] which utilizes information of the proposition processor (especially of the CML Axiom Base). Since a whole set of operations is passed to the proposition processor, set-oriented optimization of the consistency check is being studied.

The **Inference Engines** support various proof strategies for question-answering on the KB (in the current implementation, the Prolog prover with some enhancements concerning negation is the only such proof strategy). Queries are built using (open or closed) first-order logic expression over CML objects. Since the same assertion language is used in rules (see *rule* propositions above), the inference engines are also capable of evaluating rules. The inference engines may enhance their performance by lemma generation; this capability is, e.g., used in creating dependency graph objects of the GKBMS. Several time calculi may be supported by different inference engines; currently, the models of [ALLE83] and [KS86] are supported.

Finally, the **Conceptual Model Processor** uses the object processor to combine tools for the manipulation of models which consist of all objects relevant to an application of ConceptBase, e.g., the GKBMS. Models constitute highly complex muli-level object structures which are maintained in hierachies. Different models may share some objects or (sub-)models. Configuring a model for a specific application means the activation of the corresponding nodes in the lattice, i.e. making their objects accessible for the proposition processor. This work is done by the **Model Configuration** module which corresponds to a complex object database; to date, only a simple main memory version of this component has been implemented. The **ModelDisplay and Interaction** module provides tools for displaying, browsing and editing of (complex) objects as well as configurations of objects.

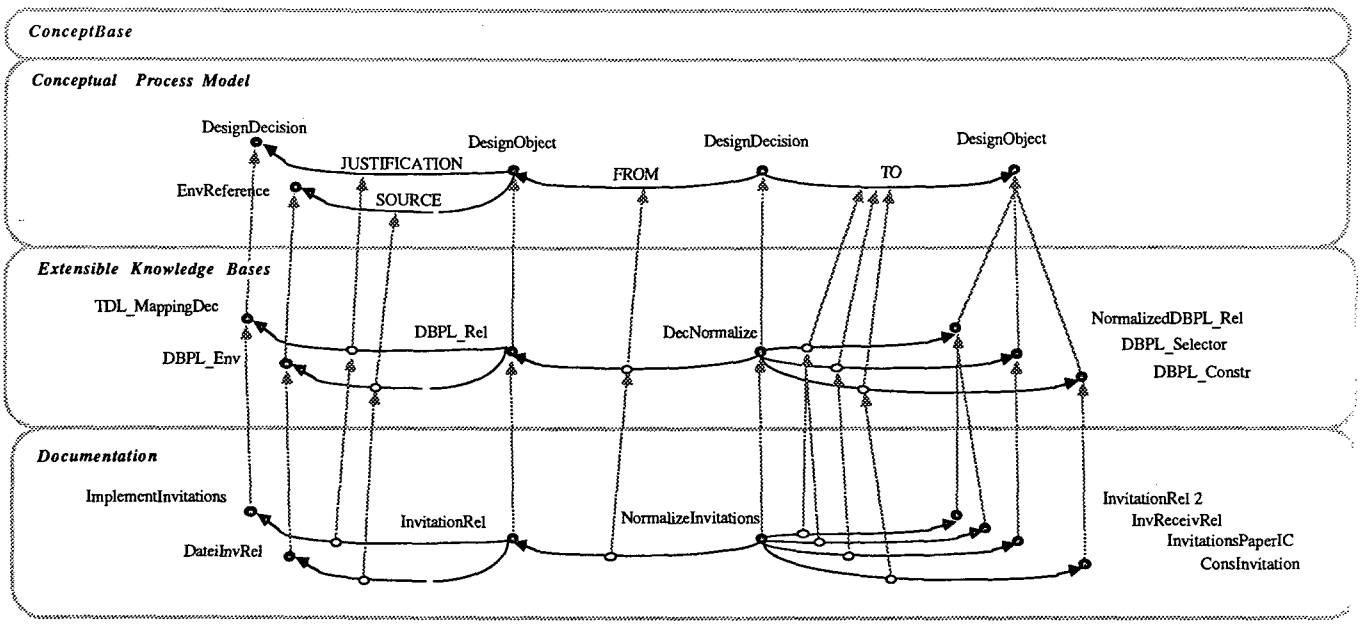


Fig. 3-3: Proposition-level representation of design decisions

3.2 CML Model of Information System Evolution

The GKBMS is implemented as a model in ConceptBase. This model realizes knowledge bases of design objects, decisions and tools. As outlined in section 2.2, the GKBMS consists of three levels of representation. Figure 3-3 shows this layered model of the GKBMS at the proposition processor level. For clarity of graphical presentation, some classes are duplicated; moreover, assertions and tool specifications are omitted.

At the conceptual level, the GKBMS introduces metaclasses to express design object and design decision classes. Formally, metaclass *DesignDecision* provides the expressive facilities to build design decision classes upon input (FROM) and output (TO) relationships (cf. section 2.2). Attributes of concrete decision classes must be instances of these properties. For example, there are two links relating decision class *DecNormalize* to object class *DBPL_Rel*, one being an instance of FROM, the other one being an instance of TO (*Normalized DBPL_Rel* is a specialization of *DBPL_Rel*). Constraints on FROM and TO define, for example, the decomposition of decision classes into PART decisions as a basis for configuration control (not shown in the figure).

Due to the instantiation principle, the design decision base of the GKBMS is extensible in case of new tools or evolved expertise. Constraints at the second level express formal input/output relationships for concrete decision classes. Instantiation of a decision class thus defines a proof obligation that these constraints are satisfied. Since tool specifications provide guarantees for certain behaviour, only those parts of the constraints *not* guaranteed by tool specfications have to be tested. This can be done in a way similar to integrity checking in transactions (the decision instance defining a, possibly nested, transaction). For example, in fig. 3-3, *normalizeInvitations* must satisfy that *InvitationRel2* and *InvReceivRel* are normalized DBPL relations with correct keys; however, as illustrated in section 2.1, the key decision may be executed manually, thus creating a proof obligation (the "proof" may be either formal or by "signature" of the decision maker).

At the lowest level of documentation, the executed decision *NormalizeInvitations* represents a decision on normalization interrelating the object instances shown in figure 3-3.

Conversely, metaclass *DesignObject* provides facilities to express the justifying decision of an design object and its source reference. Instances of *DesignObject*, such as the decision object class *DBPL_Rel*, characterize mainly the language constructs and semantic configurations of objects offered by the sub-environments (in DAIDA: the CML, TaxisDL, and DBPL objects, cf. fig. 1-1).

3.3 Decision-Based Tool Support

Due to the uniform representation of each knowledge base in CML, the GKBMS mainly supports three tasks:

- *analyzing the evolution* - analyzing information system evolution by browsing in decisions and their causal ordering; additionally, arbitrary switching between browsing of performed decisions, design objects possibly at different stages of the development process and tool specifications is provided. The latter enables a powerful navigation through development processes and outcomes.
- *decision processing* - besides pure backtracking of decisions, tool specifications enable some kind of revision support; for instance, adding an attribute in the design could be processed by the GKBMS by replaying decisions (GKBMS tests their re-applicability).
- *conceptual tool server* - based on its functionality as central repository, the GKBMS serves as a board for tool communication; tools are enabled to consider results gained by other tools. Additionally, manual modifications are supported by the analyzing facilities.

This section describes three utilities which can be built on the decision-oriented representation of the development process and the uniform formalism to present development processes and outcomes.

3.3.1 Navigation in Decision Histories

As illustrated in section 2.1, the GKBMS enables browsing along and arbitrary switching between several dimensions:

- *status-oriented*, by browsing requirements, designs, implementations, and their interrelationships;
- *process-oriented*, by following mapping and refinement relationships and their causal ordering;
- *temporal*, by focusing on system versions and following the history of design ojects and design decisions.

Such an exploration typically starts from a **focus** object or decision; tool selection for this focus (using the idea shown in fig. 2-6) will also display which of the above exploration directions are applicable to the focus in the current state under a given methodology. To support exploration by focusing, browsing, and zooming with direct manipulation, we have implemented a number of window-oriented interface tools which are formally part of ConceptBase's Model Display and Interaction module (cf. fig. 3-1):

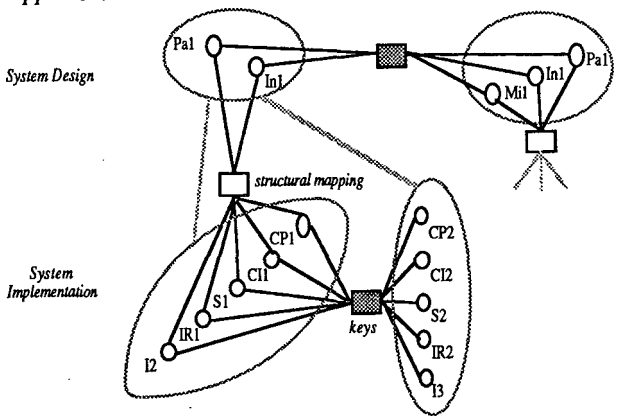
- A **text DAG browser** (fig. 2-1) allows the display and browsing of a tree-like CML structure at a dynamically defined depth and width. Basically, it consists of a recursively embedded set of windows, each variable in size and endowed with a scrolling facility.
- A **graphical DAG browser** (used in fig. 2-1 to 2-4 to show dependency graphs) offers a graphical representation of the same kinds of data structures as the text browser. A simple standard layout is offered but can be changed by the user in a persistent way.
- A **relational display** shows the properties of objects in tabular form with variable column width and scrolling (thus corresponding to the Object Processor level in fig. 3-1); the extension to a non-first normal form display of complex objects is underway. This display is associated with a **CML form editor**, to interact with the knowledge base and to work with CML code frames.
- **Focusing** in any of these structures is done by mouse selection; hierarchical **menus** (cf. fig. 2-1) with context-dependent content are used for tool selection as illustrated in section 2.1. A **dialog manager** with improved error handling and recovery facilities is under construction.

3.3.2 Version and Configuration Management

A frequent operation on a GKB will be the configuration of a complete derivation structure and its subsequent projection on one level, e.g., "configure the latest complete DBPL database program system version"; this involves excluding all non-used versions of design objects, and ensuring consistency and sufficient completeness of the remaining ones with respect to specifications and decision class definitions. As the example in fig. 2-3 and 2-4 shows, there is also a need to retain multiple versions of certain system components, without duplicating all the implementation. The decision structure described in section 3.2 can be exploited for this kind of version and configuration management:

- Allowable multi-level configurations of world/system models, designs, and implementations are those which are interrelated by mapping decisions (*vertical configuration by means of equivalences).*
- Allowable one-level (sub) configurations must be consistent, as documented by refinement decisions inside a (sub) configuration and mapping decision on coherent higher-level objects (*horizontal configuration by means of component configuration).*
- Versioning rests upon choice decisions. An alternative version is created each time an object is refined or mapped alternatively*(versioning by decisions to retract)*; typically, such a retract decision would start a (nested) sub-transaction.

Noticing similarities of these kinds of decisions to the three dimensions of equivalence, configuration, and version in [KAC86], a version and configuration management mechanism similar to the one proposed there is being considered. Fig. 3-4 represents the example of section 2.1 from this viewpoint. In this way, version and configuration management come as a natural by-product of the decision-based documentation approach.



Objects are denoted as follows:

Pa	entity class *Papers*	CP	constructor *ConsPapers*
In	entity class *Invitations*	IR	relation *InvReceivRel*
Mi	entity class *Minutes*	S	selector *InvitationPaperIC*
I	relation *InvitationRel*	CI	constructor *Invitation*

Fig. 3-4: Decision-based configurations and versions: the second implementation, whose mapping dependency is derived via the refinement decision on keys, is based on an assumption which is inconsistent under the expanded design version with respect to candidate keys

3.3.3 Reason Maintenance and Group Support

In the currently beginning second stage of DAIDA, the facilities described so far will be enhanced with three further "expert system"-like components. As an enhancement of the navigation facilities, the predicative specifications of tool and decision classes together with ConceptBase rules and constraints will be used to develop a **design explanation facility**.

The representation of decision structures supports the storage of redundant dependency information as the basis of a **reason maintenance system** [DOYL79, DJ88] which can contribute to the automatic propagation of the consequences of high-level changes. However, since current RMS can handle only fairly small dependency networks efficiently [DEKL86], we are studying their combination with the abstraction mechanisms of the GKBMS.

Often, **multiple developers** contribute to a software system. Therefore, some design database approaches study transaction concepts adapted to the organization of collaborating groups [KSUW85]. While the GKBMS browsing and explanation facilities enable information exchange in such a group, explicit mechanisms for conflict handling (beyond consistency checking) are missing. In [HJ88], we develop a proposal for enhancing the above mentioned RMS with mechanisms for multicriteria choice support, argumentation on derivation decisions, and explicit group work organization in an object-oriented context.

4 CONCLUSIONS

In this paper, we tried to demonstrate the usefulness of a decision-based conceptual modelling formalism in the management of software development and maintenance processes. The DAIDA architecture in general, and the GKBMS design in particular, address two relevant questions in the database context. From the *information systems design viewpoint*, they present a decision-based approach how to maintain large software systems developed in multiple layers and languages consistent over time, exploiting special properties of data-intensive application domains such as reuse of a world model, data-oriented and therefore often algorithmically easy programming, or selective backtracking of small design portions. These ideas, some of which have been tried in the AI area before, are embedded here in the object-oriented deductive database context of ConceptBase. From the *viewpoint of general KBMS implementation research,* DAIDA proposes a novel way of realizing KBMS by supporting them with semi-automatically-developed dedicated information systems. This idea, which requires a powerful GKBMS to be useful, will be further elaborated in a forthcoming paper.

The GKBMS is being implemented in a UNIX environment, using BIM-Prolog which offers interfaces to graphical display and external DBMS (relational and Entity-Relationship). Based on experiences with the current prototypes, a large number of efficiency questions, especially concerning deductive querying and consistency-checking of complex design objects, are scrutinized in more depth. Another area of current interest is the augmentation of the GKBMS with more rigorous development strategies, based on algebraic specifications, in dedicated application contexts such as model-based decision support.

REFERENCES

[ABRA87] Abramowicz, K., Dittrich, K.R., Gotthard, W., Längle, R., Lockemann, P.C., Raupp, T., Rehm, S., Wenner, T. (1987). Datenbankunterstützung für Software-Produktionsumgebungen, *Proc. Datenbanken in Büro, Technik und Wissenschaft,* Darmstadt, FRG, 116-131.

[ALLE83] Allen, J. (1985). Maintaining knowledge about temporal intervals, *Comm. ACM 26*, 11, 832-843.

[BAUE85] Bauer, F.L. et al. (1985). *The Munich project CIP: Volume I,* Heidelberg, FRG: Springer-Verlag.

[BERN87] Bernstein, P.A. (1987). Database system support for software engineering, *Proc. 9th Intl. Conf. on Software Engineering*, Monterey, Ca,166-179.

[BGM85] Bouzeghoub, M., Gardarin, G., Metais, E. (1985). Database design tools: an expert systems approach, *Proc. 11th Intl. Conf. Very Large Data Bases,* Stockholm, Sweden, 82-95.

[BKKK87] Banerjee, J., Kim, W., Kim, H.-J., Korth, H.F. (1987). Semantics and implementation of schema evolution in object-oriented databases, *Proc. ACM-SIGMOD Conf.*, San Francisco, 311-322.

[BM86] Brodie, M. L., Mylopoulos, J. eds. (1986). *On Knowledge Base Management Systems,* Springer-Verlag, New York.

[BORG88] 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*, Heidelberg: Springer-Verlag, to appear.

[CML87] Borgida, A., Mylopoulos, J., Vassiliou, Y. (1987). Conceptual Modeling Language: An Informal Description, Report, ESPRIT Project 107 (LOKI), Crete Research Center, Iraklion, Greece.

[DEKL86] de Kleer, J. (1986). An assumption-based TMS, *Artificial Intelligence 28*, 2, 127-163.

[DJ88] Dhar, V., Jarke, M. (1988). Dependency-directed reasoning and learning in large systems maintenance, *IEEE Trans. Softw. Eng. 14*, 2, 211- 227.

[DOYL79] Doyle, J. (1979). A truth maintenance system, AI Memo 521, MIT, Cambridge, Mass.

[ECKH85] Eckhardt, H., Edelmann, J., Koch, J., Mall, M., Schmidt, J.W. (1985). Draft report on the database programming language DBPL, Universität Frankfurt, F.R. Germany.

[GALL 85] Gallagher, J. (1985). Overall design of CML support system, Working Paper, Esprit project 107 (LOKI), SCS Hamburg, FRG.

[GALL86] Gallagher, J.: (1986). Notes on consistency checking in CML, Working Paper, Esprit project 107 (LOKI), SCS Hamburg, FRG.

[GR83] Goldberg, A., Robson, D. (1983). SMALLTALK 80, the language and its implementation, Addison Wesley, 1983

[GBM86] Greenspan, S., Borgida, A., Mylopoulos, J. (1986).A requirements modelling language and its logic, in Brodie, M.L., Mylopoulos, J. (eds.): *On Knowledge Base Management Systems,* New York: Springer-Verlag, 471-502.

[HJ88] Hahn, U., Jarke, M. (1988). A multi-agent model for group negotiation support. To appear in *Proc. IFIP WG 8.4 Working Conf. Organizational Decision Support Systems,* Lake Como, Italy.

[HK87] Hudson,S.E., King, R. (1987). Object-oriented database support for software engineering. *Proc. ACM-SIGMOD Conf.*, San Francisco, 491-503.

[JJR87] Jarke, M., Jeusfeld, M., Rose, T. (1987). A global KBMS for database software evolution: design and development strategy, Report MIP-8722, Universität Passau, FRG.

[JV87] Jarke, M., Venken, R. (1987). Database software development as knowledge base evolution. In *ESPRIT '87: Achievements and Impact,* Amsterdam: North-Holland, 402-414.

[KCB86] Katz, R., Chang, E., Bhateja, R. (1986). Version modeling concepts for computer-aided design databases, *Proc. SIGMOD Conf.*, Washington, D.C., 379-386.

[KS88] Kowalski, R., Sergot, M. (1985). A logic-based calculus of events, to appear in Schmidt, J.W., Thanos, C. (eds.): *Foundations of Knowledge Base Management*, Springer-Verlag.

[KSUW85] Klahold, P., Schlageter, G., Unland, R., Wilkes, W. (1985). A transaction model supporting complex applications in integrated information systems, *Proc. ACM-SIGMOD*, Austin, 388-401.

[MBW 80] Mylopoulos, J., Bernstein, P., Wong, H. (1980). A language for designing interactive data-intensive applications, *ACM TODS 5*, 2, 185-207.

[SCHM77] Schmidt, J.W. (1977). Some high-level language constructs for data of type relation, *ACM Trans. Database Systems 2*, 3, 247-261.

[SKW85] Smith, D.R., Kotik, G.B., Westfold, S.J. (1985). Research on knowledge-based software engineering environments at Kestrel Institute, *IEEE Trans. Software Engineering SE-11,* 11, 1278-1295.

[SPIV87] Spivey, J.M. (1987). An introduction to Z and formal specifications, Oxford University, UK.

[TDL87] Borgida, A., Meirlaen, E., Mylopoulos, J., Schmidt, J.W. (1987). The TAXIS Design Language (TDL), Report, ESPRIT Project 892 (DAIDA), Crete Research Center, Iraklion, Greece.

[WATE85] Waters, R.C. (1885). The Progammer's Apprentice: a session with KBEmacs, *IEEE Trans. Software Engineering SE-11,* 11, 1296-1320.

[WEDD87] Weddell, G.E.C. (1987). Physical design and query compilation for a semantic data model, Ph.D. Thesis, University of Toronto, Canada.