Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Document
D-94-11

Working Notes of the KI'94 Workshop:

KRDB'94
Reasoning about Structured Objects: Knowledge Representation Meets Databases

Saarbrücken, September 20-22, 1994

F. Baader, M. Buchheit, M. A. Jeusfeld, W. Nutt (Eds.)

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Postfach 20 80
D-67608 Kaiserslautern, FRG
Tel.: (+49 631) 205-3211/13
Fax: (+49 631) 205-3210

Stuhlsatzenhausweg 3
D-66123 Saarbrücken, FRG
Tel.: (+49 681) 302-5252
Fax: (+49 681) 302-5341

Working Notes of the KI'94 Workshop: KRDB'94 - Reasoning about Structured Objects: Knowledge Representation Meets Databases

F. Baader, M. Buchheit, M. A. Jeusfeld, W. Nutt (Eds.)

DFKI-D-94-11

This work has been supported by a grant from The Federal Ministry for Research and Technology (FKZ ITW-9201).

ISSN 0946-0098

Working Notes of the KI'94 Workshop:

KRDB-94
Reasoning about Structured Objects: Knowledge Representation Meets Databases

Saarbrücken, Germany, September 20-22, 1994

Organized by

Franz Baader
Lehr- und Forschungsgebiet Theoretische Informatik
RWTH Aachen
Aachen, Germany
baader@informatik.rwth-aachen.de

Martin Buchheit
German Research Center for Artificial Intelligence
Saarbrücken, Germany
buchheit@dfki.uni-sb.de

Manfred A. Jeusfeld
Lehrstuhl Informatik V (Informationssysteme)
RWTH Aachen
Aachen, Germany
jeusfeld@informatik.rwth-aachen.de

Werner Nutt
German Research Center for Artificial Intelligence
Saarbrücken, Germany
nutt@dfki.uni-sb.de

This collection of papers forms the permanent record of the KRDB'94 Workshop "Reasoning about Structured Objects: Knowledge Representation Meets Databases", that is held at the University of Saarbrücken, Germany on September 20-22, 1994, as part of the 18th German Annual Conference on Artificial Intelligence. The workshop is set up to be as informal as possible, so this collection cannot hope to capture the discussions associated with the workshop. However, we hope that it will serve to remind participants of their discussion at the workshop, and provide non-participants with indications of the topics that were discussed at the workshop.

Object-centered formalisms for domain modeling play an important role both in knowledge representation (KR) and in the database (DB) area. Nevertheless, there has been little cross-fertilization between the two areas. Research in databases was mostly concerned with handling large amounts of data that are represented in a rather inexpressive formalism, whereas KR concentrated on intensional inferences in relatively small knowledge bases. However, many of today's problems demand sophisticated reasoning on complex and large-scale objects. The workshop brings together researchers from both areas to identify and discuss problems and applications where the combination of KR and DB techniques may provide new solutions.

For the following (non-exclusive) list of questions, such a combination seems to be most promising:

- KR formalisms as schema languages in DB: Is it possible to specify realistic DBs this way? Can the inference mechanisms from KR support the schema design?
- Distributed information sources: How can one describe their interaction in a changing environment?
- Advanced query processing: How can schema knowledge be utilized for query optimization? How can it be used to generate intensional answers?

Two invited talks introduce into the topic of the workshop. Maurizio Lenzerini covers a broad range of services offered by concept logic reasoning on database schemata. Marc Scholl reports on the application of this kind of reasoning within the COCOON project.

Two sessions are devoted to schema design of data and knowledge bases. Gottfried Vossen presents core aspects for object-oriented database models. Different approaches are taken by Martin Buchheit et al. who find that concept languages subsume the structural part of object-oriented database models. Edith Buchholz and Antje Düsterhöft propose a natural language frontend resulting in a data dictionary for the database schema. Finally, Wolfgang Benn takes a data dictionary as input and puts a taxonomic layer on top of it in order to produce integrate database schemata and to reason on completeness.

Another area of interest is the relationship of knowledge representation and query languages. Ulrich Hustadt argues against the standard closed-world-assumption in database query languages and votes for an epistemic operator that can stepwisely convert a knowledge base into a database. Klaus Schild augments this argument by his investigation of null values (known from databases) as incomplete knowledge in concept logics. Manfred Jeusfeld proposes a language that defines interfaces between programs and databases by a restricted concept language. D. Beneventano et al. argue that a concept logic which explicitly distinguishes value types from object classes gives an attractive framework for schema design and query optimization in object-oriented databases. Albrecht Schmiedel concludes the area by presenting indexes for query processing that are based on the concept logic system BACK.

In the session on techniques for modeling business data, Harald Huber reports from empirical studies about the shortages of widely used data modeling languages. Ramzi Guietari et al. present a formalism called OLSEN that answers to at least some of the shortages by adding the dimensions of time, organisation, and measurement to the data modeling layer.

The last session is devoted to database implementations of KR systems. M.C. Norrie et al. map the KR language FRM to the COCOON data model which itsself is defined on top of the relational data model. Paolo Bresciani integrates a standard database as assertional knowledge (DBox) into a KR system based on concept logics.

Contents

Description Logics for Schema Level Reasoning in Databases

Maurizio Lenzerini

Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italy

Abstract

Several recent papers point out that the research on Description Logics and their associated reasoning techniques can be profitably exploited in several ways in the area of Databases. We argue that one of the most important aspects of Databases where we can take advantage of Description Logics is the one related to schema level reasoning, i.e., reasoning at the intensional level of a database. This is the case in schema design, schema maintenance, schema integration, schema translation, integrity checking, query evaluation in cooperative information systems, etc. Indeed, on the one hand Description Logics can be seen as very powerful data models, and on the other hand, they can serve as unified formalisms that capture object-oriented, semantic and conceptual data models proposed in the literature. Most importantly, they can provide useful reasoning services in all the above mentioned tasks.

This article was processed using the LaTeX macro package with LLNCS style

Database Views on KR Classification — Abstract —

Marc H. Scholl

University of Ulm, Faculty of Computer Science
D-89069 Ulm, Germany
scholl@informatik.uni-ulm.de

Abstract. The database models for Object Database Systems (ODBMSs) include many modeling concepts that originate in semantic data models, that were formerly used for database design purposes, or in (object-oriented) programming languages. To some extent, research on data models and query languages for such ODB models has already reached a consensus, not on one particular model or language, but on the core of what should be considered furtheron. Other aspects, such as view support for example, are less common. We argue that the KL-ONE style terminological logics can provide a very convenient basis for the integration of a flexible view mechanism into object databases. KL-ONE defined concepts correspond to database views (classes of objects that are derived by a qeury expression). Updates to such views can be propagated to base classes if the view classes are inserted into the global class(ification) hierarchy. Therefore, object databases need the inference services that KL-ONE systems provide (classification, subsumption, ...). We report on the experiences that we gained in the COCOON project, where this approach was pursued over the last few years.

This article was processed using the LaTeX macro package with LLNCS style

Formalization of OODB Models

Gottfried Vossen
Institut für Wirtschaftsinformatik, Universität Münster
Grevenerstraße 91, 48159 Münster

1 Introduction

Object-oriented data models represent a current end-point in the evolution of data models [23]. Their formalization has been attempted in a variety of papers, including [5; 6; 19]. This short paper indicates what we consider the common intersection of these (and other) approaches; we list the relevant features and components, and give an idea of how to formalize the notion of an object-oriented database schema.

An object-oriented data model has to capture a variety of requirements [8; 27], which differ considerably from those that traditional data models have to meet. However, many system developers seem not to care about *formal* models as a solid foundation of their system, but simply design a "data definition language" in which the relevant features can be coded. In our opinion, a formal model for object-oriented databases basically has to capture the same intuitions as models for other types of databases, which are the following:

1. It has to provide an adequate linguistic abstraction for certain database applications.
2. It should provide a precise semantics for a data definition language.
3. It has to be composed of both a specification and an operational part.
4. It represents a computational paradigm as a basis for formal investigations.

In this short note, we do not present a comprehensive survey of formal models for object-oriented databases which have been proposed in the literature, but instead try to point out the fundamentals of how such models are obtained. The result can be considered as a framework in which the essentials of the object-oriented paradigm can be expressed concisely and further studied. Indeed, we give hints to various such investigations that have recently been undertaken.

2 Core Aspects of Formal OO Models

In this section, we describe what we perceive as the core aspects of various proposals for object models, and we do so by distinguishing structural from behavioral aspects. Thus, we generally consider *schemas*, the central notion of any conceptual database description, to be pairs of the form $S = (S_{\text{struc}}, S_{\text{behav}})$; in what follows, we first consider each component in isolation and then indicate how the two interact. We mention, however, that while it is generally agreed that an object-oriented data model has to capture both structure and behavior, the former can be obtained by using the experience from the relational, nested relational and complex-object models, but the latter represents a completely new challenge to database researchers. Consequently, a consensus seems achieved for structure, but not for behavior.

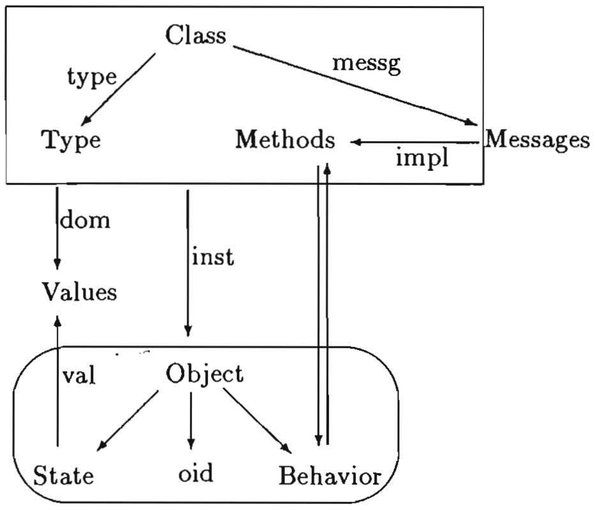

Figure 1: Core Aspects of an Object Model.

The core aspects of formal models for object-oriented databases are summarized in Figure 1, in which labels of arrows represent function names. In brief, the only structuring mechanism is the *class* which describes both structure and behavior for its instances, the *objects*. Structure is captured as a *type* for a class (in our notation, a function "type" associates a type with each class; the other function names shown above are to be interpreted similarly, see below). A type is nothing but a description of a *domain*, i.e., a set of values, and may or may not be named (in the former case, type names distinct from class names and attribute names must be provided). Values comprise the state of an object and can be as complex as the type system allows (i.e., depending on the availability of base types and constructors like tuple, set, bag, list, etc.). Behavior is manifested in a set of *messages* associated with each class (its external interface), which are internally implemented using *methods* that are executable on objects. Hence, objects have a state *and* a behavior; in addition, they are uniquely identified. Messages are specified by providing a *signature*, and by associating several signatures with the same message name, the latter gets *overloaded*. Not shown in Figure 1 is the possibility to organize classes in an *inheritance hierarchy*; also not shown is the fact that class attributes are allowed to reference other classes, thereby forming an *aggregation lattice*.

We next look at structural as well as behavioral aspects in more detail. Regarding the modeling of structure, more precisely highly-structured information, complex data types are all that is basically needed, since they serve as descriptions for domains of complex values. One way to introduce such types, i.e., to define a *type system* T, is the following:

(i) `integer, string, float, boolean` $\subseteq T$;

(ii) if A_i are distinct attributes and $t_i \in T$, $1 \leq i \leq n$, then $[A_1 : t_1, \ldots, A_n : t_n] \in T$ ("tuple type");

(iii) if $t \in T$, then $\{t\} \in T$ ("set type");

(iv) if $t \in T$, then $< t > \in T$ ("list type").

In other words, a type system is made up of *base types*, from which complex types may be derived using (eventually attributes and) *constructors*. Note that this requires nothing additional but the availability of attribute names. Clearly, other base types as well as additional or alternative constructors could straightforwardly be included. Notice also that here types are not named; for practical reasons, the use of type names may be desirable (e.g., in order to be able to reuse type definitions in various places throughout a schema), and if it is, it can easily be added to the above in the way indicated earlier.

The notion of a domain as a "reservoir" of possible values can be defined as follows; it just has to obey constructor applications:

(a) dom(`integer`) is the set of all integers; dom is analogously defined for `string, float, boolean`;

(b) $\text{dom}([A_1 : t_1, \ldots, A_n : t_n]) := \{[A_1 : v_1, \ldots, A_n : v_n] \mid (\forall\, i, 1 \leq i \leq n)\ v_i \in \text{dom}(t_i)\}$;

(c) $\text{dom}(\{t\}) := \{\{v_1, \ldots, v_n\} \mid (\forall\, i, 1 \leq i \leq n)\ v_i \in \text{dom}(t)\}$;

(d) $\text{dom}(< t >) := \{< v_1, \ldots, v_n > \mid (\forall\, i, 1 \leq i \leq n)\ v_i \in \text{dom}(t)\}$.

In a structurally *object-oriented* context, the first thing that needs to be introduced beyond complex types and domains as defined above is the possibility to *share* pieces of information between distinct types, or to *aggregate* objects from simpler ones. At the level of type declarations, an easy way to model this is the introduction of another reservoir of names, this time called *class names*, which are additionally allowed as types. In other words, *object types* are complex types as above with the following new condition:

(v) $C \subseteq T$, where C is a finite set of class names.

This states nothing but the fact that class *names* are allowed as types (below we will complement this with the requirement that classes themselves *have* types).

The intuition behind this new condition is that objects from the underlying application all are distinguished by their identity, get collected into classes, and can reference other objects (share subobjects). To provide for this at the level of domains, let us first assume the availability of a finite set OID of *object identifiers* which includes the special identifier `nil` (to capture "empty" references); next, *object domains*, i.e., sets of possible values for objects are complex values as above with the following additional condition:

(e) $\text{dom}(c) = OID$ for each $c \in C$.

Thus, classes are assumed to be instantiated by objects (class-name types take object identifiers as values, in the same way as, say, the `integer` type takes integer numbers as values). Clearly, this alone is not enough, since class instances commonly have distinct sets of object identifiers associated with them. We will show below how that (and, for example, the fact that sometimes inclusion dependencies need to hold between sets of class instances) is captured at the instance level.

The object-oriented paradigm has another dimension for organizing information besides aggregation, which is inheritance, or the possibility to define a class as a specialization of one or more other classes. To this end, a *subtyping relation* is needed through which it can be expressed that a subclass *inherits* the structure of a superclass. Such a relation can be defined in various ways; for example, it can be defined semantically by requiring that the sets of values or instances of types, where one is a subtype of the other, are in a subset relationship. We prefer a simpler, syntactical approach, which has, for example, the advantage that checking subtype relationships can be automated:

Let T be a set of object types. A subtyping relation $\leq\ \subseteq T \times T$ is defined as follows:

(i) $t \leq t$ for each $t \in T$,

(ii) $[A_1 : t_1, \ldots, A_n : t_n] \leq [A'_1 : t'_1, \ldots, A'_m : t'_m]$ if

  (a) $(\forall\, A'_j, 1 \leq j \leq m)(\exists\, A_i, 1 \leq i \leq n)\ A_i = A'_j \wedge t_i \leq t'_j$,

  (b) $n \geq m$,

(iii) $\{t\} \leq \{t'\}$ if $t \leq t'$,

(iv) $< t > \leq < t' >$ if $t \leq t'$.

With these preparations, we arrive at the following definition for objectbase schemas that can describe structure of arbitrary complexity: A *structural schema* is a named quadruple of the form $S_{\text{struc}} = (C, T, \text{type}, \text{isa})$ where

(i) C is a (finite) set of class names,

(ii) T is a (finite) set of types which uses as class names only elements from C,

(iii) type : $C \rightarrow T$ is a total function associating a type with each class name,

(iv) isa $\subseteq C \times C$ is a partial order on C which is consistent w.r.t. subtyping, i.e.,
c isa $c' \Rightarrow \text{type}(c) \leq \text{type}(c')$ for all $c, c' \in C$.

This definition resembles what can be found in a variety of models proposed in the literature, including [17; 19; 20; 25] and others. Notice that it still leaves several aspects open, like single vs. multiple inheritance; if the latter is desired, a condition needs to be added stating how to conflicts should be resolved. Also, implementations typically add a number of additional features, like attributes as functions [22; 29], a distinction of class attributes from instance attributes (the latter are shared by all objects associated with a class, while the former represent, for example, aggregate information like an average salary only relevant to the class as a whole) [7], a unique root of the class hierarchy from which every class inherits [20], a distinction between private and public attributes [12], a different set of constructors (like one with an additional array constructor to describe matrices), an explicit inclusion of distinct types of relationships between classes and their objects (in particular various forms of composition, see [18]), integrity constraints which represent semantic information on the set of valid databases instances (a proposal in that direction appears in [3; 4], where *object constraints, class constraints*, and *database constraints* are distinguished). For another example, the ODMG-93 proposal for a standardized model [10] contains explicit keys, (binary) relationships, and inverse attributes. None of these features appear in our model, the reason being that these are *not* specific to object-orientation.

The second important aspect of an object-oriented database is that it is intended to capture behavior, besides structure. To this end, the relevant intuition is that classes have attached to them a set of messages, which are specified in the schema via signatures, and which are implemented as methods. In addition, behavior can be inherited by subclasses, and message names can be overloaded, i.e., re-used in various contexts.

So a *behavioral schema* is a named five-tuple of the form $S_{\text{behav}} = (C, M, P, \text{ messg}, \text{ impl})$ where

(i) C is a (finite) set of class names as above (again needed here since references to it have to be made),

(ii) M is a (finite) set of *message names*, where each $m \in M$ has associated with it a nonempty set $\text{sign}(m) = \{s_1, \ldots s_l\}$, $l \geq 1$, of signatures; each s_h, $1 \leq h \leq l$, has the form $s_h : c \times t_1 \times \ldots \times t_p \rightarrow t$ for $c \in C, t_1, \ldots, t_p, t \in T$
(each signature has the receiver of the message as its first component),

(iii) P is a (finite) set of methods or programs,

(iv) messg : $C \rightarrow 2^M$ is a mapping s.t. for each $c \in C$ and for each $m \in \text{messg}(c)$ there exists a signature $s \in \text{sign}(m)$ satisfying $s[1] = c$,

(v) impl: $\{(m, c) \mid m \in \text{messg}(c)\} \rightarrow P$ is a partial function.

In combining structural and behavioral schemas, we finally obtain an *objectbase schema* of the form

$S = (C, (T, \text{ type}, \text{ isa},), (M, P, \text{ isa}, \text{ messg}, \text{ impl}))$.

S is called *consistent* if the following conditions are satisfied:

(i) c isa c' implies $\text{messg}(c') \subseteq \text{messg}(c)$ for all $c, c' \in C$,

(ii) if c isa c' and $s, s' \in \text{sign}(m)$ for $m \in M$ such that $s : c \times t_1 \times \ldots \times t_n \rightarrow t$, $s' : c' \times t'_1 \times \ldots \times t'_n \rightarrow t'$, then $t_i \leq t'_i$ for each i, $1 \leq i \leq n$, and $t \leq t'$,

(iii) for each $m \in \text{messg}(c)$ there exists a $c' \in C)$ s.t. c isa c' and $\text{impl}(m, c')$ is defined.

Condition (i) just says that subclasses inherit the behavior of their superclasses. Condition (ii) says that message-name overloading is done with compatible signatures, and is called the *covariance condition* in [20; 9]. The covariance condition is a significant difference from what is used at a corresponding point in programming languages, and which is known as the *contravariance* condition; for a detailed explanation, see [9]. Finally, Condition (iii) states that for each message associated with a class, its implementation must at least be available in some superclass.

It is interesting to note that various natural conditions can be imposed on the programs that are used as implementations of messages. We now sketch one of them, which is based on the view that programs are functions on domains [20]. More formally, if $m \in M$ and $s : c \times t_1 \times \ldots \times t_n \rightarrow t \in \text{sign}(m)$, then $\text{impl}(m, c)$, if defined, is a program $p \in P$ of the form

$p : \text{dom}(c) \times \text{dom}(t_1) \times \ \ldots \ \times \text{dom}(t_n) \rightarrow \text{dom}(t)$

The condition in question informally states that if message overloading appears in isa-related classes (so that the corresponding signatures satisfy the covariance condition), then the associated programs coincide (as functions) on the subclass. More formally, we have: If $|\text{sign}(m)| > 1$ for some $m \in M$, then the following holds: If $s, s' \in \text{sign}(m)$ such that $s : c \times t_1 \times \ldots \times t_n \rightarrow t$, $s' : c' \times t'_1 \times \ldots \times t'_n \rightarrow t'$, c isa c', $t_i \leq t'_i$ for each i, $1 \leq i \leq n$, $t \leq t'$, and $\text{impl}(m, c) = p$, $\text{impl}(m, c') = p'$, then p and p' agree on $\text{dom}(c) \times \text{dom}(t_1) \times \ \ldots \ \times \text{dom}(t_n)$.

A variety of formal investigations for behavioral schemata in the sense defined above can already be found in the literature, which investigate questions including termination of method executions, limited depth of method-call nestings (an issue related to precompilation of method executions), well-definedness of method calls, i.e., consistency as well as reachability considerations (issues related to type inference and schema evolution), expressiveness of method implementation languages (relative to some notion of completeness), complexity of method executions, or potential parallelism of method evaluations. To investigate such issues, our general notion of schema is made precise in various ways. For example, [15] fixes a simple imperative language for implementing methods as *retrieval programs*, contrasts them with *update programs* and shows undecidability results for the latter. [1; 2] as well as

[11] introduce distinct notions of a *method schema* to study behavioral issues of OODBS; for example, [2] investigates implications of the covariance condition using the formalism of program schemas, while [11] looks at tractability guarantees corresponding to those known for relational query languages. Also, it is pretty straightforward to define an *object algebra* for a model like the one sketched in the previous section; see, for example, the papers in [13]. That carries over to issues like query optimization, implementation of operations, and query processing. A survey of other recent investigations that have similar bases or origins can be found in [28].

We emphasize again that the model just sketched can be seen as description of the core of vastly any object-oriented model; however, this is valid only relative to the fact that many specialities, which have been proposed in the literature, or which are being built into commercial systems, are neglected here.

We conclude this section with a brief indication of how *object databases*, i.e., sets of class instances or extensions, can be defined over a given schema: For a given objectbase schema S, an *objectbase* over S is a triple $d(S) = (O, \text{inst}, \text{val})$ s.t.

(i) $O \subseteq OID$ is a finite set of object identifiers,

(ii) inst: $C \rightarrow 2^O$ is a total function satisfying the following conditions:

  (a) if $c, c' \in C$ are not (direct or indirect) subclasses of each other, then $\text{inst}(c) \cap \text{inst}(c') = \emptyset$,

  (b) if c isa c', then $\text{inst}(c) \subseteq \text{inst}(c')$,

(iii) val: $O \rightarrow V$ is a function s.t. $(\forall c \in C)(\forall o \in \text{inst}(c))\, \text{val}(o) \in \text{dom}(\text{type}(c))$.

Notice that this definition closes the problem left open earlier, namely that class domains originally were simply the set OID.

3 Open Issues

We next survey several modeling issues in object-oriented databases which have not yet received enough research attention:

1. *Entities can have roles that vary over time.* For example, some person object may at one point be a student, at another an employee, and at a third a club member; while the person's identity never changes, its type changes several times.
2. *Entities can have multiple types at the same time.* For example, a person may be a student, an employee, and a club member simultaneously. So far the only way to represent this in an object-oriented database is by multiple inheritance, but this might not be appropriate since it can result in a combinatorial explosion of sparsely populated classes [21].
3. *Objects can be in various stages of development.* For example, in a design environment it is usually necessary to maintain incomplete designs, i.e., objects whose types get completed in the course of time.
4. *Classes may contain "too few" instances.* For example, consider a database in which all persons living in a large country are represented. In this context, so many combinations of meaningful properties have to be distinguished that it might become necessary to introduce artificial name constructions for classes, like *unmarried-nonstudent-autoOwner-renter-taxpayer* [26], and each such class has only very few instances. More generally, the name space available for classes might not be sufficient.
5. *Objects and their classes might come into existence in reverse order.* A database user in a design environment like CAD creates objects in the first place, not type definitions or even classes. The usage of databases thus differs considerably from traditional applications where schema design has to be completed prior to instance creation.

We mention that one issue or the other from this list is sometimes reflected already in existing models, but never as a basic design target. Alternative approaches, which takes these issues into consideration right from the start, appear, for example, in [21; 24; 16]. A possible general concept for the solution of these problems seems the exploitation of prototype languages, which suggest to model applications *without* a classification that partitions the world into entity sets. A prototype represents default behavior for some concept, and new objects can re-use part of the knowledge stored in a prototype by saying how they differ from it. Upon receiving a message an object does not understand, it can forward (delegate) it to its prototype to invoke more general behavior. In the area of object-oriented programming languages, many people believe that this approach has advantages over the class-based one with inheritance, with respect to the representation of default knowledge and incrementally and dynamically modifying concepts. The investigation of *classless* models in the context of object-oriented databases has only recently been proposed in [26], and a concrete model is reported in [14].

4 Conclusions

In this short paper we have tried to give a rough personal account of recent work on formal models for object-oriented databases. Although there is not a single uniform such model, the foundations on which such models have to be built seem understood, and even standardization efforts have recently been launched [10]. On the other hand, a number of interesting research issues still deserve further investigation. In particular, formal models as they are currently available seem hardly suited for the nonstandard applications which initiated the consideration of object-orientation in the context of databases. A reason seems to be that many researchers have too much of a relational background, and try to exploit that as long as possible; this is more than confirmed by the ODMG-93 proposal. As was done a number of years ago, when database people discovered what programming-language or knowledge-representation people had been studying for years already, it seems again necessary to take recent developments in these areas into account, and to adopt them for solving the problems database applications have.

References

[1] S. Abiteboul, P.C. Kanellakis: The Two Facets of Object-Oriented Data Models; IEEE Data Engineering Bulletin 14 (2) 1991, 3–7

[2] S. Abiteboul, P.C. Kanellakis, E. Waller: Method Schemas; Proc. 9th ACM Symposium on Principles of Database Systems 1990, 16–27

[3] P.M.G. Apers et al.: Inheritance in an Object-Oriented Data Model; Memoranda Informatica 90-77, University of Twente 1990

[4] H. Balsters et al.: Sets and Constraints in an Object-Oriented Data Model; Memoranda Informatica 90-75, University of Twente 1990

[5] F. Bancilhon, C. Delobel, P. Kanellakis (eds.): *Building an Object-Oriented Database System — The Story of O_2*. Morgan-Kaufmann 1992

[6] C. Beeri: A Formal Approach to Object-Oriented Databases; Data & Knowledge Engineering 5, 1990, 353–382

[7] E. Bertino et al.: An Object-Oriented Data Model for Distributed Office Applications; Proc. ACM Conference on Office Information Systems 1990, 216–226

[8] E. Bertino, L. Martino: Object-oriented Database Management Systems: Concepts and Issues; IEEE Computer 24 (4) 1991, 33–47

[9] E. Bertino, L. Martino: *Object-Oriented Database Systems*; Addison-Wesley 1993

[10] R.G.G. Cattell (ed.): *The Object Database Standard: ODMG-93*. Morgan-Kaufmann 1994

[11] K. Denninghoff, V. Vianu: The Power of Methods with Parallel Semantics; UCSD Technical Report No. CS91–184, University of California, San Diego, February 1991; extended abstract in Proc. 17th Int. Conference on Very Large Data Bases 1991, 221–232

[12] O. Deux et al.: The Story of O_2; IEEE Transactions on Knowledge and Data Engineering 2, 1990, 91–108

[13] J.C. Freytag, D. Maier, G. Vossen: *Query Processing for Advanced Database Systems*; Morgan-Kaufmann 1994

[14] M. Groß-Hardt, G. Vossen: *Towards Classless Object Models for Engineering Design Applications*; Proc. 4th International Conference on Database and Expert Systems Applications (DEXA) 1993, Prag, Springer LNCS 720, 36–47

[15] R. Hull, K. Tanaka, M. Yoshikawa: Behavior Analysis of Object-Oriented Databases: Method Structure, Execution Trees, and Reachability; Proc. 3rd FODO Conference, Springer LNCS 367, 1989, 372–388

[16] T. Imielinski et al.: Incomplete Objects — A Data Model for Design and Planning Applications; Proc. ACM SIGMOD International Conference on Management of Data 1991, 288–297

[17] A. Kemper et al.: GOM: A Strongly Typed Persistent Object Model with Polymorphism; Proc. German GI Conference on "Datenbanken für Büro, Technik und Wissenschaft" (BTW) 1991, Springer Informatik-Fachbericht 270, 198–217

[18] W. Kim: *Introduction to Object-Oriented Databases*; MIT Press 1990

[19] C. Lecluse et al.: O_2, an Object-Oriented Data Model; Proc. ACM SIGMOD International Conference on Management of Data 1988, 424–433

[20] C. Lecluse, P. Richard: Foundations of the O_2 Database System; IEEE Data Engineering Bulletin 14 (2) 1991, 28–32

[21] J. Richardson, P. Schwarz: Aspects: Extending Objects to Support Multiple, Independent Roles; Proc. ACM SIGMOD International Conference on Management of Data 1991, 298–307

[22] M.H. Scholl, H.J. Schek: A Relational Object Model; Proc. 3rd International Conference on Database Theory 1990, Springer LNCS 470, 89–105

[23] H.J. Schek, M.H. Scholl: Evolution of Data Models; Proc. Database Systems of the 90s, November 1990, Springer LNCS 466, 135–153

[24] E. Sciore: Object Specialization; ACM Transactions on Information Systems 7, 1989, 103–122

[25] D.D. Straube, M.T. Özsu: Queries and Query Processing in Object-Oriented Database Systems; ACM Transactions on Information Systems 8, 1990, 387–430

[26] J.D. Ullman: A Comparison of Deductive and Object-Oriented Database Systems; Proc. 2nd DOOD Conference, Springer LNCS 566, 1991, 263–277

[27] G. Vossen: *Datenmodelle, Datenbanksprachen und Datenbankmanagement-Systeme*; 2. Auflage, Addison-Wesley 1994

[28] G. Vossen: Database Theory: An Introduction; Technical Report, University of Münster, June 1994

[29] K. Wilkinson et al.: The Iris Architecture and Implementation; IEEE Transactions on Knowledge and Data Engineering 2, 1990, 63–75

Terminological Systems Revisited: Terminology = Schema + Views*

M. Buchheit[1] and **F. M. Donini**[2] and **W. Nutt**[1] and **A. Schaerf**[2]
1. German Research Center for Artificial Intelligence (DFKI), Saarbrücken, Germany
{buchheit,nutt}@dfki.uni-sb.de
2. Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", Italy
{donini,aschaerf}@assi.dis.uniroma1.it

Abstract

Traditionally, the core of a Terminological Knowledge Representation System (TKRS) consists of a so-called TBox, where concepts are introduced, and an ABox, where facts about individuals are stated in terms of these concepts. This design has a drawback because in most applications the TBox has to meet two functions at a time: on the one hand, similar to a database schema, framelike structures with typing information are introduced through primitive concepts and primitive roles; on the other hand, views on the objects in the knowledge base are provided through defined concepts.

We propose to account for this conceptual separation by partitioning the TBox into two components for primitive and defined concepts, which we call the *schema* and the *view* part. We envision the two parts to differ with respect to the language for concepts, the statements allowed, and the semantics.

We argue that by this separation we achieve more conceptual clarity about the role of primitive and defined concepts and the semantics of terminological cycles. Moreover, three case studies show the computational benefits to be gained from the refined architecture.

1 Introduction

Research on terminological reasoning usually presupposes the following abstract architecture, which reflects quite well the structure of existing systems. There is a logical representation language that allows for two kinds of statements: in the TBox or *terminology*, concept descriptions are introduced, and in the ABox or *world description*, individuals are characterized in terms of concept membership and role relationship. This abstract architecture has been the basis for the design of systems, the development of algorithms, and the investigation of the computational properties of inferences.

Given this setting, there are three parameters that characterize a terminological system: (*i*) the language for concept descriptions, (*ii*) the form of the statements allowed, and (*iii*) the semantics given to concepts and statements. Research tried to improve systems by modifying these three parameters. But in all existing systems and almost all theoretical studies language and semantics have been kept uniform.[1]

The results of these studies were unsatisfactory in at least two respects. First, it seems that tractable inferences are only possible for languages with little expressivity. Second, no consensus has been reached about the semantics of terminological cycles, although in applications the need to model cyclic dependencies between classes of objects arises constantly.

Based on an ongoing study of applications of terminological systems, we suggest to refine the two-layered architecture consisting of TBox and ABox. Our goal is twofold: on the one hand we want to achieve more conceptual clarity about the role of primitive and defined concepts and the semantics of terminological cycles; on the other hand, we want to improve the tradeoff between expressivity and worst case complexity. Since our changes are not primarily motivated by mathematical considerations but by the way systems are used, we expect to come up with a more practical system design.

In the applications studied we found that the TBox has to meet two functions at a time. One is to declare frame-like structures by introducing primitive concepts and roles together with typing information like isa-relationships between concepts, or range restrictions and number restrictions of roles. *E.g.*, suppose we want to model a company environment. Then we may introduce the concept Employee as a specialization of Person, having exactly one name of type Name and at least one affiliation of type Department. This is similar to class declarations in object-oriented systems. For this purpose, a simple language is sufficient. Cycles occur naturally in modeling tasks, *e.g.*, the boss of an Employee is

*This work was partly supported by the Commission of the European Union under ESPRIT BRA 6810 (Compulog 2), by the German Ministry of Research and Technology under grant ITW 92-01 (TACOS), and by the CNR (Italian Research Council) under Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo, LdR "Ibridi."

[1]In [Lenzerini and Schaerf,1991] a combination of a weak language for ABoxes and a strong language for queries has been investigated.

also an Employee. Such declarations have no definitional import, they just restrict the set of possible interpretations.

The second function of a TBox is to define new concepts in terms of primitive ones by specifying necessary *and* sufficient conditions for concept membership. This can be seen as defining *abstractions* or *views* on the objects in the knowledge base. Defined concepts are important for querying the knowledge base and as left-hand sides of trigger rules. For this purpose we need more expressive languages. If cycles occur in this part they must have definitional import.

As a consequence of our analysis we propose to split the TBox into two components: one for declaring frame structures and one for defining views. By analogy to the structure of databases we call the first component the *schema* and the second the *view* part. We envision the two parts to differ with respect to the language, the form of statements, and the semantics of cycles.

The schema consists of a set of primitive concept introductions, formulated in the *schema language*, and the view part by a set of concept definitions, formulated in the *view language*. In general, the schema language will be less expressive than the view language. Since the role of statements in the schema is to restrict the interpretations we want to admit, first order semantics, which is also called descriptive semantics in this context (see Nebel 1991), is adequate for cycles occurring in the schema. For cycles in the view part, we propose to choose a semantics that defines concepts uniquely, *e.g.*, least or greatest fixpoint semantics.

The purpose of this work is not to present the full-fledged design of a new system but to explore the options that arise from the separation of TBoxes into schema and views. Among the benefits to be gained from this refinement are the following three. First, the new architecture has more parameters for improving systems, since language, form of statements, and semantics can be specified differently for schema and views. So we found a combination of schema and view language with polynomial inference procedures whereas merging the two languages into one would have led to intractability. Second, we believe that one of the obstacles to a consensus about the semantics of terminological cycles has been precisely the fact that no distinction has been made between primitive and defined concepts. Moreover, intractability results for cycles mostly refer to inferences with defined concepts. We proved that reasoning with cycles is easier when only primitive concepts are considered. Third, the refined architecture allows for more differentiated complexity measures, as shown later in the paper.

In the following section we outline our refined architecture for a TKRS, which comprises *three* parts: the *schema*, the *view taxonomy*, and the *world description*, which comprise primitive concepts, defined concepts and assertions in traditional systems. In the third section we show by three case studies that adding a simple schema with cycles to existing systems does not increase the complexity of reasoning.

2 The Refined Architecture

We start this section by a short reminder on concept languages. Then we discuss the form of statements and their semantics in the different components of a TKRS. Finally, we specify the reasoning services provided by each component and introduce different complexity measures for analyzing them.

2.1 Concept Languages

In concept languages, complex concepts (ranged over by C, D) and complex roles (ranged over by Q, R) can be built up from simpler ones using concept and role forming constructs (see Tables 1 and 2 a set of common constructs). The basic syntactic symbols are (*i*) *concept names*, which are divided into *schema names* (ranged over by A) and *view names* (ranged over by V), (*ii*) *role names* (ranged over by P), and (*iii*) *individual names* (ranged over by a, b). An *interpretation* $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ consists of the *domain* $\Delta^\mathcal{I}$ and the *interpretation function* $\cdot^\mathcal{I}$, which maps every concept to a subset of $\Delta^\mathcal{I}$, every role to a subset of $\Delta^\mathcal{I} \times \Delta^\mathcal{I}$, and every individual to an element of $\Delta^\mathcal{I}$ such that $a^\mathcal{I} \neq b^\mathcal{I}$ for different individuals a, b (*Unique Name Assumption*). Complex concepts and roles are interpreted according to the semantics given in Tables 1 and 2, respectively.

In our architecture, there are two different concept languages in a TKRS, a *schema language* for expressing schema statements and a *view language* for formulating views and queries to the system.

2.2 The Three Components

We first focus our attention to the schema. The schema introduces concept and role names and states elementary type constraints. This can be achieved by *inclusion axioms* having one of the forms:

$$A \sqsubseteq D, \qquad P \sqsubseteq A_1 \times A_2,$$

where A, A_1, A_2 are schema names, P is a role name, and D is a concept of the schema language. Intuitively, the first axiom states that all instances of A are also instances of D. The second axiom states that the role P has domain A_1 and range A_2. A *schema* § consists of a finite set of schema axioms.

Inclusion axioms impose only necessary conditions for being an instance of the schema name on the left-hand side. For example, the axiom "Employee $\sqsubseteq$ Person" declares that every employee is a person, but does not give a sufficient condition for being an employee.

A schema may contain *cycles* through inclusion axioms (see Nebel 1991 for a formal definition). So one may state that the bosses of an employee are themselves employees, writing "Employee $\sqsubseteq$ $\forall$boss.Employee." In general, existing systems do not allow for terminological cycles, which is a serious restriction, since cycles are ubiquitous in domain models.

There are two questions related to cycles: the first is to fix the semantics and the second, based on this, to come up with a proper inference procedure. As to the semantics, we argue that axioms in the

Construct Name	Syntax	Semantics
top	$\top$	$\Delta^{\mathcal{I}}$
singleton set	$\{a\}$	$\{a^{\mathcal{I}}\}$
intersection	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
union	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
universal quantification	$\forall R.C$	$\{d_1 \mid \forall d_2 : (d_1, d_2) \in R^{\mathcal{I}} \rightarrow d_2 \in C^{\mathcal{I}}\}$
existential quantification	$\exists R.C$	$\{d_1 \mid \exists d_2 : (d_1, d_2) \in R^{\mathcal{I}} \wedge d_2 \in C^{\mathcal{I}}\}$
existential agreement	$\exists Q \doteq R$	$\{d_1 \mid \exists d_2 . (d_1, d_2) \in Q^{\mathcal{I}} \wedge (d_1, d_2) \in R^{\mathcal{I}}\}$
number restrictions	$(\geq n\, R)$	$\{d_1 \mid \sharp\{d_2 \mid (d_1, d_2) \in R^{\mathcal{I}}\} \geq n\}$
	$(\leq n\, R)$	$\{d_1 \mid \sharp\{d_2 \mid (d_1, d_2) \in R^{\mathcal{I}}\} \leq n\}$

Table 1: Syntax and semantics of concept forming constructs.

Construct Name	Syntax	Semantics
inverse role	P^{-1}	$\{(d_1, d_2) \mid (d_2, d_1) \in P^{\mathcal{I}}\}$
role restriction	$(R{:}C)$	$\{(d_1, d_2) \mid (d_1, d_2) \in R^{\mathcal{I}} \wedge d_2 \in C^{\mathcal{I}}\}$
role chain	$Q \circ R$	$\{(d_1, d_3) \mid \exists d_2 . (d_1, d_2) \in Q^{\mathcal{I}} \wedge (d_2, d_3) \in R^{\mathcal{I}}\}$
self	ϵ	$\{(d_1, d_1) \mid d_1 \in \Delta^{\mathcal{I}}\}$

Table 2: Syntax and semantics of role forming constructs.

schema have the role of narrowing down the models we consider possible. Therefore, they should be interpreted under descriptive semantics, *i.e.*, like in first order logic: an interpretation $\mathcal{I}$ satisfies an axiom $A \sqsubseteq D$ if $A^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, and it satisfies $P \sqsubseteq A_1 \times A_2$ if $P^{\mathcal{I}} \subseteq A_1^{\mathcal{I}} \times A_2^{\mathcal{I}}$. The interpretation $\mathcal{I}$ is a model of the schema § if it satisfies all axioms in §. The problem of inferences will be dealt with in the next section.

The *view part* contains *view definitions* of the form

$$V \doteq C,$$

where V is a view name and C is a concept in the view language. Views provide abstractions by defining new classes of objects in terms of the concept and role names introduced in the schema. We refer to "$V \doteq C$" as the *definition* of V. The distinction between schema and view names is crucial for our architecture. It ensures the separation between schema and views.

A *view taxonomy* $\mathcal{V}$ is a finite set of view definitions such that (*i*) for each view name there is at most one definition, and (*ii*) each view name occurring on the right hand side of a definition has a definition in $\mathcal{V}$.

Differently from schema axioms, view definitions give necessary *and* sufficient conditions. As an example of a view, one can describe the bosses of the employee Bill as the instances of "BillsBosses $\doteq$ $\exists$boss-of.{BILL}."

Whether or not to allow cycles in view definitions is a delicate design decision. Differently from the schema, the role of cycles in the view part is to state recursive definitions. For example, if we want to describe the group of individuals that are above Bill in the hierarchy of bosses we can use the definition "BillsSuperBosses $\doteq$ BillsBosses $\sqcup$ $\exists$boss-of.BillsSuperBosses." But note that this does not yield a definition if we assume descriptive semantics because for a fixed interpretation of BILL and of the role boss-of there may be several ways to interpret BillsSuperBosses in such a way that the above equality holds. In this example, we only obtain the intended meaning if we assume least fixpoint semantics. This observation holds more generally: if cycles are intended to uniquely define concepts then descriptive semantics is not suitable. However, least or greatest fixpoint semantics or, more generally, a semantics based on the μ-calculus yield unique definitions (see Schild 1994). Unfortunately, algorithms for subsumption of views under such semantics are known only for fragments of the concept language defined in Tables 1 and 2.

In this paper, we only deal with acyclic view taxonomies. In this case, the semantics of view definitions is straightforward. An interpretation $\mathcal{I}$ satisfies the definition $V \doteq C$ if $V^{\mathcal{I}} = C^{\mathcal{I}}$, and it is a model for a view taxonomy $\mathcal{V}$ if $\mathcal{I}$ satisfies all definitions in $\mathcal{V}$.

A state of affairs in the world is described by *assertions* of the form

$$C(a), \qquad R(a, b),$$

where C and R are concept and role descriptions in the view language. Assertions of the form $A(a)$ or $P(a, b)$, where A and P are names in the schema, resemble basic facts in a database. Assertions involving complex concepts are comparable to view updates.

A *world description* $\mathcal{W}$ is a finite set of assertions. The semantics is as usual: an interpretation $\mathcal{I}$ satisfies $C(a)$ if $a^{\mathcal{I}} \in A^{\mathcal{I}}$ and it satisfies $R(a, b)$ if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$; it is a model of $\mathcal{W}$ if it satisfies every assertion in $\mathcal{W}$.

Summarizing, a knowledge base is a triple $\Sigma = \langle \S, \mathcal{V}, \mathcal{W} \rangle$, where $\S$ is a schema, $\mathcal{V}$ a view taxonomy, and $\mathcal{W}$ a world description. An interpretation $\mathcal{I}$ is a model of a knowledge base if it is a model of all three components.

2.3 Reasoning Services

For each component, there is a prototypical reasoning service to which the other services can be reduced.

Schema Validation: Given a schema $\S$, check whether there exists a model of $\S$ that interprets every schema name as a nonempty set.

View Subsumption: Given a schema $\S$, a view taxonomy $\mathcal{V}$, and view names V_1 and V_2, check whether $V_1^{\mathcal{I}} \subseteq V_2^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\S$ and $\mathcal{V}$;

Instance Checking: Given a knowledge base Σ, an individual a, and a view name V, check whether $a^{\mathcal{I}} \in V^{\mathcal{I}}$ holds in every model $\mathcal{I}$ of Σ.

Schema validation supports the knowledge engineer by checking whether the skeleton of his domain model is consistent. Instance checking is the basic operation in querying a knowledge base. View subsumption helps in organizing and optimizing queries (see *e.g.* Buchheit *et al.* 1994). Note that the schema $\S$ has to be taken into account in all three services and that the view taxonomy $\mathcal{V}$ is relevant not only for view subsumption, but also for instance checking. In systems that forbid cycles, one can get rid of $\S$ and $\mathcal{V}$ by expanding definitions. This is not possible when $\S$ and $\mathcal{V}$ are cyclic.

2.4 Complexity Measures

The separation of the core of a TKRS into three components allows us to introduce refined complexity measures for analyzing the difficulty of inferences.

The complexity of a problem is generally measured with respect to the size of the whole input. However, with regard to our setting, three different pieces of input are given, namely the schema, the view taxonomy, and the world description. For this reason, different kinds of complexity measures may be defined, similarly to what has been suggested in [Vardi,1982] for queries over relational databases. We consider the following measures (where $|X|$ denotes the size of X):

Schema Complexity: the complexity as a function of $|\S|$;

View Complexity: the complexity as a function of $|\mathcal{V}|$;

World Description Complexity: the complexity as a function of $|\mathcal{W}|$;

Combined Complexity: the complexity as a function of $|\S| + |\mathcal{V}| + |\mathcal{W}|$.

Combined complexity takes into account the whole input. The other three instead consider only a part of the input, so they are meaningful only when it is reasonable to suppose that the size of the other parts is negligible. For instance, it is sensible to analyze the schema complexity of view subsumption because usually the schema is much bigger than the two views which are compared. Similarly, one might be interested in the world description complexity of instance checking whenever one can expect $\mathcal{W}$ to be much larger than the schema and the view part.

It is worth noticing that for every problem combined complexity, taking into account the whole input, is at least as high as the other three. For example, if the complexity of a problem is $O(|\S| \cdot |\mathcal{V}| \cdot |\mathcal{W}|)$, its combined complexity is cubic, whereas the other ones are linear. Similarly, if the complexity of a given problem is $O(|\S|^{|\mathcal{V}|})$, both its combined complexity and its view complexity are exponential, its schema complexity is polynomial, and its world description complexity is constant.

In this paper, we use combined complexity to compare the complexity of reasoning in our architecture with the traditional one. Moreover, we use schema complexity to show how the presence of a large schema affects the complexity of the reasoning services previously defined. View and world description complexity have been investigated (under different names) in [Nebel,1990; Baader,1990] and [Schaerf,1993; Donini *et al.*,1994], respectively.

3 The Case Studies

We studied some illustrative examples that show the advantages of the architecture we propose. We extended three systems by a simple cyclic schema language and analyzed their computational properties.

As argued before, a schema language should at least be expressive enough for declaring subconcept relationships, restricting the range of roles, and specifying roles to be necessary (at least one value) or single valued (at most one value). These requirements are met by the language $\mathcal{SL}$, which was introduced in [Buchheit *et al.*,1994] and that is defined by the following syntax rule:

$$C, D \longrightarrow A \mid \forall P.A \mid (\geq 1\, P) \mid (\leq 1\, P).$$

Obviously, it is impossible to express in $\mathcal{SL}$ that a concept is empty. Therefore, schema validation in $\mathcal{SL}$ is trivial. Also, subsumption of concept names is polynomially decidable.

We proved that inferences become harder for extensions of $\mathcal{SL}$. If we add inverse roles, schema validation remains trivial, but subsumption of schema names becomes NP-hard. If we add any construct by which one can express the empty concept—like disjointness axioms—schema validation becomes NP-hard. However, in our opinion this does not mean that extensions of $\mathcal{SL}$ are not feasible. For some extensions, there are natural restrictions on the form of schemas that decrease the complexity. Also, it is not clear whether realistic schemas will contain structures that require complex computations.

In all the three cases studied, the schema language is $\mathcal{SL}$. For the view language, we propose three different languages derived from three actual systems described in the literature, namely the deductive object-oriented database system CONCEPTBASE [Jarke,1992], and the terminological systems KRIS [Baader and Hollunder,1991] and CLASSIC [Borgida *et al.*,1989]. We investigated the computational properties of the reasoning services with respect to $\mathcal{SL}$-schemas. We aimed at showing two results: (*i*) reasoning w.r.t. schema complexity is always tractable, (*ii*) combined complexity is not increased by the presence of terminological cycles in the schema.

In all three cases, we assume that view names are allowed in membership assertions and that the view taxonomy is acyclic. In this setting, every view name can be substituted with its definition. For this reason, from this point on, we suppose that view concepts are completely expanded. Therefore, when evaluating the complexity, we replace the size of the view part by the size of the concept representing the view.

We have found the following results for the three systems in which $\mathcal{SL}$ is the schema language and the concept language the abstraction of the query language of CONCEPTBASE introduced in [Buchheit *et al.*,1994], or the language offered by KRIS or CLASSIC, respectively.

CONCEPTBASE: instance checking is in PTIME w.r.t. combined complexity (view subsumption has been proved in PTIME in [Buchheit *et al.*,1994]).

KRIS: view subsumption and instance checking are PSPACE-complete problems w.r.t. combined complexity and PTIME problems w.r.t. schema complexity.

CLASSIC: view subsumption and instance checking are problems in PTIME w.r.t. combined complexity.

We conclude that adding (possibly cyclic) schema information does not change the complexity of reasoning within the systems taken into account.

4 Conclusion

We have proposed to replace the traditional TBox in a terminological system by two components: a schema, where primitive concepts describing frame-like structures are introduced, and a view part that contains defined concepts. We feel that this architecture reflects adequately the way terminological systems are used in most applications.

We also think that this distinction can clarify the discussion about the semantics of cycles. Given the different functionalities of the schema and view part, we propose that cycles in the schema are interpreted with descriptive semantics while for cycles in the view part a definitional semantics should be adopted.

In three case studies we have shown that the revised architecture yields a better tradeoff between expressivity and the complexity of reasoning.

The schema language we have introduced might be sufficient in many cases. Sometimes, however, one might want to impose more integrity constraints on primitive concepts than those which can be expressed in it. We see two solutions to this problem: either enrich the language and have to pay by a more costly reasoning process, or treat such constraints in a passive way by only verifying them for the objects in the knowledge base. The second alternative can be given a logical semantics in terms of epistemic operators (see Donini *et al.* 1992).

References

[Baader and Hollunder, 1991] Franz Baader and Bernhard Hollunder. A terminological knowledge representation system with complete inference algorithm. In *Proc. PDK-91*, LNAI, pages 67–86, 1991.

[Baader, 1990] Franz Baader. Terminological cycles in KL-ONE-based knowledge representation languages. In *Proc. AAAI-90*, pages 621–626, 1990.

[Borgida *et al.*, 1989] Alexander Borgida, Ronald J. Brachman, Deborah L. McGuinness, and Lori Alperin Resnick. CLASSIC: A structural data model for objects. In *Proc. ACM SIGMOD*, pages 59–67, 1989.

[Buchheit *et al.*, 1994] Martin Buchheit, Manfred A. Jeusfeld, Werner Nutt, and Martin Staudt. Subsumption between queries to object-oriented databases. *Information Systems*, 19(1):33–54, 1994.

[Donini *et al.*, 1992] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, Werner Nutt, and Andrea Schaerf. Adding epistemic operators to concept languages. In *Proc. KR-92*, pages 342–353, 1992.

[Donini *et al.*, 1994] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. Deduction in concept languages: From subsumption to instance checking. *Journal of Logic and Computation*, 4(92–93):1–30, 1994.

[Jarke, 1992] M. Jarke. ConceptBase V3.1 User Manual. Aachener Informatik-Berichte 92-17, RWTH Aachen, 1992.

[Lenzerini and Schaerf, 1991] Maurizio Lenzerini and Andrea Schaerf. Concept languages as query languages. In *Proc. AAAI-91*, pages 471–476, 1991.

[Nebel, 1990] Bernhard Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43:235–249, 1990.

[Nebel, 1991] Bernhard Nebel. Terminological cycles: Semantics and computational properties. In John F. Sowa, editor, *Principles of Semantic Networks*, pages 331–361. Morgan Kaufmann, Los Altos, 1991.

[Schaerf, 1993] Andrea Schaerf. On the complexity of the instance checking problem in concept languages with existential quantification. *Journal of Intelligent Information Systems*, 2:265–278, 1993.

[Schild, 1994] Klaus Schild. Terminological cycles and the propositional μ-calculus. In *Proc. KR-94*, 1994.

[Vardi, 1982] M. Vardi. The complexity of relational query languages. In *Proc. STOC-82*, pages 137–146, 1982.

Using Natural Language for Database Design

Edith Buchholz * and **Antje Düsterhöft**

Department of Computer Science
University of Rostock, A.-Einstein-Str.21
18059 Rostock, Germany
Email: {buch,duest}@informatik.uni-rostock.de

Abstract.
This paper deals with a natural language dialogue tool for supporting the database design process. We want to illustrate how natural language (German) can be used for obtaining a skeleton design and for supporting the acquisition of semantics of the prospective database. The approach is based on the assumption that verbs form a central part in defining the meaning of sentences and imply semantic roles in the sentences which have to be filled by objects. We are using a moderated dialogue for drawing the designer's attention to these objects in order to extract comprehensive information about the domain.

1 Introduction

The quality of database design is a decisive factor for the efficiency of a database application. A database designer has to use a high level of abstraction for mapping his real-world application onto an entity relationship model. The designer has to learn the model and the constraints to use it.

Natural language can be exploited in order to overcome this bottleneck. From our point of view a user-friendly design system has to have two supporting tools: firstly, a tool which makes available an interface for obtaining a natural language description of an application and secondly, a tool for paraphrasing database schemes in a natural language way (see also [FloPR85]).

[ColGS83], [TseCY92], [TjoB93] are presenting various methods dealing with natural language as input for database design systems. These systems are based on natural language texts for the requirement specification in the data base design process. This paper illustrates how natural language in a dialogue tool can be used for gathering the knowledge of the designer and how it can be transfered into an extended entity-relationship model. The dialogue together with the knowledge base will be used for drawing to the designer's attention special facts resulting from the syntactic, the semantic and the pragmatic analyses. The tool makes suggestions for completing the design applying the knowledge base.

In the database design project RAD ([ThaA94]) we have implemented a rule-based dialogue design tool for getting a skeleton design on the basis of the extended entity-relationship model HERM [Tha91]. The designer describes the structure of an application in German. The specification and formalisation of semantic constraints is one of the most complex problems for the designer. Within natural language sentences the designer uses semantic constraints intuitively. For that reason, within the natural language design process we focus on extracting comprehensive semantic information about the domain from natural language utterances. The results of the dialogue are available in the internal DataDictionary for the other tools (grahical interface, integrity checker, strategy adviser,...) of the system. Within the RAD system the designer can use these results for various forms of representation, e.g. a graphical representation. The skeleton design with the semantic constraints is also the basis for further semantic checks, e.g. of key candidates, and will restrict the search areas in the checking process.

For the theoretical and pragmatic analyses of the language used within the design dialogue it was necessary to do this with a practical example. So we decided to choose the field of library - its tasks and processes. As a method of obtaining the linguistic corpus we carried out a number of interviews with librariens and library users. The extracted corpus was analysed statistically to obtain the frequency of word forms and the occurence of synonyms and homonyms. Starting from this domain we developed relations to other domains (see [BucD94]).

The dialogue tool will be implemented in PROLOG.

2 The structure of the dialogue tool

For the acquisition of designer knowledge we decided to choose a moderated dialogue tool. A moderated dialogue can be seen as a question-answer-tool. The tool asks for input or additional questions considering the acquisition of database design information. These questions are frames which will be updated in the dialogue process. The designer can formulate the answer in natural language

* This work is supported by DFG project TH 456/2-2.

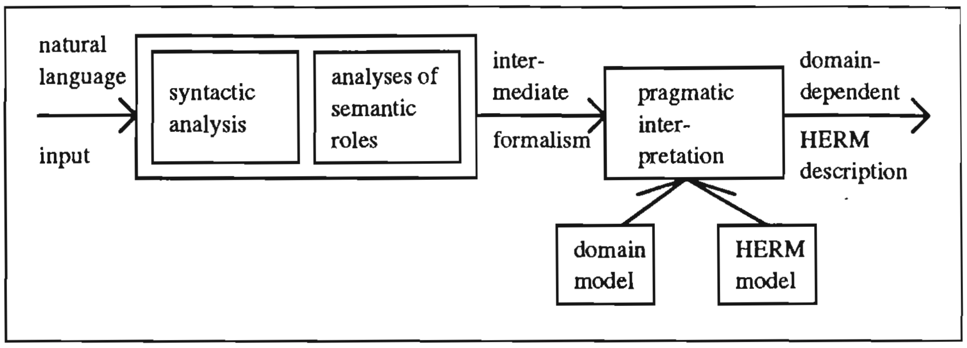

Fig. 1. Two-stage Dialogue interpretation tool

sentences. Each sentence will be analysed syntactically as well as semantically and then transformed into HERM stuctures.

Within the dialogue the results of the syntactic, semantic and pragmatic analyses will be used for controlling the dialogue. That means, if an incomplete designer input is received a question will be initiated. Inputs are incomplete if either semantic roles are not complete or the newly generated design model is incomplete. Semantic roles are filled within the semantic analysis. The pragmatics realizes the transformation of the natural language sentences into HERM structures.

2.1 Syntactic analysis

The syntactic analysis of the natural language input of the designer is based on a GPSG parser (Generalized Phrase Structure Grammar) [Gaz85]. GPSG belongs to the family of Unification Grammars. A basic feature is the introduction of ID/LP Rules (Immediate Dominance/ Linear Precedence). Immediate Dominance determines the immediate dominance of a root over its followers, Linear Precedence determines the order in which the follower, e.g. syntactic categories are to be processed.
The parser implemented in our tool uses the Earley algorithm [Ear70].

2.2 Semantic analysis

Interpreting the semantics of the designer input we are using the model of Bierwisch [Bie88] which inserts a semantic level between the syntax level and the conceptual level (HERM data model).

We assume that verbs form a central part in defining the meaning of sentences and the relationships between parts of sentences. Basically they describe actions, processes and states. We have tried to find a classification of verb semantics that can be applied to all verbs in the German language. Our aim was to keep the number of classes small and fairly general but large enough to identify their function in a sentence correctly. This classification (see also [BucD94]) is, at this stage, independent of the domain to be analysed (cf.Fig.2).

To identify the meaning of sentences we have used the model of semantic roles. Verbs of a special class imply the occurence of semantic roles. The units in a sentence or an utterance are seen to fulfil certain roles. Our role concept is mainly based on the hypothesis by Jackendoff [Jac83] and consists of the following roles which refer to the objects partaking in the action: Cause, Theme, Result/ Goal, Source, Locative, Temporal, Mode, Voice/Aspect. The following example illustrates the role concept.

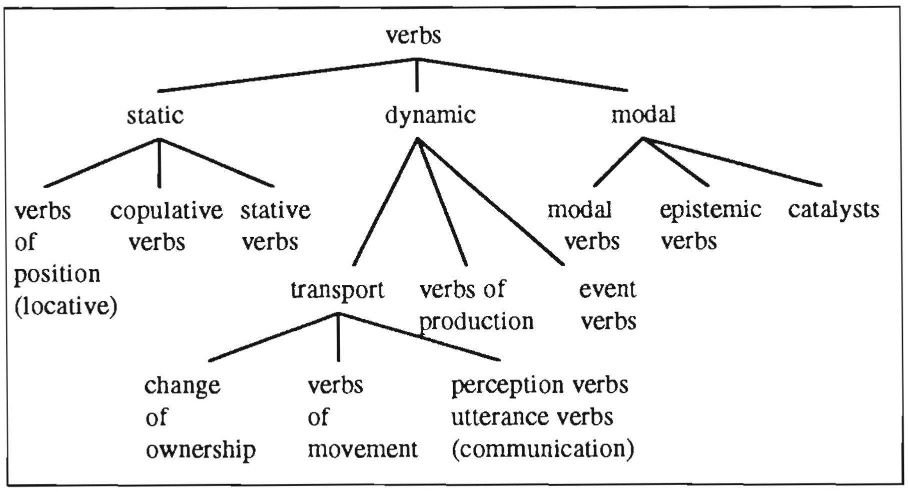

Fig.2. Verb classification

Example. 'The user borrows a book with a borrowing-slip'

```
results of the semantic analysis:
verb type:    change of ownership
subject:      the user
object:       a book
locative:     ?*
temporal:     ?*
mode:         with a borrowing-slip
```

(* an additional question will be initiated)

2.3 Pragmatic interpretation

2.3.1 Obtaining a skeleton design

The transformation of the structure of natural language sentences into EER model structures is a process which is based on heuristic assumptions, e.g., we assume that all nouns are entities. [TjoB93] illustrate a large number of such heuristics in an informal way. If we accept these heuristics then we can formalize them using contextfree and contextsensitive rules.

Example.
/* all nouns are transferred into entities */
```
N(X) → entity(NAME,X).
```

/* sentences with the main verb 'have' are transferred into an entity (the subject) and the according attribute (the object of the sentence) */

```
N(X),subject(X),V(haben),N(Y),object(Y)
  →entity(X),attre(X,Y).
```

Considering the results of the syntactic analysis of a natural language sentence we can describe these results using a tuple structure.

Example. The tuple structure of the sentence 'the user borrows a book with a borrowing-slip' is:

```
S(NP(DET(the),N(user)),
  VP(VP(V(borrows),NP(DET(a),N(book)),
                PP(PRAEP(with),
                   NP(DET(a),
                      N(borrowing-
                        slip)))))
```

The tuple can be seen as a language which can be described by a grammar, e.g. terminals are N, DET or VP. The HERM model can also be seen as a language if predicates are used to describe the elements of the model. *Now we can handle the transformation as a compiler process using an attribute grammar.* The heuristics are integrated into grammar rules as well as into semantic rules. A compiler for this purpose has been developed. The following example illustrates how the transformation is realized.

Example. Transforming the utterance 'at the library' into an entity named 'library' using a contextfree grammar formalism. (The small letters identify nonterminals, and the capital letters are terminals. '$x' is a variable. 'assert(X)' asserts 'X' to the model description.)

tuple structure:
```
S(PP(PRAEP(at),NP(DET(a),N(library))))
```

grammar rules:
```
start → S(phrase)
phrase → PP(pp_phrase)
pp_phrase → PRAEP($x),NP(np_phrase)
np_phrase → NP(det_phrase,n_phrase)
det_phrase → DET($x)
n_phrase → N($x) {assert(entity($x))}
```

The advantage of this approach is that we can define actions at the word category level as well as at the sentence phrase level. So, it is possible to define database design actions, e.g. when considering the occurence of a genitive nominal phrase connected with another nominal phrase in the sentence. The heuristics underlying is that a genitive nominal phrase has an attribute function concerning the corresponding nominal phrase.

We are using a dialogue in which the designer can formulate a description of an application in several sentences. For that reason we have to deal with the problem of inserting a new part of a design into an existing design. We have implemented a two-step approach. Firstly, a seperate design will be generated from the sentence of the user. Secondly, the design description will be updated inserting the new design part. Common heuristics are the basis of the updating process (cf. [Düs94]).

2.3.2 Extracting information on behaviour

In most cases a database will be used for complex processes. In order to be able to maintain the database we have to define transactions. (For the reasons of using transactions see [Tha94:114].) The behaviour of the database can help to make the system more efficient and faster and thus to save time and money.

Behaviour can best be gained from a knowledge base. One form of presenting the domain is by classification of the processes involved as a conceptual graph. The knowledge base will be used for gathering relevant processes of the application and is based on the results of the semantic analysis. Each application can be classified. Lending processes are identified by verbs of the class

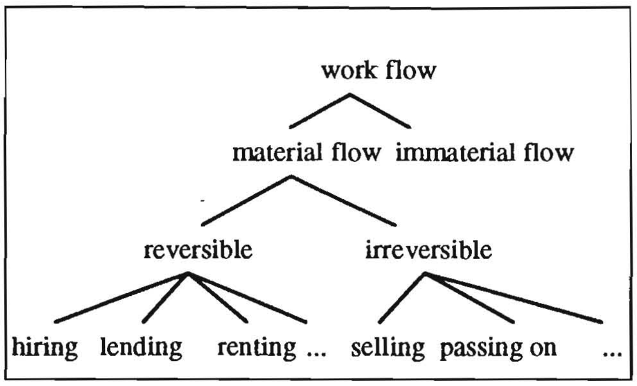

Fig. 3. Part of the process classification

'change of ownership'. The library processes or the 'rent a car' processes (cf. Fig. 3) belong to this group.

The lending process as a complex process can be further classified into a number of pre and post processes (cf. Fig. 4). These processes are included in the knowledge base. If a user input contains one of these processes a possible classification will be defined and an action within the dialogue will be initiated.The pre and post processes in Fig. 4 can be further subdivided into processes which are summarized in the above classification. Lending thus requires the processes of obtaining a user card, updating the user card if need be checking whether the book is held and available, filling in a borrowing-slip and signing it.

Example. The sentence 'the user borrows a book with borrowing-slip' implies the following general questions (borrowing has the synonym lending):

```
preprocesses:
1) Is the process 'obtaining' situated before
     'lending' ?
2) Is the process 'registration' situated
     before 'lending' ?
main processes:
3) Is the process 'document exists' situated
     before 'lending' ?
4) Is the process 'document valid' situated
     before 'lending' ?
     ...
postprocesses:
5) Is the process 'returning' situated after
     'lending' ?
```

The designer has to give correct answers.

3 Conclusions/ Future Topics

We have presented a dialogue tool consisting of a syntax analyser, a semantic role definer and a pragmatics interpreter. The dialogue tool gathers information on structure, semantics and behaviour of the prospective database. By means of transformation rules this information is mapped onto the HERM model.

The advantage of the dialogue tool is that the designer can describe the requirements of the database system in a natural language (German) and thus can specify the knowledge of a domain in a natural way. This knowledge is then employed for gathering database constructs such as entities, attributes, cardinalities, constraints, etc.

The efficiency of the database greatly depends on the exact interpretation and transformation of the natural language input analysis. The accuracy, on the other hand, depends on the size and complexity of the grammar used and the scope of the lexicon.

Work in future has to concentrate on extending the grammar to comprise all types of sentences and other hitherto excluded parts of grammar and on ways of steadily increasing the lexicon. For reasons of integrity we cannot leave updating of the lexicon to the chance designer who may have no linguistic training. Much work will have to go into completing and maintaining the linguistic background before it can finally be used for any type of systems design.

A second future topic is the application of the linguistic knowledge for acquiring further semantic information of the prospective database, e.g. acquiring key attributes or functional dependencies.

Acknowledgements

We are grateful to Bernhard Thalheim for his guidance and support of our work and for his helpful criticism and suggestions.

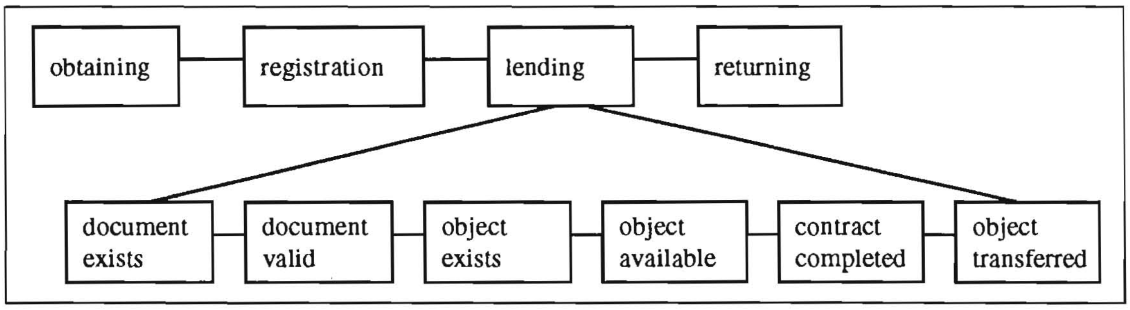

Fig. 4. Part of the knowledge base: pre, main and post processes of the act/borrowing/ lending

References

[Bie88] Bierwisch, M., Motsch, W., Zimmermann, I. : Syntax, Semantik und Lexikon. Berlin,Akademie Verlag, 1988

[BucD94] Buchholz, E., Düsterhöft, A.: The linguistic backbone of a natural language interface for database design. In: LLC ?/94, Oxford University Press

[ColGS83] Colombetti, M.; Guida, G.; Somalvico, M.: NLDA: A Natural Language Reasoning System for the Analysis of Data Base Requirements. In: Ceri, S. (ed.): Methodology and Tools for Data Base Design. North-Holland, 1983

[Düs94] Düsterhöft, A.: Zur Vorgehensweise bei der pragmatischen Inter pretation natürlichsprachiger Äußerungen Im Datenbankentwurf. Preprint 4/94, Fachbereich Informatik, Universität Rostock

[Ear70] Earley, J.: An efficient context-free parsing algorithm. Comm. ACM13:2, S.94-102

[Eic84] Eick, Ch.F.: From Natural Language Requirements to Good Data Base Definitions - A Data Base Design Methodology. In: Proc. of the International Conference on Data Engineering, pp.324-331, Los Angeles, USA, 24.-27.4.1984

[FloPR85] Flores, B.; Proix, C.; Rolland, C.: An Intelligent Tool for Information Design. Proc. of the Fourth Scandinavian Research Seminar of Information Modeling and Data Base Management. Ellivuori, Finnland, 1985

[Gaz85] Gazdar, G.; Klein, E.; Pullum, G.; Sag, I.: Generalized Phrase Structure Grammar. Havard University Press Cambridge, Mass. 1985

[GolS91] Goldstein, R.C.; Storey, V.C.: Commonsense Reasoning in Database Design. Proc. of the 10th International Conference on Entity-Relationship Approach, San Mateo, California, USA, 23.-25.October 1991, pp.77-92

[Jac83] Jackendoff, R.: Semantics and cognition. MIT Press, Cambridge Mass., 1983

[Tha91] Thalheim, B.: Intelligent Database Design Using an Extended Entity-Relationship Model. Berichte des Fachbereiches Informatik 02-1991, Universität Rostock.

[Tha94] Thalheim B.: Fundamentals of Entity-Relationship Modeling. Springer Verlag 1994, Forthcoming

[ThaA94] Thalheim, B., Albrecht, M., Altus, M., Buchholz, E., Düsterhöft, A., Schewe, K.-D.: Die Intelligente Tool Box zum Datenbank entwurf RAD. Workshop "Benutzerschnitstellen",17.-19.März1994, Kassel

[TjoB93] Tjoa, A.M., Berger, L.: Transformation of Requirements Specifications Expressed in Natural Language into an EER Model. Proceeding of the 12thInternational Conference on ER-Approach, Airlington, Texas USA,Dec. 15-17th, 1993

What's in a Federation?
Extending Data Dictionaries with Knowledge Representation Techniques

Wolfgang Benn
Chemnitz University of Technology • Management of Data
P.O. Box 964 • D-09009 Chemnitz
benn@informatik.tu-chemnitz.de

1. Introduction

Databases and knowledge representation languages have a rather different view upon data: knowledge representation languages describe a universe of discourse in a taxonomy and allow a user to ask epistemic questions against the relationships between concepts and roles. However, no data structures, data locations, nor any information about the existence or availability of data can be found in a taxonomy -- even not if it includes an assertion that describes a particular data item.

Relational databases provide users with schemata. Schemata describe in detail the data structures of sets of persistent data items. Data dictionaries, included in these systems, tell about data existence and its availability. Anyway, these tools do not provide the entity view, relationships between entities are merely implicit, and no question about the universe of discourse that is behind a schema will get an answer.

Object-oriented databases provide users with class hierarchies as schemata. They support the entity view -- is-a as well as part-of relationships are explicit. Nevertheless, an information about the universe of discourse is not given as well.

In a federation of systems -- databases and applications, for instance -- the situation gets worse. Databases may be heterogeneous in their modeling technique: some will follow the object-oriented the majority certainly follows the relational paradigm. How does a user get to know what data is available in a federation, if he wants to build a new application? How does that user get to know how he may access a particular data item? How does he know that the selected data item is semantically correct concerning the context of his application?

If he can access a federated data dictionary, it will provide him with the technical information about the data in a common data model -- similar to the global conceptual schema of a distributed database. If such a tool does not exist, the user must read all available schemata from all available federation components (i.e., he must know about all languages, data models, and dialects that the local components of the federation individually use).

In the remainder of this paper we will briefly introduce a module that coordinates a federation of systems and that hosts a central data dictionary. It is the module, which we will extend to provide users with an entity view upon the information available in a federation.
We introduce the logical architecture of a prototypical implementation of this module in section 2 and describe some extensions that we made in section 3. In section 4 we specify some ideas of the mentioned extension, conclude in section 5 and give some literature in section 6.

2. The Federal System Manager

The Federal System Manager (FSM) is a module that coordinates a federation of autonomous systems. These systems can be applications or services like databases, which may link to the FSM to form a federation for some particular tasks. Afterwards they can leave the federation and run again as autonomous systems. This idea is rather similar to the concept of multi-agent systems.

The FSM performs a minimum of three tasks: The first one is to run a protocol that enables the linkage process and guarantees a negotiation of autonomy aspects to the components, if these want to join or leave the federation. Second, the FSM must provide a uniform view upon all information that is available to applications of the federation through a so-called Common Data Model (CDM). Third, it must support an exchange of information, i.e., data types and data itself, between members of the federation. We will detail these tasks and concentrate on the second one.

Comparing an FSM with the Common Object Request Broker Architecture (CORBA) [1] the FSM is an object broker that looks at databases as service providing objects and applications as clients that request these services. Commonly known services from database components are storage, retrieval, update, etc.

Moreover, the FSM is an object itself! It provides services like data and type exchange. It contains a Federal Data Dictionary (FDD) that allows a user to retrieve the information contents of the actual federation under several aspects. It is our aim to extend this Federal Data Dictionary with knowledge representation techniques to better support users in their retrieval than before.

2.1. The FSM Prototype

The currently implemented FSM prototype has its roots in an ESPRIT project, finished in 1991 [2,3,4,5,6]. The prototype mainly follows the reference architecture for interoperable systems given in [7] and includes a repository according to the Information Resource Dictionary Standard IRDS [8].

This standard defines a four-layer architecture with (top down)

- a meta-meta layer that describes the model of the meta layer descriptions -- which is in our case the Common Data Model of the FSM, a frame work that basis on the Abstract Data Type (ADT) idea --,
- a meta layer where we find the description of schemata -- which is in our case a description of the federation components data models --,
- a schema layer where the data descriptions are located -- which is in our case the data types that are defined in schemata of databases or in type declarations of applications --, and
- an application data layer where we finally find the application data itself.

The Meta-Meta Layer

To enable the description of schema descriptions we implemented a common data model.

In the literature we found many different approaches to implement a CDM -- the approach most often used, however, was the object-oriented. Thus, we asked ourselves, what is the kernel idea of the object-oriented paradigm that makes it suitable for a CDM. We found out that it probably is the idea of Abstract Data Types.

Thus, we implemented a frame work, which is actually not a real data model but a tool box [2]. It allows a user to describe the structure and semantics of those elements, which he uses to describe a schema, similar to the ADT concept (see next paragraph).

The CDM that we implemented is very similar to the Interface Description Language (IDL) of the CORBA specification [1] -- because its purposes are rather similar. IDL is a language, which describes object services in an intermediate way and the CDM describes entities (application objects) in an intermediate way.

An IDL description is mapped into a real programming language and the object services are available for all programs written in this programming language. Application objects described in our CDM are (under certain conditions) transformable into all data models that are represented in the FSM.

The Meta Layer

An extension of the IRD standard was made for the meta layer. If the FSM supports an exchange of data between components, it must be able to transform data between the different individual data descriptions. These descriptions follow type or schema declarations, which use data model elements. Thus, our meta layer has to include a suitable sub-set of the component data model for each involved component. Moreover, it must include some rules that guide the transformation of entities between these data model sub-sets.

However, the description of a data model sub-set is somewhat more complex than the description of a schema. While a schema merely consists of data structures, a data model usually includes data types and data type semantics. The meta layer of our FSM includes both (the assignment of a set of operations to a data type that makes up the type's semantics in the data model of a component is currently under implementation).

To enable the exchange of data and schema information between components the system administrator of each federation component defines the relevant structural part of his component data model types with the CDM types and assigns some procedures that make up the semantics of these data types. He inserts the necessary data model knowledge into the meta layer using the meta-meta layer elements.

For instance, from an object oriented data model the administrator defines the structural parts of the concept CLASS and assigns at least one particular routine that performs inheritance similar to his individual data model.

This information is provided through an interface, which is the so-called Data-Model-Profile. It is an ASCII file with a particular syntax that is parsed. Then the information is kept in a knowledge base -- the FSM Meta Knowledge Base.

The Schema Layer

Databases, as components of a federation, use database schemata. Applications use data type definitions to declare their application types.

The FSM reads these schemata and declarations and interprets the used data types through the information of the meta layer. Application entities are transformed into entities of the CDM and then -- for storage purposes -- transformed into entities of a database data model.

The entity information in CDM-format is stored in the Federal Data Dictionary (FDD) for retrieval purposes.

The Application Layer

Finally the data that comes from applications is stored in databases that have joined the federation, that are represented through meta-information in the Meta Knowledge Base, and that are willing to perform the storage process after a negotiation of their autonomy rights.

Of course, the data is not stored as CDM-typed data but is typed according to the data model of the involved database system. The interpretation of binary data runs the same way as the transformation of type information: It goes from the data model of the application towards the CDM and from the CDM to the database data model, and vv.

3. Extensions of the FSM Prototype

Since 1991 the FSM prototype has been completed by some student's work.

The Federal Data Dictionary of the prototype contained information about data type declarations, the types of application entities, and the structure of these entities -- as well, access rights were included. It did not include any technical information about the availability of entities or schemata.

We extended the FDD and it now contains technical information about the federation components. The meta layer includes information about the technical system that hosts the application or the database system. The schema layer includes information about the technical availability of entities [9].

The lack of a docking mechanism and a protocol to negotiate autonomy was another problem of the original FSM prototype. It was a static system with two applications, a database system and the FSM with hard wired mechanisms to read data type declarations -- database schemata could not be read, nor was it possible to link another database system with the FSM.

Now we have implemented a link mechanism that generalizes the old one [10]. We now use a FSM-Bind module that binds a component -- either a database system or an application -- if it includes our FSM-Bind-Agent.

The FSM-Bind-Agent acts as a client to the FSM-Bind module, which is the server, and performs the link process between FSM and component. It runs an implemented protocol for start-up and shut-down situations and uses the Remote Procedure Call (RPC) technique.

After linkage the FSM-Bind-Agent passes control to a so-called FSM-Agent, which performs the information exchange and the retrieval of schema information via the Remote Data Access (RDA) protocol.

What is still missing, is a user friendly retrieval facility that completes the Federal Data Dictionary. We will describe our ideas in the next section.

3.1. Extensions of the FDD

Data dictionaries offer technical information to users -- and exactly this can be expected from our Federal Data Dictionary as it is currently implemented. If a user wants to build a new application he looks into the FDD and looks up some data structures that he wants to re-use. Then he includes the chosen data structures into his new schema (the FSM provides some commands to do so) and runs his application.

This user is unable to check whether his new schema violates the semantic integrity of the universe of discourse of the actual federation because he can not ask the FDD to present him semantic relations between entities.

We wish to provide such a user with an extended Federal Data Dictionary, which shows the contents of a federation from various levels of abstraction. If this extended data dictionary has a graphic interface the user will use a mouse to easily request the change of levels. Which are these levels?

Taxonomy Level

The highest level presented, should be a taxonomy upon the universe of discourse. It could be the union of all schemata (and may be data type declarations of applications) of local database components, which we previously transformed into the abstraction level of a concept language. This level would represent the data of a particular federation without any technical details. Here the user could look-up the real-world context of an entity and might ask questions about the relationships between entities. It is the level that KL-ONE like languages usually offer to users with their T-Box.

Concept Languages separate between the terminological (T-Box) and assertion knowledge (A-Box). The task, which we have to perform is to abstract the technical information from schemata and data type declarations to concepts of concept languages. In [11] we find a theoretical basis that allows us to express database schemata with concept languages.

Moreover, the authors show that classification is then available for entities of schemata -- and we found out that the implementation of a classificator is surprisingly supported through an algorithm, which we use within the FSM to detect data type intersections for types from different data models. This algorithm follows perfectly the above mentioned steps for a classification of concepts.

Anyway, if we make the is-a and part-of relations of entities from schemata explicit and suppress the technical information, then we can ask questions against a schema similar to the questions against a taxonomy.

The implementation of this level may use intermediate language representations that follow the idea of attributed trees. This model allows us to determine the degree of entity detail information, which we want to present, by cutting the tree at a certain level. The information above the cut is presented as concept. The rest is hidden until requests from other levels of our retrieval interface force it to become visible.

Apparently, we address some open questions if we want to extend a data dictionary with knowledge representation features:

How do we find a way to reconstruct the entity view from relational schemata with normalized relations? Any automatic evaluation of foreign keys -- which is the only data model construct that can be used to express sub-part relationships, set-inclusions, and entity-inclusions within the relational data model -- finally depends on the support of a human. A machine may solely hypothesize is-a relations between entities. Thus, our entity re-constructor can not be a completely automatic component. It has to include a dialogue component to keep in touch with a human expert, but it may be a component that is able to learn.

Schema Level

On a second level, the schema level, in a detailed view, the user should have access to the more technical details of entities and should see what attributes an entity make up, where the information resides within the federation, whether and when it is accessible for him.

This level is comparable with an extended Entity-Relationship level where we added attributes about data distribution and data availability to the usual representation of entities, attributes, and relationships.

We realize this view by an FDD retrieval, because our directory includes the structure information of entities in a neutral representation and the information about the availability of these entities.

Syntax Level

Finally, the user may get what he always got from databases: the pure schema information. If he asks for this, he will get an excerpt of a schema of one or more particular local components of the federation -- and he should decide himself whether he would like to receive this information in the format of a common data model or in the individual format of the involved local federation components.

4. First Steps toward the Taxonomy Level

Concerning the integration of abstract schema representations into one taxonomy we did some work in advance and evaluated an idea, published in [12]. It proposed the assignment of fuzzy values to relationships to determine the is-a of an entity.

We took this idea and tried to use probability values for the integration of different schemata into one -- to simulate the situation that comes up if we have to integrate abstracted schemata from components into one taxonomy. It was a first guess to cope with modeling heterogeneity.

The basic assumption behind our tests was, that the insert of knowledge into a taxonomy is an evolutionary process and that we ask "is B a A *or* a C" and not "how probably is B a A *and* a C".

We defined a value $C_T(E_i, E_j)$ for the correctness of a is-a relationship between two entities E_i and E_j in a taxonomy for the federation. Such a value is assumed to be assigned to each is-a relationship within that taxonomy. Similar to C_T we defined a $C_S(E_i, E_j)$ as a value for the correctness of a is-a relationship in a local schema.

Next we said that $S_T(E_n)$ and $S_S(E_n)$ are the sets of all super-concepts of a concept in the taxonomy and an entity in a local schema.

Finally, we defined two functions, which were necessary to calculate the probability values during the integration process.

The first function was called INIT and initialized an initial taxonomy with the value 1 for all is-a relationships: $C_T(E_i, E_j) := 1$.

The second function included a case statement and was called CALC. It calculated the initialized values according to the new schema. The first case, C_1, was used if a relationship was found in a schema -- it corresponds with the INIT function for the taxonomy -- and set $C_S(E_i, E_j) := 1$. We assume that the designer of the schema did a good and correct work.

The second case, C_2, was used, if we find a relationship within the schema but not within the taxonomy. We insert the relationship into the taxonomy and give it the value $C_T(E_i,E_j) := C_S(E_iE_j) \div \text{card}(S_T(E_i) \approx S_S(E_i))$.

This approach seems to be correct because we can not guarantee that the taxonomy was correctly initialized with relationships. Moreover, an insertion of a new relationship affects the probability value of another one because there must be a reason why a particular application domain needs this new relationship. It may be, that the already existing relationships do not have the importance, which we have expected.

Finally there is the case C_3. In this case we see a relationship within the taxonomy but miss it in a schema. We interpret that relationship as "possible but unnecessary" within this application domain and "insert" it into the schema with $C_S(E_i,E_j) := C_T(E_i,E_j) \div \text{card}(S_T(E_i))$.

Then we made three assumptions:
a) The increase of probability of one particular relationship is given by its existence in schemata and causes a decrease of probability for those relationships, which are often missed.
b) The results of calculations about the overall probability for a particular relationship is included into the taxonomy.
c) Results are calculated through the geometrical mean of the two probability values from the taxonomy and from a schema.

With these assumptions and formulas we tested the integration of six schemata into a taxonomy, which was initialized with one relationship "B is-a A". Four of these schemata included the relationship "B is-a A" (we call them the A-type schemata). Two included "B is-a C" and not "B is-a A" (we call these the C-type schemata).

In a first test, we inserted a C-type schema first and afterwards both relationships had the same value (0.71) in the taxonomy. A four-times insert of the A-type schemata brought the value of the "B is-a A" relationship up to 0.98 and the value of "B is-a C" fell down to 0.18 -- similar to the predicate "insignificant" or "incorrect". A final insert of a C-type schema, however, gave a new balance to both values, which was 0.69 for the "B is-a A" and 0.42 for the "B is-a C" relationship.

A second test gave surprising results: We inserted the two C-type schemata and then four times the A-type schemata. This gave a high value to the "B is-a C" relationship first -- the balance was 0.5 for "B is-a A" and 0.84 for "B is-a C" -- and a final value of 0.96 for "B is-a A" and 0.37 for "B is-a C".

While the first test showed that the late insert of an apparently insignificant relationship makes the value system unstable, the second test showed that an early insert of the two C-type schemata prevents the alternative relationship to fall down to an "insignificant" valuation.

Anyway, both value calculations were highly sequence dependent, and we suspected the second assumption as the reason for it. Thus we tried again without this assumption. We inserted into C_3 a variable: $V(E_i)$ counts the number of schemata without a particular relationship and the calculation C_3 changed to

$$C_S(E_i,E_j) := 1 \div (V(E_i) + 1).$$

This does not change much and we were stuck to the question: Is the insert of knowledge really an evolutionary process or is it correct to calculate probability values from the arithmetic mean of all values from schemata?

5. Conclusion

The proposed extended data dictionary gives a twofold benefit. At first, a user who wants to build a new schema for an application in a system federation can check which entities already exist, which of them he can re-use within his application, and which one he has to add or modify.
Second, an administrator can test the correctness of an existing schema against the universe of discourse. He can check the completeness of relations between entities by looking-up the taxonomy, where he would find the collection of all relationships between entities -- and eventually a probability value of the necessity or reliability of an individual relationship.

6. Literature

[1] *The Common Object Request Broker: Architecture and Specification*, OMG Document Number 91.12.1, Revision 1.1, Draft

[2] W. Benn, G. Junkermann, H. Kalweit, Ch. Kortenbreer, G. Schlageter, X. Wu: *The Conceptual Ob-*

ject Manager Document, University of Hagen, Computer Science Report N° 99, 1990

[3] W. Benn, Ch. Kortenbreer, X. Wu: *Towards Interoperability: Vertical Integration of Languages with a KBMS*, GI-Fachtagung "Datenbanksysteme in Büro, Technik und Wissenschaft" (BTW 91), Springer-Verlag, 1991

[4] W. Benn: *KBMS Support for Multiple Paradigm Applications*, in [16]

[5] W. Benn: *KBMS Support for Conceptual Modeling in AI*, 3rd International Conference on Tools for Artificial Intelligence, 1991

[6] W. Benn, Ch. Kortenbreer, G. Schlageter, X. Wu: *On Interoperability for KBMS Applications - The Horizontal Integration Task* -, 8 th Intl. Conference on Data Engineering, Phoenix, AZ, 1992

[7] A.P. Sheth, J.A. Larson: *Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases*, ACM Computing Surveys (1990) 3

[8] DIN 66 313, *Rahmenangaben für Systeme zur Verwaltung von Informationsrecourcenverzeichnissen*, DIN Deutsches Institut für Normung e.V., Berlin, 1992 (same as ISO/IEC 10 027)

[9] J. Hunstock: *Erweiterung einer Wissensbasis zur Realisierung von universellem Polymorphismus in föderativen Systemen um technische Informationen autonomer Systemkomponenten (Extending the Meta-Knowledge Base of the FSM by technical information)*, thesis for diploma, Chemnitz University of Technology, 1993

[10] M. Schöne, S. Herold: *Konzeption und Implementierung eines Protokolls und zugehöriger Systemkomponenten zur Integration von Datenbanksystemen in einer Föderation (Design and implementation of a protocol for the integration of database components into a federation)*, thesis for diploma, Chemnitz University of Technology, 1994

[11] S. Bergamaschi, C. Sartori: *On taxonomic reasoning in conceptual design*, ACM TODS (1992) 3

[12] P. Fankhauser, M. Kracker, E. Neuhold: *Semantic vs. Structural Resemblance of Classes*, ACM SIGMOD Record 20 (1991) 4

Do we need the closed-world assumption in knowledge representation?

Ullrich Hustadt*
Max-Planck-Institut für Informatik
Im Stadtwald, D-66123 Saarbrücken
e-mail hustadt@mpi-sb.mpg.de

1 Introduction

Database systems and knowledge representation systems represent and reason about some aspect of the real world. In both it is common to separate the two functions of *representation*, i.e. describing the conceptual scheme and the actual data, and *computation*, i.e. answering of queries and manipulation of data.

The database management system of a database system provides a data definition language to describe the conceptual scheme. The data definition language is used to describe the database in terms of a data model. Operations on the database require a specialized language, called a data manipulation language or query language. One of the most important data models is the *relational model* which describes the world in terms of atomic values and relations on the set of all atomic values. Data manipulation languages of the relational model comprise the relational algebra, and the domain and tuple relational calculi. The *object-oriented model* supports a more elaborated description of the world by allowing complex objects, i.e. objects constructed using record formation and set formation, classes, i.e. abstract data types describing methods, which are operations to be performed on the objects, and class hierarchies.

The data manipulation languages of these data models are based on the following assumptions.

The closed-world assumption
which says that all information that is not true in the database is considered as false.

The unique-name assumption
which says that two distinct constants (either atomic values or objects) necessarily designate two different objects in the universe.

The domain-closure assumption
which says that there are no other objects in the universe than those designated by constants of the database.

These assumptions are important to understand the way computations are performed in databases.

Knowledge representation formalisms are aimed to represent general conceptual information and are typically used in the construction of the knowledge base of a reasoning agent. A knowledge base can be thought of as representing the beliefs of such an agent. One of the most prominent knowledge representation formalisms is KL-ONE [Brachman and Schmolze,1985] which has been used in the construction of natural language processing systems.

The knowledge representation language of KL-ONE and all it's derivates can be considered as a subset of first-order logic with equality. With respect to describing structural properties of objects and conceptual schemes they are more expressive than the data definition languages corresponding to the relational or object-oriented model.

In the late eighties inference in KL-ONE was shown to be undecidable [Schmidt-Schauss,1989]. Since then the emphasis in research has been on developing and investigating systems that are computationally well behaved, i.e. are tractable or at least decidable [Brachman *et al.*,1991; Donini *et al.*,1991; Buchheit *et al.*,1993]. As a result many commonly used knowledge representation languages have restricted expressiveness and are in their current form no longer suitable for natural language applications. They are still more expressive than data definition languages, but the question can be risen whether there is an application needing this additional expressive power.

Nevertheless, data manipulation languages and query languages of knowledge representation formalisms differ considerably in their underlying assumptions.

The open-world assumption
which says that there can be true facts that are not contained in the knowledge base.

The unique-name assumption
which says that two distinct constants (either atomic values or objects) necessarily designate two different objects in the universe.

The open-domain assumption
which says that there can be more objects in the universe than those designated by constants in the knowledge base unless a constraint in the knowledge base prevents this.

That means, that even if the data definition language and the data manipulation language of a database management system and a knowledge base management system would coincide, the results of data manipulations would differ.

***Acknowledgments:** This work has been supported by the German Ministry for Research and Technology (BMFT) under grant ITS 9102 (Project Logo). Responsibility for the contents lies with the author.

In the next section I will give some examples that show the usefulness of closed-world inferences in natural language processing. Thus knowledge representation languages sticking to the open-world assumption seem to be insufficient for natural language processing.

2 Query answering in Natural Language Processing

In cooperation with the PRACMA Project[1] (Department of Computer Science, University of Saarbrücken) we have been developing a suitably extended knowledge representation system, called MOTEL [Hustadt and Nonnengart,1993], which is intended to be a module of the PRACMA system. The PRACMA Project [Jameson *et al.*,1994] is concerned with the modeling of noncooperative information-providing dialogues. An example from PRACMA's domain is the dialogue between a person $\mathcal{S}$ trying to sell her used car to a potential buyer $\mathcal{B}$. Naturally, the goals of $\mathcal{S}$ conflict in part with those of $\mathcal{B}$.

In the final implementation, the natural language analysis module of the PRACMA system will use the semantic representation language $\mathcal{NLL}$ [Laubsch and Nerbonne,1991] to represent the German-language input strings. The resulting $\mathcal{NLL}$ expressions will be stored in the pragmatic dialogue memory. Various modules will process the content of the dialogue memory, the most important one for us is the *comment and question handler.* The result of this module is transfered to the natural language generator which is responsible for verbalizing $\mathcal{NLL}$ expressions.

$\mathcal{NLL}$ contains a first-order logic core with anadic predicates, generalized quantifiers, plural reference expressions, and λ-abstraction. To fit the purposes of PRACMA the language has been extended by modal operators.

Suppose the knowledge base of the car seller S contains declarations defining that vehicles are either cars or trucks, `veh1` is a truck, and `veh2` is a vehicle. This can be represented in $\mathcal{NLL}$ in the following way.

```
(forall ?x vehicle(inst: ?x) iff
          (car(inst: ?x) or
           truck(inst: ?x))                (1)
truck(inst: veh1)                          (2)
vehicle(inst: veh2)                        (3)
```

Here `veh1` and `veh2` are constants, `vehicle`, `car`, and `truck` are predicate symbols. In $\mathcal{NLL}$ arguments of predicates are identified via keywords, e.g. `inst`, rather than positions in argument vectors. Any identifier preceded by a question mark, e.g. `?x`, is a variable. In addition we have used the boolean operators `iff` (equivalence) and `or` (disjunction), and the universal quantifier `forall` in declaration (1).

Now a question of the buyer concerning which objects are either cars or trucks is represented in the following way.

```
(?lambda ?x car(inst: ?x) or
          truck(inst: ?x))                 (4)
```

An expression of the (`?lambda ?x` P) denotes the set of all `?x` satisfying P. The answer we have to infer from the knowledge base is that `veh1` and `veh2` both belong to this set.

Obviously, this answer cannot be computed by the comment and question handler without taking declaration (1) into account. For instance, it is not possible to find the correct answer to (4) by computing the answer sets for `(?lambda ?x car(inst: ?x))` and `(?lambda ?x truck(inst: ?x))` and to return the union of the resulting sets as an answer.

A question of the buyer concerning which objects do not belong to the set of trucks is translated into the following $\mathcal{NLL}$ expression.

```
(?lambda ?x not car(inst: ?x))             (5)
```

Whereas the closed-world assumption would allow us to infer that `veh1` belongs to this set, the open-world assumption underlying $\mathcal{NLL}$ doesn't support this conclusion.

The question whether all cars are vehicles can also be formulated in $\mathcal{NLL}$. To answer this question we can try to infer

```
(forall ?x vehicle(inst: ?x) if
          car(inst: ?x))                   (6)
```

from the knowledge base. The answer to this question has to be independent of the constants currently occurring in our knowledge base. On the basis of declaration (1), the answer has to be positive.

Now let us assume that the left front seat of `veh2` is red. Choosing `lfseat` to designate the left front seat, this can be represented in the following way.

```
hasPart(inst: veh2, theme: lfseat)         (7)
seat(inst: lfseat)                         (8)
hasColour(inst: lfseat, theme: red)        (9)
```

To answer the question whether all seats of `veh2` are red we have to try to infer the following $\mathcal{NLL}$ expression.

```
(forall ?x
      hasColour(inst: ?x, theme: red)
      if hasPart(inst: veh2, theme: ?x)
         and seat(inst: ?x))               (10)
```

Because of the open-domain and open-world assumption, the answer to the question cannot be positive. Although the only seat the car seller knows to be part of `veh2` is actually red, there may be other seats of `veh2` and these seats may not be red.

Intuitively, a positive answer is much more plausible. We would assume that the car seller knows all the seats of `veh2` and knows the colour of every seat of `veh2`. It is possible to extend the knowledge base using *number restrictions* in such a way that we can infer a positive answer, e.g.

```
((= 1) ?x hasPart(inst: veh2, theme: ?x)
       and seat(inst: ?x))                 (11)
```

[1]PRACMA is short for 'PRocessing Arguments between Controversially Minded Agents.'

declares that veh2 has exactly one seat. declarations (7),(8),(9), and (11) taken together allow us to answer query (10) positively. However, it seems to be more natural to extend the language by an *epistemic modal operator* in the style of Lifschitz [Lifschitz,1991] to solve the problem. For a description of an extension of the knowledge representation language $\mathcal{ALC}$ by an epistemic operator refer to Donini et al. [Donini *et al.*,1992].

Suppose our language contains such an epistemic operator K. Then we have two possibilities to get a positive answer to the question. The first possibility is to reformulate the question slightly in the following way.

```
(forall ?x
    hasColour(inst: ?x, theme: red) if
    K(hasPart(inst: veh2, theme: ?x)
      and seat(inst: ?x)))                (12)
```

Now the question is whether all known seats of veh2 are red and the answer has to be positive. This approach causes the problem how the natural language analysis module should determine the epistemic character of question (12) opposed to the non-epistemic character of question (6).

The second possibility is to add a declaration of the following form to the knowledge base

```
not (hasPart(inst: veh2, theme: ?x)
     and seat(inst: ?x)) if
  not K(hasPart(inst: veh2, theme: ?x)
        and seat(inst: ?x))               (13)
```

This declaration allows to conclude that an object is either not part of veh2 or not a seat if it is not known to be part of veh2 and a seat.

Obviously, we are now able to turn our knowledge base system into a database system either by suitably adding epistemic operators to all the queries or by adding enough epistemic rules to the knowledge base. Therefore, the extension of knowledge representation languages with an epistemic operator is a first step to unify the database world and the knowledge base world.

3 Future Work

It is well-known that theorem proving in a first-order language containing an epistemic operator is not even semi-decidable. Although the answers to the example questions presented in the previous section seem to be derived easily, there is no hope to find a correct and complete inference mechanism which is able to deduce them.

If we need a correct inference mechanism, the only possibility we have is to restrict the knowledge representation language, i.e. we have to identify a decidable fragment of $\mathcal{NLL}$ to which we can add an epistemic operator without loosing decidability.

References

[Brachman and Schmolze, 1985] Ron J. Brachman and J. G. Schmolze. An Overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.

[Brachman *et al.*, 1991] Ron J. Brachman, Deborah L. McGuinness, Peter F. Patel-Schneider, and A. Borgida. Living with CLASSIC: When and how to use a KL-ONE-like language. In J. F. Sowa, editor, *Principles in Semantic Networks: Explorations in the Representation of Knowledge*, pages 401–456. Morgan Kaufmann, San Mateo, California, 1991.

[Buchheit *et al.*, 1993] M. Buchheit, F. M. Donini, and A. Schaerf. Decidable reasoning in terminological knowledge representation systems. Research Report RR-93-10, Deutsches Forschungszentrum für Künstliche Intelligenz, Saarbrücken, Germany, 1993.

[Donini *et al.*, 1991] F. M. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning*, pages 151–162, Cambridge, USA, April 22-25 1991. Morgan Kaufmann.

[Donini *et al.*, 1992] F. M. Donini, M. Lenzerini, D. Nardi, A. Schaerf, and W. Nutt. Adding Epistemic Operators to Concept Languages. In B. Nebel, C. Rich, and W. Swartout, editors, *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, pages 342–353, Cambridge, USA, 1992. Morgan Kaufmann.

[Hustadt and Nonnengart, 1993] U. Hustadt and A. Nonnengart. Modalities in knowledge representation. In Chris Rowles, Huan Liu, and Norman Foo, editors, *Proceedings of the 6th Australian Joint Conference on Artificial Intelligence*, pages 249–254, Melbourne, Australia, 16–19 November 1993. World Scientific.

[Jameson *et al.*, 1994] Anthony Jameson, B. Kipper, A. Ndiaye, R. Schäfer, J. Simons, T. Weis, and D. Zimmermann. Cooperating to be noncooperative: The dialog system pracma. To appear in the Proceedings of the 18th Annual German Conference on Artificial Intelligence, 1994. Springer.

[Laubsch and Nerbonne, 1991] J. Laubsch and J. Nerbonne. An Overview of $\mathcal{NLL}$. Technical report, Hewlett Packard Laboratories, May 1991.

[Lifschitz, 1991] Vladimir Lifschitz. Nonmonotonic databases and epistemic queries. In *Proceedings of the Twelfth International Conference on Artificial Intelligence*, pages 381–386, Sydney, Australia, August 24–30 1991. Morgan Kaufmann.

[Schmidt-Schauss, 1989] M. Schmidt-Schauss. Subsumption in KL-ONE is Undecidable. In R. J. Brachman and H. J. Levesque, and R. Reiter, editors,*Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, pages 421–431, Toronto, Canada, May 15-19 1989. Morgan Kaufmann.

Tractable Reasoning in a Universal Description Logic: Extended Abstract*

Klaus Schild
German Research Center for Artificial Intelligence
Stuhlsatzenhausweg 3, D-66123 Saarbrücken, FRG
e-mail: schild@dfki.uni-sb.de

1 Introduction

Description logics (also called *terminological logics* or *concept languages*) have been designed for the logical reconstruction and specification of knowledge representation systems descending from KL-ONE such as BACK, CLASSIC, $\mathcal{KRIS}$, and LOOM.[1] These systems are used to make the terminology of an application domain explicit and then to classify these definitions automatically into a taxonomy according to semantic relations like subsumption and equivalence. More precisely, automatic classification refers to the ability to insert a new concept into the taxonomy in such a way that it is directly linked to the most specific concept it is subsumed by and to the most general concept it in turn subsumes. Terminological knowledge representation systems thereby support the task to formalize an application in at least two respects. On the one hand, they urge the user to isolate the intrinsic concepts of the application; on the other hand they may detect hidden subsumption and equivalence relations between definitions or may even detect that a definition is incoherent.

A model of the application is then given by associating special objects of the domain with the concepts of the terminology. The systems mentioned above in turn automatically classify these objects with respect to the given terminology and to those membership relations which have been asserted explicitly. In this case, however, automatic classification refers to the ability to find the most specific concept the object is a member of.

Terminologies comprise two different kinds of terms, viz. so-called *concepts* and *roles*. The former are intended to represent classes of objects of a given domain, while the latter represent binary relations over this domain. Concepts can either be simple *concept names*, representing not further specified classes of objects, or structured by means of a fixed set of *concept structuring primitives*. Common concept structuring primitives are *concept conjunction* $\sqcap$ and universal quantification $\forall R{:}C$ over a role R. Concept conjunction is to be interpreted as set intersection, while the concept $\forall R{:}C$ denotes all those objects d of the domain for which each object related to d by the role R is a member of the concept C. Although there exist many other concept structuring primitives, it is commonly accepted that these two should be part of each concept language. In contrast to concepts, roles are often taken to be atomic, i.e., there are no roles other than *role names*. The standard concept language $\mathcal{ALC}$, for instance, does not comprise any role structuring primitives. However, in addition to those mentioned above, this language comprises *concept disjunction* $\sqcup$, *concept negation* $\neg$ as well as existential quantification $\exists R{:}C$ over a role R as concept structuring primitives. For details the reader is referred to [Schmidt-Schauß and Smolka, 1991].

Definitions are given by associating a concept or role T with a concept name (resp., role name) TN. Such a definition is represented by the expression $TN \doteq T$ and is called *concept* and *role introduction* respectively. *Terminologies* are just finite sets of concept and role introductions such that each concept and role name is defined at most once, i.e., for every concept and role name TN there exists at most one concept or role introduction the left-hand side of which is TN.

As already mentioned, a model of application domain is described in terms of the given terminology. More precisely, specific objects of the domain and pairs of objects can be associated with concepts and roles of the terminology, where these objects are syntactically represented by so-called *individual names*. It can either be asserted that an individual name a is an instance of a concept C or that it is related to another individual name, say, b, by a role R. Such assertions are called *assertional axioms* and are represented by the expressions $a{:}C$ and $(a,b){:}R$ respectively. A finite set of assertional axioms forms a *knowledge base*.

From a theoretical point of view, the computational service provided by terminological knowledge representation systems can be reduced to answer queries of the following form with respect to a knowledge base $\mathcal{KB}$ and to a terminology $\mathcal{T}$: a *query* can be an assertional axiom or an *inclusion axiom* of the form $T_1 \sqsubseteq T_2$, where T_1 and T_2 are either two concepts or two roles. The meaning of such a query Q posed with respect to $\mathcal{KB}$ and $\mathcal{T}$ is usually given in terms of so-called interpretations and models. An *interpretation* $\mathcal{I}$ consists of a *domain* $\Delta^{\mathcal{I}}$ and a *val-*

*This work was supported by a grant from the *Deutsche Forschungsgemeinschaft* (DFG).

[1]For a good overview of the so-called KL-ONE family the reader is referred to [Woods and Schmolze, 1992]; for KL-ONE itself cf. [Brachman and Schmolze, 1985].

uation $\mathcal{V}$ over $\Delta^{\mathcal{I}}$ along with an *interpretation function* $.^{\mathcal{I}}$. The valuation $\mathcal{V}$ over $\Delta^{\mathcal{I}}$ maps each concept name to a subset of $\Delta^{\mathcal{I}}$ and each role name to a binary relation over $\Delta^{\mathcal{I}}$. Individual names, however, are mapped to singleton sets containing exactly one element of $\Delta^{\mathcal{I}}$. The interpretation function $.^{\mathcal{I}}$, on the other hand, just extends $\mathcal{V}$ to deal with arbitrary concepts and roles in such a way that all concept and role structuring primitives are interpreted properly. The concept structuring primitives $\sqcap$, $\sqcup$, $\neg$, for instance, are to be interpreted as the corresponding set operations on $\Delta^{\mathcal{I}}$, while the interpretation of the concept $\forall R{:}C$ is defined inductively as follows: if $C^{\mathcal{I}}$ and $R^{\mathcal{I}}$ have already been defined, then $(\forall R{:}C)^{\mathcal{I}}$ is $\{d \in \Delta^{\mathcal{I}} : \forall e(\langle d,e\rangle \in R^{\mathcal{I}}), e \in C^{\mathcal{I}}\}$.

An interpretation $\mathcal{I}$ is then said to be a *model* of the inclusion axiom $T_1 \sqsubseteq T_2$ just in case that $T_1^{\mathcal{I}} \subseteq T_2^{\mathcal{I}}$ and, if a and b are individual names such that $a^{\mathcal{I}}$ is $\{\underline{a}\}$ and $b^{\mathcal{I}}$ is $\{\underline{b}\}$, then $\mathcal{I}$ is a model of the assertional axiom $a{:}C$ (resp., of $(a,b){:}R$) just in case that $\underline{a} \in C^{\mathcal{I}}$ (resp., $\langle \underline{a}, \underline{b}\rangle \in R^{\mathcal{I}}$). Not very surprising, an interpretation is a *model* of $\mathcal{KB}$ and $\mathcal{T}$ if it is a model of each of the elements of $\mathcal{KB}$ and $\mathcal{T}$. Now, Q is said to be *entailed* by $\mathcal{KB}$ and $\mathcal{T}$, written $\mathcal{KB} \models_{\mathcal{T}} Q$, if and only if every interpretation which is a model of $\mathcal{KB}$ and $\mathcal{T}$ is a model of Q as well. Moreover, we say that T_2 *subsumes* T_1 with respect to $\mathcal{T}$ if and only if it holds that $\emptyset \models_{\mathcal{T}} T_1 \sqsubseteq T_2$.

2 Terminological Reasoning is Inherently Intractable

Unfortunately, answering such queries is in most cases provably intractable, at least in terms of computational worst case complexity. This applies, for instance, to the basic inference of KL-ONE, although originally claimed to be computationally tractable. In fact, Schmidt-Schauß [1989] proved that there exists no algorithm at all which decides whether one concept of KL-ONE subsumes another one or not, even with respect to empty terminologies.

Moreover, in [Schild, 1993, 94a], , it is proved that in case of the standard concept language $\mathcal{ALC}$, every algorithm capable of deciding whether one concept subsumes another one or not uses more than polynomial time in the worst case if at least one (possibly recursive) concept introduction is taken into account. Notably, this result holds no matter which of the usual kinds of semantics for recursive concept introductions is presupposed, viz. either *descriptive semantics* or *least* or *greatest fixed point semantics*, as Nebel [1991] called them.

It is also known that even in case of the minimal concept language (comprising no concept and role structuring primitives other than concept conjunction and universal quantification over role names), there exists no polynomial time algorithm which decides with respect to acyclic terminologies whether one concepts subsumes another one or not, unless P = NP [Nebel, 1990].

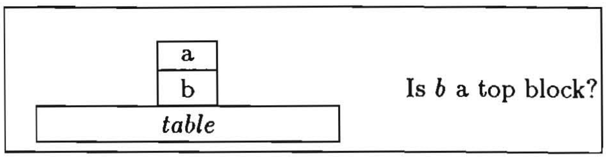

Figure 1: A sample blocks world.

$$\left\{\begin{array}{l} \forall x.block(x) \Leftrightarrow x = a \vee x = b, \\ a \neq b, a \neq table, b \neq table, \\ \forall x \forall y.on(x,y) \quad \Leftrightarrow \quad (x = a \wedge y = b) \\ \qquad\qquad\qquad\quad \vee \quad (x = b \wedge y = table) \end{array}\right\}$$
$$\overset{?}{\models} \; block(b) \wedge \neg\exists x.block(x) \wedge on(x,b)$$

Figure 2: Representing the sample blocks world by first-order formulae.

3 Model Checking Versus Theorem Proving

In the previous section, we have seen that, as Woods and Schmolze [1992] put it, "the surfeit of intractability results seems to have reached its logical end with the conclusion that practically everything of any use is intractable (in the worst case)." Recently, Halpern and Vardi [1991] proposed a possible solution to this very problem of knowledge representation. As a starting point, they re-examined the traditional approach to knowledge representation, going back to McCarthy [1968]. According to this approach the world to be modeled should be represented by a finite set of formulae of some given logic, preferably first-order logic. If a question to be answered is then formulated within the same logic, the answer depends on whether this formula is a *logical consequence* of the collection of formulae representing the world or not. In other words, it is checked whether *every* semantic structure which is a model of each of the formulae representing the world is also a model of formula corresponding to the question.

We shall illustrate this traditional approach to knowledge representation by means of an example, drawn from the famous blocks world. Suppose, for instance, we would like to represent a blocks world involving two blocks, say, a and b, where a lies on b and the latter in turn lies on a table. Suppose, furthermore, we would like to know whether b is a top block or not. Figure 1 depicts exactly this situation, while Figure 2 gives its representation in terms of first-order logic in the traditional way just described.

McCarthy's approach, however, gives rise to the problem that the need to represent all facts about the world in terms of some logic necessitates the use of very expressive logics such as full first-order logic. This, in fact, gives rise to difficulties because it is known that there exists no algorithm at all which generally decides logical consequence in full first-order logic [Church, 1936], and this remains true even when only finite interpretation domains are taken into consideration [Trahtenbrot, 1963].

At this very point Halpern and Vardi stressed that

$$\begin{array}{lcl}
\mathcal{D}om & = & \{a, b, table\} \\
[\![block]\!] & = & \{a, b\} \\
[\![on]\!] & = & \{\langle a, b\rangle, \langle b, table\rangle\} \\
\overset{?}{\models} & & block(b) \wedge \neg\exists x.block(x) \wedge on(x, b)
\end{array}$$

Figure 3: Representing the sample blocks world by a semantic structure.

in many cases the natural representation of a world to be modeled is a *semantic structure* rather than a collection of formulae. If, as in the traditional approach, queries are represented by formulae of a given logic, a query can be answered in this case depending on whether the formula representing the query is true in the *given* semantic structure or not. That is to say, it is checked whether the semantic structure is a model of the formula corresponding to the query. The fact that a (closed) formula α is true in a semantic structure $\mathcal{M}$ is usually indicated by $\mathcal{M} \models \alpha$. Resorting to this convention, Figure 3 gives such an alternative representation of the blocks world considered above.

In many cases this model checking approach has tremendous benefits, at least in terms of computational complexity. For instance, checking the truth of an arbitrary closed first-order formula[2] α in a finite semantic structure fixing the interpretation of all predicates and constants occurring in α is known to be *decidable* using at most polynomial space [Chandra and Merlin, 1977]. Recall that in contrast to this, there exists no algorithm at all which is able to decide whether an arbitrary formula of this kind is a logical consequence of a finite set of first-order formulae, even with only finite interpretation domains taken into account. However, it is also known that first-order model checking is still at least as hard as *any other* problem solvable using at most polynomial space, hence this problem is still very hard [Chandra and Merlin, 1977]. Anyway, Halpern and Vardi's intention was to forge a new approach to knowledge representation rather than to give concrete instances which allow for tractable inferences.

4 The Model Checking Approach to Terminological Reasoning

It should be clear that terminological knowledge representation, as described in the introduction, is committed to the traditional approach to knowledge representation rather than to the model checking approach. In [Schild, 1994b] we investigated the consequences of adapting Halpern and Vardi's model checking approach to terminological reasoning. It turned out that even in case of the most powerful description logic considered in the literature, answering queries become tractable just by replacing the usual kind of knowledge bases with single finite semantic structures fixing the interpretation of all *primitive* concepts and roles (i.e., those concept and role

[2]This formula should involve no function symbols other than constants.

$$\left\{\begin{array}{l}
a{:}Block, b{:}Block, table{:}\neg Block, \\
(a, b){:}on, (b, table){:}on, \\
a{:}(\neg\exists on^{-1}{:}Block), table{:}(\neg\exists on{:}Block)
\end{array}\right\}$$
$$\mathcal{T} = \{TopBlock \doteq Block \sqcap \neg\exists on^{-1}{:}Block\}$$
$$\overset{?}{\models}_{\mathcal{T}} \quad b{:}TopBlock$$

Figure 4: Representing the sample blocks world by an $\mathcal{ALC}^{-1}$-KB.

$$\begin{array}{lcl}
\mathcal{D}om & = & \{a, b, table\} \\
[\![Block]\!] & = & \{a, b\} \\
[\![on]\!] & = & \{\langle a, b\rangle, \langle b, table\rangle\}
\end{array}$$
$$\mathcal{T} = \{TopBlock \doteq Block \sqcap \neg\exists on^{-1}{:}Block\}$$
$$\overset{?}{\models}_{\mathcal{T}} \quad b{:}TopBlock$$

Figure 5: Representing the sample blocks world by a physical $\mathcal{ALC}^{-1}$-KB.

names which are mentioned somewhere in the terminology or in the query, but which are not defined).

But before engaging into details, have a look at Figure 4, which shows how to represent the already familiar blocks world in terms of $\mathcal{ALC}$ together with the inverse of roles $^{-1}$, as it would be done traditionally. Observe, however, that this representation is *incomplete* in that it solely states that block a lies on block b, while the latter in turn lies on the table, but it is left open whether there is any other block lying on b or on the table. As a matter of fact, there is no way at all to give an *accurate* representation of our blocks world in terms of $\mathcal{ALC}$, even when augmented by the inverse of roles. This means, in this case the so-called *open world assumption*,[3] traditionally made for terminological reasoning, is a nuisance rather than an advantage.

Figure 5 modifies the just considered representation in the spirit of the model checking approach. A finite semantic structure is shown there which fixes the interpretation of each primitive concept and role of $\mathcal{T}$, that is, it fixes the interpretation of *Block* and *on*. Such a semantic structure is obviously nothing but a valuation along with a domain. When taken together with a domain, the syntactic representation of such a valuation is called *physical knowledge base*, emphasizing the fact that they are intended to replace customary knowledge bases. Now, suppose $\mathcal{V}$ is such a physical knowledge base with domain $\mathcal{D}om$, $\mathcal{T}$ is an arbitrary terminology, and Q is a query. Then $\mathcal{V} \models_{\mathcal{T}} Q$ is intended to mean that every interpretation extending $\mathcal{V}$ which is a model of $\mathcal{T}$ is a model of Q as well, where an interpretation $\mathcal{I}$ is said to *extend* a physical knowledge base $\mathcal{V}$ with domain $\mathcal{D}om$ just in case that $\Delta^{\mathcal{I}} = \mathcal{D}om$ and, moreover, $\cdot^{\mathcal{I}}$ interprets all those concept and role names handled

[3]In contrast to the *closed world assumption*, usually made for databases, the open world assumption does *not* assume that all those facts that are not explicitly mentioned (or that cannot be inferred) are taken to be false.

by $\mathcal{V}$ in exactly the same way as $\mathcal{V}$ does.

In [Schild, 1994b] we investigated the computational complexity of answering such queries with respect to physical knowledge bases in the description logic $\mathcal{U}$, introduced by Patel-Schneider [1987] as a universal description logic. This concept language is *universal* in the sense that it encompasses all others considered in the literature, except for those which comprise nonstandard facilities like defaults, for instance. In addition to those of $\mathcal{ALC}$, this language comprises *number restrictions* of the form $\exists^{\geq n} R{:}C$ and $\exists^{\leq m} R{:}C$ as well as *role value maps* of the form $R \leq S$ as concept structuring primitives. Number restrictions restrict the number of role fillers (i.e., those objects which are related to an object by a role), while role value maps impose restrictions on the fillers of two roles. The concept $R \leq S$ states that all fillers of the role R are also fillers of the role S. In addition, $\mathcal{U}$ admits of individual names to occurring in concepts. The role structuring primitives of $\mathcal{U}$ are the *identity role* ϵ, Boolean operations $\sqcap$, $\sqcup$, $\neg$ on roles, the inverse R^{-1} of a role, the composition $R \circ S$ of two roles, as well as the *transitive closure* R^+ and the *reflexive-transitive closure* R^* of a role. For details cf. [Schild, 1994b] or [Patel-Schneider, 1987]. Notably, it is known that there cannot exist any algorithm which is capable of deciding subsumption between two concepts (or two roles) of $\mathcal{U}$, even with respect to empty terminologies [Schild, 1988].

The main result of [Schild, 1994b] is that even in this language $\mathcal{V} \models_{\mathcal{T}} Q$ can be decided in polynomial time provided that each of the following conditions is satisfied:

(a) $\mathcal{V}$ has a finite domain and specifies all concept and role names occurring in $\mathcal{T}$ and Q except for those which are defined in $\mathcal{T}$;

(b) Roles are not defined recursively;

(c) Concepts can be defined recursively, but then they must occur in their definition[4] *positively*, i.e., they must occur in the scope of an even number of negations, where $\exists^{\leq m} R{:}$ counts also as a negation. Moreover, each recursive definition must be given either least or greatest fixed point semantics, not necessarily in a uniform way.

Of course, each of these conditions calls for some comment. Condition (b) is commonly presupposed for terminological reasoning, while condition (c) constitutes the most liberal restriction on recursive concept definitions considered in the literature. The most important condition, however, is the first one in that it ensures all primitive concepts and roles to be specified extensionally. This restriction does make sense as these concepts and roles are exactly those which are not further specified according to the semantics. It can easily be verified that the sample query of Figure 5 obeys each of the three conditions above.

The employed algorithm capable of deciding $\mathcal{V} \models_{\mathcal{T}} Q$ in polynomial time just mimics the semantics of the concept and role structuring primitives of $\mathcal{U}$, storing already evaluated ones. To deal with recursive concept definitions, however, we exploited a technique for computing least and greatest fixed points due to Emerson and Lei [1986].

It turned out that even when relaxing condition (a) in such a way that $\mathcal{V}$ is solely required to have a finite domain, $\mathcal{V} \models_{\mathcal{T}} Q$ is still decidable in the universal description logic $\mathcal{U}$. In fact, we proved that in this case the computational complexity is essentially the same as the one of deciding ordinary subsumption between two concepts with respect to acyclic terminologies in the *minimal* concept language.[5]

We also investigated the consequences of incorporating some limited kind of incomplete knowledge by means of Reiter's *null values* [Reiter, 1984]. It turned out that, when presupposing P $\neq$ NP, admitting of null values causes intractability, even in case of $\mathcal{ALC}$. Thus our results suggest that the main source of computational complexity of terminological reasoning seems to be the ability to express *incomplete* knowledge.

5 Description Logics as Tractable Query Languages for Databases

Another interpretation of our results is that, when taken together with the least and greatest fixed point semantics, the universal concept language $\mathcal{U}$ can serve as a powerful but tractable query language for relational databases comprising solely unary and binary relations.[6] From this point of view terminologies are to be thought of as defining so-called *views*, possibly defined recursively.

At this very point, it is important to note that the universal description logic $\mathcal{U}$ is so strong in expressive power that it is even capable of *accurately* defining concepts such as directed acyclic graphs (*DAG*s), trees, or binary trees. The powerful role forming primitives of $\mathcal{U}$ actually admit of plausible and nonrecursive definitions of these concepts. As every finite graph can uniquely be represented by a physical knowledge base in a completely straightforward manner, these concepts provide views which can be used to extract from a huge collection of (connected) directed graphs exactly those which are acyclic or those which are trees or binary trees. If we additionally have recursive concept introductions along with least fixed point semantics at our disposal, we may even extract from a finite and-or-graph G (or a collection of such) exactly the *solvable* vertices, i.e., those vertices which are a root of an acyclic subgraph G_s of G such that every and-vertex of G_s has exactly those edges it has in G and, moreover, every or-vertex has at least one of those edges it has in G. Figure 6 gives the terminology of $\mathcal{U}$ defining all the concepts mentioned in this section, where the recursive concept introduction of *Solvable* should be given least fixed point semantics. This is just to demonstrate that even though the model check-

[4]In this context, a *definition* is meant to be the subterminology of $\mathcal{T}$ which contains exactly those concept introductions which are involved in the recursion.

[5]Technically speaking, in this case deciding $\mathcal{V} \models_{\mathcal{T}} Q$ in $\mathcal{U}$ is co-NP-complete.

[6]Note that unary and binary relations do suffice as far as only object-oriented databases are concerned.

$$
\begin{array}{rcl}
\mathit{DirectedGraph} & \doteq & \forall \mathit{connected}: \mathit{Vertex} \\
\mathit{connected} & \doteq & (\mathit{edge} \sqcup \mathit{edge}^{-1})^{*} \\
\mathit{Acyclic} & \doteq & \forall \mathit{connected}{:}(\mathit{edge}^{+} \leq \neg\epsilon) \\
\mathit{DAG} & \doteq & \mathit{DirectedGraph} \sqcap \mathit{Acyclic} \\
\mathit{Tree} & \doteq & \mathit{DAG} \\
 & \sqcap & \forall \mathit{edge}^{*}{:}\exists^{\leq 1} \mathit{edge}^{-1}: \mathit{Vertex} \\
\mathit{BinaryTree} & \doteq & \mathit{Tree} \\
 & \sqcap & \forall \mathit{edge}^{*}{:}\exists^{\leq 2} \mathit{edge}: \mathit{Vertex} \\
\mathit{AndOrGraph} & \doteq & \mathit{DirectedGraph} \\
 & \sqcap & \forall \mathit{connected}{:}\mathit{AndOrVertex} \\
\mathit{AndOrVertex} & \doteq & \mathit{AndVertex} \sqcap \neg \mathit{OrVertex} \\
 & \sqcup & \mathit{OrVertex} \sqcap \neg \mathit{AndVertex} \\
\mathit{Solvable} & \doteq & \neg \exists \mathit{edge}: \mathit{Vertex} \\
 & \sqcup & \mathit{AndVertex} \sqcap \forall \mathit{edge}{:}\mathit{Solvable} \\
 & \sqcup & \mathit{OrVertex} \sqcap \exists \mathit{edge}{:}\mathit{Solvable}
\end{array}
$$

Figure 6: A terminology of $\mathcal{U}$.

ing approach to terminological knowledge representation does make it possible to answer queries in polynomial time, there are actually nontrivial inferences to perform.

Acknowledgements

I would like to thank Martin Buchheit for valuable comments on earlier drafts of the abstract.

References

[Brachman and Schmolze, 1985] Ronald J. Brachman and James G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.

[Chandra and Merlin, 1977] Ashok K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational databases. In *Proceedings of the 9th ACM Symposium on Theory of Computing*, pages 77–90, 1977.

[Church, 1936] Alonzo Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58:345–363, 1936.

[Emerson and Lei, 1986] E. Allen Emerson and Chin-Laung Lei. Efficient model checking in fragments of the propositional mu-calculus (extended abstract). In *Proceedings of the 1st IEEE Symposium on Logic in Computer Science*, pages 267–278, Boston, Mass., 1986.

[Halpern and Vardi, 1991] Joseph Y. Halpern and Moshe Y. Vardi. Model checking vs. theorem proving: A manifesto. In *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, pages 325–334, Cambridge, Mass., 1991.

[McCarthy, 1968] John McCarthy. Programs with common sense. In M. Minsky, editor, *Semantic Information Processing*, pages 403–418. MIT Press, Cambridge, Mass., 1968.

[Nebel, 1990] Bernhard Nebel. Terminological Reasoning is Inherently Intractable. *Artificial Intelligence*, 43:235–249, 1990.

[Nebel, 1991] Bernhard Nebel. Terminological cycles: Semantics and computational properties. In J. Sowa, editor, *Formal Aspects of Semantic Networks*, pages 331–361. Morgan Kaufmann, San Mateo, Cal., 1991.

[Patel-Schneider, 1987] Peter F. Patel-Schneider. *Decidable, Logic-Based Knowledge Representation*. PhD thesis, University of Toronto, Toronto, Ont., 1987. Computer Science Department, Technical Report 201/87.

[Reiter, 1984] Raymond Reiter. Towards a logical reconstruction of relational database theory. In M. L. Brodie, J. Mylopoulos, and J. W. Schmidt, editors, *On Conceptual Modeling*, pages 191–233. Springer-Verlag, Berlin, FRG, 1984.

[Schild, 1988] Klaus Schild. Undecidability of subsumption in $\mathcal{U}$. KIT Report 67, Department of Computer Science, Technische Universität Berlin, Berlin, FRG, 1988.

[Schild, 1993] Klaus Schild. Terminological cycles and the propositional μ-calculus. DFKI Research Report RR-93-18, German Research Center for Artificial Intelligence (DFKI), Saarbrücken, FRG, April 1993.

[Schild, 1994a] Klaus Schild. Terminological cycles and the propositional μ-calculus. In *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning*, pages 509–520, Bonn, FRG, 1994.

[Schild, 1994b] Klaus Schild. Tractable reasoning in a universal description logic. DFKI Research Report, German Research Center for Artificial Intelligence (DFKI), Saarbrücken, FRG, 1994. Forthcoming.

[Schmidt-Schauß and Smolka, 1991] Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.

[Schmidt-Schauß, 1989] Manfred Schmidt-Schauß. Subsumption in KL-ONE is undecidable. In *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning*, pages 421–431, Toronto, Ont., 1989.

[Trahtenbrot, 1963] B. A. Trahtenbrot. Impossibility of an algorithm for the decision problem in finite classes. *American Mathematical Society Translation Series*, 23(2):1–5, 1963.

[Woods and Schmolze, 1992] William A. Woods and James G. Schmolze. The KL-ONE family. In F.W. Lehmann, editor, *Semantic Networks in Artificial Intelligence*, pages 133–178. Pergamon Press, 1992.

Generating queries from complex type definitions*

Manfred A. Jeusfeld
Informatik V, RWTH Aachen, D-52056 Aachen
jeusfeld@informatik.rwth-aachen.de

Abstract

Many information systems are implemented as application programs connected to a database system. A characteristic problem of such systems is the famous impedance mismatch, i.e., the conceptual distance between the programming and the database languages. The traditional solution is to implement an interface that transforms one representation into the other. Commercial database systems offer preprocessors that allow to embed the database language (e.g., SQL) into the programming language (e.g., C). Such an approach frees the application programmer from the task to specify details of the communication. However, the impedance mismatch is not solved but aggravated. The set-oriented database language is intermixed with the element-oriented programming language, a notorious cause for programming errors. Moreover, there is no support in mapping the restricted data representation of databases into the more complex type system of programming language. This paper proposes an intermediate language, the API modules, for specifying the relationship between the representations in the database and in the application program. The query for retrieving the information and the data types for storing it can be generated from the API module. The modules are simple enough to allow reasoning on queries generated from them.

1 Introduction

The purpose of a database system is to maintain a large amount of information for a variety of application programs. The application-specific clustering is either described as a database view definition or performed by filters inside the application program. Both approaches have their disadvantages:

- View definition languages are restricted to the type system of the database system. In the case of relational databases, only flat relations can be expressed. In the case of object-oriented databases, the type system depends on the specific data model of the database system.
- Handcoded clustering by filter procedures within the application program is error-prone and gives away the chance of reasoning on the relationship between the information in the database and in the application program.

Section 2 introduces *API modules* as the interface between the database and the application program. Base types are imported from the database. Application specific types are defined by using tuple, set, and pointer constructors. The latter allows to represent recursive concepts of the database schema.

Section 3 presents the mapping of the API modules to a logic program delivering complex terms. These terms are read by a parser that itself is generated from the API modules.

Section 4 relates the types in an API module to statements of a concept language. Thereby, types of two different API modules can be checked for subsumption and consistency.

2 Interface Modules

Interfaces between imperative-style programming languages should both reflect the major type constructors and the facilities of the database query language. The most common type constructors are *tuple* and *set*. Some languages also support *lists*. Pseudo-recursive type definitions are possible when allowing pointer types, e.g. in C and Modula-2. Common base types are *Integer* and *String*. The denotational semantics of a type expression is a potentially infinite set of values, for example *[Integer,String]* denotes the cartesian product of the semantics of the component types.

2.1 Example

Assume a database provides information about a company. An application programmer has the task to process the information about the projects of employees who work for a given department. The API module could look as in Figure 1.

The FROM clause imports concepts from the database schema. They are used like (finite) base types. Their extension is represented in the current database state. The TYPE clauses declare complex

*This work was partly supported by the Commission of the European Union under ESPRIT BRA 6810 (Compulog 2).

```
API-MODULE Emps;
FROM CompanyDb IMPORT Employee, Project,
     Department, String;
TYPE
     EmpType/Employee = [name: String;
                         project: {Project};
                         dept: DeptType];

     DeptType/Department = [deptName: String;
                            head: *EmpType];
PORT
     e: {EmpType| dept.deptName=$N};
END.
```

Figure 1: API module for the company example

data structures on top of the imported concepts. **EmpType** is a record type which represents the name of an employee, his projects, and the department. The latter is given by the name and the reference (pointer) to that employee who is the head of the department. The purpose of the pointer is to encode recursive type definitions. The PORT declaration defines which information of the database should be teransfered to the application program. Here, all employees who have a department named by **$N** are of interest. The token **$N** denotes a placeholder for a string whose value is inserted by the application program at run time.

3 Query Generation

From the database point of view, an API module is a collection of simple view definitions whose extensions are represented by terms conforming the type definitions. These views are encoded as a logic program defining a predicate **hasType(T,V)**. It formally defines the set of values V having type T, i.e., the semantics of the type T. The database system is modelled by two predicates for accessing information:

- In(X,C) denotes that the database object X is an instance of the concept C, e.g., In(e2341,Employee), In("Peter Wolfe",String).
- A(C,a,X,Y) states that the object X is related to the object Y by an attribute a which is defined in class C, e.g., A(Employee,name,e2341,"Peter Wolfe").

The logic program can automatically be generated from the type definitions by a simple top down traversing algorithm on the syntax tree of a type definition[1]:

For each concept C imported in the API module we include a clause which delivers all values of type C.

```
hasType(C,C(_X)) :-
   In(_X,C).
```

A tuple type has the general form T/C = [a1:T1,...,ak:Tk]. The decoration C is called the "class" of T. It is mapped to the clause pattern

```
hasType(T,T(_X,_Y1,...,_YK) :-
   In(_X,C),
   <map(a1:T1)>,
   ...
   <map(ak:Tk)>.
```

The parts <map(ai:Ti)> have to be mapped as follows:

- If Ti is a set type {S} where S is a type name for a tuple-valued type with arity m then <map(ai:Ti)> is replaced by

```
SET_OF(S(_Z,_Z1,...,_Zm),
    ( A(C,ai,_X,_Z),
      hasType(S,S(_Z,_Z1,...,_Zm))),
    _Yi)
```

- If Ti is a set type {*S} where S is a type name of a tuple type with class D then <map(ai:Ti)> is replaced by

```
SET_OF(REF(S,_Z),
       ( A(C,ai,_X,_Z),
         In(_Z,D)),
       _Yi)
```

- If Ti is a tuple type with arity m then the macro is replaced by

```
_Yi = Ti(_Y,_Z1,...,_Zm),
A(C,ai,_X,_Y),
hasType(Ti,_Yi)
```

- Finally, pointer types *Ti where Ti is a record type with class D are mapped to the condition

```
(_Yi = REF(Ti,_Y),
A(C,ai,_X,_Y),
In(_Y,D);
_Y = null_value)
```

The operator ';' stands for a logical disjunction. There will be no backtracking on this disjunction. Thus, _Y will either be bound to a term REF(.,.) or to the special value null_value.

The PORT clauses specify those subsets of types which are of interest to the application program. A port definition

```
PORT v: {T| a1.a2...an=$P}
```

is compile to the clause

```
askPort(_S,v,_P) :-
  SET_OF(_X,
     (hasType(T,_X),
      path(_X,[a1,a2,...,an],_P),
     _S).
```

The predefined predicate path evaluates the path expression a1.a2...an starting from _X. Note that the parameter $P becomes an argument of the askPort predicate. It is instantiated by the application program when calling the goal askPort. The result is returned in the first argument.

The restriction in the port definition can easily be extended to contain several conditions. Moreover, one can allow a constant or a second path expression instead of the parameter on the right-hand side of the equality.

[1]We adopt the syntax of Prolog to denote the clauses. Variables start with an underscore. The meta predicate SET_OF(x,c,s) evaluates s as the set of all elements x satisfying the condition c

```
hasType(String,String(_S)) :-
  In(_S,String).

hasType(Project,Project(_P)) :-
  In(_P,Project).

hasType(DeptType,DeptType(_D,_DN,_M)) :-
  In(_D,Department),
  _DN = String(_Z1),
  A(Department,_D,deptName,_Z1),
  hasType(String,_DN),
  _M = REF(EmpType,_Z2),
  A(Department,_D,head,_Z2),
  In(_Z2,Employee).

hasType(EmpType,EmpType(_E,_N,_PS,_DT)) :-
   In(_E,Employee),
   _N=String(_Z1),
   A(Employee,_E,name,_Z1),
   hasType(String,_N),
   SET_OF( Project(_Z2),
     (A(Employee,_E,project,_Z2),
      hasType(Project,Project(_Z2))),
     _PS),
   _DT = DeptType(_D,_DN,_M),
   A(Employee,_E,dept,_D),
   hasType(DeptType,_DT).

askPort(_S,e,_N) :-
   SET_OF(_X,
       (hasType(EmpType,_X),
       path(_X,[dept,deptName],_N),
       _S).
```

Figure 2: Logic program for the example

3.1 Mapping of the Example

The definition of **hasType** for the running example is presented in Figure 2.

The values of the imported concepts are represented as unary terms, e.g. `String("Peter Wolfe")`. Values of complex terms have more components according to the type definition. For example,

```
EmpType(e2341,String("Peter Wolfe"),
        [Project(p1),Project(p2)],
        DeptType(d41,String("Marketing"),
         REF(EmpType,e3331)))
```

is the term representing a value of **EmpType**. Values of set types like {`Project`} are sequences of values of the member type enclosed by brackets. The component for the **dept** attribute is avalue of type **DeptType**. This shows the representation of pointers as terms **REF(T,X)** where **X** is the identifier of the value (of type **T** pointed to. The identifier is always the first component of a term **T(X,...)**. All identifiers are constants from the database.

4 Properties of Interfaces

Termination of the logic program is guaranteed, and the types defined in API modules can be compared with the database schema and with each other.

4.1 Termination

On first sight, the generated logic program is recursive in the **hasType** clause and it contains complex terms as arguments. Thus, one has to ensure termination when evaluation it by the SLD strategy for logic programs.

Fortunately, if one makes sure that the types in the API module are defined non-recursively, then there is a partial order on the type names. If a type definition for **T1** uses a type **T2** on the right-hand side, then $T1 > T2$ holds. The definition of the logic program generator propagates this property to all clauses of the **hasType** predicate: if **hasType(T,.)** occurs in the condition of a clause **hasType(R,.)** then **T** must be smaller than **R**. Consequently, the logic program terminates on each goal **hasType(T,X)**[2].

A corrolar of this proposition is the finiteness of the sets interpreting the types in the API module.

4.2 Reasoning Services

The constructs in the API module were deliberately choses to be conformant with the concept language dialect of Buchheit *et al.* 1994. A couple of reasoning services are possible, each determing a different set of axioms to be reasoned about. We illustrate only one service, type checking against the database.

The type definitions in an API module make assumptions about the structure of the imported database concepts. In the example of Figure 1, the concepts **Employee** must at least have three attribute categories **name**, **project**, and **dept**. For the **Department** concept, two attributes categories **deptName** and **head** are required. Moreover, attribute cardinalities for the answer objects are stated:

- a set-valued attribute like **project** does not induce any cardinality constraint;
- a pointer-valued attribute like **head** restricts the the number of attribute fillers to be less or equal 1;
- the remaining attributes like **dept** must have exactly one filler.

Please note that these properties apply to the defined concepts like **EmpType** (ET) and not to the imported concepts like **Employee** (E). The concept language expression is:

$$ET = E \sqcap (= 1\ name.S) \sqcap (= 1\ dept.DT)$$
$$DT = D \sqcap (= 1\ deptName.S) \sqcap (\leq 1\ head.E)$$

As prescribed by the logic program, the pointer-valued attribute **head** of **DeptType** is not refering to **EmpType** directly but to its associated class **Employee**. Thereby, circular concept definitions are prevented.

These equalities for the type definitions are true provided the database schema has a schema consistent to it. At least it has to fulfill the following "well-typedness" axioms[3]:

[2]One has to assume that the underlying database is finite. This is however a standard assumption with databases.

[3]The symbol $\top$ stands for the most general concept.

$$E \sqsubseteq \forall name.\top \sqcap \forall project.\top \sqcap \forall dept.\top$$
$$D \sqsubseteq \forall deptName.\top \sqcap \forall head.\top$$

One can check this by adding it to the database schema and checking its consistency. The service would just make sure that all referenced attributes are defined in the database schema.

With a stricter regime, one can demand that the database schema must have the same or sharper cardinality constraints and that the well-typedness is refined to the concepts appearing as attribute types in the API module:

$$E \sqsubseteq ET \sqcap \forall name.S \sqcap \forall project.P \sqcap \forall dept.DT$$
$$D \sqsubseteq DT \sqcap \forall deptName.S \sqcap \forall head.ET$$

Here, the database schema has to fulfill the structure of the types in the API module. Consequently, all instances of the database concepts will apply to the type definitions. The type definitions would only project on the attributes of interest. Even if one regards this as a too narrow coupling, the test on consistency of the above axioms with the database schema returns useful information to the designer of an API module.

5 Programming Language Embedding

From the API modules, programming language data types can be derived. Currently, a prototype for the C++ language is implemented. The tuple types are mapped to C++ structures, the sets to linked lists, and the pointers to C++ pointers. While the concept language view makes no difference between pointer-valued attributes (like `*EmpType` and their associated class `Employee`, the representation within the application program is very different:

- A value `Employee(X)` is stored in a variable with C++ type `char*` because `X` is just an string representing a database constant.
- A value `REF(EmpType,X)` is represented as a main memory address pointing to the location where the value `EmpType(X,...)` is stored. This allows the application program to follow attribute chains by fast main memory adressing.

Communication between an applications program and the database is routed through the ports. The term representations of port `p` returns in argument `s` of the query `askPort(s,p,x1,...,xn)` are read by the application program. The arguments `x1,...,xn` contain the constants for the selection conditions[4]. The "read" procedure, basically a simple term parser, stores the values in the C++ data structures. Both the parser and the data structures are generated from the API module by a compiler. Since the `askPort` predicate can only return syntactically correct terms, an exception handling for malformed answers is superfluous.

[4]Like for types the properties of port definitions can be investigated within the framework of concept languages. If the parameters `x1,...xn` are known, then the selection conditions are path agreements. Moreover one may allow path expressions of the form $a_1.a_2...a_r = b_1.b_2...b_s$ without compromising on the theoretical complexity of the reasoning.

6 Related Work

Lee and Wiederhold 1994 present a mapping from relational database schemas to complex objects. It is more general in the sense that arbitrary arities of the relations are allowed. In this paper, we assume a totally normalized schema of the database consisting of unary relations for class membership and binary relations for attributes. The advantage of our approach is that the algorithm for the generation of the logic programm can be kept free of reasoning on foreign key dependencies.

Plateau *et al.* 1992 present the view system of O_2 as complex type definitions coupled with the database types and with prescriptions for graphical display. The type system contained in the O_2 data model. Reasoning on type correctness is done by the compiler.

The Interface Description Language IDL by Nestor *et al.* 1992 has four type constructors for records, lists, sets, and classes (unions of different record types). The base types represents boolean, integers, rationals, and strings. The values are transfered between two programs by using a term representation similar to ours. The difference is the missing formal relationship between type definitions and (database) concepts.

A recent proposal by Papakonstantinou *et al.* 1994 encodes all type information with the term representation of a value. An application program has to provide generic data structures capable of storing arbitrary values (though restricted to a fixed set of base types). The advantage is the flexibility of the approach. A disadvantage is missing compile time type checking.

Persistent object systems, esp. Tycoon by Matthes 1993, "lift" the type systems of information sources and application programs into a single type system. Because of the heterogenous information sources, the approach is more general than in O_2. Reasoning is again restricted to type checking.

7 Conclusion

We defined API modules as mediators between application programs and databases. Both programming language data types and database queries are generated from the module description. The language is simple enough to guarantee termination of the query and efficient reasoning on the type definitions. Pointer types are introduced to simulate recursive datatypes and find a natural counterpart in the database query.

In future, we plan to eliminate the distinction between application program and database in the API modules. Application programs can serve as a "database" provided they offer the ability the interpret queries on their information. Then, information flow design between a collection of programs can be supported by reasoning on the relationship between the type definitions.

Acknowledgement. Many thanks to Claudia Welter and Martin Staudt for attacking weak points in earlier versions of this paper.

References

[Buchheit *et al.*, 1994] M. Buchheit, M.A. Jeusfeld, W. Nutt, and M. Staudt. Subsumption between queries to object-oriented databases. *Information Systems*, 19(1):33–54, 1994.

[Lee and Wiederhold, 1994] B.S. Lee and G. Wiederhold. Outer joins and filters for instantiating objects from relational databases trough views. *IEEE Trans. Knowledge and Data Engineering*, 6(1):108–119, 1994.

[Matthes, 1993] F. Matthes. Persistente Objektsysteme. Springer-Verlag, 1993.

[Papakonstantinou *et al.*, 1994] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. Submitted paper, 1994.

[Plateau *et al.*, 1992] D. Plateau, P. Borras, D. Leveque, J. Mamou, and D. Tallot. Building user interfaces with Looks. In F. Bancilhon, C. Delobel, P. Kannelakis (eds.):*Building an Object-Oriented Database System - The Story of O2*, Morgan-Kaufmann, 256–277, 1992.

[Nestor *et al.*, 1992] J. R. Nestor, J. M. Newcomer, P. Giannini, and D. L. Stone. IDL - The language and its implementation. Prentice Hall, 1990.

Terminological logics for schema design and query processing in OODBs*

D. Beneventano°, S. Bergamaschi°, S. Lodi°, C. Sartori

Dipartimento di Elettronica, Informatica e Sistemistica
Università di Bologna - CIOC-CNR
°Facoltà di Ingegneria, Università di Modena

Introduction

ıe paper introduces ideas which make feasible and ective the application of Terminological Logic (TL) :hniques for schema design and query optimization in ›ject Oriented Databases (OODBs).

Applying taxonomic reasoning and TL in database vironment for traditional semantic data models led a number of promising results for database schema sign and other relevant topics, as query processing d data recognition. In particular, in [Bergamaschi d Sartori,1992] a general theoretical framework has en presented, which supports conceptual schema acisition and organization by preserving *coherence* and *inimality* w.r.t. inheritance, exploiting the framework terminological reasoning. Complex object data mod-, recently proposed in the area of OODBs, are more pressive than actually implemented TL languages in me aspects. For instance, most of the complex ob-:t data models introduce a distinction between objects th identity and values, which is not present in TL lanages. Further, complex object models usually support ditional type constructors, such as set and sequence. ɔst importantly, these models usually support the repsentation and management of cyclic classes. These oblems have found a solution in [Bergamaschi and :bel,1992; 1993], by the adoption of an extended TL, med ODL.

A real database specification always includes a set of les, the so-called *integrity constraints*, which should arantee data consistency. Constraints are expressed various fashions, depending on the data model: e.g. bsets of first order logic, or inclusion dependencies and edicates on row values, or methods in OO environents. In particular OO methods are programs whose mantics cannot be inspected by an automatic reasoner. first, necessary, improvement is to express at least a ıss of integrity constraints at schema level. Our pro-ısal is to generalize the notion of a database schema cluding a declarative specification of a set of integrity nstraints and to exploit this knowledge together with taxonomic reasoning for the different tasks of schema design and query optimzation. Let us examine separately the two aspects of schema design and query optimization.

*This research has been partially funded by the project .U.R.S.T. 40%, "Metodi Formali e Strumenti per Basi di ıti Evolute".

2 Reasoning services in schema design

Provided that an adequate formalism to express integrity constraints is available, the following question arises: Is there any way to populate a database which satisfies the constraints supplied by a designer? Means of answering to this question should be embedded in automatic design tools, whose use is recommendable or often required in the difficult task of designing non-trivial database schemas.

Our proposal is to use the tableaux-calculus technique to guarantee schema consistency, therefore including state constraint consistency. Such a solution is actually a modification of existing algorithms for Description Logics [Schmidt-Schauss and Smolka,1991; Hollunder and Nutt,1990; Hollunder *et al.*,1990; Donini *et al.*,1991].

In order to substantially enhance OODBs with reasoning features, the next step should be the design of a front-end to the DB to validate insertions and updates, with respect to the extended schema description.

2.1 Examples

Let us consider the organizational structure of a company in order to explain the purpose of our constraint validation method. Assume the following: Employees have name and salary. Managers are employees and have a level composed of a qualification and a parameter. Repositories have a denomination, wich can be either a string or a structure composed by a repository name and an address; a repository stocks a set of at least one and at most five materials. Materials are described by a name and a risk. Departments have a denomination (string), and are managed by a manager. Warehouses have all the properties of departments and repositories.

The above description is expressed in our formalism, ODL extended, as follows:

$$\sigma(\texttt{Level}) = [\texttt{qualification}:\texttt{String}, \texttt{parameter}:\texttt{Int}]$$

$$
\begin{aligned}
\sigma(\texttt{Employee}) &= \triangle[\texttt{name: String, salary: Int}]\\
\sigma(\texttt{Manager}) &= \texttt{Employee} \sqcap \triangle[\texttt{level: Level}]\\
\sigma(\texttt{Repository}) &= \triangle[\ \texttt{denomination: String} \sqcup\\
&\qquad [\texttt{rname: String},\\
&\qquad \texttt{address: String}],\\
&\quad \texttt{stock: }\{\texttt{Material}\}_{(1,5)}]\\
\sigma(\texttt{Department}) &= \triangle[\texttt{denomination: String},\\
&\quad \texttt{managed-by: Manager}]\\
\sigma(\texttt{Warehouse}) &= \texttt{Department} \sqcap \texttt{Repository}\\
\sigma(\texttt{Material}) &= \triangle[\texttt{name: String, risk: Int}]
\end{aligned}
$$

Class and type descriptions use the tuple ([ ]) and set ({}) constructors, the latter with a cardinality interval. The $\triangle$ operator enforces a distinction between object classes, preceded by it, and value types. With respect to the formalism in [Bergamaschi and Nebel,1993], the general complement ($\neg$) and the union operator ($\sqcup$), considered in many works on complex object data models [Abiteboul and Kanellakis,1989] [Lecluse and Richard,1989], have been added.

As an example of integrity constraint, let us assert that an employee must earn less than his manager:

$$
\begin{aligned}
\sigma(\texttt{Technician}) = \ & \texttt{Employee} \sqcap\\
& \triangle[\texttt{works-in: Department}] \sqcap\\
& (\triangle\texttt{salary} < \triangle\texttt{works-in.}\\
& \quad \triangle\texttt{managed-by.}\ \triangle\ \texttt{salary}).
\end{aligned}
$$

As a further example, if the class shipment is introduced, the following integrity constraint can be specified on it: for all shipments it must hold that if the risk of the material is greater than 3 then its urgency must be greater than 10, that is: "**for all** $x \in$ Shipment **if** x is of type Shipment $\sqcap$ ($\triangle$item. $\triangle$ risk > 3) **then** x is of type Shipment $\sqcap$ ($\triangle$urgency > 10)". The constraint can be embedded in the class description, obtaining the following type description for Shipment:

$$
\begin{aligned}
&\sigma(\texttt{Shipment}) =\\
&\quad \triangle[\texttt{urgency: Int, item: Material}] \sqcap\\
&\quad (\neg(\triangle\texttt{item.}\ \triangle\ \texttt{risk} > 3)) \sqcup\\
&\quad (\triangle\texttt{urgency} > 10))
\end{aligned}
$$

The coherence checking completion rules, devised by TL researchers, are a suitable starting point also to solve the corresponding problem in OODBs, as shown in [Beneventano *et al.*,1994].

3 Reasoning services in query optimization

The purpose of semantic query optimization is to use semantic knowledge (e.g. integrity constraints) for transforming a query into an *equivalent* one that may be answered more efficiently than the original version.

In database environment, semantic knowledge is usually expressed in terms of IC rules, that is *if then* rules on the attributes of a *database schema* (i.e., roughly a Tbox of a Terminological Knowledge Representation System (TKRS)). Informally, *semantic equivalence* means tha the transformed query has the same answer as th original query on all databases satisfying the IC rules The notion of semantic query optimization for rela tional databases was introduced in the early 80's b King [King,1981a; 1981b]; Hammer and Zdonik [Ham mer and Zdonik,1980] independently developed very sim ilar optimization methods. During the last decade, man efforts have been made to improve this technique and t generalize it to deductive databases [Shenoy and Ozsoyoglu,1989; Siegel *et al.*,1992; Chakravarthy *et al.*,1990] More recently, some efforts have been made to perforr semantic query optimization in OODBs [Chan,1992 Jeusfeld and Staudt,1993; Pang *et al.*,1991; Buchhei *et al.*,1994; Beneventano *et al.*,1993; Bergamaschi and Nebel,1993]. The main point is that OODBs provide a very rich type (class) system able to directly represent a subclass of integrity constraints in the database schema. By exploiting schema information as, for instance, inheritance relations between types (classes), it is possible to perform semantic query optimization.

In order to develop a theory of semantic query optimization, we propose a theoretical framework (in term of *subsumption*) which includes the main query transformation criteria proposed in the database literature and is based on inclusion statements between concepts, recently proposed in [Donini *et al.*,1993]. This new perspective perfectly fits the usual database viewpoint. In fact, actual database schemata are given in terms of *base classes* (i.e. primitive concepts); further knowledge is expressed as IC rules. In particular, structural class descriptions are expressed as rules where the antecedent is a name of the class and the consequent is the class description. More generally, rules allow the expression of integrity constraints with an antecedent and a consequent which are types of the formalism. Since query languages for OODBs are more expressive than our formalism we, following [Buchheit *et al.*,1994], ideally introduce a separation of a query into a *clean* part, that can be represented as a type in our formalism, and a *dirty* part that goes beyond the type system expressiveness. Semantic optimization will be performed only over the clean part of a given query. The clean part of a query, in the following referenced as query, corresponds to the so-called conjunctive queries or single operand queries [Kim,1989] in OODBs and is a *virtual class* (i.e. a defined concept).

The chosen strategy for optimization is the following. Prior to the evaluation of any query, we *compile*, once at all, the given schema (classes + IC rules), giving rise to an enriched schema obtained by adding (*all the new*) *isa relationships* which are logically implied by the original schema. The compilation process is based on the generation of the *semantic expansion* in *canonical form* (i.e. a form which permits to abstract from different syntactical representation of semantically equivalent types) of the schema types. Following the approach of [Shenoy and Ozsoyoglu,1989; Siegel *et al.*,1992] for semantic query expansion, the semantic expansion of a type, say EXP(S) permits to incorporate any possible restriction which is not present in the original type but is *logically implied* by the type and by the schema. EXP(S) is based on the iteration of this simple transformation: if a type implies the antecedent of an IC rule then the consequent of that rule can be added. Logical implications between these types (the type to be expanded and the antecedent of a rule) are evaluated by means of the *subsumption computation* [Brachman and Schmolze,1985; Bergamaschi and Sartori,1992; Bergamaschi and Nebel,1993].[1]

At run time, we add to the compiled schema the query Q and activate the process again for Q, obtaining EXP(Q), with possible new *isa* relationships is obtained. If new *isa* relationships are found, it is possible *to move the query down* in the schema hierarchy. The main points of our optimization strategy are:

1. The *most specialized query* among the equivalent queries EXP(Q) is computed. During the transformation, we compute also, and substitute in the query at each step, the *most specialized classes* satisfying the query.
2. A filtering activity (*constraint removal*) is performed by detecting the *eliminable* factors of a query, that is, the factors logically implied by the query.

.1 Examples

Let us extend the schema of the previous section with the class dangerous-shipment, which has the same structure of shipment. The following integrity constraint can be specified on it: for all shipments it must hold that if the risk of the material is greater than 3 then its urgency must be greater than 10 and it must belong to the class dangerous-shipment. The constraint can be embedded in the class description, obtaining the following type description for Shipment:

$$\begin{aligned}\sigma(\texttt{Shipment}) &= \triangle[\texttt{urgency}:\texttt{Int},\texttt{item}:\texttt{Material}]\\ &\sqcap(\neg(\triangle\texttt{item}.\ \triangle\,\texttt{risk} > 3)) \sqcup\\ &(\texttt{DShipment} \sqcap \triangle\texttt{urgency} > 10))\end{aligned}$$

Let us give two simple query optimization examples related to our schema.

Q: "Select all shipments involving a material with risk greater than 8"

$$Q = \texttt{Shipment} \sqcap (\triangle\texttt{item}.\ \triangle\,\texttt{risk} > 8)$$

From the rule on Shipment, we derive:

$$\begin{aligned}\text{EXP}(Q) &= \texttt{DShipment} \sqcap\\ &(\triangle\texttt{item}.\ \triangle\,\texttt{risk} > 8) \sqcap\\ &(\triangle\texttt{urgency} > 10)\end{aligned}$$

The query is optimized by obtaining the *most specialized generalization* of the classes involved in the query itself.

[1]The subsumption is similar to the *refinement* or *subtyping* adopted in OODBs [Cardelli,1984; Lecluse and Richard,1989].

Furthermore, the factor ($\triangle$urgency > 10) can be added if some advantageous access structure is available for it.

Another rewriting rule proposed in [Shenoy and Ozsoyoglu,1989; Siegel *et al.*,1992] is the *constraint removal*, i.e., removal of implied factors. We formalize constraint removal by subsumption. As an example, consider the query:

Q: "Select all the shipments involving a material with risk greater than 8 and urgency grater than 5":

$$Q = \underbrace{\texttt{Shipment} \sqcap (\triangle\texttt{item}.\ \triangle\,\texttt{risk} > 8)}_{S} \sqcap \underbrace{(\triangle\texttt{urgency} > 5)}_{S'}$$

In the schema with rules S is subsumed by S', as *explo*(S) is subsumed by S' in the schema without rules. Thus, S' can be eliminated from Q.

References

[Abiteboul and Kanellakis, 1989] S. Abiteboul and P. Kanellakis. Object identity as a query language primitive. In *SIGMOD*, pages 159–173. ACM Press, 1989.

[Beneventano *et al.*, 1993] D. Beneventano, S. Bergamaschi, S. Lodi, and C. Sartori. Using subsumption in semantic query optimization. In A. Napoli, editor, *IJCAI Workshop on Object-Based Representation Systems - Chambery, France*, August 1993.

[Beneventano *et al.*, 1994] D. Beneventano, S. Bergamaschi, S. Lodi, and C. Sartori. Reasoning with constraints in database models. In S. Bergamaschi, C. Sartori, and P. Tiberio, editors, *Convegno su Sistemi Evoluti per Basi di Dati*, June 1994.

[Bergamaschi and Nebel, 1992] S. Bergamaschi and B. Nebel. Theoretical foundations of complex object data models. Technical Report 5/91, CNR, Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo, Roma, January 1992.

[Bergamaschi and Nebel, 1993] S. Bergamaschi and B. Nebel. Acquisition and validation of complex object database schemata supporting multiple inheritance. *Applied Intelligence: The International Journal of Artificial Intelligence, Neural Networks and Complex Problem Solving Technologies*, 1993. to appear.

[Bergamaschi and Sartori, 1992] S. Bergamaschi and C. Sartori. On taxonomic reasoning in conceptual design. *ACM Transactions on Database Systems*, 17(3):385–422, September 1992.

[Brachman and Schmolze, 1985] R.J. Brachman and J.G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.

[Buchheit *et al.*, 1994] M. Buchheit, M. A. Jeusfeld, W. Nutt, and M. Staudt. Subsumption between queries to object-oriented database. In *EDBT*, pages 348–353, 1994.

[Cardelli, 1984] L. Cardelli. A semantics of multiple inheritance. In *Semantics of Data Types - Lecture Notes in Computer Science N. 173*, pages 51–67. Springer-Verlag, 1984.

[Chakravarthy *et al.*, 1990] U. S. Chakravarthy, J. Grant, and J. Minker. Logic-based approach to semantic query optimization. *ACM Transactions on Database Systems*, 15(2):162–207, June 1990.

[Chan, 1992] Edward P.F. Chan. Containment and minimization of positive conjunctive queries in oodb's. In *Principles of Database Systems*, pages 202–11. ACM, 1992.

[Donini *et al.*, 1991] F.M. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. In J. Allen, R. Fikes, and E. Sandewall, editors, *KR '91 - 2nd Int. Conf on Principles of Knowledge Representation and Reasoning*, pages 151–162, Cambridge - MA, April 1991. Morgan Kaufmann Publishers, Inc.

[Donini *et al.*, 1993] F.M. Donini, A. Schaerf, and M. Buchheit. Decidable reasoning in terminological knowledge representation systems. In *13th International Joint Conference on Artificial Inteligence*, Chambery - France, September 1993.

[Hammer and Zdonik, 1980] M.M. Hammer and S. B. Zdonik. Knowledge based query processing. In *6th Int. Conf. on Very Large Databases*, pages 137–147, 1980.

[Hollunder and Nutt, 1990] Bernhard Hollunder and Werner Nutt. Subsumption algorithms for concept languages. Technical report, Research Report RR-90-4, Deutsche Forschungszentrum fuer Kuenstliche Intelligenz GmbH, Kaiserslautern, Germany, April 1990.

[Hollunder *et al.*, 1990] Bernhard Hollunder, Werner Nutt, and M. Schmidt-Schauss. Subsumption algorithms for concept languages. In *Proc. ECAI-90, Stockholm, Sweden*, 1990.

Jeusfeld and Staudt, 1993] M. Jeusfeld and M. Staudt. Query optimization in deductive object bases. In Freytag, Maier, and Vossen, editors, *Query Processing for Advanced Database System*. Morgan Kaufmann Publishers, Inc., June 1993.

Kim, 1989] W. Kim. A model of queries for object-oriented database systems. In *Int. Conf. on Very Large Databases*, Amsterdam, Holland, August 1989.

King, 1981a] J. J. King. *Query optimization by semantic reasoning*. PhD thesis, Dept. of Computer Science, Stanford University, Palo Alto, 1981.

King, 1981b] J. J. King. Quist: a system for semantic query optimization in relational databases. In *7th Int. Conf. on Very Large Databases*, pages 510–517, 1981.

Lecluse and Richard, 1989] C. Lecluse and P. Richard. Modelling complex structures in object-oriented databases. In *Symp. on Principles of Database Systems*, pages 362–369, Philadelphia, PA, 1989.

[Pang *et al.*, 1991] H. H. Pang, H. Lu, and B.C. Ooi. An efficient semantic query optimization algorithm. In *Int. Conf. on Data Engineering*, pages 326–335, 1991.

[Schmidt-Schauss and Smolka, 1991] M. Schmidt-Schauss and G. Smolka. Attributive concept descriptions with unions and complements. *Artificial Intelligence*, 48(1), 1991.

[Shenoy and Ozsoyoglu, 1989] S. Shenoy and M. Ozsoyoglu. Design and implementation of a semantic query optimizer. *IEEE Trans. Knowl. and Data Engineering*, 1(3):344–361, September 1989.

[Siegel *et al.*, 1992] M. Siegel, E. Sciore, and S. Salveter. A method for automatic rule derivation to support semantic query optimization. *ACM Transactions on Database Systems*, 17(4):563–600, December 1992.

Semantic Indexing Based on Description Logics

Albrecht Schmiedel
Technische Universität Berlin
atms@cs.tu-berlin.de

Abstract

A method for constructing and maintaining a 'semantic index' using a system based on description logics is described. A persistent index into a large number of objects is built by classifying the objects with respect to a set of indexing concepts and storing the resulting relation between object-ids and most specific indexing concepts on a file. These files can be incrementally updated. The index can be used for efficiently accessing the set of objects matching a query concept. The query is classified, and, based on subsumption and disjointness reasoning with respect to indexing concepts, instances are immediately categorized as hits, misses or candidates with respect to the query. Based on the index only, delayless feedback concerning the cardinality of the query (upper and lower bounds) can be provided during query editing.

1 Introduction

Indexing generally involves an association between some kind of key and the actual target. The key is used to jump directly to a desired piece of information, thereby avoiding an exhaustive search through large sets of candidates. In the context of databases, keys are usually based on the set of values of a particular attribute of the objects to be indexed: if we know the value, we can move directly to the corresponding object(s).

In the following, a description logic (DL) based approach to indexing is sketched which broadens the notion of a key: instead of using attribute values, indexing elements can be arbitrary structured concepts as provided by a terminological language such as BACK (cf. [Hoppe et al.,1993]). Firstly, I will show how the construction of such an index falls out quite naturally from the normal workings of a terminological reasoner, and secondly I will discuss how such an index can be used. This approach, and an experimental implementation, is described in more detail in [Schmiedel,1993].

Basic entities:		
patient	:<	anything.
examination	:<	anything **and** **not**(patient).
observation	:<	anything **and** **not**(patient) **and** **not**(examination).
Basic relations:		
hasExam	:<	**domain**(patient) **and** **range**(examination).
hasItem	:<	**domain**(examination) **and** **range**(observation).
hasValue	:<	**domain**(observation) **and** **range**(number).
Other Primitives:		
hce	:<	examination.
bloodPressure	:<	observation.
bloodPressureSystolic	:<	bloodPressure.
bloodPressureDiastolic	:<	bloodPressure **and** **not**(bloodPressureSystolic).
normal	:<	observation.
abnormal	:<	observation **and** **not**(normal).

Table 1: Primitive concepts and roles

2 Index Construction

In a description logic such as BACK a data base is viewed as a set of distinct objects (also instances or individuals) typically representing domain entities, each of which is associated with a description.

Descriptions are terms built with

- *term-forming operators* such as **and**, **all**, **some**, etc., the logical constants provided by the language,
- *primitive concepts* and *roles* introduced by the user, and
- named *defined concepts* and *roles*.

Table 1 shows some top level primitive concepts roles for building a data base containing descriptions of patients, examinations, and observations made in examinations[1]. Patients are related to examinations via hasExam, and examinations to observations via hasItem.

```
examSomeBpSysAbnorm   :=
  examination and some(hasItem,
                       bloodPressureSystolic and
                       abnormal).

patSomeBpAbnorm       :=
  patient and some(hasExam, examSomeBpAbormal).
```

Table 2: Defined concepts

Table 2 gives two examples for named descriptions (defined concepts) using the primitives. examSomeBpSysAbnorm is an examination which has an item which is an abnormal systolic blood pressure, and patSomeBpAbnorm is a patient which has an examination which has an abnormal blood pressure. Defined concepts are syntactic sugar for abbreviating possibly complex descriptions.

```
bloodPressureSystolic and all(hasValue, gt(140))
      =>  abnormal.
bloodPressureSystolic and all(hasValue, 110..140)
      =>  normal.
bloodPressureSystolic and all(hasValue, lt(110))
      =>  abnormal.
```

Table 3: Rules

Descriptions are also used to define rules, which are expressed as implications between two descriptions. The left hand sides of the rules shown in Table 3 are descriptions of certain sets of observations which are asserted to be in the set of normal or abnormal ones by the description on the right hand side.

Table 4 shows how data is actually entered into the system. The '::' operator is used for asserting that the description on the right hand side is true for the object referenced on the left hand side. Here, there is an object *patient1*, an instance of patient, with two examinations, *hce1* and *hce2*, both instantiating the concept hce. The keyword **closed** indicates that all fillers of the hasExam role are known, i.e. there are only two examinations. The examinations each have exactly two observations, each of which has exactly one numeric value.

Based on this type of input, the system computes

- for concepts the subsumption and disjointness relation, i.e., for each pair of concepts whether one subsumes the other or whether they are disjoint,
- for each individual the set of concepts it is (and is not) an instance of.

For our example containing three kinds of entities, patients, examinations, and observations, the result of this is illustrated in Fig. 1. Concepts (primitives marked with

patient1	::	patient **and** hasExam:closed(*hce1* **and** *hce2*).
hce1	::	hce **and** hasItem:closed(*bpsys1* **and** *bpdia1*).
hce2	::	hce **and** hasItem:closed(*bpsys2* **and** *bpdia2*).
bpsys1	::	bloodPressureSystolic **and** hasValue:130.
bpdia1	::	bloodPressureDiastolic **and** hasValue:90.
bpsys2	::	bloodPressureSystolic **and** hasValue:150.
bpdia2	::	bloodPressureDiastolic **and** hasValue:95.

Table 4: Object descriptions

an asterisk) are related by subsumption links; disjointness has been left out for the sake of simplicity. The individuals at the bottom of the graph, a patient with two examinations, each of which with two observations, are linked to the most specific concepts they instantiate. For example, bpsys2 is classified under the conjunction of bPSystolic, which was explicitly told, and abnormal, due to an abnormality rule as in the example above. This leads to the classification of hce2 under examSomeBpSysAbnorm ('an examination with an abnormal systolic blood pressure') which in turn triggers the classification of patient1 as an instance of patSomeBpSysAbnorm ('a patient with an examination containing an abnormal systolic blood pressure'). Note that hce2 (patient1) was explicitly told to be only an examination (patient); the more specific concepts were derived by the system as a consequence of the role filler relations, the definitions and the rules.

In the following, two properties of description logic based systems not present in mainstream database systems play a crucial role:

- the ability to handle any degree of partial information in conjunction with an open world assumption, and
- the ability to describe individuals with complex concepts and to use these descriptions for query answering.

These two properties make it possible, for example, to remove all the information concerning observations (the shaded part in Fig. 1), but to keep all the information that was derived from observations concerning other entities. Thus, hce2 will still be known to be an instance of examSomeBpSysAbnorm, but the observations and their values from which this was derived will become unknown.

We can now define a set of individuals to be indexed (for example the set of patients), choose a set of indexing concepts (e.g., the concepts specializing patient), and store the relation which associates each indexing concept with the individuals it instantiates. This relation can efficiently be stored in two hashtables: one maps individual names to the set of most specific concepts describing them, and the other maps concept names to the set of individuals they directly instantiate, i.e. those which are not instances of any subconcept. It is also useful to store the associated cardinalities.

[1]For a more detailed description of the BACK language see [Hoppe et al.,1993].

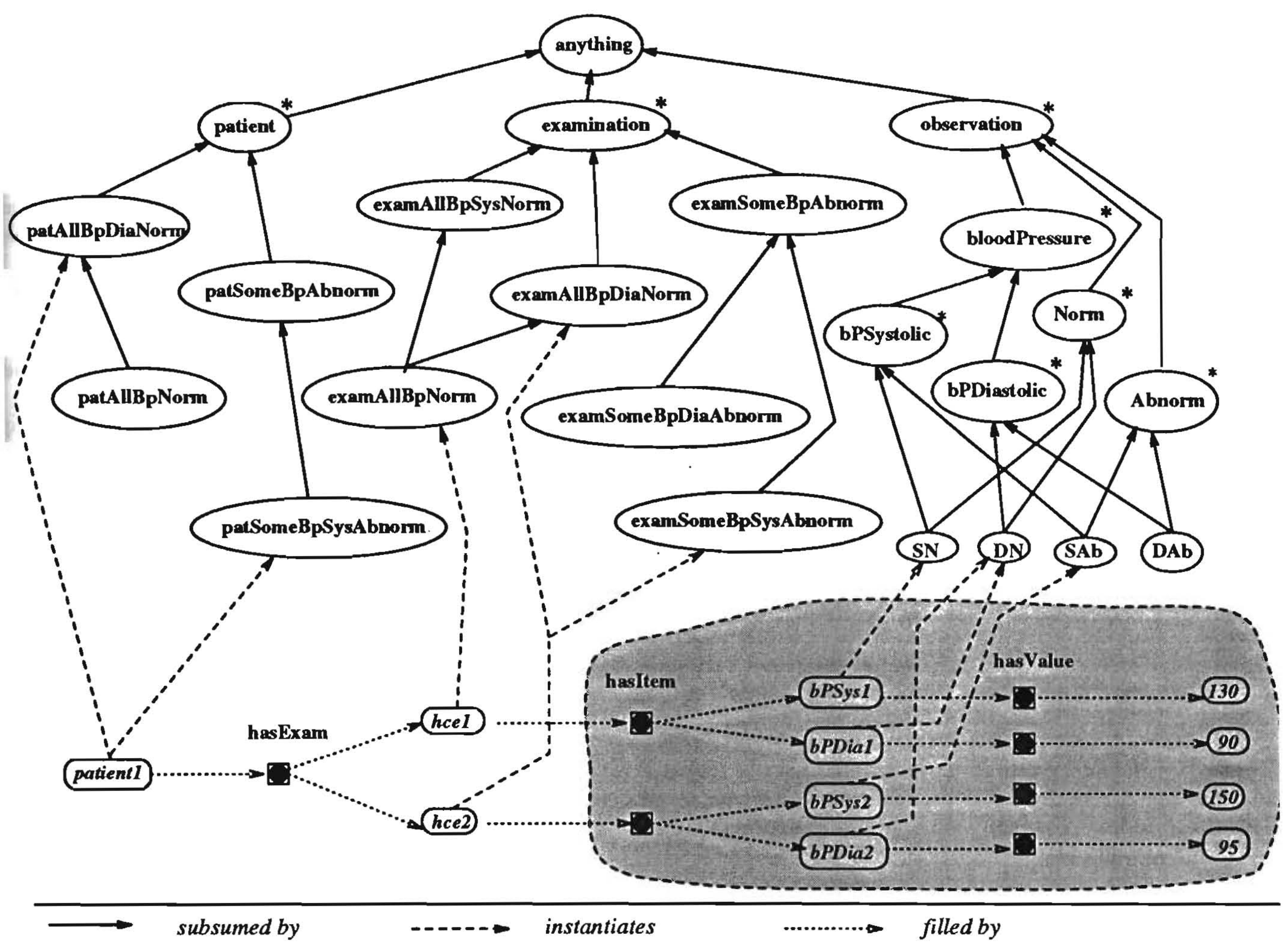

Figure 1: Example KB

3 Using the Index

Based on this stored relation and the original concept definitions, a new knowledge base can be built which contains only the classification of the individuals with respect to the indexing concepts, but lacks the full individual descriptions (see Table 5). It may thus be much smaller than the original KB. Due to the semantic properties mentioned above, it will be ignorant w.r.t to some information contained in the original KB, but it will never produce contradictory answers. This makes it useful as an index.

patient1	::	patSomeBpSysAbnorm and patAllBpDiaNorm.
hce1	::	hce and examAllBpNorm.
hce2	::	hce and examSomeBpSysAbnorm and examAllBpDiaNorm.

Table 5: Abstracted object descriptions

Queries using the index are processed in three distinct phases, each one providing progessively more information at additional costs. The first phase is designed to provide cheap and immediate feedback on the expected cardinality of the result of a query. For this only the cardinalities associated with indexing concepts need to be loaded. The query is classified, and cardinality constraints for it are computed based on the known cardinalities of indexing concepts, and their logical interrelations. Thus, the example query shown in Fig. 2 must have at least 40 instances, since there are two indexing subconcepts the cardinalities of which are added because they can be proved disjoint by the system. Similarly, there is an upper bound of 80 instances for the query, because the indexing superconcept with the least cardinality (100) has an indexing subconcept (20) disjoint from the query. Depending on this cardinality information, the user can either refine his query, specializing or generalizing it as desired, or proceed to the second phase.

The quality of the cardinality feedback depends very much on how close the query is related to already exist-

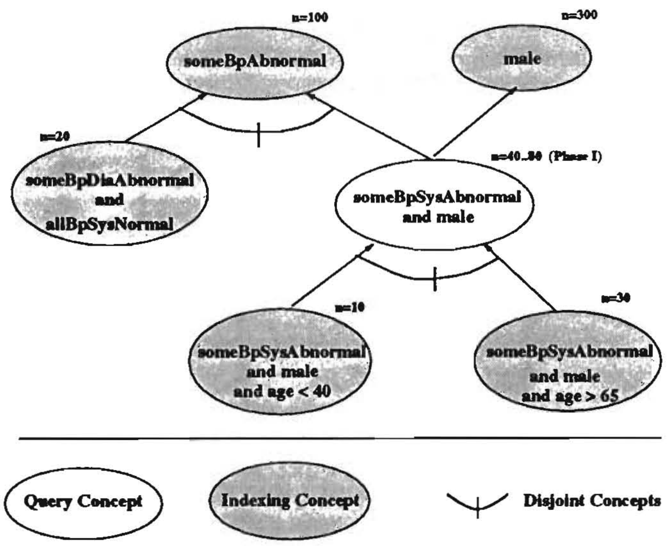
n=100
someBpAbnormal
n=300
male
n=20
someBpDiaAbnormal and allBpSysNormal
n=40..80 (Phase I)
someBpSysAbnormal and male
n=10
someBpSysAbnormal and male and age < 40
n=30
someBpSysAbnormal and male and age > 65
Query Concept
Indexing Concept
Disjoint Concepts

Figure 2: Approximating the cardinality of a query

ing indexing concepts. If we ask for a concept which is equivalent to an indexing one, we get the exact cardinality. If we ask for a concept which is totally unrelated to existing indexing concepts, i.e. there are no subsuming, no subsumed, and no disjoint ones, we will get a lower bound of 0 and an upper bound equal to the number of indexed instances. This means no information at all from the index. Typically, one should get something in between, some partial information.

The second phase additionally utilizes the actual extensions of indexing concepts also stored in the index. This generally results in much better cardinality estimates at the cost of having to load the instances, computing intersections and unions, etc. In case the query is a combination of indexing concepts, its exact extension (and cardinality) can be computed.

Otherwise there is a remaining set of candidates, the individuals for which the query is not known to be either true or false. In this case the index alone does not contain enough information to determine the extension of the query, and the third phase must be entered. For each candidate instance the original description must be accessed and explicitly tested against the query. After this has been done, the user can choose to declare the query as a new indexing concept, making the index more dense at that particular point in the semantic space.

4 Concluding Remarks

This semantic indexing mechanism is crucially dependent on reasoning with descriptions as provided by terminological systems. The indexing elements are potentially complex descriptions logically related by subsumption and disjointness. Note that incomplete algorithms for computing subsumption are not disastrous for indexing: they will simply result in a less informed, suboptimal index.

Compared with standard value-based indexes, this results in the following characteristics:
(1) A semantic index is inherently multidimensional since any combination of properties cast into a DL concept (i.e. an arbitrary query) can serve as an indexing element.
(2) As a structured concept the indexing elements are not just attribute values, but can be based on complex descriptions of related individuals.
(3) A semantic index as a whole is highly adaptable to patterns of usage. Indexing concepts can be added or removed at will, making it very dense and precise w.r.t to interesting sets of individuals, or very sparse in other, less interesting areas.
(4) Since the index is actually a set of partial descriptions for the indexed instances, lots of information (such as cardinality estimates) can be drawn from the index alone without accessing (possibly remote) individual descriptions at all.

These properties may turn out useful for building local information servers which cache information at various levels of completeness, depending on usage patterns.

References

[Hoppe et al., 1993] Hoppe, Th., Kindermann, C., Quantz, J.J., Schmiedel, A., and Fischer, M., BACK V5 Tutorial and Manual. KIT Report 100, Department of Computer Science, Technische Universität Berlin, Berlin, Germany, March 1993.

[Schmiedel, 1993] Schmiedel, A., Persistent Maintenance of Object Descriptions using BACK. KIT Report 112, Department of Computer Science, Technische Universität Berlin, Berlin, Germany, November 1993.

The Problems of Data Modeling in Software Practice

Harald Huber
USU Softwarehaus, Spitalhof, D-71696 Möglingen

Abstract

This paper presents, from the author's perspective, the problems that occur in practice during data modelling. The author's experiences are a result of a considerable number of projects which he carried out in the framework of his consultancy role at USU Softwarehaus in Möglingen (Germany).

These projects concerned the following themes:

- Corporate Datamodelling
- Comparing Datamodels
- Project (Application)- related Data modelling.

In all cases, E/R-notation was the chosen representation-form. From these experiences, the author formed an impression of the problems that occur in practice when defining a data model. These problems have, however, also led to the author's increased interest in knowledge representation, in turn leading to his usage of KR-methods in practice. This has shown itself to be quite effective.

Sections 2 and 3 briefly illustrate the recommendations and the experiences arising from their usage in projects.

1 Datamodelling in Practice - the Problems

Datamodelling was still up until recently the buzzword with which one believed to be able to solve the software crisis. CASE products concentrated on this area, meta-databases were created using a data-modelling process (E/R), and large companies invested millions in order to acquire a corporate data model. Although this trend has subsided a little, the theme in general is still of current interest. What Chen already recognised as an important benefit when presenting the E/R-Model, is today still seen as a key effect of a data model: the representation provides a standard communications basis with which understanding between DP and users is more easily accomplished.

This however, unfortunately seems to hold just for small data models. For larger areas of attention, the methodology starts to become ineffective, and no longer provides the overview required. Apparently, there are just a few 'gurus' who are able to create a complete complex data model. Often this data model quickly decreases in value, as soon as that person leaves the company. Director's offices exist in which the corporate data model is hanging up behind glass - however, this is regrettably the only place in which the data model is noticed or paid heed to.

The following problems, among others, have been recognised:

1.1 Low Expressivness of a Data Model in E/R-Form

During the analysis phase, many of the organisation's interdependencies and processes are identified. These are subsequently, to use the relativly inadequate language of the E/R-Model, abstracted and generalised. This often requires a change in terminology; in other words a unified, formal language is compulsory. What many authors (e.g. Vetter) see as an advantage of data modelling (exactly this coming-into-being of a corporate, unified terminology) often turns out to be a disadvantage: the terms used in the data model are not understood by the user departments. To make matters worse, these terms are mostly held in commentary form (if at all). Also the cross-reference of the new, unified terminology to the terms used in the departments is, in most cases, not documented at all. This makes understanding the data Model afterwards very difficult (see 1.6).

1.2 The Development of the Data Model is not Documented

A model undergoes many changes during the modelling phase. Requirements, ideas and practical examples from the user department contribute to the permanent extension and improvement of the model. Consequently, variations in the Business Processes are represented by generalisations, and classes (e.g. Subtypes) are created in order to denote similar 'things' in the model. The problem is that in nearly every case the documentation of this development is missing, i.e. reasons and reflections on which the model's structures and elements are founded will be lost after a short time. This results in difficulties if the model is changed due to further development or new requirements.

1.3 The Ideal Model is Developed

Although the user departments are consulted during the analysis phase, in practice one is often left with the impression that the DP-staff's ideal model is developed. This trend is strengthened by the fact that the creation of the data model requires a change in terminology and a certain generalisation (see 1.1). The user department staff usually see themselves therefore as incapable of effectively contradicting the 'high-flying' ideas of their DP-colleagues. The result is mostly a model which gives the impression of absolute perfection, but which neither makes the day-to-day business its priority, nor is so understandable that the user-departments can work with it.

1.4 Weak Methodology of the Developer

The possibilities of graphical development tools and the resulting excellent representation often disguises the weakness in the developer's understanding of the methodology. In this way, entities such as 'Total Turnover' and 'Turnover per Customer' can actually be modelled. Most developers tend to model concepts as entities, instead of taking the expressive character of entities in general into account. (This behaviour is also to be seen in a completely different form, where the developers come from a very technical background and mean tables or files instead of entities. Let's leave this point for the moment - it will be touched upon again in point 1.8). Another weakness is the missing experience in interview technique. Very often, the interviewer's question is formulated like "And how can I show that in E/R?" instead of "Which process stati occur in practice - let's leave E/R out of it for the moment?".

1.5 Exceptional Cases Become the Core of Model

Since the daily business of a company is in most cases comparatively simple to represent, Data Modelling projects often rush headlong into attempting to build every case imaginable into the model, as if the knowledge for treating each of these cases really had to be documented. The effect of this is that the models quickly become too detailed and difficult to understand - so much so that the user-departments, who really should judge the model's 'correctness' - more or less make this judgement on the basis of 'gut-feeling'. If they see well-known terms and recognise relationships between them that are held to be necessary, then the model seems to them to be complete and correct, even though in many cases they cannot follow it through to the lowest detail.

1.6 Assumption of Understanding

The relatively low expressiveness of an E/R-model all-too-seldom leads to recognition of this 'inadequacy in meaning'. Often this inadequacy is compensated for by an overkill of interpretation, which means that the model, which really should be the basis for a common understanding, often becomes a problem of understanding. The real world is then no longer the topic of discussion (in which the question of understanding certainly arises) - rather, one discusses entities and relationships, whose meanings are comparatively trivial and thereby are a matter of interpretation and alteration when trying to understand the 'fact-content' behind them.

1.7 Missionary character of DP

DP tends to over-estimate itself in many organisations. This inaccurate estimation doesn't particularly affect the importance of DP for the organisation's success so much (This could certainly be the subject of heated discussion both in theory and in an organisation's leadership). This obviously false judgement of one's own situation affects the implementation of standards and norms much more. The standardisation of terminology (mentioned under point 1.1) which the DP-Department carries out during data modelling is here an excellent example. It implies however, that 0.5 - 2 % of the company can dictate the terminology of the remaining employees. This over-estimation, together with the problem outlined in 1.3, means that DP doesn't model according to requirements, rather use their own ideas as basis for the 'ideal model'.

1.8 Too much Technical Thinking

Since most modellers come from a technical background (e.g. Application Development), they find it extremely difficult to ignore this technical knowledge when modelling. In the past, many cases occured where performance considerations were incorporated into the E/R-Diagram. The problem, however, goes much deeper than that. Most modellers cannot imagine any way to represent the characteristics of entities other than with attributes. Two entities with the same attributes are hastily made one, without considering that they express a classification on a logical level.

2 Suggestion for a Solution

The approach this solution takes is basically to use to best effect the developer's (and the user department's) tendency to express himself in concepts. This means that in the initial Data modelling phase, one creates a model of these concepts in the form of a semantic network. It's quite possible that other, more modern, representations are more suitable for this task. However, since the author has his roots in the Data modelling world, moving towards semantic networks was the easier way for him to come to terms with knowledge representation methodology.

The author makes the following suggestion for the development of a Data Model (relational or E/R):

- **Creation of various semantic networks for parts of total area of attention.** These semantic networks contain all statements-of-fact and requirements issued by the user-department, in order not to let any information fall by the wayside. Representative questions from the user-departments can also be noted here.
- **Consolidation of the various networks.** The aforementioned networks are consolidated. Synonyms and homonyms are not 'cleaned up'.

This means that there is no unification of language necessary. Rather, the individual terms are cross-referenced to one another.

- **Generation of an E/R-model.** The user department requirements can be generated using all of the semantic networks. The E/R-Model can be worked on using this basis and can be tested using the requirements represented in the networks. This model is then the basis for the creation of the relational model.

To make the consolidation of several semantic networks developed by several developers possible, a standardized, unified representation of the networks is suggested. This means that only two types of associations are allowed, represented by lines; all other relevant concepts and associations appear as nodes. This restriction forces the unified representation necessary for the consolidation. The following two types of associations are allowed to be represented by lines:

- Type 1, which describes just the extension of a concept
- Type 2, which defines the intention.

Note that these associations are not defined by their symbolic meaning, rather by a relatively formal context. This has the advantage that the semantics of these associations are not interpretation-dependent.

3 Experiences from Projects

The suggested methodology solves the aformentioned problems. The interviewers interview-technique is positively affected, because his annotation is not subject to the restrictions of the E/R-model. The developement of the model is also documented, whereby the supplementary information discovered during the analysis phase, is held in the model.

- The tendency to strong generalisation and 'artificial terms' is restricted - the terminology can still be understood by the user department.
- The selection process (what's an entity?) can be re-created and checked in reviews. The user-department staff can concentrate more on the model's content, thereby avoiding 'ideal structures'.
- The cabability to consolidate the various parts means that the model in the user-department stays relatively small.
- There are, however, also disadvantages.

If one uses a strictly formal representation, as suggested above, the model becomes difficult to grasp in its entirety. Furthermore, during the interviews, the interviewer requires considerable concentration in order to express the facts in the required manner. In practice, however, during the interview a somewhat less formal representation is chosen, which is subsequently translated into a formal model.

Note that the principle elements of the model are concepts, and not other elements such as entities, even if a less formal notation is used.

OLSEN: An Object-Oriented Formalism for Information and Decision System Design

Ramzi Guetari, Frédéric Piard[1], Bettina Schweyer[2]

LLP/CESALP 41 Avenue de la Plaine
BP. 806 - 74016 Annecy Cedex - FRANCE
Tel : (+33) 50.66.60.80 - Fax : (+33) 50.66.60.20
email : guetari|piard|schweyer@esia.univ-savoie.fr
[1]CIFRE contract with ANRT and Pôle Productique Rhône-Alpes
[2]CIFRE contract with ANRT and ARM Conseil

1.□Introduction

The Object oriented model has spread widely within programming languages during the last years. The principles of this model have had a great influence on analysis and design techniques. However no existing method is able to manage the whole analysis-specification-design-implementation cycle, preserving the homogeneity of the model used in different stages and the coherence by passing from one stage to the following.
We think that the global management of the life cycle cannot be solved, with the existing state of knowledge, by one unique miraculous method, which could adapt to every kind of application. We think on the contrary that the problem should be treated by a panel of methods dedicated to a particular domain.
For this reason we have developed the OLYMPIOS model at the LLP-CESALP laboratory. This model covers the life cycle of every application in the field of Information and Decision Systems for Manufacturing Firms. OLYMPIOS uses algebraic techniques, transformation rules and a predefined entity organisation to propose an original approach for object oriented design of information and decision system.

2.□OLYMPIOS Model Concepts.

The information processed in an enterprise, which we call industrial information, is a complex datum. An information and decision system (IDS) must take this complexity into account. We propose to represent industrial information through four main facets :

- data, describing the different entities handled by the IDS and the actions that they can perform or can be subjected to ;
- temporal properties of the different kinds of processes (including traceability of information) ;
- organisation, considered through information flows;
- economic facet, which describes the means of performance evaluation in relation to enterprise environment and objectives.

The OLYMPIOS model [Beauchêne□93] [BHP□93] [BHS□93] covers the different stages of such a system life cycle and proposes original solutions for its analysis, specification, design and realisation. OLYMPIOS describes activities, taking into account the assigned objectives and the resources availability. The basic modelling elements are□:

- an industrial information database, where products, resources, machines,... are described.
- Consumer-Supplier Information Systems (**CSIS**). A CSIS stands for an "atom" of organisation. It is a generalisation of the customer-supplier exchange relationship to every couple of actors in the enterprise (men, machines, software). Every CSIS is associated to an objective, transforms resources and emits a satisfaction level.
- an Objective Management System (**OMS**), whose role is to create a graph from expressed objectives, where every node is an objective associated to a CSIS.
- a Resource Management System (**RMS**), in charge of the product and resource management and sharing.
- an activation system (**AS**), producing actions plans to organise processes, taking into account the application, temporal constraints, and communications/synchronisation between CSIS.

3.□The IDS Life Cycle

The OLYMPIOS model covers the different stages of the IDS life-cycle (Fig.□1). We use an algebraic approach for the four facets of industrial information so as to obtain a coherent (i.e. sufficiently complete and consistent) specification. The design stage enables us to design the information system from specification and by analysing the "existing" system of the enterprise and its objectives. The result of this stage is a representation of the IDS using structured entities. The OLYMPIOS model introduces the uniformity of the model used from specification up to design. It uses tools proving the coherence of the system in the specification step and maintaining this coherence by automating the translation from one stage to another.

3.1.□Analysis Stage

In the analysis stage, the relevant information for the data, the temporal, the organisational and the economic facets is collected.
The result of the **data** facet analysis consists in the description of the data handled (resources etc.) in the system to design and, for each datum, the set of operations that can be realised (data dictionary). This static description can be translated into a finite state automaton in which every node represents a state of the datum in question and every edge an operation which produces a new state.

The analysis of the **organisational** aspects of the manufacturing firm results in a set of interactions between the different agents of the enterprise in the form of exchange relationships. By interviewing each of these agents we enumerate, on the one hand, the exchange relationships in which he is consumer, i.e. follows a certain objective by asking for satisfaction of the respective needs, and on the other hand, we identify the relationships in which he is supplier and performs a certain function. For each of these functions (which we would like to call basic operation) he enumerates the resources necessary for realising this operation and the algorithm he follows to obtain the wanted resource.
Thus, this interview gives us information about

- objectives and their decomposition,
- identification of the possible suppliers for the realisation of a given objective,
- the basic operations that can be performed and knowledge about how to execute the operations and which resources are needed.

Starting from this information, we can establish a knowledge base of the different ways to decompose objectives and a knowledge base for the needed resources for each basic operation. These knowledge bases will help us, in addition to the predefined structure of such an exchange relationship, to define the enterprise organisation.

The analysis of the **temporal** facet provides a dynamic description of the system. It enables us to describe the temporal behaviour of different agents and resources of the system and their interactions. For this part of the analysis, a method close to natural language is being developed which will allow a user-friendly way of describing temporal rules.
From this analysis we also obtain a description which we call *realization programs*. These programs contain the description of the CSIS functionning and of the operations which are not formally describable.

As far as the **economic** facet is concerned, we are actually working on an interview structure including fuzzy logic in order to acquire the information necessary for evaluating the system's performance.

3.2□Specification Stage

3.2.1□Data Specification

The data facet corresponds to the IDS functional and structural aspects, and aims at representing the technical and technological data. We use Algebraic Specifications of Abstract data Types (ASAT) [Guttag□78] [Jacquenet□86] [Liskov□87] so as to have efficient and simple proof techniques at our disposal. An ASAT enables us to express an entity behaviour in a high level formalism. For a given entity, an ASAT is a triple $<\Omega,\Sigma,A>$, where Ω is a set of domains containing the domain of the entity values, Σ is a set of operations on the entity, and A is a set of equations (axioms and preconditions) on these operations, which determines the entities behaviour and the relationships between them. ASAT are automatically constructed from the entities automata, which are the result of the analysis stage. This automatic construction is realised by the algorithms [Nkongo□90] developed in our laboratory.

3.2.2□Organization Specification

It starts from the analysis of the "existing system", which results (inter alia) in the identification of actors and their functions and objectives. Specifying organisation consists in formally expressing identified objectives (in the "triple" form), and in constructing their associated CSIS from standard parametrized ASAT of organisation [Beauchêne□93]. Simultaneously, one must elaborate the different graphs of objectives.

3.2.3□Temporal Specification

The specification of the industrial information temporal facet uses a synchronous process algebra, directly derived from the SCCS calculus of R. Milner [Piard□93]. We specify four kinds of processes with this language :

1- chronological and event-based clocks, essential to specify synchronisation and to measure temporal intervals;
2- behaviours of data facet entities, which are not completely determined by ASAT axioms;
3- behaviours of CSIS;
4- activation plans, elaborated by the activation system from graphs of objectives and resources to schedule the CSIS.

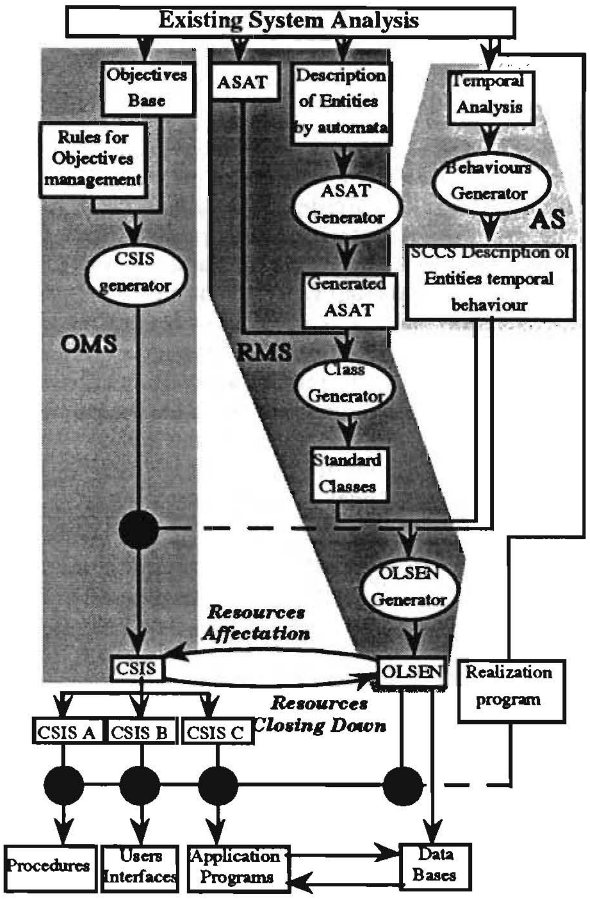

Fig. 1. The Analysis - Specification - Design Cycle in the OLYMPIOS Model

3.2.4.□Economic Specification

This facet cannot be specified independently of data and organisation. Indeed it is shared between them, and the most important part is included in the organisation facet. Works are still going on to sharpen the economic view of OLYMPIOS on the information system (with the help of performance indicators, fuzzy logic and project-based management approach).

3.3.□Design Stage

The OLYMPIOS model, in its design stage, is based on the class model. This model was extended in order to allow to take all industrial information features into account, in particular real time ones. The result of the design stage is an organisation of entities independent of possible target programming languages□: OLSEN (**OL**ympios **S**tructured **EN**tity).

An OLSEN [Guetari□94] is composed of a "class" part and another part called "scenario" which indicates the interactions with its environment. The difference between an OLSEN and a classical object is the scenario which describes the temporal behaviour generally missing in the standard class model. The OLSEN model is a "design object".

In this paper, we present only the specification and design of Activation System (AS part) and Resource Management System (RMS). The Objective Management System is the subject of a publication to come.

4.□The Transition from the Analysis to the Specification Stage

This stage consists in describing data types using finite state automata. We must first insist on the fact that every entity cannot be described by an automaton. Only if it has successive states and if it is concerned by actions passing from one state to another can it be described by an automaton. We do not use the automata as a specification tool but as a tool allowing us to shape the evolution of some kind of data type over a set of states. In this kind of automata, each transition represents an operation changing the entity's state and each node represents one state of the entity. The automata may have many transitions corresponding to the same operation, however, each state is unique. A particular state called "starting state" must always exist. It corresponds to the extremity of the transition which stands for the operation creating the type of interest□(TI).

The entities described by automata are distinguishable by the successive states that they can have. The order in which different states are occupied is well defined. The graph of state changing is oriented and has a starting state from which we can observe the evolution of the entity. This graph allows us to distinguish the constructor operations using a single method. The transitions corresponding to these operations have extremity nodes which can be reached from the starting state by only one path of the graph. The construction of axioms is done in two steps□: the construction of left parts of axioms and the construction of right parts of axioms, as it is shown below□:

The construction of left parts of axioms :

The construction of axioms left parts consists of building the following sets :

- $CT = \{c(y^*), c \in C\}$
- $OT = \{o(x, y^*), o \in O, x \in CT\}$
- $ST = \{s(x, y^*), s \in S, x \in CT\}$

OT and ST contain the left parts of specification axioms. Axioms which define the semantic of the abstract data type have their left parts in the OT set and axioms which shows the simplification of terms of $T(\Omega,\Sigma)$ have their left parts in the ST set.

The construction of right parts of axioms :

The graph of states, whose every node is a state of entities of TI type, and whose every transition is an operation, provides□:

1- $\Omega = \{TI, STATES\}$, $STATES = \{E1,E2,E3,...\}$
2- $\Sigma = \{state, \sigma1, \sigma2, \sigma3, ..., \sigma n\} = O+C+S$, $T = S + C = \{\sigma1, \sigma2, \sigma3, ..., \sigma n\}$ is the set of operations which create or transform the values of TI (represented in the automata by transitions), O={state} contains a single observer.
3- Left parts of axioms by the building of AC,AO,AT from O,C et T.
4- Right parts (y) of axioms in the form $state(c(x^*)) = y$, where $c \in C$, and y is the expression of the name of the node extremity of the path represented by $c(x^*)$ from the starting state. If there are many of these paths then the y term will be expressed in the form **if**...**then**...**else** ...
5- Right parts (y) of axioms in the form $s(c(x^*)) = y$, where $s \in S$ is a convertible operation and y corresponds to the canonical form of the state extremity of the path $c(x^*)$, i.e. the expression of the shortest path between the starting state and the state extremity of the path represented by the expression $c(x^*)$. In other terms, these axioms are represented in the automata by simple circular paths. If there are many of these paths then the y term will be expressed in the form **if**...**then**...**else** ...
6- Preconditions related to the state of arguments (membership of TI) of each operation, which are expressed by the restrictions on the domain of this operation before its execution. These restrictions are issued from the state origin of the arc representing the operation.

5.□The Transition from the Specification to the Design Stage

The transition from the specification stage (ASAT and SCCS) to the design stage is done automatically in two steps. The first step consists in taking the ASAT one by one and translating each one into a standard class. The second step is a global one and permits the organization of the communication between the obtained classes. The benefit of this automation is the preservation of the coherence obtained in the specification stage.

5.1.□The Standard Class Generation

The class attributes and methods are generated from the ASAT operations. This is done using the following rules. We note an operation : $\sigma : \Omega_1 \rightarrow \Omega_2$. Ω_1 is the set of domains and Ω_2 is the set of codomains. "TI" is the data type that we specify. We distinguish three kinds of operations :

- Case 1 : $\sigma : \Omega_1 \rightarrow \Omega_2 / TI \notin \Omega_1$ and $\Omega_2 = \{TI\}$. This kind of operation corresponds to a particular constructor. For each constructor, we generate a method "*New*" with parameters of type Ω_1.
- Case 2 : $\sigma : \Omega_1 \rightarrow \Omega_2 / \Omega_1 = \{TI\}$ and $\Omega_2 = \{\omega \neq TI\}$. This kind of operation corresponds to observers. The class structure is obtained from these observers. For each observer we generate an attribute of type Ω_2 and a method to access it.
- Case 3 : $\sigma : \Omega_1 \rightarrow \Omega_2 / TI \in \Omega_1$ and $TI \in \Omega_2$. This case corresponds to a general one. For each operation of this kind we generate a method with in parameters of type $\omega \in \Omega_1 / \omega \neq TI$ and out of parameters of type $\omega \in \Omega_2 / \omega \neq TI$.

The scenario of an OLSEN is issued from SCCS formulae. An SCCS formula contains several deterministic parts. Each part provides one script in the OLSEN scenario. The scenario generation is done in three steps : the first two provide the declarative part of a scenario, the third one provides the dynamic part. For each OLSEN, we determine the determinist parts of the corresponding BEHAVIOUR (separated by a "sum" operator). For each part, we execute the following three steps□:

- **Event Detection**. This step permits the detection and declaration of the different kinds of events. The type of each event is deduced from the SCCS syntax. A communicational event appears in at least two BEHAVIOURs, once preceded by the delay operator δ, and once without this operator. An environmental event is identified by the existence of a clock emitting this event. An event is conditional if its complementary event appears at least once in a BEHAVIOUR. When all events are declared, we proceed to the unification of the communicational events. This unification is based on the *observational equivalence* [Austry□84] and consists of giving the same name to two synchronously successive events in a SCCS formula.
- **Identification of the Set of Suppliers**. For each communicational event, we define its receiving OLSENs whose BEHAVIOURs contain this event, preceded by the delay operator δ. Any OLSEN responding to this event by applying one of its methods must be added to the suppliers list of the treated OLSEN.
- **Script Generation**. A script is generated for each determinist part. Each event described in the formula is replaced by one or several simultaneous dispatches of messages. The receivers of these messages are the suppliers defined in step 2.

6.□The Transition from the Design to the Realization Stage

This transition is based on the realization programs which we have obtained in the analysis stage.
The OLSEN formalism helps us to generate data bases on the realization stage. The application programs are obtained through the OLSEN, the realization programs and the CSIS organization.
If we target object-oriented data bases in the realization stage, we have to use the OLSEN and the realization programs. In this case, each class part of an OLSEN is directly translated into a data base object and the scenario part is used for the data access in the application programs. The realization programs allow us to implement the methods of the data base objects.
If the data bases are not object-oriented, only the structure of the OLSEN interferes for the realization of these data bases. In a relational data base, for example, the OLSEN structure is used for the table creation. The inheritance relationship is eliminated in these data bases and replaced by the result of merging the structures of a super-class and the sub-classes.

In the realization stage we can obtain three different types of CSIS translations: automatic CSIS where the actors perform totally automated processes, semi-automatic CSIS where one of the two actors performs an automated task or the manual CSIS where both actors perform manual tasks.

The first type of CSIS with the realization programs and the scenarii allow us to obtain the application programs. These programs will act upon the data bases with the classical operations like add, modify and delete. These interactions with the data base are performed through message sending between the data base objects in the case of an object-oriented data base or through primitives which are the result of the OLSEN behaviour in the case of non object-oriented data bases.
The semi-automatic CSIS form the interactions between a user and a process. These CSIS lead towards the implementation of user interfaces and external views which restrict the data base access according to the user's rights.
The manual CSIS finally, allow us to realize the manual procedure for which the automation would be too expensive.

7.□Conclusion

The OLYMPIOS model provides the means to analyse and specify coherently an industrial information and decision system. It allows then to design the specified IDS by preserving the coherence obtained in the specification stage by using algebraic techniques. The continuity and uniformity claimed by the Olympios model is the result of two factors□:

- the use of algebraic tools to specify all the components of an IDS like the data facet, the organization facet or the temporal facet,
- the use of ASAT to specify data and Objects to design them.

This care of continuity and uniformity has lead us to develop algorithms (and parts of a future CASE-Tool) to automatically generate a coherent OLSEN organisation from the analysis. Our objective is to generate a maximum of code for applications.

References

[Austry□84] AUSTRY D., BOUDOL G., Algèbres de processus et synchronisation, TCS 30(1) 1984

[Beauchêne 93] D. Beauchêne. L'information industrielle : définition et spécification. PhD thesis, University of Savoie. December 1993.

[BHP 93] D.Beauchêne, A.Haurat, F.Piard. Une méthode de spécification de l'information industrielle par types abstraits algébriques. Proceedings of ICO'93, 4-7 May 1993, Montreal Canada.

[BHS 93] D.Beauchêne, A.Haurat, B.Schweyer. Designing an information system for a manufacturing enterprise under the aspect of a CIM approach : the model OLYMPIOS. Proceedings of APMS'93, 28-30 September. 1993, Athens Greece.

[Guetari 94] R. Guetari. and F. Piard. From the Specification to the Design of an Industrial Information System: the Olympios Model. Accepted in the 1994 IEEE Conference on Systems Man and Cybernetics. San Antonio - Texas October 2 - 5 1994.

[Guttag 78] J.V. Guttag and J.J. Horning. The algebraic specification of Abstract Data Type. Acta Informatica. 1978 Vol 10. P 27-52.

[Jacquenet 86] J.P. Jacquenet, P.Lescanne. La réécriture. Techniques et Sciences Informatiques 1986. Vol 5 N° 6. p. 433-452.

[Liskov 87] B. Liskov. Data Abstraction and hierarchy. OOPSLA'87 Addendum to the proceedings. 1987.

[Nkongo 90] T. Nkongo. Spécification algébrique de types abstraits pour le modèle Olympios. DEA report Ingénierie Informatique of INSA Lyon. September 1990.

[Piard 93] F. Piard, C. Braesch - Application du calcul SCCS de Milner à la spécification de processus informationnels par types abstraits algébriques dans une entreprise manufacturière. Real Time Systems Conference, Paris 1993.

Frames, Objects and Relations: Three Semantic Levels for Knowledge Base Systems*

M. C. Norrie[1], U. Reimer[2], P. Lippuner[2], M. Rys[1], H.-J. Schek[1]

[1]Dept. of Computer Science, Swiss Federal Institute of Technology (ETH), CH-8092 Zürich, Switzerland
{norrie, rys, schek}@inf.ethz.ch
[2]Swiss Life, Informatik-Forschungsgruppe, CH-8022 Zürich, Switzerland
{reimer, lippuner}@swssai.uu.ch

Abstract

We propose an architecture for large-scale knowledge base systems based on database technologies and the three levels of semantic construct - frames, objects and relations. The intermediate object level retains the structural semantics of the frame level and is therefore beneficial in bridging the semantic gap between the frame and relational levels and enabling the use of semantic information in query optimisation. Specifically, we outline how this approach has been adopted in the hybrid knowledge base system, HYWIBAS.

1 Introduction

For knowledge base systems to be effective for large-scale applications, it is essential that they support efficient retrieval and update operations on large, shared knowledge bases. Database system research has focussed on issues of performance and concurrent access to large data sets and we wish to exploit the resulting technologies for the storage and management of knowledge bases.

Past research in this area has tended to use relational systems for the persistent storage of knowledge bases. While this strategy does meet the requirements of controlled data sharing, the large semantic gap between the knowledge representation structures and the relational structures makes it more difficult to utilise data semantics in query optimisation. We therefore adopt a two-level mapping. The first level maps a frame-based knowledge representation model, FRM [Rei 89; RL 94], to an object data model, COCOON [SLR+92], which retains much of the data semantics. The second level then maps COCOON to a relational system which is used as a simple storage system with query and update strategies controlled primarily at the object system level.

Here, we present an overview of how this approach is utilised in the (hybrid) knowledge base system HYWIBAS [RRS+93] (the hybrid aspects are not elaborated here). Section 2 introduces the three level architecture and discusses its merits. The mappings from FRM to COCOON and from COCOON to a relational system are discussed in Sections 3 and 4, respectively. Some remarks on the current status of HYWIBAS and future research plans are given in Section 5.

2 Three Level Architecture

Knowledge base systems research has tended to concentrate on issues of semantic expressiveness and inference mechanisms. For knowledge base systems to be used for large-scale applications, issues of efficient update and retrieval operations on large, shared knowledge bases must be addressed. Database systems research has focussed on these very issues in dealing with efficient, concurrent access to large data sets. The question then becomes one of how best to exploit database technologies in knowledge base systems.

Relational database technologies now have established and well-understood mechanisms to support efficient access to large sets of value tuples with techniques for concurrency control and recovery. The problem of mapping a knowledge model directly to a relational storage structure is the large semantic gap due to the lack of semantic expressiveness of the relational data model. As described in [RS 89], this can in part be overcome by mapping a knowledge model to a nested relational model which can represent complex structures directly. However, the nested relational model does not support notions of type inheritance and concept hierarchies which are fundamental to knowledge models such as FRM.

Object data models have been developed to support notions of semantic data modelling and thereby increase the semantic expressiveness of the data model. They have constructs to represent both complex structures and relationships between structures - including those that arise in classification structures, often known as *isa* hierarchies. In addition, a number of object data models have been proposed that specify operations over collections of objects in terms of an object algebra. By mapping the frame knowledge model to an object data model rather than to a relational data model, the semantic gap is reduced. However, object-oriented database management systems are not yet as well established as

* The work presented here was supported by the Swiss Priority Programme in Computer Science under Grant No. 5003-34347.

relational database management systems in terms of efficient processing of set-oriented retrieval and update operations and supported transaction mechanisms. For this reason, we choose to map our object data model to a relational storage system. This mapping is specifically tailored to support the retrieval and update patterns initiated by the frame model. As a result, we have a three level architecture as indicated in Figure 1.

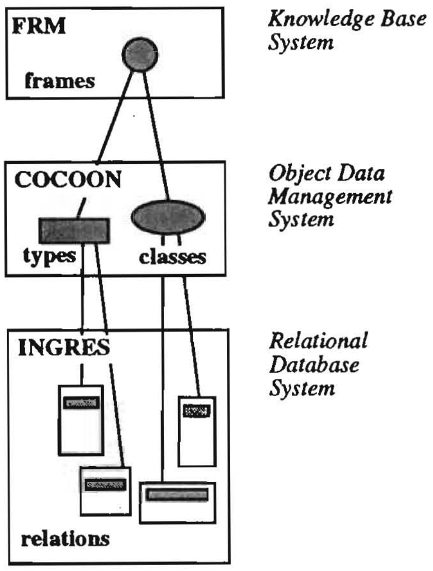

Figure 1: Three Level Architecture

The knowledge model FRM is mapped to the object data model COCOON which in turn is mapped to a relational system. At present, we use the relational data base management system INGRES, but the mapping can easily be altered for other relational systems.

3 From Frames to Objects

A discussion of the differences between the knowledge representation and semantic data modelling approaches is given in [Bor 91]. One of the main differences often quoted is that database models tend to be prescriptive rather than descriptive. Thus the underlying assumption is that the database provides a complete, current and consistent description of the application domain; any attempt to input data which is not consistent with the database model will be rejected. Knowledge models tend to be descriptive and it is quite acceptable that the model may have to be revised according to new information received into the system. This is most clearly visible in a knowledge-based system with some learning capabilities (see e.g. [Mor 91]).

A further general distinction between data models and knowledge representation languages is the fact that data models have a much clearer separation between intensional and extensional information. Intensional information is given by a database schema which is relatively stable and thus plays a predominant role in determining efficient storage, retrieval and update strategies for operations on extensional data.

Ideally, for the support of knowledge base systems, we wish to have the latter property of database models (i.e. efficiency) but not necessarily the former (i.e. being prescriptive). In this respect the COCOON object data model is a good candidate for the support of the frame model FRM.

In this paper we consider only a subset of FRM which corresponds to the common frame constructs: slots, slot entries, and cardinality restrictions. For example,

Skilled-Person $\doteq$
  (**and** *Person*
    (**all** *has-skills Skill*)
    (**exist** *has-skills Rare-Skill*)
    (**atleast** *has-skills* 3))

defines a frame class *Skilled-Person* as a subclass of *Person* with the slot *has-skills* that represents the relationship *has-skills* to the class *Skill*. The slot requires at least 3 values at an associated class instance; one of those entries must be an instance of the class *Rare-Skill*.

COCOON has a strong influence from both semantic data models and knowledge representation languages (especially KL-ONE [BS 85]) in terms of semantic expressiveness. It supports not only complex object structures but also rich classification structures and high-level operations over collections of objects. As a result, the semantic expressiveness of COCOON is at a similar level to that of FRM with the main difference between the two models stemming from the fact that FRM supports more specialised inference mechanisms. In some sense COCOON may be considered as lying somewhere between the prescriptive and descriptive paradigms. A COCOON class represents a semantic grouping of objects and may have an associated predicate condition. For example

define class *Youngsters* : *person* **some** *Persons*
  where *age* < 30;

defines a class *Youngsters* which contains objects of type *person* and is a subclass of *Persons*; further there is an associated predicate condition that specifies that its members should be less than 30 years old. The object type *person* declares what functions are applicable to an object of that type and may look like the following

define type *person* = *age* : **integer**,
  name : **string**, *has-skills* : **set-of** *skills*;

A formal mapping from frame structures to object structures and from query operations on frame knowledge bases to object bases has been defined and implemented. While concept class descriptions in FRM are based on a single representation structure – the frame, COCOON has two basic representation structures – the type and the class. Types describe what properties and relationships to other objects an object can have whereas, as stated above, classes deal with semantic groupings of objects. Only a small number of the frame constructs for concept class descriptions can be mapped to COCOON

FRM concept class description:

```
Comp_Delivery ≐ (and (all supplier Company)
                     (exist supplier Computer_Company)
                     (all recipient Company Person)
                     (atmost recipient 1)
                     (all ispart Workstation)
                     (all price [0,100])
                     (atmost price 1))
```

Corresponding COCOON type definition:

```
define type comp_delivery  =  supplier : set-of object,
                              recipient : object,
                              ispart : set-of object,
                              price : integer;
```

Corresponding COCOON class definition:

```
define class Comp_Delivery  :  comp_delivery
                     where  supplier ⊆ Company and
                            ∅ ≠ (supplier ∩ Computer_Company) and
                            recipient ⊆ (Company ∪ Person) and
                            ispart ⊆ Workstation and
                            ∅ = select [(i < 0) or (i > 100)] (i : price);
```

Figure 2: Example of Mapping an FRM Concept Class Description to COCOON Types and Classes

type definitions but all of them to COCOON class definitions. As a consequence, frames of FRM are mapped to some combination of types and classes in COCOON. To increase the possibilities for compile-time optimisation, we designed the mapping such that as much information as possible is provided on the type level.

Figure 2 shows an example of mapping an FRM concept class description to COCOON types and classes. In a first step the object type *comp_delivery* is derived from the FRM class *Comp_Delivery* such that for every **all** construct (i.e. for every slot) we have a function with the same name. In case of a slot with a maximal cardinality of 1 the function is single-valued, otherwise set-valued. In a second step the COCOON class *Comp_Delivery* of type *comp_delivery* is generated from the frame class *Comp_Delivery*. With the type reference we ensure that the class will contain only objects with the right functions being applicable. With the associated class predicate we cover the remaining features of the FRM concept class description. As a result, the COCOON class defines the same necessary and sufficient conditions on class membership as the frame class does. Note that the three object-valued functions in the type definition *comp_delivery* are all of type **object**. This is because providing more specialised function ranges (e.g. *supplier* : **set-of** *Company*) would not lead to a simpler class predicate. As this would not reduce the amount of dynamic type checking necessary we decided to keep the mapping to the type level simple and to map always to object-valued functions of type **object**. For details see [LNR+94].

The establishment of the mapping from frames to types and classes has also proved useful in providing an insight into the similarities and differences in the fundamental concepts of terminological models such as FRM and object data models.

In knowledge base systems a query for objects with certain properties is usually established as a class description. The result of the query is all the objects subsumed by that class so that in this case query evaluation amounts to inferencing. To support such queries on our COCOON-based FRM we have specified a second mapping that transforms a frame class description to be interpreted as a query into an equivalent expression of the COCOON object algebra (cf. example in Figure 3). This algebra expression is then evaluated on the COCOON object base derived from the original frame knowledge base. At that point query optimisation techniques, which are highly developed in the database area, can be employed. We hope that this will lead us to a query processing that is much more efficient than evaluating a query frame by the inference mechanism of FRM.

4 From Objects to Relations

In mapping an object data model onto a relational system, there are many choices to make concerning both the representation of objects and also of classes. For example, all the properties of an object may be stored together in a single relation or split over several relations. In the former case, there are problems of how to represent multi-valued properties. In the latter case, several join operations may be required to reconstruct an object.

With the representation of classes, the choices arise because an object may belong to many classes and the prime decision is whether to store an object only with its most specific class – or to store it in all classes – or to have some form of compromise between the two extremes. Further, some COCOON classes have associated predicates which specify necessary and sufficient conditions for membership of that class. In such a case, there is no need to store

Query Frame:

```
(and (all supplier Company)
     (all recipient Company)
     (exist recipient Insurance_Company)
     (all product
         (and Workstation
              (all has-cpu Sparc) (atleast has-cpu 2))))
```

Corresponding Algebra Expression:

$\mathbf{select}[supplier(o_1) \subseteq Company](o_1 : Objects) \cap$
$\mathbf{select}[recipient(o_1) \subseteq Company](o_1 : Objects) \cap$
$\mathbf{select}[recipient(o_1) \cap Insurance\_Company \neq \emptyset](o_1 : Objects) \cap$
$\mathbf{select}[product(o_1) \subseteq$
$\quad \mathbf{select}[has\text{-}cpu(o_2) \subseteq Sparc] \cap \mathbf{select}[\#(has\text{-}cpu(o_2)) \geq 2](o_2 : Workstation)](o_1 : Objects)$

Figure 3: Example of Mapping a Query Frame to an Object Algebra Expression (still to be Optimised)*

the class explicitly as it can be derived at access time. The trade-off here is between fast access to explicitly stored classes versus high update overheads if data is replicated unnecessarily.

In our mapping of COCOON onto a relational storage system, we employ extensive replication to minimise retrieval costs. For example, all classes are represented explicitly even those which could be specified in terms of a query expression (view) over other classes. Since an object may belong to many classes, an object representation may be replicated in several relations. The penalty associated with such an approach of massive replication is the cost of update operations; a single update operation on a specific object may require updates on a large number of relations involved in the representation of that object.

The problem then becomes one of how to speed up the time for updates. This is achieved by implementing the update operation as a number of simpler update operations which can be executed in parallel. The exploitation of intra-transaction parallelism together with multi-level transactions is a key technique towards such improved performance [WS 92].

We are currently evaluating the above approach to see under what conditions the overheads of parallelisation are compensated by the corresponding speed-up of the operations. In the future, we shall investigate dynamic methods of mapping the object data model COCOON to relational systems such that good performance is attained under various retrieval and update patterns (which finally stem from specific retrieval and update operations on the knowledge base system).

5 Conclusions

In the HYWIBAS project, we are using database technologies to support large, shared knowledge bases. We employ a three level architecture corresponding to three semantic levels of frames, objects and relations. The introduction of the object level is beneficial in reducing the semantic gap between the frame level and the relational level and enabling the utilisation of structural semantic information for query and update processing. The mapping from the object level to the relational level allows the use of well-established, efficient mechanisms for data storage, data access, data sharing and recovery under failure.

At present, we have implemented mappings for structural information from the frame model, FRM, to the object model, COCOON and from COCOON to the multiprocessor relational database system, INGRES. We also have a mapping from frame query classes to COCOON algebra. Moreover, there are some early results on the parallelisation of update operations over a COCOON database represented in INGRES [Rys 94]. Currently, we are working on the mapping of the remaining operational components and on the mapping of frame class instances to objects.

* For reasons of readability we have slightly simplified the algebra expression: The select statements should apply to classes of objects for which the functions referred to are really defined, rather than operating on the most general class *Objects*. This requires an additional meta-schema query, which we have omitted.

References

[Bor 91] A. Borgida, "Knowledge Representation, Semantic Modeling: Similarities and Differences", In *Entity-Relationship Approach: The Core of Conceptual Modelling*, ed. H. Kangassalo, North-Holland, 1991, pp. 1-24.

[BS 85] R. J. Brachman and J. G. Schmolze, "An overview of the KL-ONE knowledge representation system", *Cognitive Science*, Vol. 9, No. 2, 1985, pp. 171-216.

[LNR+94] P. Lippuner, M. Norrie, U, Reimer and M. Rys, "Mapping a Frame Model, FRM, to an Object Data Model, COCOON", HYWIBAS Working Paper, 1994. (in preparation)

[Mor 91] K. Morik, "Underlying Assumptions of Knowledge Acquisition and Machine Learning", *Knowledge Acquisition*, Vol. 3, 1991, pp. 137-156.

[Rei 89] U. Reimer, "FRM: Ein Frame-Repräsentationsmodell und seine formale Semantik. Zur Integration von Datenbank- und

Wissenrepräsentationsansätzen", Springer, 1989.

[RL 94] U. Reimer, P. Lippuner, "Syntax und Semantik von FRM", Working Paper, 1994, Informatik-Forschungsgruppe, Swiss Life, CH-8022 Zurich).

[RRS+93] U. Reimer, M. Rys, H.-J. Schek and R. Marti, "Datenbankbasierung eines Frame-Modells: Abbildung auf ein Objektmodell und effiziente Unterstützung komplexer Operationen", Beitrag zum Workshop "Verwaltung und Verarbeitung von strukturierten Objekten" während der KI 93, (also available as Technical Report 5/93, Informatik-Forschungsgruppe, Swiss Life, CH-8022 Zurich).

[RS 89] U. Reimer and H.-J. Schek, "A Frame-Based Knowledge Representation Model and its Mapping to Nested Relations", *Data and Knowledge Engineering*, Vol. 4, No. 4, 1989, pp. 321-352.

[Rys 94] M. Rys, "Parallelising Generic Update Operations in COCOON Using Multilevel Transactions". (in preparation)

[SLR+92] M. H. Scholl, C. Laasch, C. Rich, H.-J. Schek and M. Tresch, "The COCOON Object Model", Technical Report 211, Dept of Computer Scince, ETH Zurich, CH-8092 Zurich, Switzerland.

[WS 92] G. Weikum, H.-J. Schek, "Concepts and Applications of Multilevel Transactions and Open Nested Transactions", In *Database Transaction Models for Advanced Applications*, ed. A.K. Elmagarmid, Morgan Kaufmann, 1992.

Uniformly Querying Knowledge Bases and Data Bases

Paolo Bresciani
IRST, I-38050 Trento Povo, TN, Italy
bresciani@irst.it

Abstract

Present KL-ONE-like knowledge base management systems (KBMS), whilst offering highly structured description languages aside efficient concepts classification, have limited capability to manage large amounts of individuals. Data base management systems (DBMS) can, instead, manage large amounts of data efficiently, but give scarce formalism to organize them in a structured way, and to reason with them.

This paper shows how assertional knowledge of KBMS and data of DBMS can be uniformly accessed. The query answering capability of an arbitrary KBMS is augmented with the possibility of accessing external databases (DB) as a supplemental source of extensional knowledge.

The techniques presented in this paper can be easily adapted to several sources of information. From a knowledge acquisition perspective, we believe that they can be usefully applied in all those applications where several sources of informations are available independently from the knowledge bases.

1 Introduction

The two basic components of a KBMS of the KL-ONE family are the terminological box (TBox) and the assertional box (ABox). One of the tradeoff of these KBMS is between the expressiveness of the description languages characterizing their TBox and the inefficiency in managing large amounts of data in the ABox, even when they have a quite schematic form and their classification is completely a priori given. DBMS, instead, are suited to manage data efficiently, with little concern about their dimension, but their formalism for organizing them in a structured way is quite absent, as it is the capability to infer new information from the existing ones.

Here we propose to cope with both KBMS and DBMS together, using them in an integrated way to manage with several kinds of information. Of course a uniform way to retrieve information from a mixed KBMS/DBMS is needed.

In the present paper it is shown how assertional knowledge of KBMS and data of DBMS can be uniformly accessed. A technique to tightly couple KBMS with DBMS [Borgida and Brachman,1993] is described. As in [Devanbu,1993; Borgida and Brachman,1993] we let primitive concepts and relations in a KB correspond respectively to unary and binary tables/views in a DB. Unlike [Devanbu,1993; Borgida and Brachman,1993] we provide a *tight coupling* between KBMS and DBMS, i.e., a *on demand* access to the DB, instead of a *loose coupling*, that requires a pre-loading of the data from the DB into the KB. In this way we obtain the following advantages:

- more complex queries than simply asking for the instances of concepts can be done; just as an example, in our system queries like $C(x) \wedge R(x,y) \wedge D(y)$ can be made.
- no memory space is wasted in the KBMS to keep descriptions of DBMS instances.
- answers are given on the basis of the current state of the KB and the DB.
- no periodical updating of the KB with new or modified data from the DB is needed.

Basically, in our system the query answering capability of an arbitrary KBMS[1] is augmented with the possibility of accessing external information as a supplemental source of extensional knowledge. In particular a database is seen as an extension of the KBMS ABox.

2 DBox as Extension of the ABox

The ABox is the component of a DBMS where assertions about single individuals are stated. In the present paper we describe how the ABox can be extended with an external source of extensional data. We call this extension a 'DBox'. In the following, we adopt the notation of [Nebel,1990] and call a set of term descriptions (concepts and roles) a *terminology* $\mathcal{T}$[2], and a set of individual assertions a *world*

[1] Even if we implemented the ideas presented here as an extension of LOOM [MacGregor,1991], they can be easily applied to any KL-ONE-like KBMS system with a first-order-logic query-language.

[2] An important task of a KBMS is to organize the terms in a taxonomy accordingly with a specialization relation, i.e., to *classify* them; in the following, we often use $\mathcal{T}$ to denote just the set of atomic terms appearing in $\mathcal{T}$, and consider them correctly classified in the taxonomy on the basis of their definitions.

description $\mathcal{W}$[3]; we say also that the set of data expressed in a DBox constitutes a *data base* $\mathcal{D}$. Assuming that two complete query answering functions separately exist, for both the ABox and the DBox, a *knowledge base* $\mathcal{KB} = \langle \mathcal{T}, \mathcal{W}, \mathcal{D} \rangle$ can be defined in such a way that a uniform query function, based on the two answering functions, can be implemented. We do not require any special capability from the DBox, except the one of (quickly) retrieving lists of tuples of items satisfying requested conditions. These conditions are of the kind of *being in a class* – i.e., belonging to a unary table/view – or *being in relation with other items* – i.e., belonging to a binary table/view – and logical combinations of these, as it can be, for example, expressed in SQL. Since our implementation relies, in fact, on a DBMS with SQL, we assume that $\mathcal{D}$ is somehow represented by means of a relational database, and queries to the DBox can be done in SQL. Therefore in the following we will refer to tables/views – or simply tables – as they usually are intended in relational databases, and to the query answering function of the DBox as to those of SQL.[4]

¿From the point of view of users of KBMS, our experience [Bresciani,1992] suggests that, in realistic applications, knowledge bases not only can be complex, but can also involve a large number of individuals: often most of them are already completely and suitably described in some database. We faced several times, in the past, the task of transferring data from these databases to our knowledge bases. Using the techniques described in the present article it is, of course, much more recommended to *link* the databases to the knowledge bases. Using a DBox paradigm we obtain the advantage of reducing to the minimum the effort of transferring data and, moreover, they are automatically kept updated as far as the linked databases are. But also when data must be collected *ex-novo* it can be convenient[5] to manage most of them by means of a DBMS.

3 Coupling

As mentioned, in our $\mathcal{KB}$, $\mathcal{D}$ is assumed to contain no structural knowledge, but just raw data, and is not supposed to have any inferential power. The single instances are, therefore, already placed in the right tables. That is, speaking in KR terms, they are completely a-priori *realized* under the right concept. This corresponds to having each table of $\mathcal{D}$ associated with a *primitive* term[6] of $\mathcal{T}$. We will show how mixed (KBMS/DBMS) queries can be answered in a coherent way, but, to this extent, we need to *couple* the terminology $\mathcal{T}$ in $\mathcal{KB}$ with the data base $\mathcal{D}$. This coupling consists in the association of some particular terms of $\mathcal{T}$ with tables, in the DB representing $\mathcal{D}$, where the extension of these terms are to be found.

For sake of simplicity we adopt, next, some restrictions on the form of $\mathcal{KB}$, even if, as it will be noted later, they can be, at least in part, released. Given $\mathcal{KB} = \langle \mathcal{T}, \mathcal{W}, \mathcal{D} \rangle$, we assume that the following conditions are satisfied:

- **non intermediate db extension**: every $\mathcal{D}$ individual must be realized under a leaf term in $\mathcal{T}$, i.e., a term in $\mathcal{T}$ specialized by no other terms in $\mathcal{T}$.
- **homogeneous extension**: for each leaf term of $\mathcal{T}$ its associated instances are either all in $\mathcal{W}$ or all in $\mathcal{D}$.
- **db isolation**: all the leaf terms of $\mathcal{T}$ whose instances are in $\mathcal{D}$ are primitive and are not used in any other term definition in $\mathcal{T}$.

Consider that it is not difficult to design $\mathcal{KB}$ in such a way that a primitive term is introduced in $\mathcal{T}$ for each class of individuals present in $\mathcal{D}$: by this the **homogeneous extension** hypothesis can always be satisfied. The **db isolation** and the **non intermediate db extension** conditions reflect the hypothesis that $\mathcal{D}$ is just a flat collection of unstructured tables of records of data, without any reasoning capability.

Under these assumptions, all the information needed to correctly drive the query mechanism is the association of those terms in $\mathcal{T}$ whose extensions are in $\mathcal{D}$ with the corresponding tables in the DB. To this extent it is enough to know this association for those terms that are leaf, for the **non intermediate db extension** condition above. Therefore, we assume that a partial mapping $PM : \mathcal{PT} \rightarrow DBtable$ is given, where $\mathcal{PT}$ is the set of primitive terms in $\mathcal{T}$, and $DBtable$ the set of tables in the DB. So, we can define the *marking* function $M : \mathcal{T} \rightarrow 2^{DBtable}$, s.t. $M(t) = \{PM(x) \mid x \in subs(t) \text{ and } PM(x) \text{ is defined}\}$, where $subs(t)$ is the set of all the terms classified under t in $\mathcal{T}$ (including t). The marking function gives the (possibly empty) set of tables necessary to retrieve all the instances (pairs) of a given concept (relation). Therefore, it is an important part of our $\mathcal{KB}$, whose definition, to be more precise, has now to be rephrased: $\mathcal{KB} = \langle \mathcal{T}, \mathcal{W}, \mathcal{D}, PM \rangle$.

4 Query Answering

We are now ready to describe the task of answering a query. Here we will assume that a *query* to $\mathcal{KB} = \langle \mathcal{T}, \mathcal{W}, \mathcal{D}, PM \rangle$ is an expression of the kind $\lambda\overline{x}.P_1 \wedge \ldots \wedge P_n$, where $P_1, \ldots, P_n$ are predicates of the form $C(x)$ or $R(x,y)$, where C and R are respectively a concept and a relation in $\mathcal{T}$ and each of x and y appears in $\overline{x} = \langle x_1, \ldots, x_m \rangle$ or is an instance in $\mathcal{W} \cup \mathcal{D}$. As a first, informal example, let us consider the case in which all the $P_1, \ldots, P_n$ can be managed by the DBox only, that is: $M(t) \neq \emptyset$ for each $t \in subs(P_1) \cup \ldots \cup subs(P_n)$.

[3]By $\mathcal{W}$ we denote also the set of individuals described in $\mathcal{W}$.

[4]Of course, the external source of information where the DBox searches data can be of any kind, provided only that it can be accessed via a first-order-logic query language.

[5]if not necessary, considering that most KBMS cannot cope with more than some hundreds or few thousands of individuals.

[6]A primitive term is a term whose definition gives only necessary but not sufficient conditions; individuals cannot be realized under one of these terms unless it is not explicitly asserted that they belong to it or to a more specific term in the taxonomy.

4.1 Translating Queries into SQL

When each predicate in a query $q = \lambda\overline{x}.P_1 \wedge \ldots \wedge P_n$ can be made correspond to a set of tables in the DB, where the answers have to be found, it can be translated into an equivalent SQL query. Of course, the sets of tables can be easily found via the marking function M. At this point we have just to cope with the union set of tables $\{T_1, \ldots, T_h\}$ and their bindings via the variables in $\overline{x}$. For simplicity, let us suppose that the tables returned by M are composed by one column in the case of a concept (let it be called `left`), and two in the case of a relation (let them be called `left` and `right`). The SQL translation is of the kind:

```
SELECT DISTINCT  select-body
FROM             from-body
WHERE            where-body
```

where the *select-body* is a list of column names of the kind $M(P_{x_i})$`.left` or $M(P_{x_i})$`.right`, one for each variable x_i in $\overline{x}$, according to the fact that the variable x_i appears for the first time in the predicate P_{x_i} in the first place[7] or in the second place, respectively. The *from-body* is the list of all the tables involved – i.e., all the $M(P_i)$. The *where-body* is a list of SQL `where`-conditions of the kind `field2=field1` or `field2=constant`, where the first form has to be used for each variable that is used more than once, each time it is reused, and the second form occurs for each use of constants. In both the forms `field2` is a selector similar to those in *select-body*, corresponding to positions in the query where the variable is further used or where the constant appears, respectively; `field1` corresponds to the first occurence of the variable.

4.2 The General Case

In general answering, a query is more complex and requires the merging of results from the DBMS and the KBMS. *Answering* a query in $\mathcal{KB}$ means finding a set $\{\overline{x}^1, \ldots, \overline{x}^m\}$ of tuples of instances s.t., for each tuple $\overline{x}^i$, $\lambda\overline{x}.(P_1 \wedge \ldots \wedge P_n)[\overline{x}^i]$ holds in $\mathcal{KB}$. We call such tuples *answers* of the query and the set of all of them its *answer set*.

Due to the definition of answer of a query, it is obvious that, in order to avoid the generation of huge answer sets, free variables should not be used, i.e., each variable appearing in $\overline{x}$ must appear also in the query body (i.e., the part at the right of the dot). Indeed, we adopt a stronger restriction, because the former one still allows for some undesired situations. Let us consider, for example, the query: $\lambda\langle x, y, z\rangle.A(x) \wedge R(x, y) \wedge C(z)$. All the variables appear in the body, but, nevertheless, the answer set of the query can be unreasonably large, due to the fact that all the answers of the sub-query $\lambda\langle x, y\rangle.A(x) \wedge R(x, y)$ have to be *combined* with all the answers of the sub-query $\lambda\langle z\rangle.C(z)$. We say that such a query is *unconnected*. More in general, we say that a query is *unconnected* when it can be split into two or more sub-queries s.t. all the variables appearing in each of them does not appear in any other. We call these sub-queries *clusters*. It is obvious that the relevant result of answering an unconnected query is equivalent to the union of the single results of separately answering the clusters, in the sense that all the information is included in it. But, if we consider the formal definition of answer, we must consider the fact that the overall result must contain tuples longer than those resulting by submitting the single clusters; to obtain all the tuples satisfying the definition of answer the single answers have to be combined by a sort of *Cartesian product*. More exactly, if, after having reordered the variables, un unconected query is written as $\lambda\overline{x}.\varphi_1(\overline{x}_1) \wedge \ldots \wedge \varphi_n(\overline{x}_n)$ – where $\overline{x}$ is the concatenation of the other vectors ($\overline{x} = \overline{x}_1 \cdot \cdots \cdot \overline{x}_n$), and $\varphi_1(\overline{x}_1), \ldots, \varphi_n(\overline{x}_n)$ corresponds to the single clusters – and given that the asnwers sets of a generic cluster $\lambda\overline{x}_i.\varphi_i(\overline{x}_i)$ is $S_i = \{\overline{I}_i^1, \ldots, \overline{I}_i^{j_i}\}$, the answer set of the whole query is $S = \{\overline{I}_1^{j_1} \cdot \cdots \cdot \overline{I}_n^{j_n} \mid \overline{I}_1^{j_1} \in S_1, \ldots, \overline{I}_n^{j_n} \in S_n\}$.

The case of a *connected* (i.e., non unconnected) query $\lambda\overline{x}.\varphi(\overline{y})$ with unbound variables can be reduced to the case of an unconnected query $\lambda\overline{x}.\varphi(\overline{y}) \wedge T(\overline{z})$, where $\overline{z} = \langle z_1, \ldots, z_k\rangle$ contains all the variables appearing in $\overline{x}$ but not in $\overline{y}$, and $T(\overline{z}) = top(z_1) \wedge \ldots \wedge top(z_k)$, where *top* correspond to the most generic concept in $\mathcal{T}$.

It is now clear that unconnected queries and queries with unbound variables may have unreasonably large answer sets, without giving any further capability to the system. Therefore, we consider only connected queries with only bound variables.

To afford the answering of a query we need to split it into sub-queries that can be answered by the two specialized query answering functions of the KBMS and the DBMS. To this extent we need, as a first step, to ***mark*** all the possible atomic predicates, corresponding to the terms in $\mathcal{T}$, and say that a term P is:

- DB-marked iff for each $t \in subs(P) \cap \mathcal{PT}$ $PM(t)$ is defined .
- KB-marked iff for each $t \in subs(P) \cap \mathcal{PT}$, $PM(t)$ is undefined.
- Mixed-marked otherwise.

These three markings reflect the fact that the instances (pairs) of P are all in $\mathcal{W}$, all in $\mathcal{D}$, or part in $\mathcal{W}$ and part in $\mathcal{D}$, respectively. The strategy for answering to a query is based on this information. Let us, first, observe that it is easy to answer to an atomic query where the predicate is a KB-marked or a DB-marked term. In the first case it is enough to submit it to the KBMS. In the second it is enough to translate the query in a SQL equivalent, as shown above, and submit it to the associated DB. Moreover, if the query is not atomic, but made up by atomic subexpression all with the same marking, the same strategy is applied. More difficult is the case of queries with Mixed-marked predicates. Even the atomic case is quite difficult; it is necessary to transform the atomic query into the (possibly non atomic) one whose predicates correspond to all the leaf terms that specialize the only term in the original atomic query, proceed as before, and collect all the results.

Let us now consider a generic non atomic query: $\lambda\overline{x}.P_1^{KB} \wedge \ldots \wedge P_{l_{KB}}^{KB} \wedge P_1^{DB} \wedge \ldots \wedge P_{l_{DB}}^{DB} \wedge P_1^{M} \wedge \ldots \wedge P_{l_M}^{M}$

[7] or the only one in the case of concept.

where the P_i^{KB} corresponds to the KB-marked terms, the P_i^{DB} to the DB-marked terms, and the P_i^M to the Mixed-marked terms. The query can be split in the sub-queries: $q^{KB} = \lambda\overline{x}.P_1^{KB} \wedge \ldots \wedge P_{l_{KB}}^{KB}$, $q^{DB} = \lambda\overline{x}.P_1^{DB} \wedge \ldots \wedge P_{l_{DB}}^{DB}$, and $q^M = \lambda\overline{x}.P_1^M \wedge \ldots \wedge P_{l_M}^M$.

4.3 The Algorithms

As we said, the sub-queries q^{KB}, q^{DB}, q^M can be easily processed. The only difficulty is that some of the variables in $\overline{x}$ could be unbound in a sub-query. In this case, as shown before, the answer sets have to be completed, that is, the unbound variables should be made correspond to each instance in $\mathcal{KB}$, for all the found answers, by all the possible combinations. But, in this way, huge answer sets are generated, as in the following sketch of the query-answering algorithm:

1 split the query as sketched above into q^{KB}, q^{DB} and q^M.

2 submit q^{KB} to KBMS, q^{DB} to SQL (after translation) and transform each of the atomic sub-queries q_i^M of q^M into a set of atomic queries corresponding to the leaf terms in $\mathcal{T}$ that specialize q_i^M; submit them to the specific retrievers.

3 collect all the answers respectively in the answer sets $AS_{\overline{x}_{KB}}^{KB}$, $AS_{\overline{x}_{DB}}^{DB}$, and $AS_{\overline{x}_M}^M$, and *complete* them with the whole domain in the place of unbound variables, as mentioned above, generating $AS_{\overline{x}}^{KB}$, $AS_{\overline{x}}^{DB}$, and $AS_{\overline{x}}^M$.

4 the overall answer set is just $AS_{\overline{x}}^{KB} \cap AS_{\overline{x}}^{DB} \cap AS_{\overline{x}}^M$.

Of course this first algorithm is widely space wasting. Moreover, in step 3 it is not clearly stated how to collect the answers of the sub-queries q_i^M. We try here to shortly describe this operation and to show how the completions of $AS_{\overline{x}_{KB}}^{KB}$, $AS_{\overline{x}_{DB}}^{DB}$, and $AS_{\overline{x}_M}^M$ in step 3, and their following intersection in step 4, can be obtained more efficiently. To solve these problems, from step 3 ahead a compact representation for $AS_{\overline{x}}^{KB}$, $AS_{\overline{x}}^{DB}$, and $AS_{\overline{x}}^M$ is needed. Let a generic partial answer set be written as $AS_{\overline{y}}$, where the variables of the original complete variable tuple $\overline{x}$ missing in $\overline{y}$ are, $x_{p_1}, \ldots, x_{p_k}$. Its completion can be represented in a compact way with $AS_{\overline{x}} = \bigcup_{\overline{I} \in AS_{\overline{y}}} \{\overline{I}^\star\}$, where $\overline{I}^\star$ are equivalent to $\overline{I}$ except that are lengthened by filling the k missing positions $p_1, \ldots, p_k$ with any marker, e.g., a star '$\star$'. The star stands for all the individuals in $\mathcal{KB}$. Using this representation for the completion in step 3, it is now easy to rephrase step 4 of the algorithm as a merging operation. In fact answer sets $AS_{\overline{x}}^{KB}$, $AS_{\overline{x}}^{DB}$, and $AS_{\overline{x}}^M$ can be *merged* into a single answer set as follow:

4.1 let `result-list=`$\{AS_{\overline{x}}^{KB}, AS_{\overline{x}}^{DB}, AS_{\overline{x}}^M\}$

4.2 choose two answer sets, AS_1 and AS_2, in `result-list`, where answers have at least one common position filled by individuals, i.e., not $\star$.[8]

4.3 merge AS_1 and AS_2 by collecting only those answers in AS_1 where each non-$\star$ filled position is filled by the same individual or by $\star$ in some answers in AS_2, and replace in the collected answers each $\star$ with the individuals in the corresponding position in all the matching answers of AS_2

4.4 replace AS_1 and AS_2 in `result-list` with their merging computed in step 4.3

4.5 REPEAT from step 4.2 UNTIL only one item is left in `result-list`.

4.6 RETURN the only item left in `result-list`.

Now it is easy to explain how to collect the answers of the sub-queries q_i^M of step 2. It is enough, for each $q_i^M \in \{q_1^M \ldots q_h^M\}$, to collect all the answers of all its *descendant queries*, and complete these answer sets generating $AS_{\overline{x},1}^M, \ldots, AS_{\overline{x},h}^M$, as described above; it is now clear that, in the above algorithm for step 4, step 4.1 has to be so rephrased:

4.1-bis

let
`result-list=`$\{AS_{\overline{x}}^{KB}, AS_{\overline{x}}^{DB}, AS_{\overline{x},1}^M, \ldots, AS_{\overline{x},h}^M\}$.

The resulting algorithm, composed by steps 1, 2, 3 (modified as shown), 4.1-bis, and 4.2 to 4.6 has been implemented. In our system the KBMS currently in use is LOOM [MacGregor,1991], and the database query language is SQL, but, as mentioned, also other systems could be easily used.

5 Conclusion and Future Developments

We have shown how a third component, a DBox – allowing for the extensional data to be distributed among the ABox and the DBox – can be added to the traditional TBox/ABox architecture of KBMS. By means of the DBox is possible to couple the KBMS with, for example, a DBMS, and use both the systems to uniformly answering queries to knowledge bases realized by this extended paradigm. The presented query language has some restrictions, and some constraints have been imposed to the form of the knowledge bases. To overcome these limitations, some extensions of the present work can be proposed.

5.1 Constraints on the Form of $\mathcal{KB}$

In section 3 we assumed that some constraints should be imposed on the form of $\mathcal{KB}$. Indeed they can be in part released, even if this more general approach would require a deeper discussion and a reformulation of the algorithms. Here we try to give a very short account on possible developments in this direction. First, consider the **homogeneous extension** condition. It is important because it allows to make the search of the answers simpler, giving the basis for a neat separation between KB-marked, DB-marked, and Mixed-marked predicates[9]. But it

[8]Such two sets do always exist, otherwise the query would be unconnected, while we assumed to deal only with connected queries.

[9]and giving also the way to decompose the Mixed-marked predicates in sets of KB-marked and DB-marked ones.

is even more important when considered in conjunction with the **db isolation** condition. In fact we can easily cope with leaf terms having instances from both $\mathcal{W}$ and $\mathcal{D}$ by submitting the corresponding subqueries to both the specialized retrieving functions, and then proceeding with the merging as usual. But, allowing this ambiguity would make more complex the formulation of the **db isolation** condition, that could become:

- **db isolation**: all the leaf terms of $\mathcal{T}$ whose instances are even only in part in $\mathcal{D}$ are primitive and are not used in any other term definition in $\mathcal{T}$.

Indeed we can, at least in part, give up also with this condition. In fact, while keeping the fact that such term must be primitive – this is pragmatically coherent with the fact that the raw information coming from the DB cannot be inferred – we can allow such term to be used inside new, eventually even non primitive, definition. To this extent we need a much more complex schema for translating queries on DB-marked term into SQL. For example, if the query is of the kind $\lambda\langle x\rangle.C(x)$ where $C \doteq \mathbf{some}(R, D)$, its SQL translation could be:

```
SELECT  M(R).left
FROM    M(R)
WHERE   M(R).right IN M(D)
```

Similarly, a translation for the **all** operator could be given, as in [Borgida and Brachman,1993], but in this case some extra considerations about the adequacy of the standard extensional semantics of this operator, when used in a database context, would arise. In fact, the empty satisfiability of an **all** clause would be hardly suited for a DB.[10]

In the example above D is supposed to be a primitive atomic DB-marked concept. Another extension to be explored is about releasing this constraint. Again, some concerns about semantics adequacy should probably be adressed.

Also the **non intermediate db extension** condition has, after the considerations above, to be revised. In fact, even if we must still consider the informaton of $\mathcal{D}$, as they are given, as being *a priori* fully realized in the leaves of the taxonomy, because the tables in the DB, where the instances of $\mathcal{D}$ are described, are not structured in a hierarchy, it could happen that non primitive concepts specialize the DB-marked ones, as in the previous example on the **some** operator.

5.2 The Query Language

Another iussue to be explored regards the query language. Currently our system support existentially quantified conjuntions of atomic formulae.

We plan to expand its capability with the possibility of answering any first-order-logic query. We foresee that, to this extent, much attention has to be paid on the optimization of the queries.[11]

[10]As we argued even for standard knowledge bases [Bresciani,1991] the **every** operator [Franconi,1992] would be more adequate in this case.

[11]Because in our system queries to KB and to DB are managed in a uniform way, the approach of [Buchheit *et al.*,1994] can be usefully applied.

5.3 Aknowledgments

Our thanks must be addressed to Enrico Franconi, for his careful reading of several preliminary copies of the present paper and the useful suggestions he made about it. We also thank Fabio Rinaldi, for his implementation of the SQL interface.

References

[Borgida and Brachman, 1993] Alex Borgida and Ronald J. Brachman. Loading data into description reasoners. In *Proceeding of ACM SIGMOD '93*, 1993.

[Bresciani, 1991] Paolo Bresciani. Logical account of a terminological tool. In *Proc. of the IX Conference on Applications of Artificial Intelligence*, Orlando, FL, 1991.

[Bresciani, 1992] Paolo Bresciani. Representation of the domain for a natural language dialogue system. Technical Report 9203-01, IRST, Povo TN, 1992.

[Buchheit *et al.*, 1994] Martin Buchheit, Manfred A. Jeusfeld, Werner Nutt, and Martin Staudt. Subsumption between queries to object-oriented databases. *Information Systems*, 19(1):33–54, 1994.

[Devanbu, 1993] Premkumar T. Devanbu. Translating description logics to information server queries. In *Proceedings of Second Conference on Information and Knowledge Management (CIKM '93)*, 1993.

[Franconi, 1992] E. Franconi. Extending hybridity within the YAK knowledge representation system. *AI*IA notizie, the Italian Association for Artificial Intelligence Journal*, 5(2):55–58, June 1992.

[MacGregor, 1991] R. MacGregor. Inside the LOOM description classifier. *SIGART Bulletin*, 2(3):88–92, 1991.

[Nebel, 1990] B. Nebel. *Reasoning and Revision in Hybrid Representation Systems*, volume 422 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin, Heidelberg, New York, 1990.

Working Notes of the KI'94 Workshop: KRDB'94 - Reasoning about Structured Objects: Knowledge Representation Meets Databases

D-94-11
Document

F. Baader, M. Buchheit, M. A. Jeusfeld, W. Nutt (Eds.)