Rule Representation and Management in ConceptBase

Matthias Jarke, Manfred Jeusfeld
Fakultät für Mathematik und Informatik, Universität Passau
Postfach 2540, 8390 Passau, F.R. Germany

Abstract. ConceptBase is an experimental knowledge base management system for design applications, especially in the software engineering area. The knowledge representation language it supports, CML/Telos, combines the functionalities of deductive and temporal databases with structural object orientation. In this paper, we demonstrate how to exploit a process-oriented software data model that uses just the object-oriented structural language kernel, to bootstrap efficient internal representations of the rule sub-language.

Introduction

Among the proposals for extended database functionality, object-oriented and deductive databases have attracted the greatest attention. The ConceptBase system tries to integrate aspects of both of these extensions in order to support design applications, especially those in software engineering [JJR89a]. As its data description and manipulation language, ConceptBase offers a version of the knowledge representation language CML/Telos [KMSB89] which integrates predicative rules, constraints, and queries together with an embedded time calculus in a structurally object-oriented framework inspired by semantic networks.

In its implementation strategy, ConceptBase follows a bootstrapping approach: the structural concepts offered by the CML/Telos object language are exploited to derive efficient implementations of the predicative subsystem. Thus, our first goal in this paper is to answer the question: what kinds of objects are rules? In a second part, we discuss the efficient compilation and evaluation of such rule objects. We discuss the usage of rules in query processing and deductive integrity checking; also addressed is the representation of queries and derived data in the same, uniform framework. It is shown that the structural rule representation above yields fast access paths for all of these uses. Moreover, a small extension -- again based on the software process data model -- allows the elegant integration of triggered rule evaluation and assertion checking procedures (as well as their automatic generation from predicative formulations [FREY87, GD87]) into the system. In many applications, especially in software specification and verification, the rule proofs themselves are objects of interest to the user (= software developer); they are therefore also modeled in our system. Finally, we discuss some extensions we are currently working on.

Overview of CML/Telos

A CML/Telos knowledge base can be seen as (but need not be stored as) a semantic network. Links represent attributes, specialization among classes and instantiation relationships between objects and their classes. Each link is an object with its own identifier. Nodes ("individual" objects) are special links which point to themselves. Additionally, each object has two associated time intervals: one for the time during which the object is valid in the modelled world, and one for the time when the object is believed by the knowledge base. Built-in system classes provided for these structural features allow arbitrarily high meta-levels, multiple instantiation and inheritance.

```
IndividualClass Class with
  attribute
    attribute: Class;
    rule: Assertion;
    constraint: Assertion
end Class
```

Fig. 1: System object `Class`

To express implicit information beyond that provided by structural axioms for aggregation, generalization, and specialization, an *assertion language* for deductive rules and integrity constraints is provided. From the viewpoint of the CML object language, these rules and constraints are uninterpreted objects of class `Assertion` whose role is determined by the links (`rule` resp. `constraint`) which attach them to other objects. This decoupling allows the integration of several different assertion languages and uses of assertion objects. In the ConceptBase usage environment, for example, we are not only interested in standard Horn clause assertions but also in specialized languages for expressing verification conditions on software specifications or test programs which may be difficult to express declaratively.

Instantiation in CML/Telos means setting up an `instanceOf` link between the instance object and its class. Instances may instantiate the attributes of their classes to get attributes themselves. The object `Employee` instantiates the attribute `attribute` of `Class` four times, the attribute `rule` once; it does not use the `constraint` attribute class.

```
IndividualClass Employee with
  attribute
    name: String;
    salary: Money;
    dept: Department;
    boss: Manager
  rule
    bossrule: $ forall e/Employee,
                d/Department,m/Manager
                AttrValue(e,dept,d) and
                AttrValue(d,head,m)
            ==> AttrValue(e,boss,m) $
end Employee
```

Fig. 2: Instantiating `Class`

As can be seen, the "normal" CML assertion language is a many-sorted first-order calculus where variables range over classes. The `bossrule` states that if an employee works in a department which is headed by a particular manager, then this manager is the boss of the employee. [KMSB89] provide a formalization of a version of CML/Telos very close to ours.

Rules as Knowledge Base Objects

Given the flexible embedding of predicative constraints in the language, several implementation strategies can be followed. A first strategy explored for CML/Telos was the translation to equivalent logic programs [GS86, JJR88, KT89]. Each link of the network is translated to a corresponding Prolog term, and the semantics of `isa` and `instanceOf` axioms is hardcoded into the system for efficiency. To complete the logic program, rule, constraint, and query objects are translated into Prolog rules and queries [LT85].

For the management of large knowledge bases, this direct and formally nice approach has a number of disadvantages. Recent research in large-scale rule management has shown the need to represent more explicitly the structure and interrelationships of rules, the storage of intermediate results or derivation paths, as a basis for reusability in multiple query optimization [SLR88]. We therefore turn the above implementation strategy around and represent assertions as object structures to be managed, optimized and manipulated (an alternative attempt to represent the ideas below as metalevel logic programs is currently followed in the COMPULOG ESPRIT project). From earlier semantic network approaches to logic [MS81], our approach is distinguished by its use of CML's use of rather strict structural axioms, and by the application of a specific metamodel initially developed by us for the broader context of software process control [JJR89a]. Nevertheless, graph-based algorithms for deductive query processing [BR86] and integrity control can be nicely represented.

Since we wish to support a variety of assertion languages, our basic structure supports very general, non-deterministic rules (fig. 3). Motivated by the software engineering context, it is called a software process data model; non-determinism stems from human design decisions and externally provided tools whose functionality the KB only knows roughly.

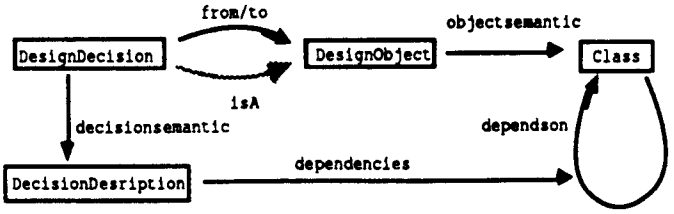

Fig. 3: Software process data model

Applied to software engineering, design objects represent programs, documentation etc. Design decision lead from existing design objects to new ones, for example from a buggy program to an error-free one. Design objects can be further described by arbitrary CML classes, and design decisions by *dependencies* among object descriptions. The CML specification of predicative assertions can now be modeled as an instance of this meta-model: design decisions relate the condition literals (`from`) deterministically to the conclusion literal (`to`).

Figure 4 shows this interpretation of the software process model in a semantic network. Unlabelled links denote instantiation relationships. For example, the class `Literal` is an instance of the metaclass `DesignObject`. The truth of the literal `AttrValue` depends on so-called attribute classes like the `dept` attribute of `Employee`. The lower third is occupied by representation of the example rule. It has two condition literals and one conclusion literal. The literals concern the attributes of `Employee` and `Department`. Note that the granularity of rule precision is a single attribute, rather than a full object class as in most languages (e.g., relational databases). The assertion compiler of ConceptBase generates this network automatically.

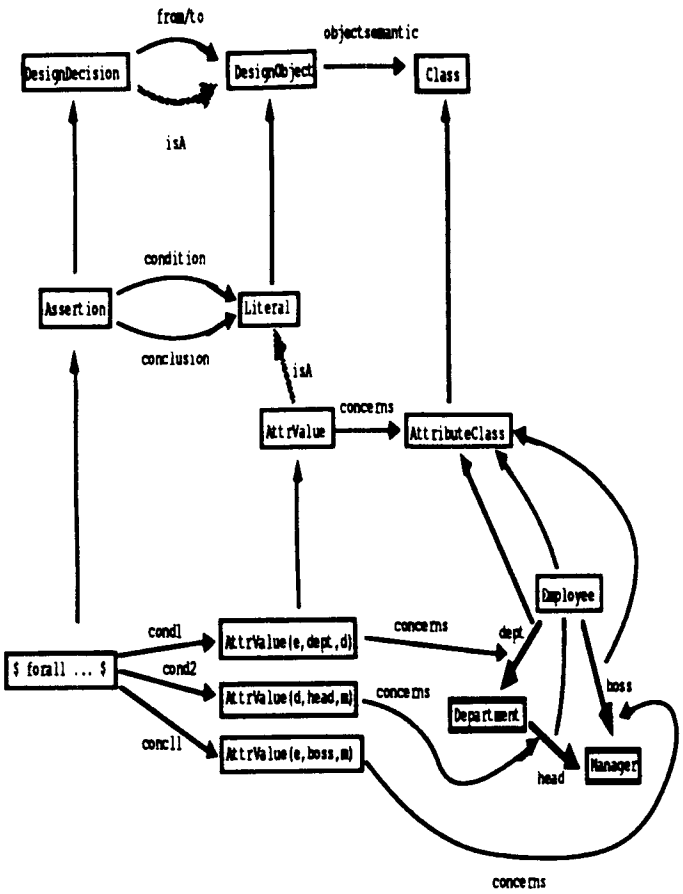

Fig. 4: The data model for assertions

Compilation of Rules -- Firing of Evaluators

The network model proposed so far relates assertions to the parts of the knowledge base they concern. Determining which rules or constraints to apply in a specific update situation still determines a complex search in these structures. The ConceptBase assertion optimizer therefore compiles the initial structures into simplified structures and executable evaluation procedures attached to those objects whose change may fire the rule or violate the constraint; in other words, rather than just making objects attributes of assertions, specialized assertion evaluators are associated as `trigger` attributes to the objects. Typically, triggers are associated with attribute classes; corresponding parameter-instantiated evaluation procedures are fired when instantiation links to these class objects are inserted respectively deleted. Since a large number of specialized procedures may be generated, techniques such as proposed in [KDM88] are explored to manage them.

For an example, figure 5 shows two related insertions provided as a transaction to ConceptBase.

```
Individual PR in Department with
  head
    ledby: mary
end PR

Individual bill in Employee with
  name
    hisname: "William B. Smith"
  salary
    earns: $20000
  dept
    worksfor: PR
end bill
```

Fig. 5: Instances for `Employee` and `Department`

According to the `bossrule` in class `Employee`, the `boss` of `Bill` is `Mary` since he works for the department `PR` which is headed by her.
The actual evaluation is done by a rule evaluator tool of the run-time system. Such tools are modeled as instances of meta-class `DesignTool`:

```
IndividualClass DesignTool
                isA DesignDecision with
  attribute
    from: DesignDecision
    to: BehaviorObject
end DesignTool
```

Fig. 6: CML definition of design tools

Design tools are formally seen as reusable design decisions which relate the specification of the tool (given by another design decision which describes the kind of transformation the tool performs) with a behavior object representing the way of invoking the tool. For example, the `bossrule` has two procedure calls: one for the case when a new department of an employee is stored (`br_dept(e,d)`) and one for the case when a new head of a department is inserted.

Fig. 7 shows the completed model with the representation of the example rule. Fig. 8 shows what happens if the attribute `head` is instantiated for the department `PR`.

The `ledby` attribute is declared to be an instance of the `head` attribute class. This event activates the head trigger to the procedure call `br_head(d,m)`. The parameters are instantiated with `PR` and `mary`. The evaluation results in a new attribute of `bill` (instance of `boss`). This new attribute may fire another rule evaluation, e.g., the constraint that no employee may earn more than his boss. In this way, the same paradigm has be used for *constraint* evaluation [KRÜG89] based on the optimization approach proposed in [BDM88]; here, the conclusion of a constraint is a special literal `Consistent`.

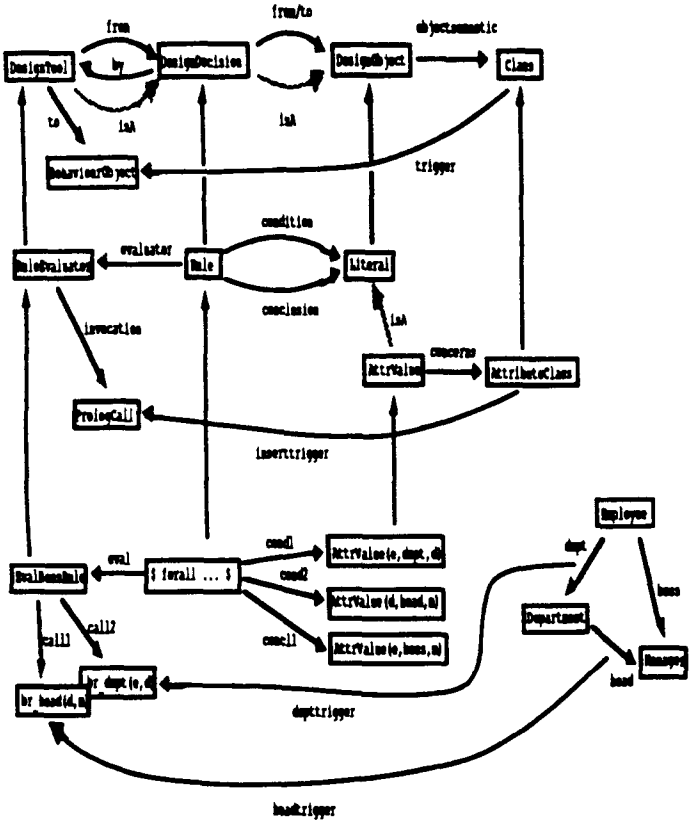

Fig. 7: Rule model with triggers

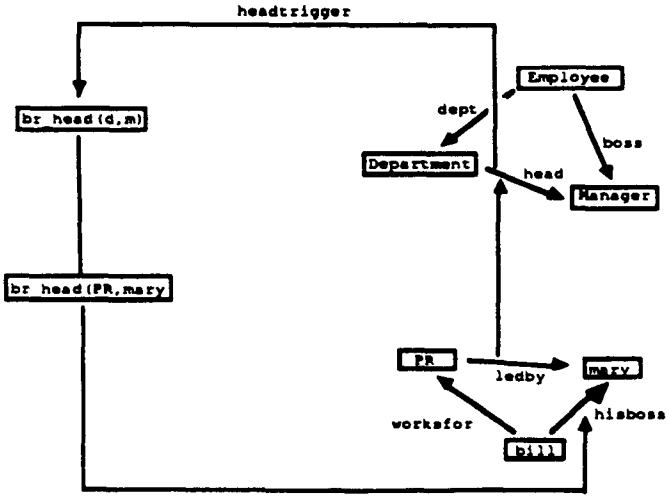

Fig. 8: Rule triggering due to attribute insertion

Similarly, *queries* to the knowledge base are expressed by instantiating a `QueryClass` with an embedded deductive rule (fig. 9). Answers are seen as instances of the query yielding instantiations of the answer variables. This uniform representation and way of accessing the results of a query suggests a formalism for handling derived data as objects; in [JJR89b], we discuss how to exploit this representation for *version and configuration management* of KB modules or sub-worlds [WA86], and for the modelling of *user interfaces*.

```
IndividualClass Bigsalaryquery in
QueryClass with
  answervariable
    x: Employee
  query
    ql: $ each x/Employee exists y/Money
          AttrValue(x,salary,y) and
          y > 10000 $
end Bigsalaryquery
```

Fig. 9: An example query in ConceptBase

The effect of this approach to derived data management is similar to the one achieved in Postgres [SHP88] but with more strict supervision of generated procedures. Explicit modelling of derived object classes also suggests a choice whether instances of these classes should be stored redundantly or recomputed; we have been looking at algorithms such as incremental view maintenance [BLAK87], the database version of Rete [SLR88], and COSMA [RS89] to explore the trade-offs. It soon became clear, however, that we should look for a more compact representation that fits better the semantic network structure we have; specifically, the documentation of derivation paths by links that generalize join indexes appears as an attractive solution. Later, we discovered that this idea has been independently (and earlier) explored in the ADMS prototype [ROUS89]; our implementation strategy is also close to that described for main-memory databases in [PT88]. From the viewpoint of the model described above, all these approaches amount to the modelling not just of derived data but also of their derivation proofs, as CML/Telos objects. Since this idea is very natural in the context of software engineering (e.g., test specifications), our model is easily extensible to this addition.

Redundant Derivations: Proofs as Objects

The evaluation of rules, queries and integrity constraints is usually regarded as an atomic operation. But in many design applications, this approach to rule evaluation and integrity control has been criticized as naive; these applications need human intervention and detailed proof analysis, even hand-coded test procedures for which a declarative expression is impractical. For example, mathematical proofs can take several days or even centuries. For some theorems, the prover has to develop new theories beforehand. Inspired by needs for specification verification and prototype testing in the DAIDA information systems development environment, we have therefore extended the basic rule model.

Essentially, a proof is documented as a process of proof steps each of which supported by its specific proof assistant tool and environment of existing lemmas and proof strategies. Fig. 10 shows the corresponding meta-level schema, whereas fig. 11 illustrates this model by the evaluation structure for the example rule introduced earlier -- of course, this is a *very* simplistic example but it may give the idea. In the DAIDA project, another prover we use is Abrial's [ABRI86] B-Tool, a theorem-proving assistant for specification refinement and verification [WNS89].

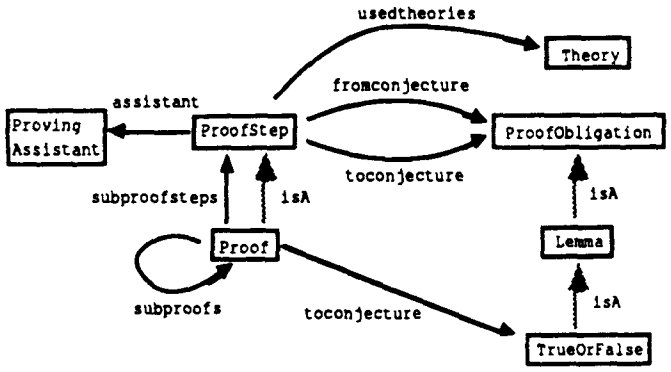

Fig. 10: CML model for proofs

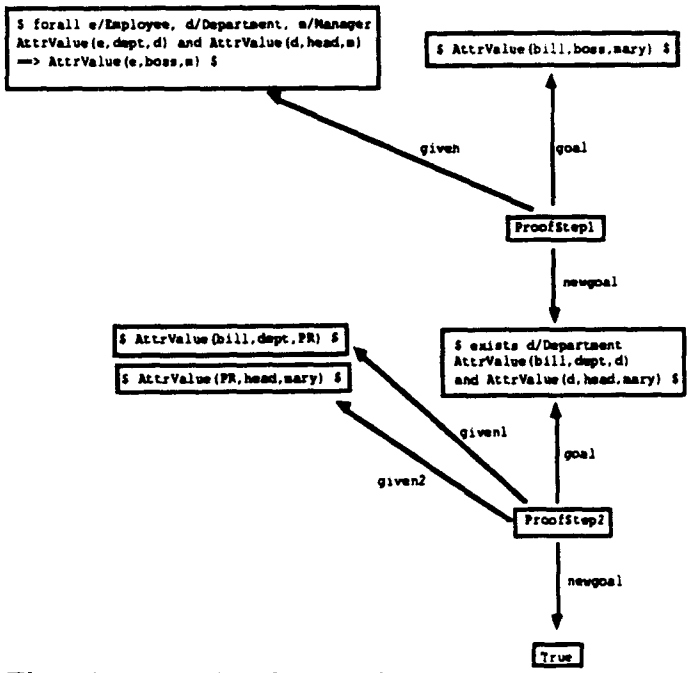

Fig. 11: Example of a proof

With this model, the software environment supports the proof process by recording which proof obligations (lemmata) have not yet been proven. One can backtrace the unsuccessful proof steps in order to find alternate solutions. The system can record which tools and theories where used for the different steps and even how these theories evolved during the process (though design decisions for changing or creating theories are not shown in the model).

As mentioned, instances of such a proof procedure model can be viewed as join indexes which, as discussed in [ROUS88], allow the efficient querying and incremental update of database structures. More detailed dependency modelling also yields the basis for truth maintenance or belief revision systems [DOYL79].

System Status and Current Work

ConceptBase has been under development in the context of ESPRIT project DAIDA since early 1986. A first prototype was completed in spring, 1988, and distributed to a number of places for experimental applications; a second prototype with the full functionality described in this paper is currently being finalized. The system is implemented in BIM-Prolog and C, using the SUNVIEW package to provide a hypertex-like user interface. ConceptBase runs on SUN under UNIX and on VAX under VMS. Further extensions to handle belief revision, multimedia data, and cooperative group design tasks are pursued in the context of the recently begun ESPRIT projects MULTIWORKS and COMPULOG. Of specific interest in our research is also the interaction of rule/constraint/query processing with temporal information on the objects; a study of related optimization issues is made in [JK89].

Acknowledgments. Work on ConceptBase is supported by the European Commission under ESPRIT Contract 892 (DAIDA), and by the Deutsche Forschungsgemeinschaft under Contract Ja445/1-1. Eva Krüger and Martin Staudt have participated in the implementation of the rule component, whereas Thomas Rose contributed strongly to the underlying software process data model. Work on proof modelling was done in collaboration with Ingrid Wetzel of the University of Frankfurt. Thanks are also due to Manolis Koubarakis and John Mylopoulos for useful discussions on CML/Telos implementation.

References

[ABRI86] Abrial, J.R. (1986) An informal introduction to B. Draft paper.

[BDM88] Bry, F., Decker, H., Manthey, R. (1988). A uniform approach to constraint satisfaction and constraint satisfiability in deductive databases. *Proc. EDBT '88*, Venice, Italy, 488-505.

[BLAK87] Blakeley, J.A. (1987). Updating materialized database views. Ph.D. thesis, University of Waterloo, Ont.

[BR86] Bancilhon, F., Ramakrishnan, R. (1986). An amateur's introduction to recursive query processing. *Proc. ACM-SIGMOD Conf.*, Austin, Tx, 16-52.

[DOYL79] Doyle, J. (1979). A truth maintenance system. *Artificial Intelligence 12*, 3, 231-272.

[FREY87] Freytag, J.C. (1987). A rule-based view of query optimization. *Proc. ACM-SIGMOD Conf.*, San Francisco, Ca, 173-180.

[GD87] Graefe, G., DeWitt, D.J. (1987). The EXODUS query optimizer generator. *Proc. ACM-SIGMOD Conf.*, San Francisco, Ca, 160-172.

[GS86] Gallagher, J., Solomon, L. (1986). CML Support System. Report, ESPRIT Project 107 (LOKI), SCS Hamburg, W. Germany.

[JJR88] Jarke, M., Jeusfeld, M., Rose, T. (1988). A KBMS for database software evolution: documentation of first ConceptBase prototype. Report MIP-8819, Univ. Passau, W.Germany.

[JJR89a] Jarke, M., Jeusfeld, M., Rose, T. (1989). A software process data model for knowledge engineering in information systems. *Information Systems*, to appear.

[JJR89b] Jarke, M., Jeusfeld, M., Rose, T. (1989). Software process modelling as a strategy for KBMS implementation. Universität Passau, W.Germany.

[JK88] Jarke, M., Koubarakis, M. (1989). Query optimization in KBMS. Report KRR-TR-89-6, University of Toronto.

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

[KMSB89] Koubarakis, M., Mylopoulos, J., Stanley, M., Borgida, A.. (1989). Telos: Features and Formalization. Technical Report FORTH/CSI/TR/1989/ 018, Computer Science Institute, Crete/Greece.

[KRÜG89] Krüger, E. (1989). Integritätsprüfung in deduktiven Objektbanken am Beispiel ConceptBase. Diploma thesis, Universität Passau, W. Germany.

[KT89] Koubarakis, M., Topaloglou, T. (1989). An implementation of Telos. Computer Science Technical Report, Univ. Toronto, Ont, forthcoming.

[LT85] Lloyd, J.W., Topor, R.W. (1985). A basis for deductive database systems. *Logic Programming 2*, 2, 93-109 and *3*, 1, 55-67.

[MS81] McKay, Shapiro (1981). Using active connection graphs for reasoning with recursive rules. *Proc. 7th IJCAI*, Vancouver, B.C.

[PT88] Pucheral, P., Thevenin, M. (1988). A graph based data structure for efficient implementation of main-memory DBMS. INRIA, Rocquencourt/ France.

[ROUS89] Roussopoulos, N. (1989). The incremental access method of view cache: concept, algorithms, and cost analysis. Report UMIACS-TR-89-15, University of Maryland, College Park.

[RS89] Regnier, M., Simon, E. (1989). Efficient evaluation of production rules in a DBMS. INRIA, Rocquencourt, France.

[SHP88] Stonebraker, M., Hanson, E., Potamianos, S. (1988). The Postgres rule manager. *IEEE Trans. Software Engineering 14*, 7, 897-907.

[SLR88] Sellis, T., Lin, L., Raschid, L. (1988). Implementing large production systems in a DBMS environment: concepts and algorithms. *Proc. ACM-SIGMOD Conf.*, Chicago, Il., 404-412.

[WA86] Wile, D.S., Allard, D.G. (1986). Worlds: an organizing structure for object-bases. *Proc. 2nd Symposium on Practical Software Environments.*

[WNS89] Wetzel, I., Niebergall, P., Schmidt, J.W. (1989). A mapping assistant for database program development. Report, ESPRIT project 892 (DAIDA), Universität Frankfurt, W. Germany.