Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Document
D-95-12

Working Notes of the KI'95 Workshop:

KRDB-95
Reasoning about Structured Objects: Knowledge Representation Meets Databases

Bielefeld, Germany, Sept. 11-12, 1995

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

September 1995

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

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

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

Deutsches Forschungszentrum für Künstliche Intelligenz

The German Research Center for Artificial Intelligence (Deutsches Forschungszentrum für Künstliche Intelligenz, DFKI) with sites in Kaiserslautern and Saarbrücken is a non-profit organization which was founded in 1988. The shareholder companies are Atlas Elektronik, Daimler-Benz, Fraunhofer Gesellschaft, GMD, IBM, Insiders, Mannesmann-Kienzle, Sema Group, Siemens and Siemens-Nixdorf. Research projects conducted at the DFKI are funded by the German Ministry for Research and Technology, by the shareholder companies, or by other industrial contracts.

The DFKI conducts application-oriented basic research in the field of artificial intelligence and other related subfields of computer science. The overall goal is to construct systems with technical knowledge and common sense which - by using AI methods - implement a problem solution for a selected application area. Currently, there are the following research areas at the DFKI:

- ☐ Intelligent Engineering Systems
- ☐ Intelligent User Interfaces
- ☐ Computer Linguistics
- ☐ Programming Systems
- ☐ Deduction and Multiagent Systems
- ☐ Document Analysis and Office Automation.

The DFKI strives at making its research results available to the scientific community. There exist many contacts to domestic and foreign research institutions, both in academy and industry. The DFKI hosts technology transfer workshops for shareholders and other interested groups in order to inform about the current state of research.

From its beginning, the DFKI has provided an attractive working environment for AI researchers from Germany and from all over the world. The goal is to have a staff of about 100 researchers at the end of the building-up phase.

Dr. Dr. D. Ruland
Director

Working Notes of the KI'95 Workshop: KRDB-95 – Reasoning about Structured Objects: Knowledge Representation Meets Databases

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

DFKI-D-95-12

This work has been supported by a grant from The Federal Ministry of Education, Science, Research and Technology (FKZ ITWM-9201).

ISSN 0946-0098

Working Notes of the KI'95 Workshop:

KRDB-95
Reasoning about Structured Objects: Knowledge Representation Meets Databases

Bielefeld, Germany, September 11-12, 1995

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
University of Science and Technology
Dept. Information and Systems Management
Clear Water Bay, Kowloon, Hong Kong
jeusfeld@usthk.ust.hk

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 Bielefeld, Germany on September 11-12, 1995, as part of the 19th 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 is intended to bring together researchers from both areas to continue the discussion about the problems and applications of a combination of KR and DB techniques, which was initiated at the predecessor workshop KRDB-94, and to identify new such questions and 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?

The first session is devoted to *extensions of knowledge representation by database techniques.* Bresciani describes an architecture that combines a KR system based on description logic and a relational DBMS by so-called close coupling. The amalgamated system presents itself like a KR system. Some parts of the class taxonomy reside in a database, however. In order to comply with the semantics of the KR system, only a fragment of the possible queries are allowed for retrieving objects from the database part. James, Gatward, and Shipley present an extension of the very expressive KR language CPL (conceptual prototyping language) by a language for describing object-oriented schemas in order to combine the management and processing of routine data with the reflection and utilization of knowledge. Lebastard proposes to define an object-oriented DBMS on top of a relational DBMS. Thus the user can choose the object model to be handled and has access to arbitrary relational databases. Reimer, Lippuner, Norrie, and Rys describe a formal mapping of DL inferences to queries of the OODB system COCOON. This extends the mapping of concept descriptions to class descriptions presented at KRDB-94.

The topic of the second session is the *extension of databases by knowledge representation techniques.* Kessel, Rousselot, Schlick, and Stern intend to combine a description logic system and a DBMS, motivated by their applications in the areas CAD and document retrieval, in which the ability to manage huge amounts of data is crucial. Simonet and Simonet present the P-type data model and show that it is closely related to description logics. The goal of their work is to transfer reasoning techniques developed for description logics, like subsumption algorithms, to P-types. Calvanese, De Giacomo, and Lenzerini introduce a new and very expressive data model for describing classes, views and links. Reasoning in this model is based on techniques, developed by the authors, for reasoning in expressive description logics allowing for, e.g., number restrictions, inverse roles, and recursive definitions. Nissen and Zemanek describe the successful usage of the KR system ConceptBase for modeling business processes and for requirements engineering. The cooperation that resulted in this work was initiated at KRDB-94.

The third session is concerned with *Queries.* Bergamaschi, Sartori, and Vincini propose the use of reasoning techniques from KR (subsumption computation) for computing intensional answers to DB queries by taking integrity constraints and the DB schema into account. Savnik, Tari, and Mohorič proposes a language that allows to manipulate and reason about the schema of a database, and to express declarative queries. Schild investigates the use of expressive description logics as database query languages. He presents a language that allows one to formulate queries that are beyond the expressivity of relational query languages, but can still be efficiently evaluated. He achieves tractability for his expressive language by exchanging the "open world" assumption usually employed in description logics by the "closed world" assumption customary in the DB area.

The three papers of the last session investigate the KR and DB issues from the viewpoint of *interoperable systems.* Boudjlida observes that complex objects play a significant role in the engineering of interoperable systems. KR-based reasoning as well as reflexivity are proposed to aid development. Edmond, Papazoglou, Russell, and Tari package conventional database systems via complex objects to provide a more flexible access. Here, reflection is used to represent information about the databases to application systems. Kusch and Saake investigate methods for partitioning complex objects in a distributed information system environment, with the goal of preserving local autonomy. Partitioning is expressed in terms of a formal specification language.

Contents

Session 1: Extending KR by DB Techniques

Session 2: Extending DBs by KR Techniques

Session 3: Queries

Session 4: Interoperability

Querying Databases from Description Logics

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

Abstract

Two different aspects of data management are addressed by description logics (DL) and databases (DB): the semantic organization of data and powerful reasoning services (by DL) and their efficient management and access (by DB). It is recently emerging that experiences from both DL and DB should profitably cross-fertilize each other, and a great interest is rising about this topic.

In the present paper our technique, that allows uniform access – by means of a DL-based query language – to information distributed over knowledge bases and databases, is briefly reviewed. Our extended paradigm integrates the separately existing retrieving functions of description logics management systems (DLMS) and of database management systems (DBMS) in order to allow, via a query language grounded on a DL-based schema knowledge, uniformly formulating and answering queries, so that uniform retrieval from mixed knowledge/data bases is possible.

In particular, some new developments extending those presented in [Bresciani, 1994] are introduced. By means of them the mapping between DL *concepts* and DB views is not more limited to primitive concepts, but also to some non-primitively defined ones.

1 Introduction

The main difference between knowledge representation (KR) and database (DB) systems is that the latter are oriented to the efficient management of large amount of data while the former seek to give a more structured representation of the universe of discourse in which data are placed. More precisely, in a KR system the universe of discourse is described by means of a collection of terms – or *concepts* – that are placed into a taxonomy. The capability of *classifying* concepts to form taxonomies is given by an appropriate calculus, whose first goal is to provide a *subsumption* algorithm. Concept Languages together with appropriate subsumption calculi are called Description Logics (DL). Databases, 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 well as the capability to infer new information from the existing ones. Thus, two different aspects of data management are addressed by description logics management systems (DLMS) and by database management systems (DBMS): the semantic organization of data (by DLMS), and their efficient management and access (by DBMS).

The importance of KR has been regarded as fundamental for the construction of good Intelligent Information Systems for more than ten years (see, e.g., [Tou *et al.*, 1982]), but only recently the theoretical foundations of a DL approach to DB have been established [Buchheit *et al.*, 1994].

From another point of view, KR-based applications and, more generally, AI-based applications can be widely enhanced by AI/DB interfaces [Pastor *et al.*, 1992; McKay *et al.*, 1990].

In particular, for the task of implementing DL-based applications, several reasons can be argued in favor of the use of external DB:

- because, in realistic applications, knowledge bases (KB) not only can be complex, but can also involve a large number of individuals, that are difficult – when not impossible – to manage with the existing DLMS ABoxes, due to their lack of efficiency in dealing with large amounts data, often it is better to manage large portions of data by means of a DBMS;
- as [Borgida and Brachman, 1993] mentions, KB based on DL are often used in applications where they need access to large amounts of data stored in *already existing* databases;
- as observed in [Bresciani, 1994; Bresciani, 1992], the task of acquiring knowledge for a real knowledge based application often includes a great amount of raw data collecting; for this subtask instead of using an ABox often it is better to use databases.

In particular we faced these problems when we were developing a large natural language system prototype [Bresciani, 1992], whose domain and linguistic model were represented using LOOM [MacGregor, 1991]. A first implementation of the ideas here presented is currently used in an enhanced version of this prototype, capable of dealing with thousands of individuals.

In such applications it is very important that the database can be queried from the DLMS in a way completely transparent to the user. This call for a semantically well founded linking between the DL knowledge base and the database. This can be obtained by *coupling* DLMS and DBMS [Borgida and Brachman, 1993]: primitive concepts and relations in a KB are made to correspond respectively to unary and binary tables in a DB. In [Borgida and Brachman, 1993] two possible way to couple DLMS and DBMS are proposed:

- *loose coupling*, that requires a pre-loading of the data from the DB into the KB;
- *tight coupling*, that implements a *on demand* access to the DB;

but in the system there presented only the loose coupling paradigm is implemented [Devanbu, 1993; Borgida and Brachman, 1993].

Instead, our system is based on tight coupling, allowing the following advantages:

- complex compound conjunctive queries involving unary and binary predicates can be done;
- no memory space is wasted in the DLMS in order to keep descriptions of DBMS data;
- answers are given on the basis of the current state of the KB and the DB, without needing periodical updating of the KB with new or modified data from the DB.

Our technique [Bresciani, 1994] will be in the following briefly reviewed. This approach is here extended with the possibility of mapping a wider set of DL *concepts* into DB *views*: in this way less restrictions about the form of the KB are necessary.

2 TBox, ABox and DBox

The basic idea of our approach is to extend the traditional DL ABox with a *DBox*,[1] by which the standard TBox/ABox architecture is coupled with one or more, possibly heterogeneous and distributed, databases, so that the user can make queries to this extended system without any concern on which DB or the KB has to be accessed.

A mapping – called PM (see section 3) – between the TBox and the DBox is needed. Therefore, a *knowledge base* $\mathcal{KB} = \langle \mathcal{T}, \mathcal{W}, \mathcal{D}, PM \rangle$ [Bresciani, 1994] is formed by a *terminology* $\mathcal{T}$ and a *world description* $\mathcal{W}$ as usual [Nebel, 1990], plus a *data base* $\mathcal{D}$ and the mapping function PM. A uniform query answering function to $\mathcal{KB}$, based on the two distinct complete query answering functions (one for the ABox and one for the DBox), can be implemented. For the sake of simplicity, it will be assumed here that $\mathcal{D}$ is represented by means of a relational database, and queries to the DBox can be done in SQL.

3 Coupling

Coupling the terminology $\mathcal{T}$ with the data base $\mathcal{D}$ corresponds to associating some terms (concepts and roles) of $\mathcal{T}$ with tables or views in the DB. The coupling of $\mathcal{T}$ with $\mathcal{D}$ is performed in two steps. First, a partial mapping PM between primitively defined terms and the tables in the DB must be given. Giving a mapping of a primitively defined term into a DB-table corresponds to giving its extension in the DB. Let the terms for which PM is defined be called $\mathcal{D}$-terms. Then, using PM, also non-primitively defined concepts can be recursively mapped into views of the DB. If the (expanded) definition of a non-primitively defined concept contains both $\mathcal{D}$-terms and non-$\mathcal{D}$-terms, the view in which the concept is mapped does not contain all the instances of the concept. Therefore, non-primitively defined concepts with (expanded) definition containing both $\mathcal{D}$-terms and non-$\mathcal{D}$-terms cannot be completely managed in our system. Thus, the following constraints must be imposed on $\mathcal{KB}$:

1. Every table in $\mathcal{D}$ must correspond to one primitively defined term in $\mathcal{T}$, called $\mathcal{D}$-term; $\mathcal{D}$-terms cannot be used in the (expanded) definition of any primitively defined term in $\mathcal{T}$.
2. The (expanded) definitions of non-primitively defined concepts of $\mathcal{T}$ must contain only $\mathcal{D}$-terms or no $\mathcal{D}$-term at all.

The aim of the constraint 1 is to avoid any need of consistency checking in case of conflicts between defining and defined concepts. If, to ensure the avoidance of such conflicts, an exhaustive checking – that could involve also the extensional analysis of DBox-tables – were provided, this constraint could be released.

As mentioned, all the information needed to correctly drive the query mechanism is the association of $\mathcal{D}$-terms with the corresponding tables in the DB. Thus, defined the partial mapping:

$$PM : \mathcal{PT} \rightarrow DBtable$$

where $\mathcal{PT}$ is the set of primitive terms in $\mathcal{T}$, and $DBtable$ is the set of tables in the DB, the views corresponding to non-primitive concepts can be built via a recursive partial mapping:

$$RM : \mathcal{T} \rightarrow DBtable \cup DBview$$

where $DBview$ is the (virtual) set of views in the DB. RM maps DL-expressions into corresponding SQL-expressions.

In the following, to simplify the description, it is assumed that concepts are mapped into unary tables with one column called `lft`, and roles into binary tables with two columns called `lft` and `rgt`. As an example, assume that non-primitively defined concepts that contain $\mathcal{D}$-terms in their (expanded) definition are constrained to use the sub-language with the only `AND` and `SOME` operators; in this case RM can be defined as follows:[2]

[1] D for data.

[2] Note that `R` stands for a role name, i.e., for an atomic role in $\mathcal{T}$, while C and D stand for concept names or expressions. In general, the `TYPEWRITER` font will be used for atomic terms.

RM((AND C D)) =

SELECT DISTINCT	lft
FROM	$RM(C), RM(D)$
WHERE	$RM(C)$.lft =$RM(D)$.lft

if both $RM(C)$ and $RM(D)$ are defined;

RM((SOME R D)) =

SELECT DISTINCT	lft
FROM	RM(R)
WHERE	RM(R).rgt IN $RM(D)$

if both RM(R) and $RM(D)$ are defined;

RM(T) = PM(T)
if PM(T) is defined;

and

RM(T) =	SELECT DISTINCT	*
	FROM	T_1
	UNION	
	⋮	
	SELECT DISTINCT	*
	FROM	T_n

if M(T) = $\{T_1, \ldots, T_n\}$, and $n > 0$.

Note that the last part of the above definition (see below for the definition of M) allows to take into account also all the tables and views corresponding to terms subsumed by T, whatever T is.

Of course RM could be extended to more general concepts, but in some cases the mapping would have to be carefully handled, due to the different semantics of DL and DB (see, e.g., the ALL and the NOT operators).

Note that, due to limitations of SQL in using subqueries, the SELECT used in the definition of RM are non exactly legal, due to the recursive application of RM. This problem can be easily overcome if a CREATE VIEW corresponds to each application of RM, and the names of the corresponding views are placed in lieu of the recursive applications of RM.[3]

The function:

$$M : \mathcal{T} \rightarrow 2^{DBtable \cup DBview}$$

used in the definition of RM returns the (possibly empty) set of tables/views necessary to retrieve all the instances (pairs) of a given concept (role) from the DB, that is:

$$M(\mathtt{T}) = \{RM(x) \mid x \in subs(\mathtt{T}) \wedge RM(x) \text{ is defined}\}$$

where $subs$(T) is the set of the terms classified under T in $\mathcal{T}$. Observe that RM and M are built starting from PM; this justifies the use of the only PM in the definition of $\mathcal{KB}$ given in section 2.

4 Query Answering

A *query* to $\mathcal{KB}$ is an expression:

$$\lambda \overline{x}.(P_1 \wedge \ldots \wedge P_n)$$

where $P_1, \ldots, P_n$ are predicates of the form C(x) or R(x, y), where C and R are concepts and roles in $\mathcal{T}$, respectively, and each of x and y appears in the tuple of variables $\overline{x} = \langle x_1, \ldots, x_m \rangle$ or is an individual constant in $\mathcal{W} \cup \mathcal{D}$. *Answering* a query in $\mathcal{KB}$ means finding a set $\{\overline{x}^1, \ldots, \overline{x}^m\}$ of tuples of instances such that, for each tuple $\overline{x}^i$, $\lambda \overline{x}.(P_1 \wedge \ldots \wedge P_n)[\overline{x}^i]$ holds – either explicitly or implicitly – in $\mathcal{KB}$. Let such tuples be called *answers* to the query and the set of all of them the *answer set*.

From the definition of answer to a query, it is obvious that, to avoid the generation of huge answer sets, free variables must not be used, that is, each variable appearing in $\overline{x}$ must appear also in the query body. Indeed, even stronger restrictions are adopted (see [Bresciani, 1994]).

To be answered, a query must be split into subqueries that can be answered by the two specialized query answering functions of the DLMS and the DBMS. To this end, a *marking* of all the possible atomic predicates, corresponding to the terms in $\mathcal{T}$, is needed; a term P is said to be:

- KB-marked iff $RM(P)$ is undefined;
- Mixed-marked otherwise.

These two markings reflect the fact that the instances (pairs) of P are all in $\mathcal{W}$, or part in $\mathcal{W}$ and part in $\mathcal{D}$, respectively. The case of queries in which the predicates are all KB-marked terms is trivial (it is enough to submit it to the DLMS answering function). The case of queries with also Mixed-marked predicates is more difficult.

Let a generic query be written as:

$$\lambda \overline{x}.(P_1^{KB} \wedge \ldots \wedge P_m^{KB} \wedge P_1^M \wedge \ldots \wedge P_n^M)$$

where the P_i^{KB} correspond to the KB-marked terms, and the P_i^M to the Mixed-marked terms. The query can be split in the two sub-queries:

$$q^{KB} = \lambda \overline{x}_{KB}.(P_1^{KB} \wedge \ldots \wedge P_m^{KB}),$$

$$q^M = \lambda \overline{x}_M.(P_1^M \wedge \ldots \wedge P_n^M).$$

Because each predicate in q^M corresponds to a view in the DB – where the answers have to be searched in addition to those in the ABox – a translation of them into equivalent SQL queries can be provided. Of course, the views can easily be found via the recursive mapping RM. For each of the P_i^M in q^M the translation into an equivalent view is simply given by $RM(P_i^M)$. Thus, the SQL query corresponding to $q_i^M = \lambda \overline{y}.P_i^M$ – where $\overline{y}$ is the sub-tuple of $\overline{x}$ containing the only one or two variables used in P_i^M – is:

SELECT DISTINCT	*select-body*
FROM	$RM(P_i^M)$
WHERE	*where-body*

where the *select-body* contains $RM(P_i^M)$.lft, $RM(P_i^M)$.rgt, or both, according to the fact that P_i^M is of the kind C(x) or R(x, a), R(a, y), or R(x, y), respectively – with x and y variables, and a constant. The WHERE clause is present only in the case of P_i^M = R(x, a) or P_i^M = R(a, y); in this case the *where-body* is RM(R).lft = a or RM(R).rgt = a, respectively.

[3] Of course, this requires a pre-compilation step of the DB with respect to the KB, but this is not a real overload of the presented query mechanism.

In this way n partial answer sets (one for each P_i^M) are obtained. Of course, the queries have to be submitted also to the DLMS, in case there are also $\mathcal{W}$-individuals satisfying them.

Now, it is, ideally, enough to get the intersection of all the partial answer sets obtained by processing the sub-queries of q^M and q^{KB}, but, due to the scope of the variables of the queries, this cannot be performed in a direct way: a *merging* of the results is needed. In fact, in each sub-query some of the variables in $\overline{x}$ may be unbound – that is, the proper tuple of variables $\overline{y}$ of the sub-query may be a sub-tuple of $\overline{x}$. Therefore, the corresponding answer set has to be *completed*, that is, each unbound variable in $\overline{x}$ must be made to correspond to each instance in $\mathcal{KB}$, for all the found answers, considering all the possible combinations. However, in this way huge answer sets would be generated.

To solve this problem a compact representation for the answer sets is needed. If $AS_{\overline{y}}$ is a generic partial answer set of a sub-query, and the variables of the original complete variable tuple $\overline{x}$ missing in $\overline{y}$ are $x_{p_1}, \ldots, x_{p_k}$, the completion of $AS_{\overline{y}}$ can be represented in a compact way as $AS_{\overline{x}} = \{\overline{I}^\star \mid \overline{I} \in AS_{\overline{y}}\}$, where each $\overline{I}^\star$ is equal to $\overline{I}$ except that it is lengthened by filling the k missing positions $p_1, \ldots, p_k$ with any marker, e.g., a star '$\star$', that stands for any individual in $\mathcal{KB}$. Using this representation it is possible to formulate an algorithm to efficiently cope with the *merging* of answers sets, as described in [Bresciani, 1994].

5 Conclusions

Our approach to deal with the task of integrating DLMS and DBMS, so that KB and DB can be uniformly queried from a DLMS, has been presented. With our technique, a third component – a DBox, allowing spreading extensional data among the ABox and databases – can be added to the traditional TBox/ABox architecture of DLMS. By means of the DBox it is possible to couple the DLMS with several, possibly distributed and heterogeneous, DBMS, and to use all the systems for uniformly answering queries to knowledge bases realized with this extended paradigm.

In our first implementation of the system[4] the DLMS is LOOM [MacGregor, 1991], and the database query language is SQL, but also other systems could be easily used.

At present our tool is used in a natural language dialogue system prototype [Bresciani, 1992], whose domain and linguistic knowledge is represented in a LOOM KB and, for some large amount of raw data, in an INGRES DB. Currently, our system support a more expressive query language than the one previously presented: existentially quantified conjunctions of atomic formulæ can also be used. The study of the use of even more complex query-languages is part of our future plans.

[4] Indeed, the answering algorithm has been implemented in a more sophisticated way than the one presented in section 4, including also optimizations for reducing the number of accesses to the DB (see [Bresciani, 1994]). The pre-compilation part of the method shown in section 3 – that allows dealing with non primitive concepts – is presently not yet fully implemented.

References

[Borgida and Brachman, 1993] Alex Borgida and Ronald J. Brachman. Loading data into description reasoners. In *Proceeding of ACM SIGMOD '93*, 1993.

[Bresciani, 1992] Paolo Bresciani. Use of loom for domain representation in a natural language dialogue system. Technical Report 9203-01, IRST, Povo TN, March 1992. presented at LOOM Users Workshop, Los Angeles, March 23-24, 1992.

[Bresciani, 1994] Paolo Bresciani. Uniformly querying knowledge bases and data bases. In F. Baader, M. Buchheit, M. A. Jeusfeld, and W. Nutt, editors, *Working Notes of the KI'94 Workshop: KRDB'94*, number D-94-11 in DFKI Documents, pages 58–62, Saarbrücken, Germany, September 1994.

[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.

[MacGregor, 1991] R. MacGregor. Inside the LOOM description classifier. *SIGART Bulletin*, 2(3):88–92, 1991.

[McKay *et al.*, 1990] Don P. McKay, Tim W. Finin, and Anthony O'Hare. The intelligent database interface. In *Proc. of AAAI-90*, pages 677–684, Boston, MA, 1990.

[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.

[Pastor *et al.*, 1992] Jon A. Pastor, Donald P. McKay, and Timothy W. Finin. View-concepts: Knowledge-based access to databases. In *Proceedings of Second Conference on Information and Knowledge Management (CIKM '93)*, Baltimore, 1992.

[Tou *et al.*, 1982] F. Tou, M. Williams, R. Fikes, A. Henderson, and T. Malone. Rabbit: An intelligent database assistant. In *Proc. AAAI'82*, 1982.

Using a Knowledge Representation Language to capture both Knowledge and Routine Data

Anne E. James and Richard A. Gatward and Steve Shipley
School of Mathematical and Information Sciences
Coventry University, Priory Street, Coventry
Tel. 0203 838991, Fax: 0203 221608
E-mail: csx118@uk.ac.coventry

Database systems typically have a simple structure designed to facilitate the management of large amounts of abstracted, structured data with a high degree of uniformity. Knowledge representation languages on the other hand typically embody much greater richness with the aim of reflecting information that is not quite as abstracted or structured and which has a far lesser degree of uniformity. We contend that although the simplicity of structure of current database systems is useful for coping with many routine tasks, systems of the future should be able to combine the management and processing of routine data with the reflection and utilisation of knowledge. This should result in more sophisticated systems that are more user-supportive and less prone to human error. We are currently involved in a project which aims to combine knowledge and database techniques for modelling engineering applications.

The approach we are taking involves the use of a knowledge representation language, CPL (Conceptual prototyping Language, see [1]) to capture both knowledge and routine data. CPL is based on linguistic theory (Functional Grammar, FG, see [3]) and uses the semantic basis of predicate calculus. The motivation behind the development of CPL was to produce a knowledge-based modelling language that had the power to express any kind of knowledge that one might want to incorporate into a system. In particular CPL includes the implementations of logics to allow for the specification of vague knowledge, knowledge about events and obligations and knowledge about temporal aspects [2].

One of the key ideas of FG used in the development of CPL is that of the semantic function. This is used in the context of CPL to specify roles defined by the use of certain verbs (called relations in CPL) in the application domain. Capel and Wistra [1] give a list of semantic functions used in their interpretation of the language. These include both those defined in the theoretical specification of Functional Grammar, along with those that they have added for the purpose of the modelling language. The kind of semantic function that can be applied to a particular slot in the predicate depends upon the nature of the slot. Predicates that have been extended with satelites [3] can have a different set of semantic functions applied to the satellite position from those that can be applied to a basic predicate. In the theory of functional grammar the parts of the expression represented by satellites roughly corresponds to prepositional clauses and similar appendages that are added to the basic utterance to give additional information about a stae of affairs to that which is required as a minimum by the use of the main verb in the expression. CPL has some special relations such as 'is-a' and 'has' where the semantic functions are predefined. In other cases the user will select appropriate semantic functions for a relation.

CPL is very rich and mirrors natural language structure. Since we can express most information in natural language it follows that CPL, as a formalised version thereof, can be used to express most knowledge. There is a problem however, in that operational information that typically needs to be recorded is more conveniently recorded using simpler frameworks. Therefore we propose to use CPL as defined for knowledge representation and an extension thereof for handling routine, uniform data sets.

The proposed extension to CPL introduces an explicit meta-level for defining routine data according to the object-oriented paradigm. A new statement type METAFACTUAL will allow for the definition of uniform data sets. This statement type will be used to define object classes, their operations and object class sub-type and containment hierarchies. Special relations with predefined semantics such as 'is-object-class', 'is-operation', 'has-operations', 'has-objects', 'is-sub-type' and 'is-instance' will be introduced for this purpose. An implementation of extended CPL would then need to include the operational semantics of the object-oriented approach as well as that of functional grammar and predicate calculus.

The above gives an overview of the type of approach we are taking to integrating knowledge and database techniques. We feel the approach is novel in that most other work we have seen adds knowledge constructs to database model formalisms wheras we have taking the opposite approach of extending a knowledge representation formalism to capture data model concepts. The work is at an early stage and planned future work will involve further prototyping of the ideas, defining the necessary CPL extensions more rigourously, examing the underlying semantics of a combined formalism and developing suitable user interfaces.

References

[1] Capel C. and Wistra D. LIKE, MSc Thesis, V U Amsterdam, 1987.

[2] Dignum F. and Van de Riet R. P. Knowledge Base Modelling Based on Linguistics and founded in Logic. *Data and Knowledge Engineering*, 7, pp 1-34, 1991.

[3] Dik S. C. Functional Grammar, North Holland, 1980.

Is an object layer on a relational database more attractive than an object database ?

Franck Lebastard
CERMICS/INRIA - BP 93
F-06902 Sophia-Antipolis Cedex, France
Voice +33 93 65 77 42
E-mail : Franck.Lebastard@sophia.inria.fr

1 Introduction

As researchers in Artificial Intelligence, our first aim was to allow our expert system shell SMECI [Sme90] the access to relational databases during reasoning. We also needed to save in a database complex objects that seemed interesting for further utilization, in particular the knowledge bases and the results of reasoning.

To this end, we have defined generic correspondences [Leb93] between relational concepts and some of the object concepts that are common to most object models. These correspondences allow to translate relational data into complex objects and conversely. They generalize the mapping proposals that we found in the literature [Lee90; WBL+91; KJA93].

An implementation of these definitions has been realized. This is the DRIVER system [Leb92] whose specificity is to define an object oriented DBMS (OODBMS) on a relational DBMS (RDBMS).

In DRIVER, a correspondence scheme must describe how to use a particular relational database that is, the object representation and the relational representation to bring together and the concrete mapping between them. It can be given by the user or automatically generated. The database is then available as an object oriented database.

DRIVER can be used with many object models. The system performs all operations on the objects through a functional interface that must be instanciated for the chosen model. In particular, it creates objects in memory and reads and writes their slots through this interface. This way to handle objects ensures that persistency is a property effectively orthogonal to the model. Of course, only selected object concepts can become persistent. The other properties are simply ignored.

DRIVER is operational and is used by several industrial partners.

2 Our generic correspondences

Let us see now the generic correspondences we have defined. They make possible to manage relational data in the form of complex objects and to express objects as relational data.

2.1 Classes and relational tables

We have associated the concept of relational table with the concept of class. More precisely, we have associated tables with class hierarchies because we enforce all subclasses of a class to be mapped on the same table. The most general class mapped to a table is called a *main class*. Its associated table is its *main table*.

We also allow to associate more than one table to a class. The extra tables are called the *secondary tables* of the class. The main table and the secondary tables are the *elementary tables* of the class. They must imperatively be linked all together with joins which are also called elementary, that means in our definition that each one binds one tuple of a table with one tuple of another. The elementary tables of a class are utilized to map its fields. In a hierarchy, any class may use one or several additional elementary tables to store specific data.

The figure 1 shows a possible mapping for the `Employee` class. `emp` is its main table and `person` is a secondary table, both are its elementary tables.

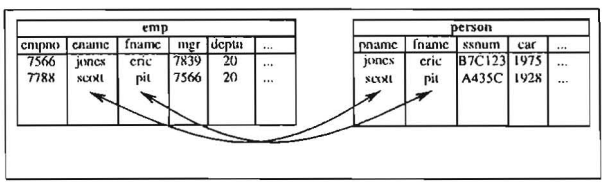

Figure 1: The elementary tables of the Employee class

We can notice that there is always a path, a join *chain* linking any elementary table to the main table.

2.2 Objects and tuples

As we have brought together both notions of relational table and class we also associate the concept of relational tuple with the concept of object. Both are data, occurrences of their own structures. In DRIVER, the correspondence of an object is a set of tuples, one for each elementary table of the class the object is instance of. We have chosen to compose its unique reference, its "oid" using the name of its class main table and its tuple key in this table. This way, every tuple in the main table is candidate to be the correspondence of an object of the associated

main class.

The object candidate is an object when :

- all the elementary tables of the main class contain a tuple for it. These tuples are found by joining the main table.
- its value is compatible with the constraints that any instance of this class must check.

These constraints can be set on the atomic fields of the class or on the attributes of the associated elementary tables. They define a kind of filter that tells which tuples correspond to objects. Those which are not selected are simply ignored and everything works at the object level as if they don't exist. The potential objects that corresponds to the selected tuples are calles the *relational objects.*

In figure 1, both tuples "jones" and "scott" of the emp table are object candidates for the class Employee. Since the attribute emp.empno is the key of the table emp, their "oids" are for example emp/7566 and emp/7788. Let us assume that a constraint "emp.empno > 7000" is enforced to any tuple of emp to be considered as a relational object of the class Employee. Since our both tuples comply this constraint and since it exists for each of them in the table person a tuple found back by the elementary join, they are considered as Employee relational objects, liable to be filtered by an object request involving Employees.

When an elementary table set is associated with a class hierarchy and when a main table tuple has been selected as an object of the associated main class, one infers the precise class it is instance of from its implementation in the elementary tables and from the constraints defined for each class it complies or not. Indeed, a class is different from its superclass, over and above its possible own fields that complete those it inherits from the superclass :

- by the possible use of new elementary tables in its correspondence. If the object is at least instance of this class, there must be a tuple for it in each of the elementary tables of the class.
- by stronger constraints enforced on its instances. These constraints must also define an object set that is disjoint from the sets defined by the other subclasses of the same superclass. Since the constraint sets are organized in a tree, to be instance of a class depends on their satisfaction along the considered hierarchy.

While a filtering in the database, DRIVER automatically classifies the chosen relational objects and gives them the most precise class depending on their values and implementation in the base.

Before ending the description of our correspondences at the class and object level, let us point out that the term "table" we have used up to here actually represents more a logical table than the "table as a structure in the database". In other words, in DRIVER, for our correspondences, we can define as many logical tables as we need on a same user table of the base. These logical tables allow us for example to map an object in different tuples of the same user table. They also allow to map independent class hierarchies on logically different main tables that actually represent the same user table in the base. There are indeed applications where one wants to supply independent classes with persistency in a unique table.

file

tnum	objnum	fnum	fval1	fval2
824	7566			employee
825	7566	1		jones
826	7566	2		eric
827	7566	3	7839	
828	7566	4	20	
...	...	...	...	...
841	7788			employee
842	7788	1		scott
843	7788	2		pit
844	7788	3	7566	
844	7788	4	20	
...	...	...	...	

Figure 2: The file table

For example, let us consider the table file presented figure 2. In DRIVER, we can associate our class Employee with this table file as easily as we did with emp et person. To have access to the different tuples that make up the relational object, one only has to define elementary joins between for example mainfile(file), the main table of Employee, and file1(file), file2(file), etc, defined as secondary tables. Here is an example of an elementary join that allows to have access to the names of the Employees :

```
mainfile.objnum=file1.objnum
      and file1.fnum=1
```

We must also set a constraint that precises which tuples of mainfile correspond to objects. In our example, this constraint can be :

```
mainfile.fnum is null
and mainfile.fval2='employee'
```

With this correspondence of the class Employee, the "oids" of our two relational objects jones and scott are this time mainfile/824 and mainfile/841.

This way to store all the objects in a unique table is not so odd since it is effectively used, for example in the OODBMS *MATISSE* [Int92].

2.3 Relational correspondences of the field types

Correspondence of the atomic types

We have associated atomic type fields with attributes. This way, any attribute of a class elementary table can be used as the correspondence of any of the atomic fields of this class and vice versa.

If we consider our association Employee - (emp, person) again, and if the class has an atomic

field `social-security-number`, its mapping can be `person.ssnum`.

Correspondence of the object type

An object field represents an oriented link from an object to another. We associate this link between classes with a relational link, more precisely with a join between one of the elementary tables of the first class and the main table of the second class. This join that we call an *object join* must be an equijoin that compares elementary attributes of the first —attributes called *referential attributes*— and the attributes composing the key of the second.

This restriction allows to know the value of an object field just by knowing the values of the corresponding referential attributes. An object field is empty if any of the referential attributes is containing a NULL.

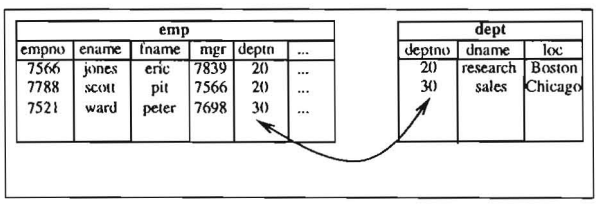

Figure 3: The mapping of the field `dpt`

Let us consider an object field `dpt` of the class `Employee`. It makes each `Employee` referring the `Department` it belongs to. This field can be mapped on the join shown figure 3 that links `emp` to the main table of the class `Department`. Then the referential attribute associated with the field is `emp.deptn`.

We can point out that this object field correspondence offers a way to modelize more or less strong links between objects : if any referential attribute is constrained by a clause unique, not null or both, the possible values for the corresponding object field are restricted. The stronger link is set when the referential attributes are also the key of their table. In that case :

- they must be valued : the associated object field cannot be empty.
- once the containing object or the contained object is persistent, its key value is fixed. As the key value of the other is settled at the same time (because of the join), and both objects are linked together for their life (!).

Correspondence of the set type

The correspondence of this field type is a *set join*. A set join is an equijoin between one of the elementary tables of the class the field belongs to and a *set table*. For a given object, the set table contains as many tuples as there are members in its set field value.

When the set is an atom set, the set join is completed with an attribute (of the set table) which contains the set members in the base.

When the set is an object set, the set join is generally completed with another join, this time between the set table and the main table of the referenced objects class. This second join must be an an equijoin that compares referential attributes and the key attributes of the joined main table.

Table emproj	
projno	empno
101	7566
103	7566
101	7788
...	...

Table project		
projno	pname	budget
101	alpha	250000.
102	beta	175000.
103	gamma	95000.
...	...	...

Figure 4: Tables `emproj` and `project`

For example, let us consider the table `emproj` shown figure 4. It is the representation in the base of the participation of every `Employees` to some `Projects`. If our class `Employee` owns a field `projects` (object set type), its mapping can be the join sequence (J[`emp, emproj`] J[`emproj, project`]) where the join expressions are respectively `emp.empno=emproj.empno` and `emproj.projno=project.projno`. In this example, the set table is of course the table `emproj`.

We also make possible to define the correspondence of an object set field in the form of a unique join between an elementary table of the class it owns to and the main table of the referenced objects class. In that case, the set table and the joined main table may be the same table. It happens when the join represents a N:1 relation. Then adding or removing members (objects) in a set finds expression in the database in updating the corresponding main tuples whereas it usually causes insertions or deletions of tuples in the set table.

An example of such a correspondence can be proposed for the field `employees` of the class `Department`. Indeed we can associate it with the join shown figure 3. Then, the new assignment of an `Employee` to a `Department` causes an update of the object tuple in the table `emp` : in this database, an `Employee` cannot work for more than one `Department`.

Correspondence of the list type

The correspondence of this type is quite similar to the one of the set type. It is made up by one or several attributes of the set table which values allow to arrange the list members and to differenciate the tuples corresponding to doubles. The first attribute defines the primary order, the second the secondary order, etc. For each of them, the sorting out can be in an ascending order or in a descending order.

We show an example of list correspondence with the `emproj2` table of figure 5. Here the arrangement of members is determined by the attribute `emproj2.order`.

In the case of the object list, doubles are allowed only if the set table is not the main table of the referred objects and if the order attributes are not chosen in their elementary tables.

Table emproj2		
projno	empno	order
101	7566	237
103	7566	121
101	7788	310
...	...	...

Figure 5: The emproj2 table

3 Benefits of the DRIVER approach

Let us now present the benefits of the DRIVER approach.

Firstly, the user chooses the object model to be handled. Thus, the accessed databases are directly viewed in his own object model, even if it is really a very own ad hoc model. More, he can easily supply volatile objects with persistency, at his convenience. Here, the OODBMS (i.e. DRIVER) doesn't impose which object model the application must work with. Thereby the OODBMS is not the main piece of the system any more. The DBMS is just a partner that simply offers a service to the application. Indeed that should be the only role of a persistency service for many applications.

Secondly, DRIVER gives an object access to very big amounts of data since the relational model is by now the most used DBMS standard. All relational databases are immediately available as object databases and conversely, all object databases built with DRIVER are of course immediately available and accessible to the numerous RDBMS users.

Industry has invested a lot in relational databases owing to the maturity of this technology. A lot of people aspire now to pass on to the object technology without giving up existing applications soon. To propose an OODBMS on top of a RDBMS lives up to this expectation. This solution does not upset the usual RDBMS users and allows the new users who need the object technology to access the same databases in the suitable form. Generally speaking, the DRIVER philosophy is a good solution to share data with many users. Data are expressed in a relational form –the simplest– in the database but they are used by everyone in another model, his own model with his own representation, an optimal choice of classes, relevant to his application.

Lastly, we believe that it is necessary to completely separate the object level, i.e. the knowledge representation level, from the physical level, i.e. the file manager level, to be able to make easily evolve the persistent object model. This separation is not complete in "classical" OODBMS. DRIVER uses RDBMS as intelligent file managers and proposes object models on top of them. Here the level separation is actual and the models should easily evolve with time.

References

[KJA93] A.M. Keller, R. Jensen, and S. Agarwal. Persistence software : Bridging object-oriented programming and relational databases. In *Proceedings of International Conference on Management of Data.* ACM SIGACT-SIGMOD, May 1993.

[Leb92] F. Lebastard. DRIVER v1.34, Reference manual. Technical Report 92-7 (in french), CERMICS-INRIA, Sophia-Antipolis (France), October 1992. 119 pages.

[Leb93] F. Lebastard. *DRIVER : A persistent virtual object layer for reasoning on relational databases.* Ph.D.Thesis (in french), CERMICS-INRIA Sophia-Antipolis (France), March 1993. 380 pages.

[Lee90] B.S. Lee. *Efficiency in Instanciating Objects from Relational Databases through views.* PhD thesis, Stanford University, Stanford (California), 1990. STAN-CS-90-1346.

[Int92] Intellitic International. MATISSE : Open semantic database - product overview. Technical report, Saint-Quentin-en-Yvelines (France), 1992. 16 pages.

[Sme90] Ilog, Gentilly (France). *SMECI Version 1.65, Reference manual*, May 1990. 470 pages.

[WBL+91] G. Wiederhold, T. Barsalou, B.S. Lee, N. Siambela, and W. Sujansky. Use of relational storage and a semantic model to generate objects : the PENGUIN project. In *Database'91 : Merging policy, standards and technology.* The armed forces communications and electronics association, Fairfax (VA), June 1991.

Terminological Reasoning by Query Evaluation: A Formal Mapping of a Terminological Logic to an Object Data Model*

U. Reimer and P. Lippuner
Swiss Life
Information Systems Research Group
CH–8022 Zürich Switzerland
⟨name⟩@swssai.uu.ch

M. Norrie and M. Rys
Swiss Federal Institute of Technology (ETH)
Dept. of Computer Science
CH–8092 Zürich, Switzerland
⟨name⟩@inf.ethz.ch

Abstract

The paper starts by giving concise introductions into the terminological logic FRM and the object data model COCOON. It then briefly outlines a semantic-preserving mapping from FRM class descriptions to COCOON types and classes and shows how the terminological inference of classification is mapped to a set of equivalent COCOON queries. Since these queries can (mostly) be submitted as a whole to the underlying database system we can take full advantage of all the results on query optimisation, on providing efficient physical access structures, as well as on parallelisation that are available in the database area to make terminological inferences more efficient. This will play a crucial role in realising knowledge base systems capable of dealing with very large knowledge bases.

1 Introduction

The fields of knowledge representation and databases are converging: The former is more and more concerned with efficiency for supporting large knowledge bases, while the latter is increasingly interested in providing higher representation constructs that better serve the construction of a domain model. Consequently, it seems to be a fruitful endeavour to combine the approaches of both areas. Our approach to combining the strengths of knowledge representation and database approaches takes advantage of the conceptual similarity of terminological logics and object data models. We realise a knowledge base system by mapping a terminological logic to an object data model which has an efficient implementation on top of a relational storage system [NRL+94]. To ensure that the potential for optimisation provided by the database system will really be available for the terminological system, the mapping from terminological structures to object structures preserves as much of the semantics of the terminological logic as possible.

*The work reported here was supported by the Swiss Priority Programme for Computer Science (Schwerpunktprogramm Informatik) under grant No. 5003-034347.

There are a few former approaches to mapping terminological logics (or frame models) to data models. In the mappings to the relational data model described in [HMM87] and [SB89], one frame structure corresponds to several database structures. As a consequence, there is little correspondence between the representation structures the database system manages and the original frame structures. Therefore, the database system is deprived of most of its optimisation capabilities.

Another former approach to mapping a frame model to a data model is described in [RS89]. It preserves the frame structure as a complex object structure in the nested relational model to which it is mapped. The major drawback with that approach results from the lack of type polymorphism in the nested relational model because this makes it difficult to host the concept hierarchy of the frame model.

Some of the existing data models that support complex objects provide constructs that are similar to constructs of a terminological logic (e.g. [KL89; BGL+91]). Their main difference is that they do not provide terminological reasoning services (besides inheritance), although offering deductive question answering.

Sections 2 and 3 introduce the basic concepts of the terminological logic FRM and the object data model COCOON used in our approach. Section 4 describes the mapping of the terminological inference of classification to COCOON queries and illustrates the mapping of class descriptions of FRM to type and class constructs of COCOON. Section 5 concludes the paper.

2 Basic Constructs of the Terminological Logic FRM

The syntactic constructs and the model-theoretic semantics of FRM [RL95; Rei85] are given in Figure 1. We distinguish two kinds of relations, namely properties and semantic relationships. A *property* denotes a relation between individuals and string or integer values (see the constructs **all-p** and **exist-v**). A *semantic relationship* denotes a relation between individuals (see the constructs **all-r, exist-c** and **exist-i**).

Unlike other terminological logics, FRM only al-

Syntactic form	Interpretation
$a \doteq t$	$\varepsilon[a] = \varepsilon[t]$
$a \dot{\leq} t$	$\varepsilon[a] \subseteq \varepsilon[t]$
(**and** $c_1 \ldots c_n$)	$\bigcap_{i=1}^{n} \varepsilon[c_i]$
(**all-p** *prop* $r_1 \ldots r_n$)	$\{x \in D \mid \exists y : \langle x,y\rangle \in \varepsilon[prop] \wedge \forall y : (\langle x,y\rangle \in \varepsilon[prop] \Rightarrow y \in (\varepsilon[r_1] \cup \ldots \cup \varepsilon[r_n]))\}$
(**all-r** *rel* $c_1 \ldots c_n$)	$\{x \in D \mid \exists y : \langle x,y\rangle \in \varepsilon[rel] \wedge \forall y : (\langle x,y\rangle \in \varepsilon[rel] \Rightarrow y \in (\varepsilon[c_1] \cup \ldots \cup \varepsilon[c_n]))\}$
(**exist-v** *prop* v)	$\{x \in D \mid \langle x,v\rangle \in \varepsilon[prop]\}$
(**exist-c** *rel* c)	$\{x \in D \mid \exists y \in \varepsilon[c] : \langle x,y\rangle \in \varepsilon[rel]\}$
(**exist-i** *rel* i)	$\{x \in D \mid \langle x,\varepsilon[i]\rangle \in \varepsilon[rel]\}$
(**at-least** *rp* n)	$\{x \in D \mid \|\{y \in D : \langle x,y\rangle \in \varepsilon[rp]\}\| \geq n\}$
(**at-most** *rp* n)	$\{x \in D \mid \|\{y \in D : \langle x,y\rangle \in \varepsilon[rp]\}\| \leq n\}$
thing	D

Figure 1: Syntax and Semantics of FRM

lows class descriptions that refer to other classes by their name and not by including their structure. This restriction does not have any effect on the expressiveness. It only requires that every concept class being used must independently be introduced and assigned a name. However, FRM provides an extended syntax for class descriptions that may occur as queries to a knowledge base (see [RLN+95]).

Terminological logics have evolved from frames and semantic networks. One difference is that terminological logics offer a greater flexibility for formulating class descriptions. This syntactic flexibility makes it difficult to define a mapping of a terminological logic to any data model because there is no fixed concept structure. However, any class description formulated in FRM can be interpreted as a frame structure, i.e., as consisting of *slots* and *slot entries*. Thus, the FRM constructs **all-p** and **all-r** correspond to slots. We call **all-p** *property slots* and **all-r** *relationship slots*. The construct **exist-c** specifies a concept class as a slot entry and **exist-i** an individual as a slot entry. **exist-v** sets a value as a slot entry in a property slot.

Since the syntax of FRM as introduced above allows to introduce a slot entry without (explicitly) defining a corresponding slot, we must consider the implications shown in Figure 2 to properly interpret an FRM class descriptions as a frame with slots and entries. For example, the first implication given there states that the introduction of a slot entry (by the **exist-c** construct) implicitly introduces a slot (as expressed by the **all-r** construct). Thus, the following two class definitions would be semantically equivalent:

$g \doteq$ (**exist-c** manufactured-by big-company)

$h \doteq$ (**and** (**all-r** manufactured-by **thing**)
(**exist-c** manufactured-by big-company))

In Section 4 we assume the existence of a normalisation function *norm* that augments a class description with all implied features. For example, with respect to the class descriptions g and h above we get the equivalence $\varepsilon[norm(g)] = \varepsilon[norm(h)]$. The normalisation function covers many further implications not shown in Figure 2.

3 Basic Constructs of the Object Model COCOON

The constructs of the terminological logic FRM are mapped to the object data model COCOON and its associated language COOL [SLR+94]. COCOON resembles a functional data model in that object properties are modelled as single- and multi-valued functions. However, it also supports the dynamic grouping of objects into a class hierarchy based on predicates over object properties (cf. Fig.3).

The COOL query and update language is based on an algebra of operations over classes and can be considered as an extension of the nested relational algebra [ScS91]. The basic operations are *select*, *project*, *extend* (provides object type extension) and the set-based operations of *union*, *intersection* and *difference* (cf. Fig.6). The language also supports type guards for dynamic type checking.

Update operations may change the properties, class memberships and even the structure of database objects during their lifetime. Since COCOON allows objects to be grouped into classes based on their properties, objects are automatically reclassified within the class hierarchy after updates.

4 Mapping The Classification Inference

Figure 3 gives an example of our mapping of FRM concept descriptions to COCOON types and classes. Due to the limited space, the mapping is not described in this paper but we provide remarks where appropriate (for a detailed des cription see [RLN+95]). In the following, we give a brief description of how the classification inference of FRM is mapped to appropriate COCOON queries.

Let $\preceq$ denote the subsumption relation and let $\lhd$ be its transitive reduction. The concept hierarchy can then be conceived of as an undirected graph where the nodes represent all introduced concepts (C) and the edges represent the relation $\lhd$. Thus, classifying a concept c means to determine the following two sets:

$$L_c = \{l \in C \mid l \lhd c\} \qquad U_c = \{u \in C \mid c \lhd u\}$$

(exist-c r c**)**	implies	**(all-r** r **thing)**	
(exist-i r i**)**	implies	**(all-r** r **thing)**	
(exist-v p v**)**	implies	**(all-p** p $*$**)**	
(at-least rp n**)**	implies	**(all-r** rp **thing)**	if rp is a relation, **(all-p** rp $*$**)** else
(at-most rp n**)**	implies	**(all-r** rp **thing)**	if rp is a relation, **(all-p** rp $*$**)** else

Figure 2: Some of the Implications being Considered by a Normalisation Function for Class Descriptions

Sun-Del $\dot{\leq}$ **(and** **(all-p** costs $[0, 100000]$**)**
(at-most costs 1**)**
(all-r receives Company Person**)**
(at-most receives 1**)**
(all-r goods Workstation**)**
(all-r delivers Company**)**
(exist-i delivers Sun**))**

define type sun-del =
costs : **integer**,
receives : **objects**,
goods : **set of objects**,
delivers : **set of objects**;

define class Sun-Del : sun-del **where**
costs ≥ 0 **and** costs ≤ 100000 **and**
receives $\subseteq$ (Person $\cap$ Company) **and**
delivers $\subseteq$ Company **and**
Sun $\in$ delivers **and**
goods $\subseteq$ Workstation;

Figure 3: A Concept Class Introduction (left) and its corresponding type and class definitions (right)

The elements of L_c are called the *most general subconcepts* of c, and the elements of U_c the *most specific superconcepts* of c. As the computation of the two sets L_c and U_c is symmetric we only discuss the ca se of L_c. It can be computed by traversing the concept hierarchy bottom-up and determing all subconcepts of c that have no superconcept which is a subconcept of c. This traversal can be done by different variations of the common ıt depth-first search algorithm [BHN+92]. Apart from such modifications, the main algorithm used in existing systems is always the same: Classification is done by traversing the concept hierarchy while testing subsumption relations.

In our approach, we compute the set L_c completely differently. Instead of searching the concept hierarchy for the appropriate position we obtain L_c as the result of two COCOON queries:

1. The first query Q_1 yields all subconcepts of c: $L_c^+ = \{l \in C \mid l \preceq c\}$
2. The second query Q_2 yields the most general concepts from L_c^+: $L_c = \{l \in L_c^+ \mid l \lhd c\}$

To formulate Q_1 we first have a closer look at the subsumption relation of FRM. There is a well-defined set of update operations that, when applied to a concept c, lead to a more specific concept $\tilde{c}$, i.e. $\tilde{c} \preceq c$:

I: Concept Level (applicable to any concept):
- Add a new slot to the concept.

II: Slot Level (applicable to any slot of a given concept):
- In case of a property slot: Restrict the set of permitted entries to a subset. In case of a relationship slot (**all-r** r $c_1 \ldots c_n$): Remov e one or more of the range classes $c_1, \ldots, c_n$ and/or specialise a range class.
- Add further slot entries.
- In case of a relationship slot and a class occurring as an entry (**exist-c** construct), specialise this class, or substitute it by an instance of it, thus substituting the **exist-c** construct with an **exist-i** construct.
- Restrict the cardinality to a smaller interval.

We are now able to define the subsumption relation *syntactically* by referring to the concept descriptions (instead of the usual model-theoretic definition). To this end, we require for $c_1 \preceq c_2$ to hold that c_1 can be obtained from c_2 by applying one or more of the above operations. The corresponding definition (in a declarative fashion) is given in Figure 4. It makes use of the notation introduced in Table 1 and of the predicate $inst(i, c)$ which is true if i is an instance of the class c. The completeness of this subsumption definition very much depends on the normalisation function discussed in Section 2. iffi-cult to be handled. Since we are still working on the normalisation function, our subsumption algorithm is currently not complete.

Based on the syntactic definition of the subsumption relation it is now straightforward to formulate the COCOON query Q_1 that determines L_c^+ for a given concept description c. It consists of an intersection of mutually independent subqueries that can be computed concurrently. Each subquery deals with one of the slots of the concept to be classified. The resulting query schema for a slot s_i is shown in Figure 5[1]. As the whole condition (ISA) is mapped

[1]The query is formulated using functions defined for objects in the meta-schema, each object being a description of one object class in the COCOON database. We do not go into the details of the meta-schema here and use the function names of Table 1 with a subscript "ms" so that the correspondence to definition (ISA) can be seen. The functions *supc* and *subc* yield all superclasses,

$$c_1 \preceq c_2 \Leftrightarrow \forall s \in slots(c_2) : (s \in slots(c_1) \wedge prange(c_1, s) \subseteq prange(c_2, s) \wedge \quad \text{(ISA)}$$
$$\forall r_1 \in rrange(c_1, s) : \exists r_2 \in rrange(c_2, s) : r_1 \preceq r_2 \wedge$$
$$\forall e_2 \in entries\text{-}c(c_2, s) : (\exists e_1 \in entries\text{-}c(c_1, s) : e_1 \preceq e_2 \vee$$
$$\exists e_1 \in entries\text{-}i(c_1, s) : inst(e_1, e_2)) \wedge$$
$$entries\text{-}i(c_1, s) \supseteq entries\text{-}i(c_2, s) \wedge$$
$$entries\text{-}v(c_1, s) \supseteq entries\text{-}v(c_2, s) \wedge$$
$$minCard(c_1, s) \geq minCard(c_2, s) \wedge$$
$$maxCard(c_1, s) \leq maxCard(c_2, s))$$

Figure 4: Syntactic Definition of the Subsumption Relationship

$slots(c)$	$= \{ rp \mid c \dot{\leq}$ (**and** ... (**all-r** $rp\ c_1 \ldots c_n$) ...) or $c \dot{\leq}$ (**and** ... (**all-p** $rp\ range$) ...) $\}$
$rrange(c, s)$	$= \{ c_1, \ldots, c_n \mid c \dot{\leq}$ (**and** ... (**all-r** $s\ c_1 \ldots c_n$) ...) $\}$
$prange(c, s)$	$= r$, where $c \dot{\leq}$ (**and** ... (**all-p** $s\ r$) ...)
$entries\text{-}c(c, s)$	$= \{ c_e \mid c \dot{\leq}$ (**and** ... (**exist-c** $s\ c_e$) ...) $\}$
$entries\text{-}i(c, s)$	$= \{ i \mid c \dot{\leq}$ (**and** ... (**exist-i** $s\ i$) ...) $\}$
$entries\text{-}v(c, s)$	$= \{ v \mid c \dot{\leq}$ (**and** ... (**exist-v** $s\ v$) ...) $\}$
$minCard(c, s)$	$= n$, where $c \dot{\leq}$ (**and** ... (**at-least** $s\ n$) ...)
$maxCard(c, s)$	$= n$, where $c \dot{\leq}$ (**and** ... (**at-most** $s\ n$) ...)

Table 1: Functions for Accessing Parts of a (Normalised) Concept Definition (analogously for $\doteq$)

to a single query we can take full advantage of the query optimiser in the underlying database system.

Since classification is an inference on the structure of concept descriptions, this query schema accesses the *meta-schema*. As discussed in Section 4, most of the information about an FRM concept is encoded in the class predicate of the corresponding object class. However, the class predicate is just a string-valued attribute in the meta-schema and can only be queried as a whole. This means that certain structures of a concept description cannot be queried directly. Therefore, we extended the COCOON meta-schema by an application-specific part where we store the information about the concept classes in a well-structured way (in a certain sense, we model FRM in COCOON). While the meta-schema extension has to be administered by the mapping algorithm the "standard" part of the meta-schema is updated automatically by the COCOON system.

Figure 6 shows part of the COCOON query that returns the set of all subconcepts of 'Sun-Del' (see Figure 3) in the current knowledge base. This part completely deals with the slot 'delivers'.

As introduced above, query Q_2 of our classification inference is concerned with extracting the most general subconcepts L_c from the set of all subconcepts L_c^+. Assuming that the variable L holds the result from Q_1 (i.e., the set L_c^+), query Q_2 can be formulated in COCOON as: $\mathbf{select}[\emptyset = supc(l) \cap L](l : L)$.

5 Conclusions

We proposed to map terminological inferences to queries of an object data model. The resulting, complex queries can be split into several subqueries and evaluated independently. Thus, besides making use of more standard database optimisation techniques, like query optimisation and specialised access structures, we can also exploit parallelisation. We expect this implementation of subsumption to be much more efficient for large knowledge bases than a standard implementation, while for small knowledge bases the overhead introduced nd the database system will be greater than the efficiency gained by the optimisations.

We are currently implementing the mappings described in this paper. To provide efficient retrieval and update services the object model COCOON is mapped to a relational storage system which makes use of massive data replication (to minimise retrieval costs) and parallelisation of update operations (to minimise update costs). In a subsequent step, we will set up experiments to evaluate the efficiency gain and to pinpoint further possible improvements.

or subclasses, resp. The function $objects(c)$ returns all objects in the class c.

References

[BGL+91] S. Benzschawel, E. Gehlen, M. Ley, T. Ludwig, A. Maier, B. Walter: LILOG-DB: Database Support for Knowledge Based Systems. In: O. Herzog, C.-R. Rollinger (eds): Text Understanding in LILOG. Berlin: Springer-Verlag, 1991, pp.501-594.

[BHN+92] F. Baader, B. Hollunder, B. Nebel, H.-J. Profitlich, E. Franconi: An Empirical Analysis of Optimization Techniques for Termninological Representation Systems. In: B. Nebel, C. Rich, W. Swartout (eds): Principles of Knowledge Representation and Reasoning. Proc. of the Third Int. Conf., 1992, pp.270-281.

[HMM87] Th. Härder, N. Mattos, B. Mitschang: Abbildung von Frames auf neuere

$$
\begin{aligned}
&\textbf{select}[\emptyset \neq \textbf{select}[name(s) = name(s_i) \textbf{ and} \\
&\qquad prange_{ms}(s) \subseteq prange_{ms}(s_i)]\ (s : slots(c_{sub}))](c_{sub}\text{:Concepts}) \quad \cap \\
&\textbf{select}[\emptyset \neq \textbf{select}[name(s) = name(s_i) \textbf{ and} \\
&\qquad rrange_{ms}(s) = \textbf{select}[\emptyset \neq (supc(r) \cup r) \cap rrange_{ms}(s_i)](r : rrange_{ms}(s))] \\
&\qquad\quad (s : slots(c_{sub}))](c_{sub}\text{:Concepts}) \quad \cap \\
&\textbf{select}[\emptyset \neq \textbf{select}[name(s) = name(s_i) \textbf{ and} \\
&\qquad entries\text{-}c_{ms}(s_i) = \\
&\qquad \textbf{select}\ [\emptyset \neq (subc(e) \cup objects(e) \cup e) \cap entries\text{-}c_{ms}(s)](e : entries\text{-}c_{ms}(s_i))] \\
&\qquad\quad (s : slots(c_{sub}))](c_{sub}\text{:Concepts}) \quad \cap \\
&\textbf{select}[\emptyset \neq \textbf{select}[name(s) = name(s_i) \textbf{ and} \\
&\qquad entries\text{-}i_{ms}(s) \supseteq entries\text{-}i_{ms}(s_i)]\ (s : slots(c_{sub}))](c_{sub}\text{:Concepts}) \quad \cap \\
&\textbf{select}[\emptyset \neq \textbf{select}[name(s) = name(s_i) \textbf{ and} \\
&\qquad entries\text{-}v_{ms}(s) \supseteq entries\text{-}v_{ms}(s_i)]\ (s : slots(c_{sub}))](c_{sub}\text{:Concepts}) \quad \cap \\
&\textbf{select}[\emptyset \neq \textbf{select}[name(s) = name(s_i) \textbf{ and} \\
&\qquad minCard_{ms}(s) \geq minCard_{ms}(s_i)]\ (s : slots(c_{sub}))](c_{sub}\text{:Concepts}) \quad \cap \\
&\textbf{select}[\emptyset \neq \textbf{select}[name(s) = name(s_i) \textbf{ and} \\
&\qquad maxCard_{ms}(s) \leq maxCard_{ms}(s_i)]\ (s : slots(c_{sub}))](c_{sub}\text{:Concepts})
\end{aligned}
$$

Figure 5: Query Schema for a Slot s_i of a Concept Description c (Used to Determine all Subconcepts of c)

$$
\begin{aligned}
&\textbf{select}[\emptyset \neq \textbf{select}[name(s) = \text{'delivers' } \textbf{and} \\
&\qquad allr_{ms}(s) = \textbf{select}[\emptyset \neq (supc(r) \cup r) \cap \{\text{'Company'}\}](r : allr_{ms}(s))] \\
&\qquad\quad (s : slots(c_{sub}))](c_{sub}\text{:Concepts}) \quad \cap \\
&\textbf{select}[\emptyset \neq \textbf{select}[name(s) = \text{'delivers' } \textbf{and} \\
&\qquad exists\text{-}i_{ms}(s) \supseteq \{\text{'Sun'}\}]\ (s : slots(c_{sub}))](c_{sub}\text{:Concepts}) \quad \cap \\
&\textbf{select}[\emptyset \neq \textbf{select}[name(s) = \text{'delivers' } \textbf{and} \\
&\qquad maxCard_{ms}(s) \leq 1]\ (s : slots(c_{sub}))](c_{sub}\text{:Concepts}) \quad \cap \\
&\textbf{select}[\emptyset \neq \textbf{select}[name(s) = \text{'receives' } \textbf{and} \ldots
\end{aligned}
$$

Figure 6: Part of the COCOON Query that Yields all Subconcepts of Concept 'Sun-Del' in Figure 3

Datenmodelle. In: K. Morik (Ed): GWAI-87. 11th German Workshop on Artificial Intelligence. Berlin: Springer, 1987, pp. 396-405.

[KL89] M. Kifer, G. Lausen: F-Logic: A Higher-Order Language for Reasoning about Objects, Inheritance, and Scheme. In: Proc. of the ACM SIGMOD Int. Conf. on Management of Data, 1989, pp.134-146.

[NRL+94] M. C. Norrie, U. Reimer, P. Lippuner, M. Rys, H.-J. Schek: Frames, Objects and Relations: Three Semantic Levles for Knowledge Base Systems. In: Proc. Workshop on Reasoning about Structured Objects: Knowledge Representation meets Databases (KRDB-94), Saarbrücken, Germany, 1994, pp.53-57.

[Rei85] U. Reimer: A Representation Construct for Roles. In: Data & Knowledge Engineering, Vol.1, No.3, 1985, pp.233-251.

[RL95] U. Reimer, P. Lippuner: Syntax und Semantik von FRM. Working Paper, 1995, Information Systems Research Group, Swiss Life, CH–8022 Zurich.

[RLN+95] U. Reimer, P. Lippuner, M. Norrie, M. Rys: Terminological Reasoning by Query Evaluation: A Formal Mapping of a Terminological Logic to an Object Data Model. Extended Version. Technical Report, 1995, Information Systems Research Group, Swiss Life, CH–8022 Zurich.

[RS89] U. Reimer, H.J. Schek: A Frame-Based Knowledge Representation Model and its Mapping to Nested Relations. In: Data & Knowledge Engineering, Vol.4, No.4, 1989, pp.321-352.

[SB89] R. Studer, S. Börner: An Approach to Manage Large Inheritance Networks within a DBS Supporting Nested Relations. In: S. Abiteboul, P.C. Fischer, H.-J. Schek (Eds): Nested Relations and Complex Objects in Databases. Berlin: Springer, 1989, pp. 229-239.

[ScS91] M.H. Scholl, H.-J. Schek: From Relations and Nested Relations to Object Models. In: M.S. Jackson, A.E. Robinson (eds): Aspects of Database Systems. Butterworth-Heinemann, 1991, pp.202-225.

[SLR+94] M.H. Scholl, C. Laasch, C. Rich, H.-J. Schek, M. Tresch: The COCOON Object Model, Technical Report 211 (revised version), Dept. of Computer Science, ETH Zurich, CH-8092 Zurich.

Accessing Configuration-Databases by means of Description Logics

T. Kessel and **M. Schlick** and **O. Stern**
ERIC – ENSAIS, Strasbourg, France
email: ⟨kessel, schlick, stern⟩@steinway.u-strasbg.fr

1 Introduction

Description Logics (DL) has become one of the most interesting formalisms in the Knowledge Representation field [Woods and Schmolze, 1992]. There exists now a wide range of implemented systems (for example C3L, CLASSIC, KRIS, LOOM, BACK), a well studied theory with respect to expressive power (for instance the representation of time or uncertainty), inferential services and enhancements based on other paradigms (rules, constraints).

Until now, less attention has been paid to an integration of Knowledge Representation and *Data Base Management Systems* (DBMS) technology. The research in the first area was focused on the interpretation of the semantic links between data which transform them into knowledge, whereas the latter was in charge of handling large amount of data. In our point of view, both fields can be considered as complementary.

The interfacing between DBMS and Knowledge Representation systems may cover two topics:

1. The interface is exclusively focused on the exchange and transformation of data. For instance, information about a mechanical piece is retrieved from a CAD data base system and converted into the internal format of the DL system.
2. The DL system constitues a kind of application layer that is built on top of a DBMS. This top layer provides a maximum of inferential services and expressive power. Furthermore it allows the conception, optimisation or distribution of queries that are mapped to the underlying DBMS.

We propose to elaborate the second scenario because it combines the best of both worlds. Thanks to the embedded DL system, the application layer may use domain knowledge to optimise queries or to send it to one of the distributed DBMS, whereas the DBMS itself handles the storage, retrieval and recovery of data. For instance, our major application will be the configuration of modular bus systems from various, available modules for the electronic bus system of a vehicle, as described in [Keith *et al.*, 1995]. This research project requires a semantic integration of heterogeneous knowledge sources. They denote for instance, constraints which specify possible combinations of bus modules, rules which guide the configuration process by means of heuristics, concepts which describe the functional composition of electromechanical components within a motor and individuals that represent variants of the bus modules. The DL system we would like to employ in this context is the C3L system [Kessel *et al.*, 1995] which is implemented within a frame-based structure and smoothly integrated in an object-oriented programming and development environment.

At the moment we have no experience in the specific research field of combining DL and DBMS yet, although it seems to be very promising with respect to the above mentioned research project . The inferential services of a DL system, in particular the classification of concepts and the realization of individuals, are indispensable to structure such a large model as the description of a modular bus system in a vehicle. Open world reasoning seems appropriate to us, because we have to deal with incomplete knowledge which may evolve continually during the configuration process. Furthermore, an important feature of DL is the deduction of implicit information by means of propagation of recently added knowledge in the ABox. The implementation of a multilayered typology of user-programmed inferences, attached to concepts, which reason about individuals, like in classical knowledge-based system shells, for instance KAPPA or Nexpert Object, would be helpful to include procedural knowledge. A substantial contribution of DL comes from the power of its intelligent retrieval queries that allow, first to normalize completely different retrieval descriptions (which need not necessarily make use of the inheritance information), and second the generation of abstract concepts which are deduced from role resp. attribute values.

What may be the impact of such a project on our research about DL systems ? We suggest to focus on the study of retrieval functionalities provided by DL systems. This issue concerns essentially all kinds of ABox services, for instance the retrieval or realization of instances. Within this framework it is worth to examine the ABox's performance with respect to its architecture in order to reduce the average retrieval costs. What are the possible drawbacks of our approach ? The coupling between the DL system and the DBMS may induce some performance

losses, due to the intensive communication exchange between two different systems. Another aspect concerns the mapping of the complex concept or individual structures to a relational or object-oriented database scheme which has been undertaken yet.

The rest of the paper is structured as follows. A brief C3L system description is presented in the second section. The following chapter tackles the integration of procedural knowledge in a DL system, notably C3L. How to integrate an object-oriented DBMS within such a system, exemplified at C3L, is discussed in the fourth section. Afterwards the significance of the retrieval inference and related topics are studied. Applying a DL system for configuration purposes constitutes the principal issue of the sixth chapter. Last, but not least, we conclude our work.

2 C3L - a system description

At the moment, two different C3L versions exist: one academic research prototype, built in Common Lisp, and another version including an object-oriented data base management system and written in C++, but which is at the current state limited to the TBox. The later system is called C3L++ (for obvious reasons) and will serve as the implementation base for future development and enhancements. The porting of C3L from Common Lisp to C++ is motivated by the idea to scale small knowledge bases up to large ones, which require more sophisticated means for handling huge amounts of knowledge.

The C3L system may be characterized in some terms as follows:

- it is a descendant of the description logics (or KL-ONE) family
- it provides reasonable expressive power (e.g. conjunction, all, one-of, fills, at-least, at-most)
- it incorporates useful inferential services (e.g. subsumption, classification, recognition)
- it is implemented in a frame-based based system
- it contains declarative as well as procedural knowledge
- it is concieved in the perspective to serve as a knowledge representation module for a hybrid development environment (the term IKME, intelligent knowledge management environment, describes it best)

Special features of C3L which distinguish it from other description logics systems are:

- a reflexive object-oriented, frame-based architecture
- the integration of methods

The initial design philosophy of C3L was to combine ideas coming from the communities of frame languages and description logics [Carré *et al.*, 1995]. Having (partially) achieved this goal, we realized that it seems necessary to include the database community as well, in order to be able to handle large quantities of data without significant performance losses and keeping powerful reasoning mechanisms. Anyway, the description logics system C3L can be considered as a kind of application layer which hides the underlying database system and allows a completely transparent management of data for the user.

Two inhouse research projects constitute the testbed for the C3L system. The first is the domain modeling of a configuration system for a modular electronic bus system in vehicles, whereas the second addresses the representation of features in CAD and their links to technological information which are needed e.g. for production purposes. Both projects are rather small-scale research projects and still in progress, but they already provided us with very helpful feedback.

After having tested our system in the above mentioned two application scenarios, we identified two major requirements for the improved successor C3L++ of the academic research prototype C3L:

- high performance: the currently used data structures are not optimized for high performance, because it is an exploratory implementation
- large scale knowledge bases: due to memory restrictions the number of defined concepts, roles and individuals has been quite restricted so far

C3L enabled us to acquire a lot of valuable experience about description logics system design and building such architectures for information systems. A complete redesign of the C3L system is now in progress, a port of the TBox to C++ is the very first result of this effort. We think of testing it by means of a huge random knowledge base which will be automatically generated. Such an approach might allow to obtain reliable, empirical data of the systems performance and behaviour. A pre-version of C3L++ has showed a considerable increase of performance which will be studied in more detail in the near future.

3 Integrating procedural knowledge

Motivated by the requirements of studied applications and the need for an efficient manipulation of knowledge, we are actually working on the integration of C3L in an object-oriented programming environment. Obviously it is not satisfactory to provide simply methods, but to offer a formalism which enables the experienced user on the one hand to fulfill his particular needs and on the other hand to control the side effects of the methods. The objective is to support a multi-layered typology of methods whose consequences can be easily supervised and tested.

Suggestions for formalizing the notion of methods come from the area of specification languages for the development of knowledge-based systems in the field of knowledge engineering [Fensel and Harmelen, 1994]. The SCARP system [Willamowski, 1994], set on top of SHIRKA, influences as well our decision to include methods in form of *tasks*. Anyway the employed approach cannot deny the impact of classical ideas coming from the field of hierarchical and sceletal planning.

Tasks are on the one hand sufficiently complex, declarative means to abstract from simple methods

and on the other hand they are close to the implementation level by incorporating source code of the underlying programming language or by calling other subtasks. A task is defined by numerous properties or attributes which allow better validation and coherence tests of the task.

At the moment, three complementary categories of procedural attachments which are listed below are in work:

- methods: they represent general purpose attachments and are linked to concepts
- demons: they are triggered by value changes of roles, for instance if-added, if-deleted, if-changed, if-needed, are classical demons
- events: they survey the concepts instances and are launched if one instance fulfills the events conditional part

One major problem of procedural attachments are the consequences on the recognition and retraction inferences provided by the ABox. One possible solution is to execute demons first, list the concerned objects and pass them to the recognition service afterwards, in order to avoid costly recomputation of the individuals status during the chained manipulation of data. A more interactive approach may suppose that the user has to demand explicitly the (re-)recognition of an individual which was modified in the past.

Summarizing the benefits of embedding the underlying programming language in a description logics system may result in the following advantages:

- increased flexibility and maintenance of the complete system
- high performance for complex (mathematical) computations
- use of the programming languages high expressive power and performance

Offering an almost exhaustive library of primitive tasks (or methods) is the consequent next step of the evolution of our development environment. In most cases the user may only select the appropriate existing task or use pre-defined tasks which have to be instantiated. The programming effort would be reduced to calling tasks or profiting of the hierarchical task structure. Basic tasks are elementary retrieval or manipulation methods which constitute the principal task layer and which can be completed step by step.

One important feature we are working on is the automatic classification of tasks with respect to certain properties, for instance parameters or agents. The principal underlying idea is to map a task scheme to a normalized concept and to call the usual classification service. Afterwards you may reason about tasks like about normal individuals. This may be particular useful to support programmers who look for special properties of an incompletely specified task.

4 Integrating an object-oriented DBMS

The current C3L++ system which is ported to C++ is particulary optimized for performance issues and synthesizes the design experience we obtained in building the former Common Lisp version. One major system requirement is to get an almost platform independent description logics system.

After having evaluated the possibility to concieve a special DBMS interface for C3L++ within the systems architecture, which provides its own storage and caching strategies, we opted for a more commercial solution by building C3L++ on top of the POET [POE, 1995] database. POET is in fact a pre-compiler which generates (documented) C++ code and it takes the complete memory handling in charge. Therefore the system programmer can concieve the system as whether he had a large, but finite (virtual) memory space, persistent objects only have to be marked in their class definitions.

The chosen solution implies several benefits:

- the object-oriented DBMS system does what it can the best, e.g. memory caching strategies, even if they are not optimized for a description logics system and its specific inferences and data structures
- using such a DBMS simplifies the design, maintenance and documentation of source code with respect for data storage, enabling the system designer to focus on the essential system properties
- the solution is easy to implement and fast thanks to the employment of available classes and methods; we hope also to obtain a real gain of programming efficiency

The principal disadvantage of the solution is that there does not exist anymore a clear distinction between source code of the POET DBMS and of the description logics system. Both parts become very closely intertwined and inseparable.

Nevertheless we are convinced that the above mentioned solution lets us enough space for improving and adapting the C3L++ system for specific applications. Another aspect that was not discussed so far is the mapping from queries of the retrieval language to the underlying DBMS. All topics concerning this issue are studied in the appropriate retrieval section.

5 Significance of retrieval

In industrial applications databases are more often queried than updated. To fulfill this condition, efficient retrieval mechanisms have to be provided. In this context SQL-like query languages are often too difficult to learn for the average employee. As is known from different investigations, three out of four query attempts are non-successful, resulting in an immense loss of time and money. A possible solution to this problem is the usage of DL as a front-end to the database system. Here the user can communicate with the knowledge base by means of simple

operators and intuitively understandable object descriptions. The DL system can then optimize the queries and transform them into terms of the underlying database language. This step is completely transparent for the employee, resulting in an increasing acceptance of database applications.

To meet all requirements of database users, we have to provide two different classes of query operators. First, it must be possible to access the descriptions of database objects, for example to receive information on a specific mechanical component. Second, we need dedicated retrieval facilities to find objects by means of arbitrary descriptions. For example an engineer could be interested in finding all bus components transferring high data rates on a specific bus segment. In these cases an intuitive description of such a component is easily constructed, in comparison to the joining of several relations in a relational database system employing SQL.

In C3L the first class is represented by the operators **showall**, **show** and **ask**. Each of them occurs in three different contexts: for roles, concepts and individuals. A **showall**-operation returns a list of the elements of the specified type which are known in the database. The **show**-operator provides the most important properties of the object in question, for example the dependencies from other objects or the values of all its attributes. The **ask**-operator finally allows to specify the properties and attributes of interest for an object, for example if we are curious to know the data transfer rates of a special bus component. All these operations can be easily mapped to database queries without the need for dedicated reasoning mechanisms. In contrast, the operator for the second class, **search**, uses the inferential capabilities of the DL system. It is tightly connected to the retrieval inferences for roles, concepts and individuals. This operator takes an arbitrary object description and returns all database elements matching it.

The retrieval algorithms for roles and concepts can be constructed from the basic TBox inferences for subsumption and classification. The retrieval of ABox individuals is far more complex. In the following we will therefore concentrate on this mechanism. The processing of a query involves five steps [Stern, 1995]:

1. Analysis of the query.
2. Optimization of the query.
3. Choice of the appropriate resolution strategy.
4. Execution of a number of retrieval primitives following the chosen strategy.
5. Verification of the results.

Step four involves real database access, for example by means of SQL. But this is absolutely transparent for the user, the DL system is in charge of the whole transformation process.

The optimization phase first detects inconsistencies in the query. It then tries to simplify the description to accelerate the further processing. Numerical intervals, for example, are normalized and subsuming roles are eliminated.

The choice of the resolution strategy depends on a rather large number of properties of the query. If we are confronted with distributed databases, for example, we have to decide which ones are relevant for the query and when and how to access these knowledge sources. The strategy is also different for various formats of the query. Short ones are processed in another way then queries with lots of roles and attributes, and the treatment is distinct for queries comprising a conceptual description or lacking this. Furthermore, various types of conceptual parts are processed differently. By means of such distinctions we are able to reuse a maximum of already derived facts (recognition inference) to guarantee high performance of the retrieval inference.

To gain a maximum of speed there exists a huge number of retrieval primitives, implemented as independent methods. They can be freely combined or used as single mini-inferences. We have also provided primitives for the most frequent combinations of these basic methods. There are methods to access the different precomputed facts of the ABox and TBox, for example by evaluating the semantic indexing structure or the information inside the role hierarchy. We can classify the conceptual parts of a query or even generate appropriate concept descriptions from role lists if the given description seems not clear enough. And, what is self-evident, there are primitives to access the database interface of C3L which performs the transformation of basic queries into the database language.

As a result of step four we receive a set of individuals that possibly match the query. Due to limitations in the optimization and calculation steps, mainly to minimize the number of real database accesses, this set may contain elements that do not exactly match all role restrictions of the query. This makes a final verification step necessary. There we match the candidate instances with the possibly offended role restrictions by means of a dedicated subsumption algorithm for individuals. The verified objects are finally returned to the user.

The entire retrieval algorithm is correct and nearly complete. It is even more complete than the recognition inference of C3L. This could be achieved by deducing further implicit facts during step four of the algorithm. In all test cases so far, the retrieval inference was capable of calculating all the instances matching a query. Hopefully, there will be only very few cases where some implicit dependencies can not be detected.

The coupling between C3L and the database system can be described by employing a meta-model. This model comprises all the different aspects of the integration of a DL system with a database, like the construction of queries in terms of the database language, the distribution of queries in distributed environments, and the access methods of the database interface of C3L. By means of this model the interface to the database management system can be easily adopted to any commercial product. We can entirely avoid changements to the inference mechanisms. Only the mapping of basic queries of C3L to the database language has to be modified. In the

future we will also try to make use of already existing databases. To perform this difficult task, it will be necessary to extract generic concept descriptions and individual definitions from the database contents to use them for the queries. An automatic transformation seems, at the current state of our research, rather difficult if not impossible. But even an extraction by hand could be worth the trouble, compared to the benefits of using DL as a query component.

When we have a closer look on the meta-model, we can distinguish the different tasks of the DL system and the DBMS in our application scenario. The database system is only concerned with the storage of large amounts of data, whereas C3L is in charge of all problems involving some reasoning:

- Construction and verification of queries.
- Detection of inconsistencies.
- Optimization and distribution of queries.
- Generalization of queries.

The last point in this list is worth some more explanations. In the analysis step of the retrieval inference we can detect sub-queries that occur very often. A considerable speed-up for such queries can now be achieved be generalizing the sub-query, resolving it and caching the results. In subsequent queries these parts have not to be processed, it is sufficient to use the stored answers.

Concerned with industrial applications, for example in the domain of the configuration of bus systems for vehicles, we have learned that it is not sufficient to provide only system defined retrieval capabilities. Most applications show a need for dedicated facilities specially adopted to the domain in question. To meet this requirement, C3L can be extended by procedural knowledge. Within a syntactically and semantically regulated framework, the user can add retrieval inferences implemented in the host language. For this purpose, most of the retrieval primitives and optimization methods are accessible through a programmers interface. They can now be used to implement domain-specific retrieval functions and strategies. A little drawback of this approach is that these user programmed methods can lead to inconsistencies in the user defined query answering process. But the necessity of a careful implementation style seems to be a little inconvenience compared to the possibility to adopt the system to the special requirements of a real world application. Furthermore, this is a feature heavily missing in database-only systems that use, for example, SQL.

As already mentioned, the actual coupling of C3L with a database system is performed by means of a dedicated interface. The only purpose of this module is to perform the transformation between basic DL queries and queries in terms of the database language. We could identify a small number of such basic queries that are sufficient to provide C3L with all necessary information from the underlying database. A realization of this interface exists for relational databases that use SQL as their query language. Actual work is in progress for the integration of the object oriented database system POET. This OODBMS will function as the back-end data store in the C++ version of C3L. In our experience so far, the presented approach is well suited for relational DBMS as well as OODBMS.

6 Configuration as an application

Configuration can be defined as the design of a technical system, according to a specification, by choosing and assembling different modules taken from a module catalogue. If this is done by hand, especially for more complex problems, it often results in errors like inconsistency or missing parts. Therefore the aim is to develop a system to support the configuration process or to do configuration automatically. We propose a system based on Description Logics.

The problem of solving the configuration task by means of DL was already studied by the AT&T research group in the framework of the PROSE project [Wright *et al.*, 1993]. We want to focus on the advantages that Description Logics offers for the treatment of large amounts of data needed for the configuration process.

Databases are necessary to store the large module catalogues. It is important for the economic success of a configuration system that module descriptions of newly developed modules can be integrated in the DBMS as fast as possible. Using a normal DBMS, a domain specialist and a DBMS specialist are only together capable to formalize the information about a module and to add the resulting description to the DBMS. This results in a loss of time and money. Furthermore, consistency checking between the module descriptions is indispensable for the configuration process. Ordinary databases are not capable of performing this task.

The use of a Description Logics system as an application layer that is built on top of the DBMS can solve these problems. Domain and knowledge engineering experts have to work together to build the terminological part of the knowledge. In this part, the domain vocabulary and principles are described. Once done, this part only has to be changed if the description is no longer sufficient. In contrast to rare updates of the terminological knowledge the assertional knowledge has to be modified rather frequently, because all descriptions of the new modules are integrated as individuals. Because of the easier access methods of DL and the possibility to use the domain vocabulary, we expect that this could be done directly by the domain expert. This would make it possible to integrate the updating of the DB in the module development process. Additionally, the DL system automatically guarantees a maximum of consistency of the knowledge base.

A DL system does not only improve the management of a knowledge base. As described above, the use of retrieval and classification offers possibilities to accelerate the access to the different facts. For example a common problem during the configuration process is, to find a module which combines the properties of two or more other different modules. We could for example be interested in finding an integrated automobile motor management unit which integrates the ignition and injection management.

The problem is to retrieve a module description that fits to a list of various properties. With a normal DBMS such a search would be very expensive. With a DL system, the query strategy can be individually optimized which results in a higher performance and a better acceptance of the configuration system.

The use of a DL system also poses one problem: Is it possible to change between an open and a closed world assumption? The open world is convenient for the knowledge acquisition step, to enable the user to integrate new facts easily into the knowledge base. During the configuration process, a closed world assumption seems to be more adequate. If, for example, the configuration system excludes the first of two possible modules it can choose the second. An open world assumption would not allow this conclusion.

We have started to model the communication flow among different components linked to the electronical bus system of a vehicle. Modeling the domain using an object-oriented approach, like Description Logics, is more appropriate than conceptual modeling for DBMS. Domain experts have less problems to intuitively understand the resulting models.

7 Conclusion

In this paper we tried to motivate the benefits of coupling a DBMS system with a knowledge representation system, in particular the description logics system C3L++. The most important requirements for such a synergetic combination are:

- the need for large scale knoledge bases
- the potential performance gains

The implementation of C3L++ which incorporate such features is still in progress. Starting with a brief presentation of the academic research prototype C3L, we familiarize the reader with its idiosyncrasies, for instance the integration of procedural knowledge by means of methods, demons and events. Some decision criteria, for choosing an object-oriented DBMS (in our case: POET) and the reasons for setting C3L++ on top of it, are studied in the following section by emphasizing the system development aspect. Discussing the impact of retrieval for our configuration application and its consequences for the DBMS coupling form the major topics of the successive part. Finally, some problems posed by applying description logics to the configuration of electronic bus systems are elaborated.

8 Acknowledgments

We would like to thank the anonymous referees for their valuable hints on earlier versions of this paper, and our colleagues for very fruitful discussions on the subject. In particular we owe a lot to the Robert Bosch company, Germany, with who we cooperate in the research project on configuration of modular bus systems for vehicles.

References

[Carré *et al.*, 1995] B. Carré, R. Ducournau, et al. Classification et objets: programmation ou représentation. In *PRC-GDR Intelligence Artificielle*. TEKNEA, 1995. In French.

[Fensel and Harmelen, 1994] D. Fensel and F. Harmelen. A comparison of languages which operationalize and formalize KADS models of expertise. *The Knowledge Engineering Review*, 9(2):105–146, 1994.

[Keith *et al.*, 1995] B. Keith, T. Kessel, M. Schlick, and O. Stern. A description logics based approach to the configuration of diagnostic systems. In *Proceedings of the IAR conference on Automatic Control and Signal Processing*, 1995. Forthcoming.

[Kessel *et al.*, 1995] T. Kessel, F. Rousselot, and O. Stern. From frames to concept: Building a concept language within a frame-based system. In *Proceedings of the International Description Logics Workshop at Rome*, 1995.

[POE, 1995] POET Software Corporation, San Mateo. *POET Release 3.0*, 1995.

[Stern, 1995] O. Stern. Entwicklung der assertionalen Komponente ERICA für das terminologische Wissensrepräsentationssystem C3L. Master's thesis, Universität Karlsruhe (TH), 1995. In German.

[Willamowski, 1994] J. Willamowski. *Modélisation de tâches pour la résolution de problèmes en coopération système-utilisateur*. PhD thesis, université Joseph Fourier - Grenoble 1, 1994. In French.

[Woods and Schmolze, 1992] W. Woods and J. Schmolze. The kl-one family. In *Semantic Networks in Artificial Intelligence*, pages 133–177. Pergamon Press, 1992.

[Wright *et al.*, 1993] J. Wright, E. Weixelbaum, et al. A knowledge-based configurator that supports sales, engineering and manufacturing at AT&T Network Systems. In *Proceedings of the Innovative Applications of Artificial Intelligence Conference*, 1993.

The P-type Model : from Databases to Knowledge Bases

Ana Simonet and Michel Simonet
Laboratoire TIMC-IMAG,
38706 La Tronche cedex - FRANCE
e-mail : (Ana,Michel).Simonet@imag.fr

1 The P-type Model

The p-type data model was conceived in the early eighties as an answer to database needs [12]. It was expressed within the Algebraic Data Types (ADT) paradigm [7] [8] and its main concern was the sharing of objects by several kinds of users seeing them through one or several views. A p-type is organized in a hierarchy of classes, where classes model database views. An object belongs to one and only one p-type, and to several views. Multiple specialisation is not necessary to express that an object belongs to several subclasses (views of a p-type). It is used only to specify a subset of the views intersection.

To specify a p-type one first gives its minimal view then its other views by simple or multiple strict specialisation, adding attributes and/or assertions. The root of the hierarchy of views is called the minimal view in that all the objects of the p-type must satisfy its properties. The ADT of a p-type is derived from its views declaration. This type contains all the attributes and methods which appear in the views of the p-type, including the minimal view. An object belongs to a view iff it satisfies its assertions. Objects which are instances of a p-type may belong to several views, among which only the minimal view is mandatory.

A p-type is defined as an algebraic data type $< S, F, E >$ where S is a set of sorts $\{s_1, ..., s_n\}$, the carrier of the type, F a set of functions $s_i \times s_j \times ... \times s_k \rightarrow sl$ and E a set of equations [12]. One sort, T, called the set of interest of the type, is central, in that the aim of the type definition is to establish the elements of the type and define their behaviour. In general, the type is given the name of its set of interest : T. Among all possible functions, we call attributes those of the form $T \rightarrow s, s \in S$. Other functions are called methods.

The algebraic type of the p-type is derived from the views declarations (including the minimal view). The type *PERSON* contains all the attributes and methods which appear in its views. The domain of an attribute in type *PERSON* is the union of its domains in the views where it is declared.

Let $t_{min} : < S, Fmin, Emin >$, $t_1 : < S, F_1, E_1 >$, $t_2 : < S, F_2, E_2 >$, ... be the views of a p-type T. T is defined as $< S, F, E >$ where S is the support set of T, $F = \bigcup_i F_i$ and $E = E_{min}$.

As a simple example, consider a p-type PERSON whose minimal view has the attributes Name, Age, and Sex, and its different views are ADULT : PERSON (Age≥ 18), SENIOR : PERSON (Age>65), and STUDENT, characterized by specific attributes. In the graph presented in figure 1, a student aged between 18 and 65 belongs to the views PERSON (the minimal view is mandatory), STUDENT, and ADULT, provided it satisfies the properties of these views.

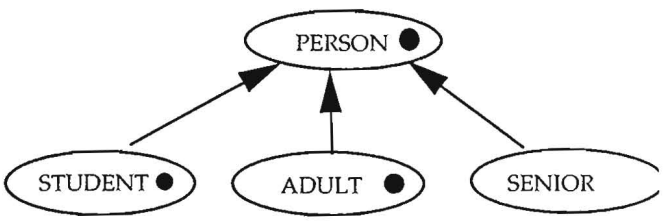

Figure 1: Graph of p-type *PERSON*

The set of interest (domain) of the minimal view person is identical to that of the p-type *PERSON*. The domain of another view is a subset of the domain of the view it specializes, or of the intersection of the domains of the views it specializes in case of multiple specialization.

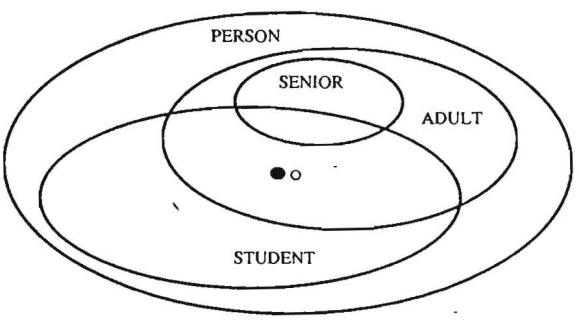

Figure 2: Inclusion set of p-type *PERSON*

In the general case, any view may be a strict specialisation of one or more views, and have its own attributes and/or assertions. Assertions are Horn clauses with literals of the form $\forall x$ *Attribute(x) in Domain*, called Domain predicate. An example of such an assertion is *Age(x)> 18 → MilitaryService(x) in {done, deferred, exempt}*. Assertions reduced to

a single Domain predicate, such as *Age(x) in [18, 65]* may stand for an attribute domain definition in a view.

Unlike most OODBMS such as the O2 proposal [9], attributes whose values are calculated by a method (or a procedure) are true attributes, and therefore are not themselves considered as methods. Any attribute may be stored or not, and may be calculated or not. A calc-stored attribute is calculated from the values of other attributes (e.g., Age from BirthDate and CurrentDate) and automatically updated whenever necessary.

2 An Example

A base schema is made up of several p-type definitions. In general, these p-types are not independent. In OSIRIS the interrelationships between different p-types of a schema are expressed by attribute definitions and by Inter-Object Dependencies (IODs).

We present the main features of the p-type description language and of the Inter-Object Dependencies through a very simple OSIRIS example. The universe modelled is that of persons and vehicles. Persons may be and/or students, teachers, trainee-teachers, professors, sportsmen. They are also either adults or minors according to their age. A given person is a model of the minimal view and may belong to none, any or several other views. The view TRAINEE, which inherits STUDENT and TEACHER, is not necessary to express that a person can be a student and a teacher at the same time. It has been created to designate a subset of their intersection, characterized by some more assertions, which restrict its domain.

```
class PERSON – Minimal view of p-type PERSON
attr
 Name : P_NAME; – P_NAME is declared elsewhere
  Children : setof PERSON;
  Sex : CHAR;
  Age : INT;
  MilitaryService : STRING;
  IncomeTax : REAL calc; – procedural attachment
  CarsOwned : setof CAR;
            – CAR is a view of a p-type VEHICLE
key  Name – External key
methods – other functions specification
assertions
– Domain Assertions
   Sex in { "f", "m" };
   0 ≤Age ≤ 120
   MilitaryService in
     { "yes", "no", "deferred", "exempt" };
– Inter-Attribute Dependencies
   Age < 18 ⇒ MilitaryService = "no";
   Age ≥ 18 ⇒ MilitaryService in
     { "yes", "deferred", "exempt" };
   Sex = 'f' ⇒ MilitaryService = "no";
end;
```

The minimal view automatically contains a private attribute OID : t_{oid}.

```
view STUDENT : PERSON ...
view TEACHER : PERSON ...

view PROFESSOR : TEACHER ...

view TRAINEE : STUDENT , TEACHER
       – specializes STUDENT and TEACHER
assertions
  Status = "trainee";
  Studies = "graduate";
  Diplomas contain  "degree";
end;
⋮
view ADULT : PERSON
assertions
  Age ≥ 18;
end;
view SENIOR : ADULT
assertions
  Age > 65;
end;

  implementation PERSON
    – stored attributes
    – body of methods
end;
```

The attributes of the type *PERSON* are those of the minimal view, PERSON, plus those defined in other views : Studies, Year, Status, Diplomas. Within a given view, the user may only access the attributes inherited from its super-views and the attributes proper to the view, if any.

Objects which are instances of the p-type *PERSON* may satisfy one or several views, among which only the minimal view is mandatory.

A part of the description of the p-type *VEHICLE* migh be :

```
class VEHICLE
attr
  Type : STRING;
  Year : DATE;
...
assertions
  Type in { "car", "truck", "bus", "tractor"};
end;

view CAR : VEHICLE
attr
 Owner : PERSON;
...
assertions
  Type = "car";
end;
```

Within the scope of the definition of p-type *PERSON* and view CAR of p-type *VEHICLE*, the interrelationships between cars and persons are expressed through the attributes CarOwner and Owner of the p-types *PERSON* and *VEHICLE* respectively. To express that these two attributes are reciprocal, one writes an Inter Object Dependency :

PERSON.CarsOwned reverse CAR.Owner

CarsOwned in p-type *PERSON* being declared as the reverse function of Owner in p-type *VEHICLE*,

the OSIRIS system ensures integrity maintenance. In particular, every car whose owner is a person X must belong to the set of cars of X. For example, suppressing a car Y with owner X implies that Y no longer belongs to the set of cars owned by X. Similarly, adding a car Y with Owner X would trigger the checking that Y belongs to the set of cars owned by X, and adding it if necessary. Thus referential integrity is checked and automatically maintained. This deductive aspect (deducing a new CarsOwned value from the insertion of a new car) is also present in Inter Attribute Dependencies (e.g. value "no" for MilitaryService can be deduced from an Age less than 18).

When modelling the universe of persons, i.e., characterizing its subclasses, the modeller has to make choices. For example, the SENIOR view can be defined as an ADULT whose Age is > 65, or as a PERSON with the same constraint on the age. Both views would be considered equivalent by the Osiris system. However, different consequences might result from either choice. If the view ADULT is modified, e.g., enriched with some new property, the view SENIOR will inherit this property only if it has been explicitly defined from the view ADULT or any subview of it.

3 The Classification Space

The key to implementation is definition of the partitioning of the object space based on the Domain Predicates of the p-type. Each Domain Predicate defines a partitioning of the attribute it covers. The product of partition of an attribute by all the predicates of the p-type [13] [14], determines a partitioning of the domain of that attribute into Stable-Sub-Domains (SSD). An instance whose attribute values change within the same SSD satisfies the same Domain Predicates, hence the same assertions. This is the stability property on which the whole system relies.

In the example given above, the partitioning of the attribute domains is :

$$\text{Domain (Age)} = d_{11} \bigcup d_{12} \bigcup d_{13}$$
$$\text{Domain (MilitaryService)} = d_{21} \bigcup d_{22}$$
$$\text{Domain (Sex)} = d_{31} \bigcup d_{32}$$

where

d_{11} = [0, 18[, d_{12} = [18, 65], d_{13} =]65, 120]
d_{21} = {"no"},
d_{22} = {"yes", "deferred", "exempt"}
d_{31} = {"m" }, d_{32} = {"f"}

By definition, each subdomain d_{ij} has the following property : when the value of attribute $Attr_i$ changes within the subdomain d_{ij}, all domain predicates maintain their truth value and consequently the assertions do likewise. Divisions d_{ij} are therefore stability zones for the assertions, hence their name : Stable Subdomains (SSDs). Domain Predicates are transformed into elementary predicates of the form $Attr_i \in d_{ij}$, where the d_{ij} are the SSDs of $Attr_i$. Introducing a new assertion with predicate Age > 40 would cause the splitting of d_{12} into [18, 40] and]40, 65], and the corresponding internal rewriting of the concerned assertions.

The partitioning of each attribute domain is extended to the object space. This partitioning, whose elements are named Eq-classes, is called the Classification Space. It is the quotient space of the object space with respect to the equivalence relation *'satisfy the same subset of Domain Predicates'*. The classification space of a p-type is the cartesian product of the sets of Stable Subdomains of its classifying attributes. Elements of the classification space are called Eq-classes. For a p-type with n classifying attributes [1], Eq-classes are n-tuples $(d_{1i}, d_{2j}, \ldots d_{nl})$.

The classification space can be illustrated in a 3-dimensional space by the figure shown figure 3, obtained by considering only attributes Age, Military-Service and Sex, and the domain constraints above, leading to a partitioning into 3x2x2 = 12 Eq-classes.

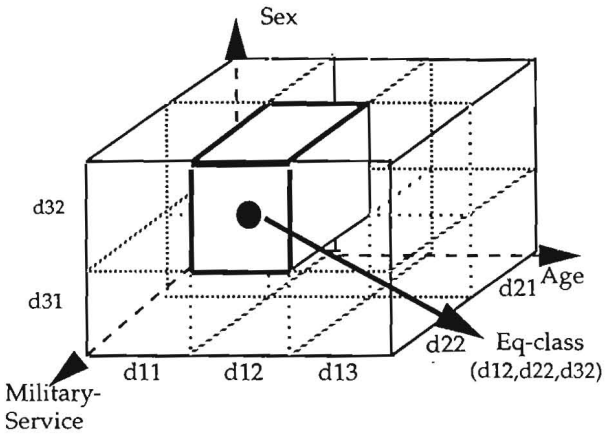

Figure 3: Space partitioning

It is possible to determine at compile time the set of views to which each Eq-class belongs. Classifying an instance, i.e., determining the set of views to which it belongs, is no longer performed by following the hierarchy of views. Classification is performed by a boolean (propositional) solver, based on the structure of the Classification Space, in time linear to the number of views and of SSD, and polynomial to the number of attributes and p-type assertions [2]. When the object is not completely known, its known attribute values determine several Eq-classes instead of a single one. These Eq-classes determine which views are valid, invalid, or potential, i.e., views whose validity still depends on missing attribute values.

Eq-classes are never represented explicitly in their totality. They up hold and direct the compilation process, and at execution time, they index the actual objects of the database. Ther number of actual Eq-classes is therefore limited by the objects which are really entered in the base [14].

Primary indexing through Eq-classes also enables semantic query optimization. The query (PERSON | Age < 30) would automatically select individuals from Eq-classes having SSD d_{11} as a component, and reject those corresponding to d_{13}. Only the elements of those indexed by d_{12} (18≤Age≤65) have to be checked for the condition Age < 30.

[1] Classifying attributes are attributes whose domain is partitioned in at least two SSDs.

4 Databases vs Knowledge Bases

Besides security and the ability to efficiently manage large quantities of data, concurrency and data sharing are important features of databases. In typical database applications, an object is assigned one class and the database has to deal with further evolution of its attribute values. In a knowledge base, objects are often not completely known, and object evolution mainly consists in the determination of unknown attribute values, but rarely in value changes. The objective is to obtain the most refined information about the object, including its valid and potential classes, deduced attribute values or value ranges, and explanations about all inferred information.

Another important feature of p-types is that an object can belong to several views and change views (not its p-type) in its lifetime, whereas is OODBMS instanciation is made in one class and is final, as in programming languages. Belonging to a view is a property which is defined as satisfying the assertions of the view, i.e., both its proper and inherited assertions. A mandatory assignation to a view, as is usually the case in a database situation (create p as V) will cause the assertions of V to be verified, considering them as integrity constraints. The way view determination has been designed of p-types [14] consists in determining all the valid views of a given object, extending this determination to that of those possible when the object is not completely known. This process is the very process of instance classification in knowledge bases.

Dealing with incomplete information is an important aspect of knowledge bases. In Osiris, it may happen that incomplete information leads to an absolutely certain conclusion, without having to make hypotheses about unknown values of attributes. In some way, all possible hypotheses have been "compiled" through the Eq-classes. When probabilistic information is available about the distribution of the values of the attributes in its SSDs, the classifying process is able to evaluate the probability assigned to each view when some attributes have unknown values [2]. Classifying a completely known instance is then a particular case of the probabilistic classification : the SSD of a known attribute value has a probability value 1 and the others 0. As a result of classification, views known to be certain have a probability 1, and those impossible a probability 0. When the actual probability of SSDs is not known, assigning to them an arbitrary probability value (e.g., equi-probability), will lead to 0, 1, and non-zero-one values, still characterizing impossible, certain, and potential views. However, in this case, the probability value is not significant and only indicates that the view is potential (i.e., neither certain nor impossible).

The consistency of the base is ensured by the integrity constraints expressed by the assertions. Integrity constraints verification is a by-product of the classification process. In effect, classifying an object in a given view means that the object is a valid interpretation of its assertions. When the user assigns an object to a given view, which is the usual situation in databases, checking the integrity constraints of that view is performed by checking that this view belongs to the objects views.

Other consistency aspects may be considered in a knowledge base context : class validity and assertion contradiction. We also define Domain-inconsistency which is weaker than logical inconsistency and indicates a probable distortion between several assertions (possibly written by several users).

Within a p-type a view may be defined with assertions which make it inconsistent, i.e. no object instance of the p-type can be a model of its assertions (inherited and proper assertions). This is detected by an empty set of valid Eq-classes for the view.

Assertions can be checked for logical inconsistency, which is possible in spite of their first order general form, because the static process enables their transformation into an equivalent set of propositional formulas. Assertions :

a1: Age< 18 ⇒ MilitaryService = "no"
a2: Age ≥ 18 ⇒ MilitaryService ∈ {"yes", "deferred", "exempt"}
a3: Sex = "f" ⇒ MilitaryService = "no"

may be transformed into a propositional system where attributes are implicitly universally quantified, and where p_{ij} is the proposition expressing that attribute $Attr_i$ is in SSD d_{ij} :

a1': $p_{11} \Rightarrow p_{21}$
a2': $p_{12} \bigvee p_{13} \Rightarrow p_{22}$
a3': $p_{32} \Rightarrow p_{21}$

along with propositions of the form

$p_{ij} \Rightarrow$ not p_{ik} for all k ≠ j

expressing the mutual exclusion of stable subdomains for the same attribute :

$$(\forall i) d_{ij} \bigcap d_{ik} = \emptyset \text{ for all } k \neq j.$$

Domain inconsistency is weaker than logical inconsistency. An assertion is said to be domain-inconsistent when its antecedent is always invalidated by other assertions of the type. In the context of the above example, the assertion 'Sex = "f" and $Age > 30 \Rightarrow$ some conclusion' is always valid, whatever its conclusion, because its antecedent is always false, being contradictory to assertions a1-a3, which impose that there cannot be any female aged over 18 in the base [2]. Assertion a2 should have been written : $Age \geq 18$ and $Sex = 'm' \Rightarrow$ $MilitaryService \in \{"yes", "deferred", "exempt"\}$. One can assume that such Domain-inconsistent assertions are not written deliberately and their detection is essential to the designer. Once they have been detected, it is up to the user to decide whether to maintain them or not. Domain-inconsistencies may be intended by the programmer; they may be harmless, but they may have unwanted hidden consequences, hence the interest of their detection.

P-types were designed in a database perspective and the Osiris implementation fulfills the usual database requirements. Persistency, transactions, concurrency, etc., are provided through the use of a set of persistent C++ classes (called the Osiris

[2] This is due to assertion a2: $Age \geq 18 \Rightarrow$ $MilitaryService \in \{"yes", "deferred", "exempt"\}$

kernel) which will be implemented in two ways : by an object manager [1] and a relational database [10][11]. The relational version of the kernel will implement data sharing [3] and a nested transaction mechanism similar to that described in [4]. The main objective for a relational implementation was to inherit the qualities of the second generation relational DBMS. Among these, efficient storage of large data volumes, concurrency control, and confidentiality management.

5 Conclusion

To conclude, we add that the p-type data model resembles more nearly Terminological Logics which can classify an instance into several concepts, than the data model of most OODBMS in which an instance must be created in exactly one class and cannot change its class in its lifetime [4]. Work remains to extend OSIRIS to view subsumption, which may be expressed as the inclusion of sets of Eq-classes in the Classification Space. The complexity of view subsumption with respect to the class of assertions taken into account, i.e., Horn clauses with Domain Predicates as literals, is still to be evaluated.

Although no commercial OODBMS has until now incorporated a view mechanism, the idea that views need to be included is becoming widely accepted. In 1992, E. Bertino acknowledged that "several questions about a suitable view model for OODBMS still need to be addressed in current research" [3].

Views are a primary concept in p-types, and are not superimposed to a given object model. A p-type is a semantic unit for the grouping of subclasses, namely views. A real world entity is instanciated in one and only one p-type, and may belong to several views : those of which it satisfies the properties. Grouping subclasses as views of a p-type is the corollary of considering the unity of the object, which is indeed the basis of object modelling. A person is unique, whether considered as a student, a sportsman, an adult, etc. In P/FDM, a prolog-based implementation of a functional data model, a given object may also be instanciated in several subclasses, with the same OID [6]. P. Gray remarks that this approach is equivalent to views, which we acknowledge.

We would also like to mention Date's opinion that "the process of inserting a row can be regarded as a process of inserting that row into the database (rather than into some specific table)"[5]. In an object-oriented perspective, this argues well for automatic classification of objects in views.

References

[1] E. Abecassis, *YOODA user's guide*, APIC systeme Arcueil, France, 1994.

[2] C. G. Bassolet, *Reseaux de Neurones de Classement dans le modele des p-types*, Rapport de Recherche IMAG, Grenoble, to appear.

[3] E. Bertino, M. Negri, G. Pelagatti, L. Sbattella, Object-Oriented Query Languages : The Notion and the Issues, IEEE Trans. on Knowledge and Data Engineering, Vol. 4, No 3, June 1992.

[4] R. G. G. Cattell, *The Object Database Standard : ODMG-93*, Morgann Kaufmann Publishers, 1994.

[5] C. J. Date and David McGoveran, *A New Database Design Principle*, In Database Programming and Design, July 1994.

[6] P. Gray, G. Kemp, *Object-Oriented Systems and Data Independence*, OOIS'94, London, Dec. 1994.

[7] J. Guttag, J. Horning, *The Algebraic Specification of Abstract Data Types*, Acta Informatica, 1978.

[8] B. Liskov, B. Zilles, *Programming with Abstract Data Types*, Proc. of a Symp. on Very High Level Language, Sigplan Notices 9, 4, April 74.

[9] O. Deux et al., The O2 System, CACM, October 1991, Vol 34, No. 10, pp 34-48.

[10] M. Quast, *Osiris et le modele relationnel*, Memoire d'ingenieur CNAM, to appear, TIMC-IMAG, 1995.

[11] M. Quast, A. Simonet, M. Simonet *A Relational implementation of a View-based Object System*, OOIS'95, Dublin, Dec. 1995 (accepted).

[12] A. Sales-Simonet, *Types abstraits et bases de donnees : formalisation du concept de partage et analyse statique de contraintes d'integrite*, These de Docteur-Ingenieur, USMG, Grenoble, 1984.

[13] A. Simonet, M. Simonet, *Objects with Views and Constraints : from Databases to Knowledgebases*, OOIS'94, London, Springer Verlag, 1994.

[14] A. Simonet, M. Simonet, *Osiris : an OO system unifying databases and knowledge bases*, KBKS'95 (Building and Sharing of Very Large-Scale Knowledge Bases), p217-227, IOS Press, 1995.

[3] The P of p-types stand for the french *partage*, which means *shared*. P-types were conceived to be shared by several users, while groups of users might have their own views of the set of objects represented by the p-type

Increasing the Power of Structured Objects

Diego Calvanese and Giuseppe De Giacomo and Maurizio Lenzerini
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy
{calvanese,degiacomo,lenzerini}@dis.uniroma1.it

1 Introduction

We have recently proposed a new object-oriented data model, called $\mathcal{CVL}$ (for Classes, Views, and Links), that extends the expressive power of known formalisms in several directions by offering the following possibilities:

- To specify both necessary and sufficient conditions for an object to belong to a class; necessary conditions are generally used when defining the classes that constitute the schema, whereas the specification of views requires to state conditions that are both necessary and sufficient [1]. With this feature, supported in $\mathcal{CVL}$ through class and view definitions, views are part of the schema and can be reasoned upon exactly like any class.
- To specify complex relations that exist between classes, such as disjointness of their instances or the fact that one class equals the union of other classes.
- To refer to navigations of the schema while defining classes and views; in particular, both forward and backward navigations along relations and attributes are allowed, with the additional possibility of imposing complex conditions on the objects encountered in the navigations.
- To specify relations that exist between the objects reached following different links; in particular, to specify that the set of objects reached through an attribute A is included in the set of objects reached through another attribute B, thus imposing that A is a subset of B.
- To use (n-ary) relations with complex properties and to declare keys on them.
- To impose cardinality ratio constraints on attributes.
- To model complex, recursive structures, simultaneously imposing several kinds of constraints on them. This feature allows the designer to define inductive structures such as lists, sequences, trees, DAGs, etc..

One of the most important aspects of the model we propose is that it supports several forms of reasoning at the schema level. Indeed, the question of enhancing the expressive power of object-oriented schemas is not addressed in $\mathcal{CVL}$ by simply adding more and more constructs to a basic object-oriented model, but by equipping the model with reasoning procedures which are able to make inference on the new constructs. Notably, we have shown that the main reasoning task in $\mathcal{CVL}$, namely checking if a schema is consistent, is decidable, by providing a sound and complete algorithm that works in worst-case deterministic exponential time in the size of the schema. Such worst-case complexity is inherent to the problem, proving that consistency checking in $\mathcal{CVL}$ is EXPTIME-complete.

2 The $\mathcal{CVL}$ data model

In this section we formally define the object-oriented model $\mathcal{CVL}$, by specifying its syntax and its semantics.

2.1 Syntax

A $\mathcal{CVL}$ *schema* is a collection of class and view definitions over an alphabet $\mathcal{B}$, where $\mathcal{B}$ is partitioned into a set $\mathcal{C}$ of *class* symbols, a set $\mathcal{A}$ of *attribute* symbols, a set $\mathcal{U}$ of *role* symbols, and a set $\mathcal{M}$ of *method* symbols. We assume that $\mathcal{C}$ contains the distinguished elements `Any` and `Empty`[1]. In the following C, A, U and M range over elements of $\mathcal{C}$, $\mathcal{A}$, $\mathcal{U}$ and $\mathcal{M}$ respectively.

As we mentioned before, for defining classes and views we refer to complex links which are built starting from attributes and roles. An *atomic link*, for which we use the symbol l, is either an attribute, a role, or the special symbol $\ni$ (used in the context of set structures). A *basic link* b is constructed according to the following syntax rule, starting from atomic links:

$$b ::= l \mid b_1 \cup b_2 \mid b_1 \cap b_2 \mid b_1 \setminus b_2.$$

Two objects are connected by $b_1 \cup b_2$ if they are linked through b_1 or b_2, whereas two objects are connected by $b_1 \cap b_2$ ($b_1 \setminus b_2$) if they are linked through b_1 and (but not) by b_2. Finally, a generic *complex link* L is obtained from basic links according to:

$$L ::= b \mid L_1 \cup L_2 \mid L_1 \circ L_2 \mid L^* \mid L^- \mid \underline{\text{identity}}(C).$$

Here, $L_1 \circ L_2$ means the concatenation of link L_1 with link L_2, L^* the concatenation of link L an arbitrary finite number of times, and L^- corresponds to link L taken in reverse direction. The use of $\underline{\text{identity}}(C)$

[1] We may also assume that $\mathcal{C}$ contains some additional symbols such as `Integer`, `String`, etc., that are interpreted as usual, with the constraint that no definition of such symbols appears in the schema.

is to verify if along a certain path we have reached an object that is an instance of class C.

Usually, in object-oriented models to every class there is an associated type which specifies the structure of the value associated to each instance of the class. In $\mathcal{CVL}$, objects are not required to be of only one specified type. Instead, we allow for polymorphic entities, which can be viewed as having different structures corresponding to the different roles they can play in the modeled reality. Therefore we admit rather rich expressions for defining structural properties. A *structure expression*, denoted with the symbol T, is constructed as follows, starting from class symbols:

$$T ::= C \mid \neg T \mid T_1 \wedge T_2 \mid T_1 \vee T_2 \mid [A_1{:}T_1, \ldots, A_n{:}T_n] \mid \{T\}.$$

The structure $[A_1{:}T_1, \ldots, A_n{:}T_n]$ represents all tuples which have at least components $A_1, \ldots, A_n$ having structure $T_1, \ldots, T_n$, respectively, while $\{T\}$ represents sets of elements having structure T. Additionally, by means of $\wedge$, $\vee$, and $\neg$, we are allowed not only to include intersection and union in structure expressions (as in [2]), but also to refer to all entities that do not have a certain structure. Note that, since we allow for entities having multiple structure, intersection cannot be eliminated from the definition of structure expressions (contrast this property with the model presented in [2]).

Class and view definitions are built out of structure expressions by asserting constraints on the allowed links and by specifying the methods that can be invoked on the instances of the class. A class definition expresses necessary conditions for an entity to be an instance of the defined class, whereas a view definition characterizes exactly (through necessary and sufficient conditions) the entities belonging to the defined view. Our concept of view bears similarity to the concept of *query class* of [14].

Class and *view definitions* have the following forms (C is the name of the class or of the view):

class C	view C
structure-declaration	*structure-declaration*
link-declarations	*link-declarations*
method-declarations	*method-declarations*
endclass	endview

We now explain the different parts of a class (view) definition.

(i) A *structure-declaration* has the form

$$\text{is a kind of } T$$

and can actually be regarded as both a type declaration in the usual sense, and an extended ISA declaration introducing (possibly multiple) inheritance.

(ii) link-declarations stands for a possibly empty set of *link-declarations*, which can further be distinguished as follows:

– *Universal-* and *existential-link-declarations* have the form

$$\text{all } L \text{ in } T \quad \text{and} \quad \text{exists } L \text{ in } T.$$

The first declaration states that each entity reached through link L from an instance of C has structure T and the second one states that for each instance of C there is at least one entity of structure T reachable through link L. Therefore such link-declarations represent a generalization of existence and typing declarations for attributes (and roles).

– A *well-foundedness-declaration* has the form:

$$\text{well founded } L.$$

It states that by repeatedly following link L starting from any instance of C, after a finite number of steps one always reaches an entity from which L cannot be followed anymore. Such a condition allows for example to avoid such pathological cases as a set that has itself as a member. This aspect will be discussed in more detail in section 4.

– A *cardinality-declaration* has the form:

$$\text{exists } (u,v)\ b \text{ in } T \quad \text{or} \quad \text{exists } (u,v)\ b^- \text{ in } T,$$

where u is a nonnegative integer and v is a nonnegative integer or the special value ∞. Such a declaration states for each instance of C the existence of at least u and most v different entities of structure T reachable through the basic link b (b^-)[2]. Existence and functional dependencies can be seen as special cases of this type of constraint.

– A *meeting-declaration* has the form:

$$\text{each } b_1 \text{ is } b_2 \quad \text{or} \quad \text{each } b_1^- \text{ is } b_2^-.$$

It states that each entity reachable through a link b_1 (b_1^-) from an instance o of C is also reachable from o through a different link b_2 (b_2^-). Such a declaration allows for representing inclusions between attributes, and is a restricted form of role-value map, a type of constraint commonly used in knowledge representation formalisms [15].[3]

– A *key-declaration* has the form:

$$\text{key } A_1, \ldots, A_m, A_1'^-, \ldots, A_{m'}'^-, U_1, \ldots, U_n, U_1'^-, \ldots, U_{n'}'^-.$$

It is allowed only in class definitions and states that each entity o in C is linked to at least one other entity through each link that appears in the declaration, and moreover the entities reached through these links uniquely determine o, in the sense that C contains no other entity o' linked to exactly the same entities as o (for all links in the declaration).

(iii) method-declarations stands for a possibly empty set of *method-declarations*, each having the form:

$$\text{method } M\ (C_1, \ldots, C_m) \text{ returns } (C_1', \ldots, C_n').$$

It states that for each instance of C, method M can be invoked, where the type of the input parameters (besides the invoking object) that are passed to, output parameters that are returned from the method are as specified in the declaration.

[2]Note that requiring the link to be basic (and not generic) is essential for preserving the decidability of inference on the schema.

[3]Note that the restricted form of role-value map adopted here does not lead to undecidability of inference, which results if this construct is used in its most general form.

2.2 Semantics

We specify the formal semantics of a $\mathcal{CVL}$ schema through the notion of *interpretation* $\mathcal{I} = (\mathcal{O}^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\mathcal{O}^{\mathcal{I}}$ is a nonempty set constituting the *universe* of the interpretation and $\cdot^{\mathcal{I}}$ is the *interpretation function* over the universe. Note that an interpretation corresponds to the usual notion of database state. Differently from traditional object-oriented models, we do not distinguish between objects (characterized through their object identifier) and values associated to objects. Instead, we regard $\mathcal{O}^{\mathcal{I}}$ as being a set of *polymorphic entities*, which means that every element of $\mathcal{O}^{\mathcal{I}}$ can be seen as having one or both of the following structures (entities having none of these structures are called *pure objects*):

(1) The structure of *tuple*: when an entity o has this structure, it can be considered as a property aggregation, which is formally defined as a partial function from $\mathcal{A}$ to $\mathcal{O}^{\mathcal{I}}$ with the proviso that o is uniquely determined by the set of attributes on which it is defined and by their values. In the sequel the term tuple is used to denote an element of $\mathcal{O}^{\mathcal{I}}$ that has the structure of tuple, and we write $[A_1{:}\,o_1, \ldots, A_n{:}\,o_n]$ to denote any tuple t such that, for each $i \in \{1, \ldots, n\}$, $t(A_i)$ is defined and equal to o_i (which is called the A_i-component of t). Note that the tuple t may have other components as well, besides the A_i-components.

(2) The structure of *set*: when an entity o has this structure, it can be considered as an instance aggregation, which is formally defined as a finite collection of entities in $\mathcal{O}^{\mathcal{I}}$, with the following provisos: (i) the view of o as a set is unique (except for the empty set $\{\}$), in the sense that there is at most one finite collection of entities of which o can be considered an aggregation, and (ii) no other entity o' is the aggregation of the same collection. In the sequel the term set is used to denote an element of $\mathcal{O}^{\mathcal{I}}$ that has the structure of set, and we write $\{|o_1, \ldots, o_n|\}$ to denote the collection whose members are exactly $o_1, \ldots, o_n$.

The interpretation function $\cdot^{\mathcal{I}}$ is defined over classes, structure expressions and links, and assigns them an *extension* as follows:

- It assigns to $\ni$ a subset of $\mathcal{O}^{\mathcal{I}} \times \mathcal{O}^{\mathcal{I}}$ such that for each $\{|\ldots, o, \ldots|\} \in \mathcal{O}^{\mathcal{I}}$, we have that $(\{|\ldots, o, \ldots|\}, o) \in \ni^{\mathcal{I}}$.
- It assigns to every role U a subset of $\mathcal{O}^{\mathcal{I}} \times \mathcal{O}^{\mathcal{I}}$.
- It assigns to every attribute A a subset of $\mathcal{O}^{\mathcal{I}} \times \mathcal{O}^{\mathcal{I}}$ such that, for each tuple $[\ldots, A{:}\,o, \ldots] \in \mathcal{O}^{\mathcal{I}}$, $([\ldots, A{:}\,o, \ldots], o) \in A^{\mathcal{I}}$, and there is no $o' \in \mathcal{O}^{\mathcal{I}}$ different from o such that $([\ldots, A{:}\,o, \ldots], o') \in A^{\mathcal{I}}$. Note that this implies that every attribute in a tuple is functional for the tuple.
- It assigns to every link a subset of $\mathcal{O}^{\mathcal{I}} \times \mathcal{O}^{\mathcal{I}}$ such that the following conditions are satisfied:

$$
\begin{aligned}
(b_1 \cap b_2)^{\mathcal{I}} &= b_1^{\mathcal{I}} \cap b_2^{\mathcal{I}} \\
(b_1 \setminus b_2)^{\mathcal{I}} &= b_1^{\mathcal{I}} \setminus b_2^{\mathcal{I}} \\
(L_1 \cup L_2)^{\mathcal{I}} &= L_1^{\mathcal{I}} \cup L_2^{\mathcal{I}} \\
(L_1 \circ L_2)^{\mathcal{I}} &= L_1^{\mathcal{I}} \circ L_2^{\mathcal{I}} \\
(L^*)^{\mathcal{I}} &= (L^{\mathcal{I}})^*
\end{aligned}
$$

$$
\begin{aligned}
(L^-)^{\mathcal{I}} &= \{(o, o') \mid (o', o) \in L^{\mathcal{I}}\} \\
(\underline{\text{identity}}(C))^{\mathcal{I}} &= \{(o, o) \in \mathcal{O}^{\mathcal{I}} \times \mathcal{O}^{\mathcal{I}} \mid o \in C^{\mathcal{I}}\}.
\end{aligned}
$$

- It assigns to every class and to every structure expression a subset of $\mathcal{O}^{\mathcal{I}}$ such that the following conditions are satisfied:

$$
\begin{aligned}
\texttt{Any}^{\mathcal{I}} &= \mathcal{O}^{\mathcal{I}} \\
\texttt{Empty}^{\mathcal{I}} &= \emptyset \\
C^{\mathcal{I}} &\subseteq \mathcal{O}^{\mathcal{I}} \\
(\neg T)^{\mathcal{I}} &= \mathcal{O}^{\mathcal{I}} \setminus T^{\mathcal{I}} \\
(T_1 \wedge T_2)^{\mathcal{I}} &= T_1^{\mathcal{I}} \cap T_2^{\mathcal{I}} \\
(T_1 \vee T_2)^{\mathcal{I}} &= T_1^{\mathcal{I}} \cup T_2^{\mathcal{I}} \\
[A_1{:}\,T_1, \ldots, A_n{:}\,T_n]^{\mathcal{I}} &= \{[A_1{:}\,o_1, \ldots, A_n{:}\,o_n] \in \mathcal{O}^{\mathcal{I}} \mid \\
&\qquad o_1 \in T_1^{\mathcal{I}}, \ldots, o_n \in T_n^{\mathcal{I}}\} \\
\{T\}^{\mathcal{I}} &= \{\{|o_1, \ldots, o_n|\} \in \mathcal{O}^{\mathcal{I}} \mid \\
&\qquad o_1, \ldots, o_n \in T^{\mathcal{I}}\}.
\end{aligned}
$$

The elements of $C^{\mathcal{I}}$ are called *instances* of C.

In order to characterize which interpretations are legal according to a specified schema we first define what it means if in an interpretation $\mathcal{I}$ an entity $o \in \mathcal{O}^{\mathcal{I}}$ *satisfies* a declaration which is part of a class or view definition:

- o satisfies a type-declaration "is a kind of T" if $o \in T^{\mathcal{I}}$;
- o satisfies a universal-link-declaration "all L in T" if for all $o' \in \mathcal{O}^{\mathcal{I}}$, $(o, o') \in L^{\mathcal{I}}$ implies $o' \in T^{\mathcal{I}}$;
- o satisfies an existential-link-declaration "exists L in T" if there is $o' \in \mathcal{O}^{\mathcal{I}}$ such that $(o, o') \in L^{\mathcal{I}}$ and $o' \in T^{\mathcal{I}}$;
- o satisfies a well-foundedness-declaration "well founded L" if there is no infinite chain $(o_1, o_2, \ldots)$ of entities $o_1, o_2, \ldots \in \mathcal{O}^{\mathcal{I}}$ such that $o = o_1$ and $(o_i, o_{i+1}) \in L^{\mathcal{I}}$, for $i \in \{1, 2, \ldots\}$.
- o satisfies a cardinality-declaration "exists (u, v) b in T" if there are at least u and at most v entities $o' \in \mathcal{O}^{\mathcal{I}}$ such that $(o, o') \in b^{\mathcal{I}}$ and $o' \in T^{\mathcal{I}}$; a similar definition holds for a cardinality-declaration involving b^-;
- o satisfies a meeting-declaration "each b_1 is b_2" if

$$\{o' \mid (o, o') \in b_1^{\mathcal{I}}\} \subseteq \{o' \mid (o, o') \in b_2^{\mathcal{I}}\};$$

a similar definition holds for a meeting-declaration involving b_1^- and b_2^-.

Finally, a class C satisfies a key-declaration "key $L_1, \ldots, L_m$", if for every instance o of C in $\mathcal{I}$ there are entities $o_1, \ldots, o_m \in \mathcal{O}^{\mathcal{I}}$ such that $(o, o_i) \in L_i^{\mathcal{I}}$, for $i \in \{1, \ldots, m\}$, and there is no other entity $o' \neq o$ in $C^{\mathcal{I}}$ for which these conditions hold.

Note that the method-declarations do not participate in the set-theoretic semantics of classes and views. For an example on the use of method declarations in the definition of a schema we refer to Section 4.

An interpretation $\mathcal{I}$ *satisfies a class definition* δ, say for class C, if every instance of C in $\mathcal{I}$ satisfies all declarations in δ, and if C satisfies all key-declarations in δ. $\mathcal{I}$ *satisfies a view definition* δ, say for view C, if the set of entities that satisfy all declarations in δ is exactly the set of instances of C. In other words, there are no other entities in $\mathcal{O}^{\mathcal{I}}$ besides those in $C^{\mathcal{I}}$ that satisfy all declarations in δ.

If $\mathcal{I}$ satisfies all class and view definitions in a schema $\mathcal{S}$ it is called a *model* of $\mathcal{S}$. A schema is

said to be *consistent* if it admits a model. A class (view) C is said to be *consistent in* $\mathcal{S}$, if there is a model $\mathcal{I}$ of $\mathcal{S}$ such that $C^{\mathcal{I}}$ is nonempty. The notion of consistency is then extended in a natural way to structure expressions.

3 Reasoning in $\mathcal{CVL}$

One of the main features of $\mathcal{CVL}$ is that it supports several forms of reasoning at the schema level. The basic reasoning task we consider is *consistency checking*: given a schema $\mathcal{S}$ and a structure expression T, verify if T is consistent in $\mathcal{S}$. This reasoning task is indeed the basis for the typical kinds of schema level deductions supported by object-oriented systems, such as checking schema consistency and class subsumption, and computing the class lattice of the schema. All these inferences can be profitably exploited in both schema design and analysis (for example in schema integration) and also provide the basis for type checking and type inference.

In general, schema level reasoning in object-oriented data models can be performed by means of relatively simple algorithms (see for example [13]). The richness of $\mathcal{CVL}$ makes reasoning much more difficult with respect to usual data models. Indeed the question arises if consistency checking in $\mathcal{CVL}$ is decidable at all. One of our main results is a sound, complete, and terminating reasoning procedure to perform consistency checking. The reasoning procedure works in worst-case deterministic exponential time in the size of the schema. Notably, we have shown that such worst-case complexity is inherent to the problem, proving that consistency checking in $\mathcal{CVL}$ is EXPTIME-complete.

Space limitations prevent us from exposing our inference method, which is based on previous work relating formalisms used in knowledge representation and databases to modal logics developed for modeling properties of programs [5; 9; 10]. For more details we refer to [4].

4 Expressivity of $\mathcal{CVL}$

In this section we discuss by means of examples the main distinguished features of $\mathcal{CVL}$ with the goal of illustrating its expressivity.

4.1 Object polymorphism

In $\mathcal{CVL}$, entities can be seen as having different structures simultaneously. In this way we make a step further with respect to traditional object models, where the usual distinction between objects (without structure) and their unique value may constitute a limitation in modeling complex application domains. As an example, Condominium in the schema of Figure 1 is regarded as a set of apartments, as a record structure collecting all its relevant attributes and as an object that can be referred to by other objects through roles (in our example manages).

4.2 Well founded structures

In $\mathcal{CVL}$, the designer can define a large variety of finite recursive structures, such as lists, binary trees,

```
class Condominium
    is a kind of (Apartment) ∧
                 [loc: Address, budget: Integer]
    key loc
    exists (1, 1) manages⁻ in Manager
endclass

class CondominiumManager
    is a kind of [ssn: String, loc: Address]
    key ssn
    exists manages in Condominium
endclass
```

Figure 1: Schema of a condominium

trees, directed acyclic graphs, arrays, depending on the application need. The schema in Figure 2 shows an example of definitions of several variants of lists. Observe the importance of the well-foundedness-declaration in the definition of List.

Notably, recursively defined classes are taken into account like any other class definition when reasoning about the schema. We argue that the ability to define finite recursive structures in our model is an important enhancement with respect to traditional object-oriented models, where such structures, if present at all, are ad hoc additions requiring a special treatment by the reasoning procedures [6; 3].

Well-foundedness-declarations also allow us to represent well-founded binary relations. An interesting example of such possibility is the definition of the *part-of* relation, which has a special importance in modeling complex applications [8]. This relation is characterized by being finite, antisymmetric, irreflexive, and transitive. The first three properties are captured by imposing well-foundedness, while transitivity is handled by a careful use of the $*$ operator. More precisely, in order to model the part-of relation in $\mathcal{CVL}$, we can introduce a basic_part_of role, assert its well-foundedness for the class Any, and then use the link basic_part_of $\circ$ basic_part_of* as part-of. Notice that by the virtue of meeting-declarations, we can also distinguish between different specializations of the part-of relation.

4.3 Classification

We show an example of computation of the class lattice in which the reasoning procedure needs to exploit its ability to deal with recursive definitions. Figure 3 shows the definitions of classes and views concerning various kinds of (directed) graphs. Our

```
view List
  is a kind of Nil ∨
    [first: Any, rest: List]
  exists (0, 1) rest⁻ in Any
  well founded first ∨ rest
endview

class ListOfPersons
  is a kind of List
  all rest* ∘ first in Person
endclass

class Nil
  is a kind of Any
  all first ∨ rest in Empty
endclass

class ListOfThreePersons
  is a kind of ListOfPersons
  exists rest ∘ rest in Any
  all rest ∘ rest ∘ rest in Empty
endclass
```

Figure 2: Schema defining lists

```
class Graph
  is a kind of [label: String]
  all edge in Graph
endclass

view FiniteDAG
  is a kind of Graph
  well founded edge
endview

view FiniteTree
  is a kind of Graph
  all edge in FiniteTree
  well founded edge
  exists (0,1) edge⁻ in Any
endview

view BinGraph
  is a kind of Graph
  all edge in BinGraph
  exists (0,2) edge in Any
endview

view FiniteBinTree
  is a kind of Graph
  all edge in FiniteBinTree
  well founded edge
  exists (0,1) edge⁻ in Any
  exists (0,1) left in Any
  exists (0,1) right in Any
  each left ∪ right is edge
  each edge is left ∪ right
  each left is edge \ right
endview
```

Figure 3: Schema defining graphs

reasoning method can be used to compute the corresponding class lattice shown in Figure 4. Observe that several deductions involved in the computation of the lattice are not trivial at all. For example, in computing subsumption between FiniteBinTree and BinGraph, a sophisticated reasoning must be carried out in order to infer that every instance of FiniteBinTree satisfies exists (0,2) edge in Any.

4.4 Methods

Consider a schema $\mathcal{S}$ in which the definition of a class C contains the method declaration "method M (D_1, D_2) returns (D_3)". Suppose now that in specifying manipulations of the corresponding database we use three objects x in class C, y_1 in class D_1' and y_2 in class D_2', respectively. Let us analyze the behavior of the type checker in processing the expression $x.M(y_1, y_2)$. If the type checker follows a strong type checking policy, then the expression would be considered well typed if and only if D_1' is subsumed by D_1 and D_2' is subsumed by D_2 in $\mathcal{S}$. On the other hand, if a weaker type checking policy is adopted, in order to guarantee well typedness, it is sufficient that both $D_1 \wedge D_1'$ and $D_2 \wedge D_2'$ are consistent in $\mathcal{S}$. Moreover, in both cases it can be easily inferred that the type of the expression is in D_3. All these inferences can be carried out by relying on the basic reasoning task introduced in the previous section.

5 Concluding remarks

The combination of constructs of the $\mathcal{CVL}$ data model makes it powerful enough to capture most common object-oriented and semantic data models presented in the literature [12; 11], such as O_2 [3], ODMG [6], and the entity-relationship model [7]. In fact, by adding suitable definitions to a schema we can impose conditions that reflect the assumptions made in the various models, forcing such a schema to be interpreted exactly in the way required by each model.

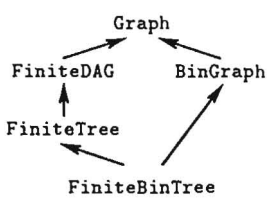

Figure 4: A lattice of graphs

References

[1] S. Abiteboul and A. Bonner. Objects and views. In J. Clifford and R. King, editors, *Proc. of ACM SIGMOD*, pages 238–247, New York (NY, USA), 1991.

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

[3] F. Bancilhon, C. Delobel, and P. Kanellakis. *Building an Object-Oriented Database System – The story of O_2*. Morgan Kaufmann, Los Altos, 1992.

[4] D. Calvanese, G. De Giacomo, and M. Lenzerini. Structured objects: Modeling and reasoning, 1995. To appear in Proc. of DOOD-95.

[5] D. Calvanese, M. Lenzerini, and D. Nardi. A unified framework for class based representation formalisms. In J. Doyle, E. Sandewall, and P. Torasso, editors, *Proc. of KR-94*, pages 109–120, Bonn, 1994. Morgan Kaufmann, Los Altos.

[6] R. G. G. Cattell, editor. *The Object Database Standard: ODMG-93*. Morgan Kaufmann, Los Altos, 1994. Release 1.1.

[7] P. P. Chen. The Entity-Relationship model: Toward a unified view of data. *ACM Trans. on Database Systems*, 1(1):9–36, Mar. 1976.

[8] V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl. From structured documents to novel query facilities. In R. T. Snodgrass and M. Winslett, editors, *Proc. of ACM SIGMOD*, pages 313–324, Minneapolis (Minnesota, USA), 1994.

[9] G. De Giacomo and M. Lenzerini. Boosting the correspondence between description logics and propositional dynamic logics. In *Proc. of AAAI-94*, pages 205–212. AAAI Press/The MIT Press, 1994.

[10] G. De Giacomo and M. Lenzerini. What's in an aggregate: Foundations for description logics with tuples and sets. In *Proc. of IJCAI-95*, 1995. To appear.

[11] R. Hull. A survey of theoretical research on typed complex database objects. In J. Paredaens, editor, *Databases*, pages 193–256. Academic Press, 1988.

[12] R. B. Hull and R. King. Semantic database modelling: Survey, applications and research issues. *ACM Computing Surveys*, 19(3):201–260, Sept. 1987.

[13] C. Lecluse and P. Richard. Modeling complex structures in object-oriented databases. In *Proc. of PODS-89*, pages 362–369, 1989.

[14] M. Staudt, M. Nissen, and M. Jeusfeld. Query by class, rule and concept. *J. of Applied Intelligence*, 4(2):133–157, 1994.

[15] W. A. Woods and J. G. Schmolze. The KL-ONE family. In F. W. Lehmann, editor, *Semantic Networks in Artificial Intelligence*, pages 133–178. Pergamon Press, 1992. Published as a special issue of *Computers & Mathematics with Applications*, Volume 23, Number 2–9.

Knowledge Representation Concepts Supporting Business Process Analysis

Hans W. Nissen
Informatik V
RWTH Aachen
Ahornstraße 55
52056 Aachen
Germany

Georg V. Zemanek
USU Softwarehaus
Unternehmensberatung GmbH
Spitalhof
71693Möglingen
Germany

Abstract

Modeling and analysing business processes is a frequent job in professional consulting projects, but adequate commercial tools or even formal methods supporting this task hardly exist. This paper reports about the successful application of the knowledge representation system ConceptBase to this task. Based on generic modeling facilities on the one hand and powerful query mechanism on the other ConceptBase is able not only to represent and analyse the final complex model but also to support and record intermediate states together with transitions between them. Our experience has shown that a logic based knowledge representation language is *not* inconvenient for practical modeling tasks but even urgently needed to handle large and complex models in an adequate way.

1 Introduction

During the first KRDB workshop in 1994 a first contact between the information systems group of Informatik V at the RWTH Aachen and the consulting firm USU was established. Now, at the second workshop in this series, we can report on a successful cooperation project. Within this project the deductive object base manager ConceptBase [4] developed in the group of Matthias Jarke was used to model and reason about business processes.

Early phases of consulting projects concerning the introduction of sophisticated information systems include the analysis of existing business processes together with the derivation of requirements as the fundamental goal. For this task only rudimentary tool support or fragments of formal methods exist. Almost all existing tools contain a fixed view of the world, an extension of the supported concepts is not possible. As a consequence, the tools prescribe the analysis procedure and not vice versa.

The aim of this project was to develop concepts and a prototype for a comprehensive support of the current USU-PFR method used to capture information about the domain of interest. The result should be easy to use and to understand, such that also customers are able to use it, and on the other hand be powerful enough for a convincing analysis.

The next section describes the USU method to business process modeling together with some of the arising problems in tool support. The knowledge representation formalism Telos is introduced in section 3. Section 4 presents the application of Telos and ConceptBase to this task while the last section summarizes our experiences.

2 The USU Method to Business Process Modeling

Requirements are captured from multiple, sometimes unforeseen perspectives: content and structure analysis of existing documents, interviews with individuals describing their current situation and wishes, informal textual or visual conceptual models developed in planned or unplanned meetings of stakeholder groups, reverse analysis of existing systems, or goal analysis from a business or individual perspective. The study of each of these sources may lead to new questions, to be answered from new sources until a somewhat coherent picture of requirements emerges.

A typical USU consulting project follows the so called PFR method (Analysis of Presence and Future Requirements) [1] The aim of this method is to generate a shared and agreed understanding of the current business processes, the problems, and a first vision of the target system. The main part consists of two phases.

In a cooperative fashion a set of involved persons generate in a first phase a rough overview of the existing processes (mostly in terms of information exchange among organisational units). Based on the result of the first phase people working within the identified units describe in a second step in detail the sequence of their activities together with relationships to other persons in the organisation. This step has the goal of testing the initial vision against the existing and expected organizational context, and to elaborate it, both in terms of deepened *understanding* and in terms of more formal *representations* (e.g. in the form of activity sequences, data flow models, entity relationship diagrams or object models). This step also includes an analysis of exchanged media in order to capture hints for further process optimiza-

tion.

From a representational viewpoint, the PFR methodology comprises a set of source perspectives as captured in the first two steps, and a set of result perspectives which represent the delivered requirements (with the intent of presenting them to users or to use them in subsequent design tasks). The details of these perspectives may change with the individual customers and projects

The source perspectives are:

- *The information exchange between organisational units.* This perspective aims to produce a visual overview of the current or future situation including the identification of weak spots of the process under investigation. It is represented in an informal collage style employing a fixed set of graphical symbols and pictograms. Although its semantics is a bit vague, it provides a valuable overview of the current situation and its limitations. This representation is not only used to produce a picture of the current situation, but also to visualize a first version of the target conception.
- *The individual activity sequence of stakeholders.* This perspective is captured for each stakeholder by individual interviews and describes in form of a detailed flow chart the sequence of activities, the required and produced information, and inter-relationships with other stakeholders. In the same way information from already existing workflow documents, as, e.g., the quality management handbook, is represented.
- *The structure of exchanged media.* This perspective identifies the pieces of information that reside on forms, documents and other kinds of media that are exchanged between stakeholders resp. organisational units. This breakdown of a medium into the pieces of information it carries is necessary for a detailed analysis of the activities performed by stakeholders.

Cross-perspective analysis applies these source perspectives and mainly consists of a comparison of the perspectives to detect discrepancies, modeling errors, gaps, and properties of the business process. The results of this comparison activity guide further interviews to clarify the inconsistencies and to complete the models. During these changes in individual perspectives, the corresponding derived knowledge about the conflicts has to be maintained, as old conflicts may disappear and new problems may surface. USU's experience in applying this method to a large number of projects has shown that an analysis by hand is a time-consuming and error-prone task. A supporting tool should therefore

- represent the information from all perspectives in a natural way (which may be different from customer to customer or from project to project) such that they can be easily communicated to stakeholders,
- enable the comparison of diagrams represented according to different (semi-)formal notations to detect discreparencies, modeling errors, gaps, etc., and maintain the detected relationships over time,
- be able to automatically generate function-oriented and data-oriented perspectives on the provided information to be used as a starting point in subsequent analysis and design steps.

Use of existing CASE tools proved unsatisfactory for these tasks, as they were too rigid in their hard-coded consistency analyses which were developed for other purposes.

Frequently, the set of perspectives has to be customized by aspects which are specific to a particular project but do not occur sufficiently often to include them in the standard methodology. Or a customer organization uses an existing methodology in subtly different ways than others.

What is needed, is a *simple* formalism which is *extensible* to the needs of specific methodologies or even application projects but still provides the formal background for *integrating* all the perspectives used. This combination of simplicity of basic formalism, extensibility, and formal integratability proved crucial to the success of ConceptBase.

3 The Knowledge Representation Language Telos

In this cooperation we used the deductive object manager ConceptBase. ConceptBase is a prototype system that is based on the knowledge representation language Telos [7]. Telos is especially designed to offer modelers the flexibility to define and use their particular understanding of the world, and to relate this understanding to that by others. Telos offers a simple generic data model that is extensible to specific application needs and provides mechanisms for perspective integration.

The kernel model of Telos consists of just two concepts: nodes and links. To allow any kind of formalization, we need a third concept, that of an assertion. Finally, to talk about different notations, we need at least one abstraction mechanism – classification – which enables us to talk about classes and their instances. The kernel of the Telos language is just that. All other language facilities can be bootstrapped from this kernel of nodes and arcs, assertions, and classification.

Tailoring Telos to specific application data models is done by first embedding the structural regulations of the language (i.e. its syntax) into Telos, second giving the new modeling constructs a formal semantics by defining appropriate rules and constraints, and third introducing the diagrammatic presentation of the language by assigning graphical type descriptions to the modeling constructs. In this section we concentrate on the structural and semantic extensibility.

Structural Extension. The infinite levels of classification available in Telos enable the creation of (meta) models. Such a meta model extends the admissible set of modeling constructs to the models considered on an abstraction level lower than the meta model. This technique can be used to integrate the structural part of other modeling languages

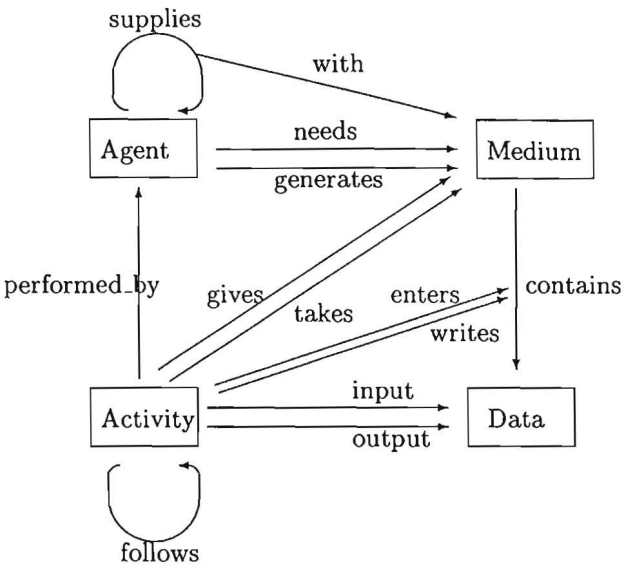

Figure 1: The Telos meta model for the PFR analysis method

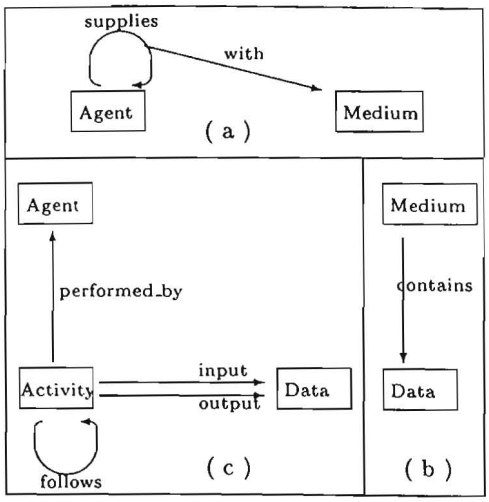

Figure 2: The PFR perspectives within the meta model

and to make the modeling concepts of that language available. The meta model acts then as a conceptual model of the structural part (syntax) of the modeling technique.

Semantic Extension. Most implemented approaches to meta modeling cover the structural part well [6; 9] but offer semantic extension only within a predefined set of constraint types (e.g. cardinality constraints). Telos assertion objects make it possible to specify the semantics of language extensions as part of the corresponding meta model. The formal behavior, defined in the form of integrity constraints and deductive rules, can be directly attached to the corresponding class definition. In ConceptBase, semantic extensibility is assisted by so-called *meta formulas* [5]. We allow formulas to make statements across several instantiation level. Thus, they are able to specify the behavior of objects which reside two or more instantiation levels below the objects of the meta model.

4 Extension of Telos Towards the PFR Method

The meta model shown in figure 1 was derived from a cumulative analysis of the perspectives typically discussed in USU's RE projects. By emphasizing the relevant objects in the meta model, we show in the following how the different perspectives described in section 2 are captured in this meta model. Based on this description, we also present a number of query classes for analyzing conflicts among these perspectives. For each of these query classes, a set of possible explanations and related courses of action have been developed in order to help USU analysts in conflict resolution.

The meta model covers all PFR source perspectives. Figure 2 presents the individual perspectives and also visualizes the overlaps of them. Part (a) highlights the part of the meta model used to represent the *information exchange between organisational units*, as captured in the "collage" of the initial workshop We model an organisational unit as an abstract `Agent` who `supplies` another agent `with` a `Medium`. The earliest version of the meta model had this simply as a data flow but, observing the participants of the first pilot project, we recognized that agents do not really exchange information, but the medium that act as the data or information carrier. A medium can be something persistent, like a piece of paper, a form or a disk, or a transient thing like the voice that carries words.

The model in part (b) therefore represents *the structure of exchanged media* by explicitly distinguishing the `Medium` and the `Data` it `contains`. This distinction is essential to talk about phenomena such as empty and completed forms, reading from and writing to a medium, replicating the same piece of data on multiple media manually or automatically, and agents that get a medium but perform no activity that needs or produces any data located on that media. For example, one project revealed that the same data was captured and re-captured several times in a workflow, with very good and expensive quality controls, except in the last step! Here, the meta model helped to explain why there was bad quality despite high quality control costs.

The conceptual model of the *individual activity sequence of stakeholders* is shown in part (c). An `Activity` is `performed_by` an `Agent`. A partial order on activities (workflow) is expressed by the `follows` relation. An activity is an atomic action that takes some information or `Data` as `input` and generates new `Data` as `output`. Our semantics of the `output` relation is very rigorous: The activity must create this data for the first time, i.e. no other activity can also create this data. Every piece of data is created exactly once. The motivation behind this is that the data once created gets never lost.

As indicated in the description above, the perspectives are strongly interrelated by overlaps and redundant information. The USU application projects identified more than 70 constraints describing the consistency of the captured information. This in-

cludes consistency of knowledge *within* an individual perspective as well as the consistency *between* different perspectives.

In Telos, we can formally include consistency checks by attaching integrity constraints to the appropriate objects of the meta model. As a consequence, the system will reject every update that violates one of these constraints. This rigid consistency enforcement strategy is not well suited for RE workers: The distributed knowledge acquisition process and the overlapping perspectives lead to numerous conflicts, which then always have be solved before inserting new information into the knowledge base. This delay hampers the analyst and the whole acquisition and analysis process. It also forces perspective reconciliation to take place outside the system, and without traceability.

In contrast, Telos query classes offer a more flexible way to analysis and enforcement. Queries are represented as classes (i.e they are first class objects in a Telos model) and the answers become the virtual instances of that class. Applied to our problem, the answers to the query are interpreted as consistency violations.

USU did not only formulate queries to detect errors within and between perspectives, but also to analyse the properties of the finally reconciled business process model. This includes questions like "What is the trace of form X305 ?", to detect the reason for the long handling time of the form X305. All together USU produced over 80 query classes. To further support the analyst, we developed guidelines for applying the queries. For each query class, they include a set of possible answer interpretations in the light of business processes as the application domain together with appropriate repair suggestions. In addition, we established a sequence of the queries that proved to be reasonable within our experiment projects.

4.1 Some Analysis Examples

In this subsection we present some concrete examples of query classes and answer interpretations. We first give a brief to the syntax of query classes: A query class is formulated in the Telos frame syntax, and has the following form:

```
QueryClass <name> isA <superclasses> with
  attribute
    <answer attributes>
  constraint
    <condition>
end
```

We can distinguish four important parts:

1. The name of the query class is given by `<name>`.
2. The `<superclasses>` part specifies the superclasses of the query class. The set of possible answer objects of the query are then restricted to the common instances of the superclasses. If this part is omitted, `Object` becomes the superclass which enables all objects of the knowledge base to join the answer set.
3. The `<answer attributes>` part defines the attributes of the answers to the query. The attributes either already exist in the knowledge base, or are deduced during query evaluation.
4. The `<condition>` part contains the query condition which can be an arbitrary closed formular. The symbol `this` used within the condition refers to potential answer objects, i.e., the instances of the superclasses.

Analysis of a single perspective
Consider the activity sequence perspective. The query class below realizes the constraint that data can only be used by an activity (indicated by the `input` relation) after it was created (via the `output` relation). The query deduces data that are used as `input` before they are produced.

```
QueryClass Data_UsedBeforeProduced isA Data
   with  attribute
    early_user : Activity
  constraint
    c : $ (early_user input this) and
          (producer output this) and
        (producer trans_follows this) $
end
```

The query class uses the `trans_follows` relation, which denotes the transitive closure of the `follows` relation and is deduced by a Telos recursive rule. The answer can be interpreted as

1. an error, if the interviewed agent indicated a wrong sequence.
2. an error, where the interviewer misinterpreted a statement and modeled an `input` relation to `Data` instead of a `gives` relation to `Medium`.
3. nothing else, since this model represents a real existing and running process where data cannot be used before it is produced.

Analysis of interrelationship among multiple perspectives
In the consulting projects we often detected contradictions between the high-level information exchange perspective acquired mainly from managers and the detailed activity sequence perspective captured from the real working agents. An often violated interrelationship states that the medium flow among agents must correspond to the data demand of agent's activities, i.e. the supplied media must contain some data that is required by an activity and, conversely, all required data must be contained on some delivered medium. The following query class implements the first part and deduces all media that is supplied to an agent who performs no activity that needs any data carried by that medium.

```
QueryClass NotUsedMedium isA Medium with
  attribute
     not_user : Agent
  constraint
     c : $ (supply in Agent!supplies)
             and (supply to not_user)
             and (supply with this)
             and not exists (
       (action performed_by not_user)
       and (this contains info)
       and ((action input info)
```

```
        or (action output info))     ) $
end
```

The answers are the media together with the agent who gets the media but does not use it. They can be interpreted as follows:

1. There exists a mismatch between the captured perspectives: the management and the concrete employees view the process in different ways. Further clarification interviews are necessary to reconcile the contradicting views.
2. The model is correct and the agent actually gets and sends the medium without any interest on the data. In this case the business process can be further improved by optimizing the media flow.
3. The model is correct and the business process is ok, but the activity that works on the medium does not require any information from the medium.

In practice, we often observed the problem described in 2. As an example of interpretation 4, a secretary collected the monthly reports of the employees of a department to give them as one piece to the manager of that department.

5 Conclusions

The applicability of ConceptBase and Telos to the task of business process modeling and analysis has been successfully proved within this project. We developed a specialized knowledge representation tool containing an adequate meta model, over 80 analysis queries together with predefined answer interpretations and guidelines how to use this system within further projects. The world model (i.e., the meta model) can easily be tailored to specific application needs, and the modeler can individually decide when to use which predefined queries for checking and analysing purposes. It exactly fits the methods used in the company, without precluding future evolution of these methods or customization to individual projects. The information exchange, document structure, and activity sequence can be represented within one meta model; a number of useful observations about the practicality of modeling features (e.g. distinguishing media and data, granularity of modeling required) were made.

An *extensible formal language* like Telos is able to provide a valuable complementary support for informal, teamwork-oriented methods. Since we can tailor the language to the specific application needs, we must also be able to formulate specific analysis queries - a fixed set of predefined queries as provided by most CASE environments will not fulfill this. This requires a powerful declarative assertion and query language based on a well-defined formal semantics.

The experiences confirm the usefulness of *requirements freedoms* and explicit tolerance of inconsistencies within and across multiple viewpoints, as postulated by researchers such as Balzer [2], Feather and Fickas [3], Finkelstein and colleagues [8]. This may seem in contrast to the old paradigm of consistently refining an initially consistent specification – the only known way to create provably correct software. However, recall that we are concerned with an early phase of analysis; its end result should still be consistent so that the consistent refinement approach may still be used in conjunction with our approach.

Conflicts during analysis force discussions and increase the understanding of the domain under investigation. Exactly for that reason we developed a meta model that potentially includes a lot of conflicts. A systematic way of developing such meta models in general is a subject of further research.

References

[1] P. Abel, "Description of USU-PFR analysis method", *Technical Report USU*, 1995.

[2] R. Balzer. Tolerating inconsistency. In *Proc. of the 13th ICSE*, 1991.

[3] M.S. Feather and S. Fickas. Coping with requirements freedom. In *Proc. Intl. Workshop Development of Intelligent Information Systems*, 1990.

[4] M. Jarke, R. Gallersdoerfer, M. Jeusfeld, M. Staudt, S. Eherer "ConceptBase - A Deductive Object Manager for Meta Databases" *Journal of Intelligent Information Systems* 4(2), 1995.

[5] M.A. Jeusfeld. *Update Control in Deductive Object Bases.* PhD thesis, University of Passau (in German), 1992.

[6] K. Lyytinen, P. Kerola, J. Kaipala, S. Kelly, J. Lehto, H. Liu, P. Marttiin, H. Oinas-Kukkonen, J. Pirhonen, M. Rossi, K. Smolander, V.-P. Tahvanainen, and J.-P. Tolvanen. MetaPHOR: Metamodeling, principles, hypertext, objects and repositories. Technical Report TR-7, University of Jyväskylä, December 1994.

[7] J. Mylopoulos, A. Borgida,M. Jarke, and M. Koubarakis, "Telos: a language for representing knowledge about information systems", in *ACM Trans. Information Systems* 8(4), pp. 325–362, 1990.

[8] B. Nuseibeh, J. Kramer, and A. Finkelstein. Expressing the relationships between multiple views in requirements specification. In *Proc. of ICSE 93*, 1993.

[9] M. Saeki, K. Iguchi, K. Wen-yin, and M. Shinohara. A meta-model for representing software specification & design methods. In N. Prakash, C. Rolland, and B. Pernici, editors, *Proc. of the IFIP WG8.1 Working Conference on Information System Development Process*, Como, Italy, 1-3 September 1993.

[10] G.V. Zemanek, "Project USU–ConceptBase: The results" *Technical Report USU*, 1995.

DL Techniques for Intensional Query Answering in OODBs

Sonia Bergamaschi* and Claudio Sartori° and Maurizio Vincini*
CIOC - CNR
* Dipartimento di Scienze dell'ingegneria, Università di Modena
Via G. Campi 213/B, I-41100 Modena, Italy
°Dipartimento di Elettronica, Informatica e Sistemistica, Università di Bologna
Viale Risorgimento 2, I-40136 Bologna, Italy
e_mail: { sbergamaschi, csartori}@deis.unibo.it

1 Introduction

It is well known that in general, a query issued on a database can be rewritten in many ways maintaining, as a result, the same set of items (say, records or objects, depending on the data model). Such rewriting has been devised with the main purpose of query optimization, i.e. to minimize the execution costs. Traditionally, database theory focused on algebraic rewriting, which depends only on formal properties of the data model and manipulation language. Some works introduced also the idea of semantic query optimization[Shenoy and Ozsoyoglu, 1987; Beneventano *et al.*, 1993; Beneventano *et al.*, 1994; Ballerini *et al.*, 1995], which rewrites queries also on the basis of semantic problem-specific knowledge, such as integrity constraints.

In this paper we exploit the idea of rewriting a query not only for the semantic optimization task, as proposed by the authors in [Beneventano *et al.*, 1993; Beneventano *et al.*, 1994], but also for another query-related task: intensional query answering. In particular, we focus on Object Oriented Databases and give a general definition of *semantic transformation* and of *semantic expansion* of a query. Then we will show how this concepts can be exploited in intensional query answering.

2 Semantic transformation and expansion of a query

Actual database schemata are, in fact, given in terms of base classes (i.e. primitive concepts) while further knowledge is expressed with *Integrity Constraints* (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) to guarantee data consistency. In general, integrity constraints go beyond data model expressiveness and are expressed in various fashions, depending on the database data model: e.g. subsets of first order logic, inclusion dependencies and predicates on row values, procedural methods in OO environments. In this context, we can say that a query Q' is a *semantic transformation* of the query Q if it gives the same result of Q for any database instance which satisfy the given IC rules.

In [Beneventano *et al.*, 1993; Beneventano *et al.*, 1994] the authors proposed a method for semantic query optimization, applicable to the class of conjunctive queries, based on two fundamental ingredients. The first one is the *ODL* description logics proposed as a common formalism to express: class descriptions, a relevant set of IC rules and queries as ODL types. The second one is the subsumption inference technique exploited to evaluate the logical implications expressed by IC rules and, thus, to produce the *semantic expansion* of a given query. The semantic expansion of a query is a semantic transformation of a query which incorporates any possible restriction which is not present in the original query but is ***logically implied*** by the query and by the overall schema (classes + IC rules).

ODL (Object Description Logics) was proposed in [Bergamaschi and Nebel, 1994] and extends the expressiveness of implemented description logics languages in order to represent the semantics of complex object data models (*CODMs*), recently proposed in the areas of deductive databases [Abiteboul and Kanellakis, 1989] and object oriented databases [Lecluse and Richard, 1989]. In particular, class types and complex value-types are differentiated. They are based on base types: integers, strings, reals, and are constructed with the recursive use of the set and record constructors. The present version of ODL allows the declarative formulation of a relevant set of database integrity constraints. In particular, ODL includes *quantified path types* and IC rules. The former extension has been introduced to deal easily and powerfully with nested structures. Paths, which are essentially sequences of attributes, represent the central ingredient of OODB query languages to navigate through the aggregation hierarchies of classes and types of a schema. In particular, we provide *quantified* paths to navigate through multi-valued attributes. The allowed quantifications are existential and universal and they can appear more than once in the same path.

Viewing a database schema as a set of ODL *inclusion statements* allows the declarative formulation of another relevant set of integrity constraints, expressing *if then rules* whose antecedent and consequent are ODL *virtual* types (i.e. defined concepts). For example, it is possible to express correlations between structural properties of the same class or sufficient conditions for populating subclasses of a given class. A *generalized database schema* can be

thus defined as a set of inclusion statements between general ODL types.

A relevant set of queries, corresponding to the so called single-operand queries [Kim, 1989], can be expressed as *virtual* ODL types. Subsumption computation, incoherence detection and canonical form generation proposed in [Bergamaschi and Nebel, 1994] can be used to produce the *semantic expansion* $EXP(Q)$ of a query Q. Following the approach of [Shenoy and Ozsoyoglu, 1987], we perform the semantic expansion of the types included at each nesting level in the query description. Type expansion 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 *subsumption computation* [Bergamaschi and Nebel, 1994].

Semantic expansion is an iterative process which produces, at any step, a query which is semantically equivalent to the original one. During the transformation, we compute and substitute in the query, at each step, the maximal subsumed classes, among the classes of the schema, satisfying the query. Therefore, each of the intermediate results of semantic expansion is a valid semantic transformation of the query and is a candidate for the intensional answer. The result of semantic expansion of a query coincides with the *lowest query* in the taxonomy among all the semantically equivalent ones [Beneventano *et al.*, 1993].

In general, semantic expansion can also lead to introduce redundant terms, i.e. terms which are *logically implied* by other terms. In the literature, this problem is generally addressed as *constraint removal*, that is the removal of the constraints which are logically implied by the query. We can then detect in the expanded query, again by subsumption, all the eliminable factors and, eventually, eliminate them [Ballerini *et al.*, 1995].

3 DL techniques for intensional query answering

An overview of the various intensional query answering techniques is given in [Motro, 1994]. On the basis of that classification, intensional query answering can be evaluated according to three main features: intensional-only (*pure*) versus intensional/extensional (*mixed*); *independence* from the database instance versus dependence; *completeness* of the characterization of the extensional answer.

In general, a query is expressed as a class of the schema (target class) restricted with additional selection predicates, which include conditions on objects of the aggregation rooted at the target class. The many queries obtained by semantic expansion will differ from the original one either for the target class or for the predicates. Each transformed query is a possible intensional answer, which is *pure*, since it does not contain reference to any extensional element, and also *independent*, since it is computed according to general IC rules which hold in any database state. Thus it is also *intension-equivalent**.

For example, in a database with an integrity constraint stating that all employees who lead a department are managers, a query on the employees who lead a department and earn more than $ 50000 is equivalent to a query on the managers who earn more than $ 50000. Conversely, in a database with an integrity constraint stating that all engineers earns over $ 40000, a query on the engineers who earn over $ 30000 is equivalent to a query on all the engineers.

When several different intensional answers are available, a main issue is to determine which answer is the "best". We give the following criteria for the best answer:

1. the target class is the most specialized among the classes of the schema that can be substituted for the original one in the query, therefore it gives a concise description of the answer which is more informative than the original query;
2. the classes included in the query predicates are the most specialized satisfying the query, giving a more significant, though semantically equivalent, predicate;
3. redundant predicates are removed as a contribution to conciseness.

The three above criteria are satisfied by the application of semantic expansion and constraint removal. In particular, according to the criterion 1 a query like *which are the* X *such that* p_1 *and ... and* p_n gets the answer *all the* X' *such that* p_1 *and ... and* p_m, where X' is subsumed by X and $m \leq n$. If we consider the first example above, we substitute the target class "employees", which can contain many thousands of items, with the class "managers" which can contain few hundreds of items and the answer, though purely intensional characterizes the result in terms of a more restricted class than the original query.

As far as completeness of intensional characterization is concerned, our rewriting method is exact, therefore each rewrited answer is a complete characterization of the original query.

With reference to the completeness of our method, which is based on subsumption, it is well known that it is greatly influenced by the complexity of the knowledge representation model or, in our case, of the data and integrity constraint definition language. If the language does not allow completeness of subsumption, the intensional answer we get is not necessarily the most concise.

Given a query Q, subsumption can also be used to compute its Greatest Lower Bound (GLB) and Least Upper Bound (LUB) among the classes of the schema. For simplicity, let us suppose in the following that the two bounds are unique. In this case $LUB_Q \sqsupseteq Q \sqsupseteq GLB_Q$ and each bound can be seen as a *partial* intensional answer to the query.

A different approach could be the generation of an intensional answer which is equivalent to the original one only for the present database instance. For example, let us suppose that, for a given database state, a query on the employees who earn between $30000 and $50000 return only employees who are

engineers. In this case, the answer "all the engineers" is *pure* and *dependent*, i.e. it is *extension-equivalent* to the original one. Unlike the previous case, this method does not avoid data access, but can be driven by schema knowledge. For example, given the query Q and its bounds LUB and GLB, the query Q is extensionally-equivalent to B if $LUB_Q - GLB_Q = \emptyset$. This result can be obtained without accessing the extension if the database system provides an efficient way to deal with classes cardinalities.

Hybrid reasoning can be used to obtain *mixed* intensional answers. In this case, the aswer contains intensional concepts and lists of positive and negative extensional items. Given an algorithm for the *instance problem*, which can decide if an object belogns to a given class, the answer to a query Q can one of the following:

$$\begin{array}{lll} LUB_A & - & \{i_1, i_2, \ldots, i_n\} \\ & where & i_j \in \{LUB_Q - GLB_Q\} \wedge i_j \notin Q \\ LUB_A & \cup & \{i_1, i_2, \ldots, i_n\} \\ & where & i_j \in \{LUB_Q - GLB_Q\} \wedge i_j \in Q \end{array}$$

For instance, the query "who earns more than $30000" could get the answer "all the engineers extept John Smith".

The usability of this technique is obviously related to the efficiency of the algorithm for the instance problem, since it has to be computed many times.

As a final remark, we mentioned in the beginning that the rewriting activity is based on a schema including integrity rules. Of course, if more integrity rules are available more rewritings are possible. For the sake of intensional answers, one could apply *data mining* techniques to discover new rules [Cercone and Tsuchiya, 1993]. The rewritings made possible by these rules give answers which are *dependent* from the present database state.

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.

[Ballerini *et al.*, 1995] J.P. Ballerini, D. Beneventano, S. Bergamaschi, C. Sartori, and M. Vincini. A semantics driven query optimizer for oodbs. In A. Borgida, M. Lenzerini, D. Nardi, and B. Nebel, editors, *DL95 - Intern. Workshop on Description Logics*, pages 59–62, Roma, June 1995.

[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*, 1993.

[Beneventano *et al.*, 1994] D. Beneventano, S. Bergamaschi, S. Lodi, and C. Sartori. Terminological logics for schema design and query processing in oodbs. In *KRDB'94 - Reasoning about Structured Objects, Knowledge Representation Meets Databases*, Saarbruecken, Sept., 1994.

[Bergamaschi and Nebel, 1994] 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*, 4:185–203, 1994.

[Cercone and Tsuchiya, 1993] Nick Cercone and Mas Tsuchiya, editors. *IEEE Transactions on Knowledge and Data Engineering, Vol.5, N.6.* IEEE, 1993. Special issue on Learning and Discovery in Knowledge-Based Databases.

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

[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.

[Motro, 1994] A. Motro. Intensional answers to database queries. *IEEE Trans. on Knowledge and Data Engineering*, 6(3):444–454, 1994.

[Shenoy and Ozsoyoglu, 1987] S. Shenoy and M. Ozsoyoglu. A system for semantic query optimization. *ACM-SIGMOD*, pages 181–195, May 1987.

Using schema information for querying databases

Iztok Savnik[a], Zahir Tari[b], Tomaž Mohorič[c]

[a]Computer Systems Department, Jožef Stefan Institute, Slovenia
[b]School of Information Systems, Queensland University of Technology, Australia
[c]Faculty of Electrical Eng. and Computer Science, University of Ljubljana, Slovenia

Abstract

We propose the set of operations for querying the conceptual schema of an object-oriented database. The operations form the basis of an algebra for objects called OVAL. They are defined using the constructs introduced for our formalization of the object-oriented database model. The operations allow a user to query: (i) associations among individual objects, (ii) relationships between individual objects and class objects, and (iii) relationships among class objects themselves.

1 Introduction

Object-oriented database model provides a rich set of modeling constructs that make the conceptual schema of an object-oriented database more expressive than relational schemas. We observe that, comparing a relational database to an object-oriented one, some information about the modeling environment has been moved from the *data* part to the *schema* part of the database. Hence, some aspects of the modeling environment can be, using an object-oriented database model, represented and stored in a database by means of a database schema. Consequently, the *schema* part of an object-oriented database should be treated in a similar manner as the *data* part of the database: it is, like ordinary data, the subject of the user's inquiry and modification.

In general, there are two types of queries which relate to the conceptual schema. Firstly, the user should be able to query the relationships between the instances and the conceptual schema of a database. Secondly, due to the frequently very complex conceptual schema, a user should be able to query it in order to obtain a precise mental image of the structure and the behavior of stored information [10].

In this paper we present the operations of the *algebra for objects* called OVAL, which are used for querying conceptual schema. The following section briefly overviews the work related to OVAL. Next, the basic constructs used for the formalization of the OVAL's data model are defined in Section 3. The basic operations of the algebra OVAL are presented in Section 4. Finally, the concluding remarks are given in Section 5.

2 Related work

The constructs that recent query languages provide for querying database schema are briefly presented in this section. To our knowledge, recently proposed database algebras (e.g., Query algebra [13], Excess [14] or Complex Object Algebra [1]) do not include such facilities.

Firstly, most recent query languages (e.g., query languages of ORION [9] or O_2 DBMS [4]) provide the constructs for using the class extensions [5] in queries. In [5] Bertino proposes the use of operator $CLASS\_OF$, which returns the class of an object at run-time. The resulting class can be further used in a query. Next, ORION [9] provides a set of operations for modifying database schema at different levels: modification of inheritance, class properties, methods and inheritance hierarchy of classes.

In [8] Kifer and Lausen propose a declarative language based on logic, called F-Logic, which includes the capabilities for querying database schema. The relationships between instances and classes, which are based on the *isa* hierarchy of database objects, can be in F-Logic queried using the predefined predicates for testing class membership and subclass relationship. Further, F-Logic provides the capabilities to explore the properties of individual and class objects by treating attributes and methods as objects that can be manipulated in a similar manner to other database objects. In this way, some types of non-trivial relationships among objects such as the analogy and the similarity relationships can be expressed in F-Logic.

Next, the query language XSQL [7] includes a set of constructs for querying database schema. XSQL queries can include variables that range over class objects. Therefore, classes can be queried on the basis of their properties and the properties of their instances. The XSQL operation $subclassOf$ can be in this context used to inquire about the relationships among classes which are based on the inheritance hierarchy of classes. In a similar manner to F-Logic, XSQL also treats attributes and methods as objects that can be queried; hence, a user can inquire about the properties of individual and class objects.

Finally, in [10] Papazoglou suggests a set of high-level operations for expressing *intensional* queries which aid a user to understand the meaning of stored data. The proposed operations can express the following types of queries: relate individual objects to classes, browse the *isa* hierarchy of classes, inquire about the class properties described using attributes, compute associations among classes which are not related by *isa* relationship, locate objects on the similarity basis and inquire about the dynamic evolution of objects represented by *roles*.

3 Data Model of OVAL

The algebra for objects OVAL is tightly related to its data model which provides, in addition to the basic constructs of the object-oriented database model [3], an uniform view of the database by treating classes as abstract objects.

This section overviews the basic features of our formalization of the object-oriented database model which serve as the platform for the development of the algebra OVAL. More details about the formalization can be found in [11].

3.1 Objects and Classes

An *object* is defined as a couple $< i, v >$, where i is the object identifier and v its corresponding value. An *object identifier* (oid) is a reference to an object, and an *object value* represents the state of the object, called an *o-value* [2]. The o-value is either: (i) a constant, (ii) an oid, (iii) a set of objects $\{o_1, \ldots, o_n\}$, where o_i-s represent o-values, or (iv) a tuple object defined as $[A_1 : o_1, \ldots, A_n : o_n]$, where o_i-s represent o-values and A_i-s are attribute names.

The data model supports two types of objects: *class objects* and *individual objects*. The class object represents an abstract concept and acts as a representation of a set of objects which share similar static structure and behavior. The interpretation of a class object is the set of objects that are called the *members* of a given class object. The interpretation of class c is denoted by $I(c)$. Furthermore, the interpretations of two classes are non-overlapping sets of object identifiers. Therefore, an individual object has exactly one parent class object.

The set of classes from a given database is organized according to the partial ordering relationship *is_a_subclass*, which we denote $\preceq_i$. The partially ordered set of classes is extended to include individual objects. The member of a given class is related to this class by the relationship $\preceq_i$. Formally, $o \in I(c) \Longrightarrow o \preceq_i c$, where o represents an individual object and c is a class object.

The *inherited interpretation* [2; 14] of class c, denoted by $I^*(c)$, includes all instances of class c, i.e. the members of class c and the members of its subclasses. Formally, $I^*(C) = \bigcup_{C_j \preceq_i C \wedge C_j \in \mathcal{V}_C} I(C_j)$, where $\mathcal{V}_C$ denotes the set of all classes from a given database.

3.2 O-Values and Types

A *type* is a pair in the form of $(S; P)$, where S represents the structure of a set of objects and P describes their behavior. This sub-section includes the description of the structural part of a type, which we call *static type*. The behavioral part of a type is not presented in this paper; its description can be found in [11]. The static type can be: (i) a primitive type, (ii) a reference type, (iii) a set-structured type and (iv) a tuple-structured type.

The primitive types are: *int*, *real* and *string*. A reference type is specified by a class object. The object identifier of the class *person*, for instance, denotes a reference type whose instances are references, i.e. object identifiers that are the elements of the class *person* interpretation. A set-structured type is defined as $S = \{S_1\}$, where S_1 is again a static type. A tuple-structured type is in the form of $S = [a_1 : S_1, \ldots, a_n : S_n]$, where a_i-s represent attribute names and S_i-s are again static types.

The interpretation of a static type is the set of o-values, the structure of which is defined by a given type. The interpretation of the primitive type is the set of constants of that type. The interpretation of a reference type is defined using the inherited class interpretation. The interpretation of a tuple structured type is $I([a_1 : T_1, \ldots, a : n : T_n]) = \{[a_1 : o_1, \ldots, a_n : o_n]; o_i \in I(T_i), i \in [1..n]\}$. Finally, the interpretation of a set-structured type is $I(\{S\}) = \{s; s \subseteq I(S)\}$.

In addition to the partial ordering relationship $\preceq_i$, a partial ordering relationship among o-values, denoted by $\preceq_o$, is defined. We call it the relationship *more_specific*. First, the partial ordering of static types is introduced. The partial ordering relationship defined among types is usually called a *subtype* relationship [14]. Intuitively, if type S is the subtype of type T, then the type S is more specific than (or refines) the type T. The reference type T_1 is the subtype of T_2 whenever there exists the *subclass* relationship between T_1 and T_2, i.e. $T_1 \preceq_i T_2$. Next, the type $\{S_1\}$ is the subtype of $\{S_2\}$, if S_1 is the subtype of S_2. Finally, $[A_1 : T_1 \ldots, A_k : T_k]$ is the subtype of $[A_1 : S_1 \ldots, A_n : S_n]$, if $k \geq n$ and T_i is the subtype of S_i, where $i \in [1..n]$. Again, as with the partially ordered set of oids, the partially ordered set of types is extended to include the instances of types. Formally, $v \in I(T) \Longrightarrow v \preceq_o T$, where T is a static type and v is an o-value. The obtained partially ordered set includes all o-values from a given database.

In a similar way to the inherited interpretation of classes, we define the *inherited interpretation* of types. Given the type T, the inherited interpretation of the type T includes the union of interpretations of the type T and all its subtypes. Formally, $I^*(T) = \bigcup_{T_j \preceq_o T \wedge T_j \in \mathcal{V}_T} I(T_j)$, where $\mathcal{V}_T$ denotes the set of all types from a given database.

Finally, the *extended interpretation* of structural types is defined. The extended interpretation of the type T, denoted by $I^\circ(T)$, includes all o-values that are more specific than T. Formally, $I^\circ(T) = \{o; o \preceq_o T\}$. The extended interpretation is used to define the semantics of OVAL variables.

4 Algebra for Objects

The algebra OVAL includes two types of operations: *model-based* and *declarative* operations. The former are used for the manipulation of object prop-

erties that are defined by the use of the database model constructs. The later are defined for expressing declarative queries on a database.

4.1 Model-based operations

The model-based operations are closely related to the constructs of the previously presented formalization of the object-oriented database model. They are intended to inquire about: (i) associations among individual objects, (ii) relationships between individual objects and class objects, and (iii) relationships among class object themselves. The use of operations is described by examples that are expressed in a predicate calculus notation.

Valuation operator

Given an object, the information describing the static properties of this object can be derived by means of the valuation operator *val*. Let us present an example of using the valuation operator. In the following expression the operator *val* is used to obtain the static properties of the class object *student*.

$$student.val = [name : string, age : int, \quad (1)$$
$$friends : \{person\},$$
$$lives\_at : string, attends : \{course\}]$$

If the valuation function is followed by the attribute name, the expression can be abbreviated using the operator denoted by "->" as it is common in procedural programming languages.

Extension operators

Two types of extension operators are defined. Firstly, the extension operator denoted by *ext* corresponds to the ordinary class interpretation which maps a class to the set of its members. Secondly, the extension operator denoted by *exts* realizes the inherited interpretation of the class that maps a class to the set of its instances.

The following query illustrates the use of extension operators. It computes the set of persons who are either employees younger than 22, or student assistants.

$$\{o; o \in person.exts \ \wedge \ (o->age < 22 \ \wedge \quad (2)$$
$$o \in employee.ext \ \vee \ o \in student\_assistant.ext)\}$$

Poset comparison operations

The simplest and most natural way to express object properties that relate to the partial ordering of objects and o-values is by using the partial ordering relationship $\preceq_o$ which is introduced in Sub-section 3.1. The comparison operations that are related to the relationship $\preceq_o$ are $\prec_o, \succ_o, \succeq_o$. Their semantics is defined in a usual manner, e.g. $a \prec_o b if and only if a \preceq_o b \wedge a \neq b$. We call these operations *poset comparison operations*.

Before illustrating the use of the poset comparison operations by some examples, the function that maps an instance object to its parent class object is defined. We name this function *class_of*. It is defined as follows: $x.class\_of = c if and only if x \in I(c)$, where $I(c)$ denotes class c interpretation. Note that an instance belongs to the exactly one class interpretation.

The following query specifies objects that are more specific than the class *lecturer* and, in the same time, the elements of either the class *student_assistant* or some more general class.

$$\{o; o \in person.exts \ \wedge \ o \prec_o lecturer \ \wedge \quad (3)$$
$$student\_assistant \preceq_o o.class\_of\}$$

In the above example the poset comparison operations are used to relate objects. In the following example we present the use of poset comparison operations for relating o-values. The query (4) filters the values of objects of the class *person*. The selected tuples have to include the values for the attributes *manager*, *friends* and *lives_at*. The value of the attribute *lives_at* has to be "Brisbane", and the value of attribute *manager* is required to be more specific than the class *lecturer*. In the similar way, the value of the attribute *friends* is required to be more specific than the type $\{student\}$. The query is formulated as follows.

$$\{v; o \in person.exts \ \wedge \ v = o.val \ \wedge \quad (4)$$
$$v \prec_o [manager : lecturer,$$
$$friends : \{student\},$$
$$lives\_at : "Brisbane"]\}$$

Closure operations

The closure operations *subcl* and *supcl* are defined on class objects. Given an argument class c, the operation *subcl* returns all subclasses of c including the class c itself. The operation *supcl* returns all superclasses of the argument class c including the class c.

The closure operations can express similar relationships among objects to those that can be expressed using the poset comparison operations. The expression $x \preceq_o y$, where x and y are classes, for instance, can be expressed as $x \in y.subcl$. The query (3) can be restated as follows.

$$\{o; o \in person.exts \ \wedge \quad (5)$$
$$o.class\_of \in lecturer.subcl \ \wedge$$
$$o.class\_of \neq lecturer \ \wedge$$
$$o.class\_of \in student\_assistant.supcl\}$$

While the poset comparison operations can serve only for expressing relationships among objects, the result of the closure operation is a set of classes that can be further queried.

Operations *lub-set* and *glb-set*

The algebra OVAL includes the operations for computing the nearest common more general and more specific objects for a given set of objects with respect to the relationship $\preceq_i$. Let consider first the use of the operation which computes the nearest common more general objects. As an example, the nearest common more general object of the set of classes $\{professor, student\_assistant\}$ is the class *lecturer*. The class *lecturer* includes all properties which are common to the class objects from the argument set.

The operator that computes the nearest common more general objects of a set of objects with respect to the relationship $\preceq_i$ is called *lub-set*. Next, the operation *glb-set* is defined to compute the set of nearest common more specific objects for a given set

of objects. Resulting objects include the properties which relate to *all* objects from the argument set.

The use of operation *lub-set* is presented in the following example. The presented expression first determines the nearest common more general objects of objects referenced by object identifiers: *peter*, *student_assistant* and *jim*. The members of the resulted classes are selected by the query. Note that *peter* and *jim* are individual objects, while the oid *student_assistant* refers to the class object.

$$\{o;\ c \in \{peter, student\_assistant, jim\}.lub\text{-}set \ \wedge \ o \in c.ext\} \quad (6)$$

Equality

The algebra OVAL provides two types of equality operations which reflect the features of the underlying data model. The first operation is the identity equality [13] denoted by the symbol "==". Two instances are *identical* if they have equal object identifiers. The second equality operation is the *value equality*. It compares objects on the basis of their values. We distinguish between two types of value equality: *complete equality* and *local equality*.

The complete equality compares two instances by comparing the values of all operand components. The operator is denoted by the symbol "=". The local equality allows the comparison of instances on the basis of the properties that pertain to the particular class. This operation is denoted by "=/class". To be able to compare two instances on the basis of the properties of the class, say C, these instances should inherit from the class C. This, of course, does not imply that they have the same parent classes. Let us present the use of local value equality by an example.

Assume that we want to compare two instances ($i1$, [*name*:*tone*, *age*:40, *works_at*:*ijs*, *salary*:10000]) and ($i2$, [*name*:*vanja*, *age*:24, *works_at*:*ijs*, *salary*:10000, *cour*:{$c1, c2$}]). The first instance is derived from the class *employee*, whereas the second one is the member of the class *student_assistant* which is a subclass of *student* and *employee*. These two instances are not value equal if all properties are considered. However, they are value equal if the local properties of the class *employee* are considered, i.e. *works_at* and *salary*.

4.2 Declarative operations

The algebra OVAL includes a set of *declarative* operations which are intended for querying a database. This set includes operations for: applying a query to the set of objects, set filtering, object restructuring, applying a query to the arbitrary nested component of object and computing transitive closure of a set of objects. The operations can be combined using the composition operator and the higher-order operations to form more complex queries.

In the following sub-sections we present some of the basic declarative operations of OVAL. The examples of using these operations for querying database schema are given.

The types of variables in queries are defined similarly to C++ variable definitions. For instance, the expression "T v;" defines the variable v of type T. The semantics of variables is defined using the extended interpretation of types I°.

Apply

The operation *apply*(f) is used to evaluate a parameter function f on the elements of the argument set. The parameter function f can be an attribute, an operation or a query.

Let us present an example of using the operation *apply*. The query described below maps a set of students into a set of student names. The *identity* function *id* is used to identify the elements of the set *studs* which is an argument of the operation *apply*.

```
{student} studs;                                    (7)
{string} str;

str = studs.apply( id->name );
```

Selection

The operation *select*(p) is used for filtering an argument set of o-values using a parameter predicate. The parameter predicate p specifies the properties of selected o-values. It can be composed of o-values and variables related by arithmetic operations, previously presented model-based operations and boolean operations. Let us illustrate the use of operation *select* for querying database schema using some examples.

The queries (3) and (4) are restated in the following two examples to illustrate the use of poset comparison operations in the context of OVAL declarative operations.

```
{person} ps;                                        (8)

ps = person.exts.
     select( id < lecturer and
             student_assistant =< id.class_of );
```

```
{person.val} pvs;                                   (9)

pvs = person.exts.
      apply( id->val ).
      select( id < [ manager:lecturer,
                     friends:{student},
                     lives_at:"Brisbane" ]);
```

The following query illustrates the use of the operation *lub-set*. The set of instances of the class *employee* is filtered by selecting the employees who work for the Computer Systems Department and are younger than 25. The operation *lub-set* than computes the closest common more general classes of the selected set of objects.

```
{employee} s;                                       (10)

s = employee.exts.
    select( id->works_at = csd and
            id->age < 25 ).
    lub-set;
```

The use of local equality is illustrated by the query (11) which selects student assistants that have the properties that relate to their role of being employees equal to the properties of an employee referenced by the variable *peter*.

```
{student_assistant} s;                                    (11)
employee peter;

s = student_assistant.exts.
    select( id.val =/employee peter.val );
```

Tuple

The operation $tuple(a_1 : f_1, \ldots, a_n : f_n)$ is a generalization of the relational *projection*. Given a set of objects as an argument of the operation, a tuple is generated for each object from the argument set. Each component of the newly created tuple is specified by the corresponding *tuple* parameter which includes the attribute name a_i and the parameter query f_i.

The query in the following example constructs the tuple for every subclass of the class *person*. Each tuple is composed of the class object identifier and the value of the class object.

```
{[ pclass: person;                                        (12)
   ptype: person.val ]} ptypes;

ptypes = person.subcl.
         tuple( pclass: id,
                ptype: id.val );
```

The tuple constructed for the class *student*, for instance, is $[pclass{:}student, ptype{:}[name{:}string, age{:}int, attends{:}\{course\}]]$. Note that the role of operator *subcl* in the above query is similar to the role of extension operator.

Group

The operation $group(a : f, b : g)$ is used for grouping of o-values resulted from the query g evaluation with respect to the result of the "key" query f. Therefore, the result of evaluating the operation $group(a : f, b : g)$ on a set of o-values is a two column table, where the first column, labeled a, stores the distinct values of the query f evaluation, and the second column, labeled b, includes the corresponding values of the query g evaluation.

In the following example the operation *group* is used for grouping the instances of the class *employee* with respect to their parent classes.

```
{[ class: employee,                                       (13)
   emps: { employee }]} EmpGroups;

EmpGroups = employee.exts.
            group( class:id->class_of,
                   emps:id );
```

5 Concluding remarks

The operations of the algebra for objects called OVAL, which are intended for querying database conceptual schema, are presented in this paper. These operations are called *model-based operations* since they are based on the concepts introduced for our formalization of the object-oriented database model. As the consequence, a tight correlation between the database model and the algebra for objects is established. Such correlation allows the algebra to support *all* aspects of the underlying database model.

References

[1] S.Abiteboul, C.Beeri, *On the Power of the Languages For the Manipulation of Complex Objects*, Verso Report No.4, INRIA, 1993

[2] S. Abiteboul, P.C. Kanellakis, *Object Identity as Query Language Primitive*, ACM SIGMOD 1989

[3] M. Atkinson et al. *The Object-Oriented Database Systems Manifesto*, Proc. First Int'l Conf Deductive and Object-Oriented Databases, Elsevier Science Publisher B. V., Amsterdam, 1989, pp. 40-57.

[4] F.Banchilion, S.Cluet, C.Deobel, *A Query Language for the O_2 Object-Oriented Database System*, Proc. 2nd Workshop on Database Programming Languages, 1989

[5] E.Bertino et al, Object-Oriented Query Languages: The Notion and Issues, IEEE TKDE, vol.4, No.3, June 1992

[6] P.Buneman, R.E.Frankel, *FQL- A Functional Query Language*, ACM SIGMOD, 1979

[7] M.Kifer et al, *Querying Object-Oriented Databases*, ACM SIGMOD 1992

[8] M.Kifer, G.Lausen, J.Wu, *Logical Foundations of Object-Oriented and Frame-Based Languages*, Technical Report 93/06, Dept. of Computer Science, SUNY at Stony Brook

[9] W.Kim, et al, *Features of the ORION Object-Oriented Database System*, 11th Chapter in *Object-Oriented Concepts, Databases and Applications*, editor W.Kim

[10] M.P. Papazoglou, *Unraveling the Semantics of Conceptual Schemas*, to appear in Comm. of ACM

[11] I.Savnik, *A Query Language for Complex Database Objects*, Ph.D. thesis, IJS-DP Technical Report, Ljubljana 1995

[12] I.Savnik, Z.Tari, T.Mohorič, *A Database Algebra for Objects*, Submitted for publication, 1995

[13] G.M.Shaw, S.B.Zdonik, *A Query Algebra for Object-Oriented Databases*, Proc. of Data Eng., IEEE, 1990

[14] S.L.Vandenberg, *Algebras for Object-Oriented Query Languages*, Ph.D. thesis, Technical Report No. 1161, University of Wisconsin, 1993

The Use of Description Logics as Database Query Languages

Klaus Schild
Daimler-Benz AG
Research and Technology
Knowledge Based Systems
Alt-Moabit 96a, D-10559 Berlin, Germany
e-mail: schild@DBresearch-berlin.de

Description Logics are knowledge representation languages set up by the development of the KL-ONE system [1]. They are used to capture the taxonomy of an application domain and to describe the application domain itself in terms of this taxonomy. These specific logics employ user-friendly variable-free notations. One of their major characteristics is their clear semantics. Without such a formal semantics, it would be impossible to state what exactly is represented by a particular representation. In other words, without formal semantics, representations would have no meaning outside the particular system in which they reside—preventing the knowledge fixed in the representation from being re-used.

The line of research set up by the KL-ONE project can be called successful in the long run. An indication for its success certainly is that the most recent successor of KL-ONE, AT&T's CLASSIC system, eventually reached the realm of a large-scale industrial application [6]. This success, however, should not obscure the fact that there is a fundamental dilemma from which all description logics suffer. In fact, despite their limited expressive power, basic inferential services such as classifying new terms into a taxonomy cannot be implemented efficiently. In particular, it is known that even in the very smallest description logic's setting, basic inferences are co-NP-hard [2].

In [4] we have shown that this fundamental dilemma can in principle be circumvented. In particular, we were able to demonstrate that tractability can generally be obtained just by eliminating any *incompleteness* from a knowledge base while the taxonomy is left unchanged. This remains true even for the most powerful description logic ever considered. The description logic we have paid attention to can be called with full right *universal* in that it encompasses all language repositories known from traditional description logics. This enables the universal description logic to define many standard data structures such as trees or directed acyclic graphs in an elegant way. In addition to traditional constructs, the universal description logic includes a general means of recursion. As is not unusual in computer science, we handled recursion with the help of least and greatest fixed-point operators. The technique employed is actually a generalization of the technique presented in [3]. These fixed-point operators turned out to be indispensable as soon as more involved concepts such as balanced trees are to be modeled. As it stands, this tractability result is of great importance. Actually, it is the very first tractability result established for a description logic which takes taxonomies into account.

On the other hand, our result can be viewed as building a bridge between traditional knowledge representation and databases. As a matter of fact, our tractability result heavily depends on the presupposition that any incomplete knowledge can be eliminated from a knowledge base. The ability to express incomplete knowledge is, of course, the very characteristic separating knowledge representation from databases. A knowledge base which is complete in this sense is, in fact, nothing but a relational database. Consequently, when viewed from the database point of view, our tractability result demonstrates that a universal description logic can be used as a powerful but tractable query language for relational databases. In this connection, it is important to note that our tractability result is to be understood in the sense of [5] in terms of the combined complexity rather than the far weaker notion of data complexity. Of course, this database point of view on description logics gives rise to several questions hardly investigated up till now. These include questions of the following kind.

1. We have shown that a universal description logic can serve as a tractable query language for databases. This means that queries to databases phrased in this description logic can be evaluated in polynomial time. But is it also the case that the universal description logic covers *all* polynomial queries?
2. How does the query power of the universal description logic relate to other more traditional database query languages?
3. Is it possible to extend our tractability result to deal with essential additional features common in relational databases such as null values?

We discussed all these questions at the workshop. A thorough investigation can be found in [4].

References

[1] R. J. Brachman and J. G. Schmolze. An overview of the KL-ONE knowledge representation

system. *Cognitive Science*, 9(2):171–216, 1985.

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

[3] K. 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.

[4] K. Schild. *A Tractable Query Language for Knowledge and Data Bases Based on a Universal Description Logic and Logics of Programs.* PhD thesis, German Research Center for Artificial Intelligence–DFKI GmbH, Saarbrücken, FRG, 1995. Forthcoming.

[5] M. Y. Vardi. The complexity of relational query languages. In *Proceedings of the 14th ACM Symposium on Theory of Computing*, pages 137–146, San Francisco, CA, 1982.

[6] J. R. Wright, E. S. Weixelbaum, G. T. Vesonder, K. Brown, S. R. Palmer, J. I. Berman, and H. H. Moore. A knowledge-based configurator that supports sales, engineering, and manufacturing at AT&T network systems. *AI-Magazine*, 14(3):69–80, 1993.

Knowledge in Interoperable and Evolutionary Systems

Nacer Boudjlida
CRIN - Bat. LORIA - Université Henri Poincaré Nancy 1, B.P. 239
54506 - Vandœuvre Lès Nancy Cedex (France)
E-mail: nacer@loria.fr

Databases, Logic Programming and Artificial Intelligence fields successfully cooperated in the area of deductive databases. Substantial results were gained in querying, albeit results on *updates and revision* are less impressive, especially from a computational perspective. Cross-fertilisation among the fields seems also very promising in the domain of dynamic and reactive systems that behavelike systems that supervise on-going activities: they must *execute* actions, *reason* about these, *gather* information about ongoing activities, *predict* possible illfunctioning, *control and coordinate* the activities, etc. New database application domains, like databases for CAD/CAM or software engineering, require this kind of functionalities. In particular, Software Engineering Environments (SEE) that support Software Process Models (SPM) falls in this category of systems: these are also called Process Centred SEE (PCSEE). SEEs usually concentrate on the support for software products development. PCSEEs provide additional supports for the activities and the agents that are implicated in to the development and the management of software projects. In this framework, SPMs is an abstract specification of how the software related activities *should* be carried out. The specification at least encompasses descriptions of the object types that are produced by the activities together with descriptions of the activities themselves and policies to be obeyed to. A PCSEE includes a *knowledge* base that contains SPMs, an object base that contains SPMs instances and software products. The PCSEE's *Software Process Engine* interprets (*enacts*) a SPM to drive the development of a software project in conformance with an *instantiated SPM*. The *Process Engine* is a set of mechanisms that controls the ongoing activities and provides a set of assistance facilities like predicting future states of the objects, explaining how a given state has been reached or can be reached, and so on. The mechanisms that constitute the *Process Engine* share and inter-operate on the knowledge base that contains the specifications of the SPMs and the object base that contains the products being developed and the gathered information about the activities that have been performed.

The environment is viewed as a collection of *tools* that cooperate in the support of the activities, that communicate and exchange objects, messages and events, and inter-operate on the ob jects in the environment's bases. Considering the variety of tools in a PCSEE, their ability to *inter-operate* on a same set of objects is crucial for the evolution of the environment. Inter-operability may be achieved through a common representation of the knowledge and the ob ject bases or through specific mechanisms that restructure objects, i.e that adapt their representation to the in ter-operable tools. Syntactic-based approaches to object interchange for inter-operability, like those based on an Interface Definition Language, must be extended by knowledge on the objects contents. We experimented a knowledge-based implementation of object restructuring and we are currently investigating the potential mutual contributions of the works on data interchange (like Common Data Interchange Format) and knowledge interchange (like KIF and KQML)to incorporate more knowledge in to object descriptions and to exploit it in the object restructuring process (this process can be viewed as a dynamic knowledge-based mechanism to achieve *ad hoc polymorphism with coercion*).

Objects in PCSEEs are no more "classical" database objects as are "flat" relations in relational databases. Objects, like design documents or source code, are complex objects with possible nesting (*is_part_of relationship*) and specialized/generalized objects (*is_a relationship*). Moreover, the associated Data Base Management System, called Object Management System, must be extendible with new object types. It must also support cooperative work, active rules, long- term activities and object versioning. Indeed, experimental activities like software engineering often require going back to previous steps or previous states of objects: versioning is then "a must" as it is to enable various evolutions in PCSEEs. *Evolution* can take place at different levels: the environment's hosting platform, the SPM level, as well as the SPMs' instances and the object base levels. Existing knowledge and objects *must be adapted* to the changes, i.e. multiple versions of the knowledge and the object bases may be maintained, every version corresponding to a version of the knowledge base and the object base specification, or alternatively, the existing knowledge base and object base may *migrate* to meet their respective new specifications. This appeals for mechanisms to *manage* knowledge and object schema *evolution*

and versioning, mechanisms to *re-use* existing SPMs and objects, and means to *analyse the impacts of a change*, etc. Change impact-analysis and change side-effects propagation meet the *frame* and the *ramification problems* in knowledge bases revision.

In this position paper, we argue that management and reasoning on structurally complex objects in the framework of dynamic systems, like PCSEEs, require knowledge concerning the knowledge itself, the objects and the actions that may be performed on the objects. It also requires a kind of "*reflexivity*" to enact (i.e. execute) the knowledge provided by the Process Models and to manipulate it, notably to ensure its evolution. Reflexivity is the fact that Software Process Models are considered as objects: so they can be updated and revised as any other object. At any level it occurs, evolution requires impact analysis similar to the resolution of the frame and ramification problems. Further, similarly to multi-agents systems like blackboard systems, interoperability of the tools must be ensured not only to enable them cooperate in carrying out the activities, but also to adapt existing knowledge and objects to possible evolutions. This feature favours knowledge and object re-use and must be founded on objects' structure and content.

Packaging Knowledge into Metaobjects

David Edmond, Mike Papazoglou, Nick Russell and Zahir Tari
School of Information Systems, Queensland University of Technology
GPO Box 2434 Brisbane Queensland 4001 Australia
email: {davee,mikep,nickr,zahirt}@icis.qut.edu.au

Abstract

The use of reflection [Mae87; HY88; Pae90] is particularly applicable to multi-database systems and to cooperating systems in general. We view such systems as (1) being distributed over a common communication network, and (2) working towards some common goal. Cooperation is achieved by coordinating and exchanging information and expertise. Conventional database systems are *not* cooperative: the knowledge they contain is inaccessibly buried within application code.

In [EPT95], we discuss the R-OK Model and suggest that this problem may be overcome by surrounding each local database system with a layer of special reflective metaobjects. The term *meta* object is used only to indicate the relation of such an object to the object it describes. A metaobject is just another object, with structure and behaviour. These objects are used to capture domain and operational knowledge, and to describe, at least in part, remote systems and to monitor task-oriented activities. In this way, we can turn interconnected conventional database systems into a set of cooperating knowledge-based systems. In the R-OK model, every object has access to four metaobjects:

1. A `state` metaobject knows the structure of any associated object, naming each attribute and specifying its type. For example, if the application domain was a simple savings bank, then a savings object might be described by a `state` metaobject as having an account Id, a balance and a minimum-balance-this-month attributes. By its nature, such a metaobject provides only a static picture of an object.

2. A `can` metaobject knows about the behaviour of any associated object – it knows what an object *can* do. This object may also be associated with a number of domain objects, all of which share the same (outward) behaviour. In this metaobject, activities are described in terms of pre- and post-conditions. In the bank example, the post-condition of the *Withdraw* methods might require that if the new balance is less than the previous minimum, then the minimum is reset. Such a metaobject allows a system to consider possible behaviour and its consequences to the object(s) concerned. It also allows a system to investigate alternative ways of achieving some goal. Should it be necessary, for example, to increase the balance of an account, it may be that there are two ways of accomplishing this – either through a conventional deposit *or* by applying interest to the account.

3. A `loc` metaobject knows how to *locate* attributes and *execute* the methods of an object. This metaobject contains:

- A *Lookup* table which indicates how each attribute of the associated domain object is materialised. This reification is accomplished by *surrogate* objects. These metalevel objects have specific knowledge of the location of data.
- A *Do* table which contains, for each method, procedural descriptions of how that method is effected. Should an interpreter be used to execute such code, it will use the *Lookup* table to resolve symbols that it does not recognise.

4. An `act` metaobject knows about the *activity* in which some group of objects is involved. It is a task-oriented object that monitors the activities of the collection of objects that constitute *its* domain.

Reflection, by means of these four metaobjects, not only allows descriptions of the capabilities of existing information systems and their inter-relationships but also facilitates the specification and implementation of a new system by means of composition, that is, by drawing upon the functionality of existing systems.

Because a metaobject is just another object, with structure and behaviour, we may ask whether it too has access to descriptions of itself. In [EPT94], we use these constructs to penetrate aspects of information systems that are usually closed to us; on particular, we look at two examples of how knowledge of behind-the-scene actions may be used to enable cooperation.

In this presentation, we will discuss how the model may be used to provide translations from an object-oriented model into a relational database.

References

[EPT94] Edmond D., Papzoglou M. and Tari Z. (1994) "Using Reflection as a Means of Achieving Cooperation", Procs of FGCS'94 Workshop on Heterogeneous Cooperative Knowledge-Bases, Tokyo.

[EPT95] Edmond D., Papzoglou M. and Tari Z. (1995) "R-OK: A Reflective Model for Distributed Object Management", Procs of RIDE'95 (Research Issues in Data Engineering), Taiwan.

[HY88] Y. Honda and A. Yonezawa, "Debugging Concurrent Systems Based on Object Groups", Procs of ECOOP'88: European Conference on Object-oriented Programming, Oslo, Norway.

[Mae87] Maes P. (1987). "Concepts and Experiments in Computational Reflection", OOPSLA'87.

[Pae90] Paepcke A. (1990). "PCLOS: Stress Testing CLOS", OOPSLA'90.

Supporting Autonomy for Information Systems in a Changing Environment

J. Kusch and G. Saake
Institut für Technische Informationssysteme, Abteilung Datenbanken,
Otto-von-Guericke-Universität Magdeburg, D–39106 Magdeburg, Germany,
E-Mail: {kusch|saake}@iti.cs.uni-magdeburg.de

Abstract

Availability and scalability are important features of information systems. To gain this kind of requirements, we propose a distributed schema catalog in conjunction with appropriate development phases. The distributed schema catalog has to support distribution transparency including partitioning and replication to be flexible for changes within organization structures. Additionally, phases of autonomy design have to appoint the adequate usage of distribution transparency aspects to enable execution autonomy by a minimum of replication.

This paper only gives a brief overview concerning the development of distributed information systems based on object-oriented structures.

1 Motivation

In the development of information systems, object-oriented specification (e.g. TROLL [Jungclaus *et al.*, 1995]) is useful for conceptually modeling of the universe of discourse. Viewing an information system as a collection of communicating objects is close to the intuitive perception of such systems on a conceptual level. A uniform lifecycle model of objects (or agents) covers the description of structural and behavioral aspects. Nowadays complex information systems (e.g. knowledge bases) are an integral part of organizations [Fasnacht, 1993]. In order to be conducive for interconnecting departments, information systems have to be flexible to accommodate for changes in organization structures. Mandatory features are *distribution* for decentralization support and *scalability* for modular system increase [Simon, 1995]. Furtheron processing in parallel enables an important *speedup* [Gray, 1995]. Thus, the task is to map the global conceptual model to computational reality as a distributed infrastructure. However the advantages of distribution are only usable by an adequate support of software.

Object-oriented specification consists of a global abstract description as conceptual model without non-functional requirements (e.g. distribution, persistence, exception handling). A distributed infrastructure consists of a set of inter-connected loosely coupled nodes with own processors and disk-spaces. *On which 'level' distribution has to be combined with the conceptual model?* A complex information system structure, which is described as conceptual entirety, is transparently distributed over several nodes. Important is a partial use of the information system, although not all nodes are available or reachable due to a site failure. *How to achieve a maximum of node autonomy for a distributed system (C. J. Dates first (of 12) rule for distributed database systems [Date, 1990]) supported by the distributed schema catalog?*

Developing highly available information systems, we propose a *distributed schema catalog* in conjunction with *autonomy design phases*:

Distributed schema catalog. In general, distributed database systems have to deal with a lot of *tradeoffs* [Rahm, 1994; Bell and Grimson, 1992; Özsu and Valduriez, 1991], e.g. data replication versus data transfer, reuse versus autonomy, transparency versus efficiency. Our goals in gaining distribution for an object-oriented data model are:

- *Distribution transparency*: For scalability support transparency of location and migration enables objects to be used without knowledge of their location and movement of objects within a system without affecting the operations [Herbert, 1989]. In conjunction with horizontal and vertical class partitioning, the system could be expanded in scale without changing the specification or the references.
- *Execution autonomy*: For decentralization support, neither federation does interfere with local (or subsystem) operations nor any knowledge of the federation is needed for performing local (or subsystem) operations [Kalathil and Belford, 1994; Veijaleinen and Popescu-Zeletin, 1988].

Existing approaches in the area of database systems mostly do not pay enough attention to the opportunities of autonomy and scalability, which are getting increasingly important

by a new generation of *parallel hardware clusters* [Gray, 1995].

Additionally design phases are mandatory to control the transparency aspects of distribution with the view to execution autonomy.

Autonomy design phases. Multiple allocation of data within a distributed environment promises an increase of performance and availability and a decrease of communication. Contrary to this disadvantages are memory consumption and consistency maintenance in case of a site failure. Thus we aspire execution autonomy enabled by a minimum of replication, which has to be guaranteed by an extended development process:

- In *former development phases* autonomy modules have to be modeled conceptually, which are based on informal autonomy requirements.
- In *later development phases* the initial distribution structure has to be appointed, which is based on the modeled autonomy modules. Altogether this effects the transparency aspects of distribution, e.g. object class location and partitioning and object location and replication.

Evolution requirements of the distribution structure are mostly not equal to those of the conceptual model. Thus the additional autonomy design phases has to be performed independently from the remaining development phases.

To focus our approach within the area of object-oriented specification and distributed databases, this work is based on a homogeneous integrated schema and covers only *structural aspects* including integrity constraints.

2 Object-Oriented Structures

Conceptual modeling of information systems requires the description of the application domain, the so-called *universe of discourse*, on a high abstraction level. Looking at an information system and its environment as a collection of interacting *objects* seems to be a very natural way for conceptualizing information structures and processes. Objects have a local state, show a specific behavior, communicate with other objects and may be themselves composed from smaller objects. This observation is confirmed by the current success of object-oriented analysis and design frameworks, e.g. [Rumbaugh *et al.*, 1991].

This paper emphasizes only the *structural aspects* of the object model as base of data maintenance. Thus, execution autonomy refers to data model operations, e.g. creation, deletion, migration of objects and object classes. Behavioral features, e.g. processes, synchronization and transactions, are not regarded. Mandatory structural features of object-oriented database systems and information systems as pointed out in i.e. [Jungclaus *et al.*, 1995; Cattel, 1994; Ahmed *et al.*, 1991; Rumbaugh *et al.*, 1991; Atkinson *et al.*, 1990] are

- *class types*,
- *objects* with a global, immutable and system wide unique *object identity*,
- *object classes*,
- *specialized classes* supporting semantic inheritance [Saake, 1993],
- *component relations* to model complex objects and
- object preserving *views*.

These abstractions of the conceptual model are grouped into an object base. Thus, each object base contains a set of classes, which consist of a class type and a set of objects.

With the view to a later implementation, the chosen abstractions gain a "small is beautiful" object model.

3 Distributed Schema Catalog

Supporting autonomy for information systems in a changing environment, a schema catalog is introduced which enables *distribution transparency*, and in conjunction with internal structures an increase of availability. To characterize the distributed schema catalog, the aspects *data definition interface*, *data-logical architecture* and *meta schema* are briefly represented.

Data definition interface. Trends about information systems and knowledge bases point out the everlasting complexity increase for data processing [Hwang and Briggs, 1989]. Representing complex data in an adequate way, a variety of abstractions are needed. To take some of the load off the development phases, the data definition language of a distributed schema catalog has to support the abstractions of the conceptual model, i.e. object classes, specialized classes, component relations, and views.

This has an effect on the development phases, which are discussed in section 4 in detail. The development process gets simplified, since there is no transformation necessary between the conceptual model and the phase of implementation. Furtheron the phase of distribution design is superfluous in early stages of development. Considering changing environments, aspects of distribution, i.e. location, partitioning and replication, are not statically decidable in advance. Due to distribution transparency, the phase of distribution design could be displaced as a later *distribution tuning phase*, dependent of the data access profile. This strategy seems to be more adequate for the design of distributed information systems.

Data-logical architecture. Base of the distributed schema catalog is a distributed infrastructure, consisting of a graph of nodes and inter-node-connections. Each node contains a set of instance-buffers for persistently maintaining data. The inter-node-connections are established via *broadcast channels* to neglect network partitioning problems.

Due to the increase of hardware performance and complexity, the schema catalog has to perform the mapping from the conceptual specification structure

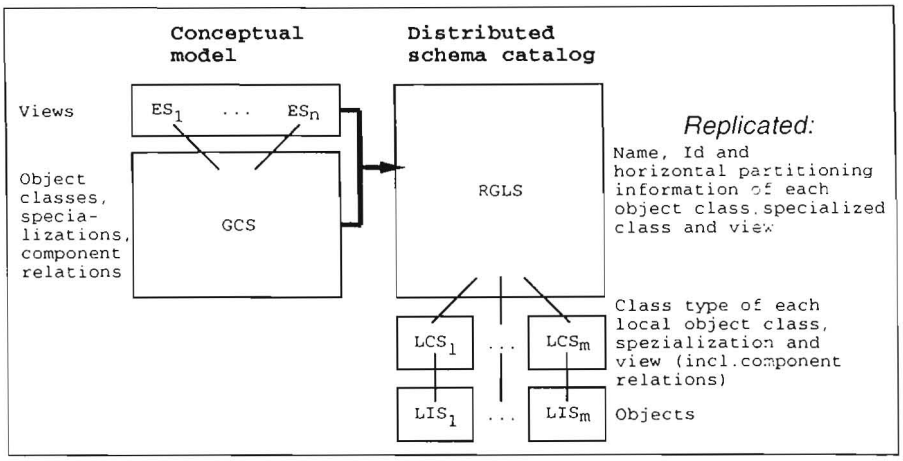

Figure 1: Data-logical Architecture of the Distributed Schema Catalog

to the distributed infrastructure. To support execution autonomy and scalability within a changing environment, location and partitioning of object classes and location and replication of objects have to be archived in an transparent way. Object classes should be located to a set of nodes (horizontal partitioning). Objects of specialized classes should be located to a set of instance-buffers of different nodes (vertical partitioning). The consideration of object encapsulation leads to the fact, that specialization is the only way of vertically partitioning objects. An object should be located to several nodes within a horizontally partitioned object class (replication).

As architecture of the distributed schema catalog (figure 1) our approach proposes a *replicated global location schema* (RGLS) with distributed local conceptual schemas (LCS's) and appropriate local internal schemas (LIS's). This depicts a specialization of the ANSI-4-level-architecture. The RGLS, which is replicated to each node, contains the name, identification and horizontal partitioning information of each object class. As a *virtual global schema*, the RGLS offers with respect to autonomy minimal global knowledge for the access of remote information. The LCS's contain class type information of each object class, which is replicated to those sites, where parts of their extension are located. Thus, objects are co-located with their types. At least the LIS's contain objects, which are maintained via a set of instance-buffers.

Further internal details, which are mandatory for scalability and execution autonomy are briefly depicted:

- Object identifiers are built as *compound object identifier* of a class Id and a class internal Id. Class internal Id's are maintained via areas of free Id's.
- Relations between and within object classes are maintained through object identifier references, which implicitly contain location information (via class information).
- Replication of key attributes has to be maintained implicitly.
- If a failed node gets online, several complex data updating operations have to be performed on the whole object base.

This allows to perform consistency preserving operations even if nodes in the context of this operations are offline.

Access to object properties is always directed to the RGLS, which determines a path over a hierarchical structure of LCS's and LIS's. In conjunction with compound object identifiers *location transparency* is enabled.

Meta schema. The conceptual model has to be free of non-functional requirements. For orthogonally influencing distribution, a meta schema is introduced, which is modeled as a reflexive system [Maes, 1988] exclusively with the selected abstractions of the object model. To maintain transparency aspects, the meta schema is modeled as *a set of vertically partitioned classes* for each node of the distributed infrastructure, specialized from *a totally horizontally partitioned class* with replicated objects. Replicated objects contain replicated schema information of horizontally partitioned classes, whereas the specialized parts are managing node related class information (i.e. only local objects of an object class and related instance-buffers). This meta schema architecture enables a sound maintenance of horizontally partitioned classes and replicated objects. Distributed administration is performed by the events of the meta schema.

4 Autonomy Design

Nowadays information systems are based on distributed infrastructures to manage the requirements of the users. To take advantage of a decentralizable platform several existing disadvantages [Rahm, 1994; Bell and Grimson, 1992; Özsu and Valduriez, 1991] of distributed database systems have to be regarded. Due to the possibility of node failures (e.g. power failure, hardware- or software failure, installation or maintenance tasks or user control failure) autonomy considerations have to be taken into account. This should save failure costs of the whole information system.

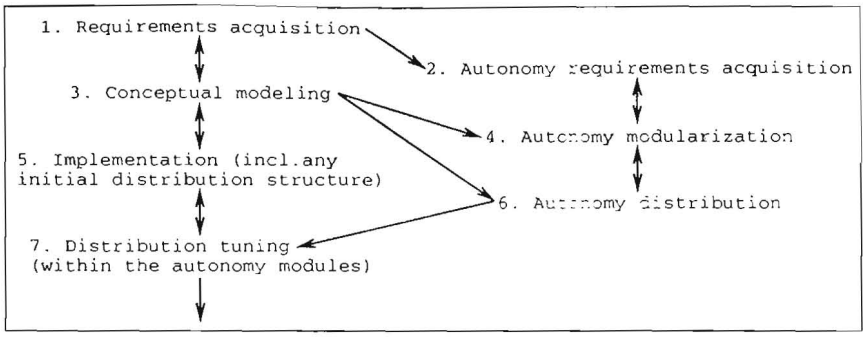

Figure 2: Phases of Autonomy Design

Developing highly available information systems, the development process has to be extended. Base of the development process is the presented distributed schema catalog which supports location, partitioning and replication transparency. Additional design phases for information systems are mandatory to control the transparency aspects of distribution, and thus serve the availability requirements by a minimum of data replication. Results of these additional design phases are the initial location including partitioning of object classes and the initial location and replication of objects.

We propose the following phases of design (figure 2) to develop highly available information systems:

1. *Requirements acquisition*: Informal description of the universe of discourse.
2. *Autonomy requirements acquisition* (based on requirements acquisition): Informal description of availability requirements within the organization structure.
3. *Conceptual modeling* (based on requirements acquisition): Formal specification of a global object model.
4. *Autonomy modularization* (based on autonomy requirements acquisition and conceptual modeling): Formal specification of a set of autonomy modules which represent autonomously maintainable areas of organization structures. Each autonomy module consists of a set of object classes, and a set of related nodes, on which this classes have to be at least located.
5. *Implementation* (based on conceptual modeling): Due to the abstraction level of the data definition interface, the conceptual model could be directly implemented. The replicated global location schema (RGLS) of the distributed schema catalog offers a "virtual global schema" to each node, independently from the initial location of the implementation.
6. *Autonomy distribution* (based on conceptual modeling and autonomy modularization): Autonomy conflicts arise through *autonomy module overlapping relations* between abstractions of the object model, i.e. specified by integrity constraints, component and specialization relations. Thus, an algorithm, which cannot be presented here in detail, automatically generates *a set of locations for each object class* (as horizontal partitioning), which is the base of mandatory object replication. Additionally *a set of birth events for objects classes* is generated to control location and replication of created objects. Altogether, a distribution structure is generated, which achieves execution autonomy by a minimum of object replication.
7. *Distribution tuning* (based on implementation and autonomy distribution): Dependent on later data access, the distribution structure within the autonomy modules could be optimized.

Phases 1, 3. 5 and 7 (figure 2) enable an evolutionary development strategy, likewise the phases 2, 4 and 6. Due to the different evolution requirements of the conceptual model and the distribution structure, the phases of autonomy design could be performed independently.

5 Outlook

The presented *specification language independent* work depicts fundamentals for the development of highly available and scalable information systems.

Object-oriented specification integrates structural and behavioral aspects of modeling. Thus one important enhancement of our approach is the consideration of *object behavior* with the view of autonomy and parallelism. Here, transactions and commit protocols within distributed systems have to be taken into account. To support implicit parallelism, we propose asynchronous communication with implicit synchronization. This could be performed by dataflow driven data evaluation [Lee and Hurson, 1994] via pipelining.

The disadvantages of optimization within our approach, which are evoked by transparent distribution, could be improved by dynamic query optimization. *Query objects* [Kusch, 1994; Jungclaus *et al.*, 1991] with internal knowledge of the state of the infrastructure are a first approach for this problem.

References

[Ahmed *et al.*, 1991] S. Ahmed, A. Wong, D. Sriram, and R. Logcher. A Comparison of Object-Oriented Database Management Systems for En-

gineering Applications. Technical report, Massachusets Institute of Technology, Department of Civil Engineering, 1991.

[Atkinson *et al.*, 1990] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik. The Object-Oriented Database System Manifesto. In W. Kim, J.-M. Nicolas, and S. Nishio, editors, *Deductive and Object-Oriented Databases*, pages 223–240. North Holland, 1990.

[Bell and Grimson, 1992] D. Bell and J. Grimson. *Distributed Database Systems.* Addison-Wesley, 1992.

[Cattel, 1994] R. G. G. Cattel. *The Object Database Standard: ODMG-93.* Morgan Kaufmann, 1994.

[Date, 1990] C. J. Date. *An Introduction to Database Systems, 5th Edition.* Addison-Wesley, 1990.

[Fasnacht, 1993] D. Fasnacht. *Koordination verteilter und heterogener Datenbanksysteme.* Verlag Josef Eul, 1993.

[Gray, 1995] J. Gray. Super-Servers: Commodity Computer Clusters Pose a Software Challenge. In G. Lausen, editor, *Datenbanksysteme in Büro, Technik und Wissenschaft*, pages 30–47. Springer-Verlag, 1995.

[Herbert, 1989] A. Herbert. *The ANSA reference manual.* Cambridge, U.K.: Architecture Projects Management Limited, 1989.

[Hwang and Briggs, 1989] K. Hwang and F. A. Briggs. *Computer Architecture and Parallel Processing.* McGraw-Hill, 1989.

[Jungclaus *et al.*, 1991] R. Jungclaus, G. Saake, and C. Sernadas. Using Active Objects for Query Processing. In R. Meersman, W. Kent, and S. Khosla, editors, *Object-Oriented Databases: Analysis, Design and Construction (Proc. of the 4th IFIP WG 2.6 Working Conf. DS-4, Windermere (UK), 1990)*, pages 285–304. North-Holland, Amsterdam, 1991.

[Jungclaus *et al.*, 1995] R. Jungclaus, G. Saake, T. Hartmann, and C. Sernadas. TROLL – A Language for Object-Oriented Specification of Information Systems. *ACM Transactions on Information Systems*, 1995. *To appear.*

[Kalathil and Belford, 1994] B. J. Kalathil and G. G. Belford. Supporting Local Autonomy in a Distributed Object-Oriented Database. In T. Özsu, U. Dayal, and P. Valduriez, editors, *Distributed Object Management*, pages 347–352. Morgan Kaufmann Publishers, 1994.

[Kusch, 1994] J. Kusch. Ein Ansatz zur Operationalisierung deskriptiver Anfragen durch Anfrageobjekte. In S. Conrad, P. Löhr, and G. Saake, editors, *"Grundlagen von Datenbanken"*, pages 92–96. Otto-von-Guericke-Universität Magdeburg, Institut für Technische Informationssysteme, Bericht 94-01, 1994.

[Lee and Hurson, 1994] B. Lee and A. R. Hurson. Dataflow Architectures and Multithreading. *IEEE Computer*, 27(8):27–39, 8 1994.

[Maes, 1988] P. Maes. *Meta-Level Architectures and Reflection.* Elsevier Science Publishers B. V., 1988.

[Özsu and Valduriez, 1991] M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems.* Prentice Hall, 1991.

[Rahm, 1994] E. Rahm. *Mehrrechner-Datenbanksysteme.* Addison-Wesley, 1994.

[Rumbaugh *et al.*, 1991] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-oriented modeling and design.* Prentice-Hall, 1991.

[Saake, 1993] G. Saake. *Objektorientierte Spezifikation von Informationssystemen.* Teubner, Stuttgart/Leipzig, 1993.

[Simon, 1995] A. R. Simon. *Strategic Database Technology: Management for the year 2000.* Morgan Kaufmann Publishers, Inc., 1995.

[Veijaleinen and Popescu-Zeletin, 1988] J. Veijaleinen and R. Popescu-Zeletin. Multidatabase systems in ISO/OSI environments. In N. Malagardis and T. Williams, editors, *Standards in Information Technology and Industrial Control*, pages 83–97. North-Holland, 1988.

Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

-Bibliothek, Information
und Dokumentation (BID)-
PF 2080
67608 Kaiserslautern
FRG

Telefon (0631) 205-3506
Telefax (0631) 205-3210

e-mail
dfkibib@dfki.uni-kl.de
WWW
http://www.dfki.uni-sb.de/dfkibib

Veröffentlichungen des DFKI

Die folgenden DFKI Veröffentlichungen sowie die aktuelle Liste von allen bisher erschienenen Publikationen können von der oben angegebenen Adresse oder (so sie als per ftp erhaeltlich angemerkt sind) per anonymous ftp von ftp.dfki.uni-kl.de (131.246.241.100) im Verzeichnis pub/Publications bezogen werden. Die Berichte werden, wenn nicht anders gekennzeichnet, kostenlos abgegeben.

DFKI Publications

The following DFKI publications or the list of all published papers so far are obtainable from the above address or (if they are marked as obtainable by ftp) by anonymous ftp from ftp.dfki.uni-kl.de (131.246.241.100) in the directory pub/Publications.
The reports are distributed free of charge except where otherwise noted.

DFKI Research Reports

1995

RR-95-11
Anne Kilger, Wolgang Finkler
Incremental Generation for Real-Time Applications
47 pages

RR-95-09
M. Buchheit, F. M. Donini, W. Nutt, A. Schaerf
A Refined Architecture for Terminological Systems: Terminology = Schema + Views
71 pages

RR-95-07
Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, Werner Nutt
The Complexity of Concept Languages
57 pages

RR-95-04
M. Buchheit, H.-J. Bürckert, B. Hollunder, A. Laux, W. Nutt,
M. Wójcik
Task Acquisition with a Description Logic Reasoner
17 pages

RR-95-03
Stephan Baumann, Michael Malburg, Hans-Guenther Hein, Rainer Hoch,
Thomas Kieninger, Norbert Kuhn
Document Analysis at DFKI
Part 2: Information Extraction
40 pages

RR-95-02
Majdi Ben Hadj Ali, Frank Fein, Frank Hoenes, Thorsten Jaeger,
Achim Weigel
Document Analysis at DFKI
Part 1: Image Analysis and Text Recognition
69 pages

1994

RR-94-39
Hans-Ulrich Krieger
Typed Feature Formalisms as a Common Basis for Linguistic Specification.
21 pages

RR-94-38
Hans Uszkoreit, Rolf Backofen, Stephan Busemann, Abdel Kader Diagne,
Elizabeth A. Hinkelman, Walter Kasper, Bernd Kiefer, Hans-Ulrich Krieger,
Klaus Netter, Günter Neumann, Stephan Oepen, Stephen P. Spackman.
DISCO–An HPSG-based NLP System and its Application for Appointment Scheduling.
13 pages

RR-94-37
Hans-Ulrich Krieger, Ulrich Schäfer
TDL - A Type Description Language for HPSG, Part 1: Overview.
54 pages

RR-94-36
Manfred Meyer
Issues in Concurrent Knowledge Engineering. Knowledge Base and Knowledge Share Evolution.
17 pages

RR-94-35
Rolf Backofen
A Complete Axiomatization of a Theory with Feature and Arity Constraints
49 pages

RR-94-34
Stephan Busemann, Stephan Oepen, Elizabeth A. Hinkelman,
Günter Neumann, Hans Uszkoreit
COSMA – Multi-Participant NL Interaction for Appointment Scheduling
80 pages

RR-94-33
Franz Baader, Armin Laux
Terminological Logics with Modal Operators
29 pages

RR-94-31
Otto Kühn, Volker Becker, Georg Lohse, Philipp Neumann
Integrated Knowledge Utilization and Evolution for the Conservation of Corporate Know-How
17 pages

RR-94-23
Gert Smolka
The Definition of Kernel Oz
53 pages

RR-94-20
Christian Schulte, Gert Smolka, Jörg Würtz
Encapsulated Search and Constraint Programming in Oz
21 pages

RR-94-18
Rolf Backofen, Ralf Treinen
How to Win a Game with Features
18 pages

RR-94-17
Georg Struth
Philosophical Logics—A Survey and a Bibliography
58 pages

RR-94-16
Gert Smolka
A Foundation for Higher-order Concurrent Constraint Programming
26 pages

RR-94-15
Winfried H. Graf, Stefan Neurohr
Using Graphical Style and Visibility Constraints for a Meaningful Layout in Visual Programming Interfaces
20 pages

RR-94-14
Harold Boley, Ulrich Buhrmann, Christof Kremer
Towards a Sharable Knowledge Base on Recyclable Plastics
14 pages

RR-94-13
Jana Koehler
Planning from Second Principles—A Logic-based Approach
49 pages

RR-94-12
Hubert Comon, Ralf Treinen
Ordering Constraints on Trees
34 pages

RR-94-11
Knut Hinkelmann
A Consequence Finding Approach for Feature Recognition in CAPP
18 pages

RR-94-10
Knut Hinkelmann, Helge Hintze
Computing Cost Estimates for Proof Strategies
22 pages

RR-94-08
Otto Kühn, Björn Höfling
Conserving Corporate Knowledge for Crankshaft Design
17 pages

RR-94-07
Harold Boley
Finite Domains and Exclusions as First-Class Citizens
25 pages

RR-94-06
Dietmar Dengler
An Adaptive Deductive Planning System
17 pages

RR-94-05
Franz Schmalhofer, J. Stuart Aitken, Lyle E. Bourne jr.
Beyond the Knowledge Level: Descriptions of Rational Behavior for Sharing and Reuse
81 pages

RR-94-03
Gert Smolka
A Calculus for Higher-Order Concurrent Constraint Programming with Deep Guards
34 pages

RR-94-02
Elisabeth André, Thomas Rist
Von Textgeneratoren zu Intellimedia-Präsentationssystemen
22 Seiten

RR-94-01
Elisabeth André, Thomas Rist
Multimedia Presentations: The Support of Passive and Active Viewing
15 pages

1993

RR-93-48
Franz Baader, Martin Buchheit, Bernhard Hollunder
Cardinality Restrictions on Concepts
20 pages

RR-93-46
Philipp Hanschke
A Declarative Integration of Terminological, Constraint-based, Data-driven, and Goal-directed Reasoning
81 pages

RR-93-45
Rainer Hoch
On Virtual Partitioning of Large Dictionaries for Contextual Post-Processing to Improve Character Recognition
21 pages

RR-93-44
Martin Buchheit, Manfred A. Jeusfeld, Werner Nutt, Martin Staudt
Subsumption between Queries to Object-Oriented Databases
36 pages

RR-93-43
M. Bauer, G. Paul
Logic-based Plan Recognition for Intelligent Help Systems
15 pages

RR-93-42
Hubert Comon, Ralf Treinen
The First-Order Theory of Lexicographic Path Orderings is Undecidable
9 pages

RR-93-41
Winfried H. Graf
LAYLAB: A Constraint-Based Layout Manager for Multimedia Presentations
9 pages

RR-93-40
Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, Werner Nutt,
Andrea Schaerf
Queries, Rules and Definitions as Epistemic Statements in Concept Languages
23 pages

RR-93-38
Stephan Baumann
Document Recognition of Printed Scores and Transformation into MIDI
24 pages

RR-93-36
Michael M. Richter, Bernd Bachmann, Ansgar Bernardi, Christoph Klauck,
Ralf Legleitner, Gabriele Schmidt
Von IDA bis IMCOD: Expertensysteme im CIM-Umfeld
13 Seiten

RR-93-35
Harold Boley, François Bry, Ulrich Geske (Eds.)
Neuere Entwicklungen der deklarativen KI-Programmierung — *Proceedings*
150 Seiten

Note: This document is available for a nominal charge of 25 DM (or 15 US-$).

RR-93-34
Wolfgang Wahlster
Verbmobil Translation of Face-To-Face Dialogs
10 pages

RR-93-33
Bernhard Nebel, Jana Koehler
Plan Reuse versus Plan Generation: A Theoretical and Empirical Analysis
33 pages

RR-93-32
David R. Traum, Elizabeth A. Hinkelman
Conversation Acts in Task-Oriented Spoken Dialogue
28 pages

RR-93-31
Elizabeth A. Hinkelman, Stephen P. Spackman
Abductive Speech Act Recognition, Corporate Agents and the COSMA System
34 pages

RR-93-30
Stephen P. Spackman, Elizabeth A. Hinkelman
Corporate Agents
14 pages

RR-93-29
Armin Laux
Representing Belief in Multi-Agent Worlds via Terminological Logics
35 pages

D-93-08
Thomas Kieninger, Rainer Hoch
Ein Generator mit Anfragesystem für strukturierte Wörterbücher zur Unterstützung von Texterkennung und Textanalyse
125 Seiten

D-93-07
Klaus-Peter Gores, Rainer Bleisinger
Ein erwartungsgesteuerter Koordinator zur partiellen Textanalyse
53 Seiten

D-93-06
Jürgen Müller (Hrsg.)
Beiträge zum Gründungsworkshop der Fachgruppe Verteilte Künstliche Intelligenz, Saarbrücken, 29. - 30. April 1993
235 Seiten

Note: This document is available for a nominal charge of 25 DM (or 15 US-$).

D-93-05
Elisabeth André, Winfried Graf, Jochen Heinsohn, Bernhard Nebel,
Hans-Jürgen Profitlich, Thomas Rist, Wolfgang Wahlster
PPP: Personalized Plan-Based Presenter
70 pages

D-93-04
Technical Staff
DFKI Wissenschaftlich-Technischer Jahresbericht 1992
194 Seiten

D-93-03
Stephan Busemann, Karin Harbusch (Eds.)
DFKI Workshop on Natural Language Systems: Reusability and Modularity - Proceedings
74 pages

D-93-02
Gabriele Schmidt, Frank Peters, Gernod Laufkötter
User Manual of COKAM+
23 pages

D-93-01
Philipp Hanschke, Thom Frühwirth
Terminological Reasoning with Constraint Handling Rules
12 pages

D-94-03
Franz Schmalhofer
Maschinelles Lernen: Eine kognitionswissenschaftliche Betrachtung
54 Seiten

Note: This document is no longer available in printed form.

D-94-02
Markus Steffens
Wissenserhebung und Analyse zum Entwicklungsprozeß eines Druckbehälters aus Faserverbundstoff
90 pages

D-94-01
Josua Boon (Ed.)
DFKI-Publications: The First Four Years
1990 - 1993
75 pages

1993

D-93-27
Rolf Backofen, Hans-Ulrich Krieger, Stephen P. Spackman,
Hans Uszkoreit (Eds.)
Report of the EAGLES Workshop on Implemented Formalisms at DFKI, Saarbrücken
110 pages

D-93-26
Frank Peters
Unterstützung des Experten bei der Formalisierung von Textwissen INFOCOM - Eine interaktive Formalisierungskomponente
58 Seiten

D-93-25
Hans-Jürgen Bürckert, Werner Nutt (Eds.)
Modeling Epistemic Propositions
118 pages

Note: This document is available for a nominal charge of 25 DM (or 15 US-$).

D-93-24
Brigitte Krenn, Martin Volk
DiTo-Datenbank: Datendokumentation zu Funktionsverbgefügen und Relativsätzen
66 Seiten

D-93-22
Andreas Abecker
Implementierung graphischer Benutzungsoberflächen mit Tcl/Tk und Common Lisp
44 Seiten

Note: This document is no longer available in printed form.

D-93-21
Dennis Drollinger
Intelligentes Backtracking in Inferenzsystemen am Beispiel Terminologischer Logiken
53 Seiten

D-93-20
Bernhard Herbig
Eine homogene Implementierungsebene für einen hybriden Wissensrepräsentationsformalismus
97 Seiten

D-93-16
Bernd Bachmann, Ansgar Bernardi, Christoph Klauck, Gabriele Schmidt
Design & KI
74 Seiten

D-93-15
Robert Laux
Untersuchung maschineller Lernverfahren und heuristischer Methoden im Hinblick auf deren Kombination zur Unterstützung eines Chart-Parsers
86 Seiten

D-93-14
Manfred Meyer (Ed.)
Constraint Processing – Proceedings of the International Workshop at CSAM'93, St.Petersburg, July 20-21, 1993
264 pages

Note: This document is available for a nominal charge of 25 DM (or 15 US-$).

D-93-12
Harold Boley, Klaus Elsbernd, Michael Herfert, Michael Sintek,
Werner Stein
RELFUN Guide: Programming with Relations and Functions Made Easy
86 pages

D-93-11
Knut Hinkelmann, Armin Laux (Eds.)
DFKI Workshop on Knowledge Representation Techniques — Proceedings
88 pages

Note: This document is no longer available in printed form.

D-93-10
Elizabeth Hinkelman, Markus Vonerden, Christoph Jung
Natural Language Software Registry (Second Edition)
174 pages

D-93-09
Hans-Ulrich Krieger, Ulrich Schäfer
TDLExtraLight User's Guide
35 pages

DFKI Documents

1995

D-95-09
Antonio Krüger
PROXIMA: Ein System zur Generierung graphischer Abstraktionen
120 Seiten

D-95-07
Ottmar Lutzy
Morphic - Plus
Ein morphologisches Analyseprogramm für die deutsche Flexionsmorphologie und Komposita-Analyse
74 pages

D-95-06
Markus Steffens, Ansgar Bernardi
Integriertes Produktmodell für Behälter aus Faserverbundwerkstoffen
48 Seiten

D-95-05
Georg Schneider
Eine Werkbank zur Erzeugung von 3D-Illustrationen
157 Seiten

D-95-03
Christoph Endres, Lars Klein, Markus Meyer
Implementierung und Erweiterung der Sprache $\mathcal{ALCP}$
110 Seiten

D-95-02
Andreas Butz
BETTY
Ein System zur Planung und Generierung informativer Animationssequenzen
95 Seiten

D-95-01
Susanne Biundo, Wolfgang Tank (Hrsg.)
Beiträge zum Workshop „Planen und Konfigurieren", Februar 1995
169 Seiten

Note: This document is available for a nominal charge of 25 DM (or 15 US-$).

1994

D-94-15
Stephan Oepen
German Nominal Syntax in HPSG
— On Syntactic Categories and Syntagmatic Relations —
80 pages

D-94-14
Hans-Ulrich Krieger, Ulrich Schäfer
TDL - A Type Description Language for HPSG, Part 2: User Guide.
72 pages

D-94-12
Arthur Sehn, Serge Autexier (Hrsg.)
Proceedings des Studentenprogramms der 18. Deutschen Jahrestagung für Künstliche Intelligenz KI-94
69 Seiten

D-94-11
F. Baader, M. Buchheit, M. A. Jeusfeld, W. Nutt (Eds.)
Working Notes of the KI'94 Workshop: KRDB'94 - Reasoning about Structured Objects: Knowledge Representation Meets Databases
65 pages

Note: This document is no longer available in printed form.

D-94-10
F. Baader, M. Lenzerini, W. Nutt, P. F. Patel-Schneider (Eds.)
Working Notes of the 1994 International Workshop on Description Logics
118 pages

Note: This document is available for a nominal charge of 25 DM (or 15 US-$).

D-94-09
Technical Staff
DFKI Wissenschaftlich-Technischer Jahresbericht 1993
145 Seiten

D-94-08
Harald Feibel
IGLOO 1.0 - Eine grafikunterstützte Beweisentwicklungsumgebung
58 Seiten

D-94-07
Claudia Wenzel, Rainer Hoch
Eine Übersicht über Information Retrieval (IR) und NLP-Verfahren zur Klassifikation von Texten
25 Seiten

D-94-06
Ulrich Buhrmann
Erstellung einer deklarativen Wissensbasis über recyclingrelevante Materialien
117 Seiten

D-94-04
Franz Schmalhofer, Ludger van Elst
Entwicklung von Expertensystemen: Prototypen, Tiefenmodellierung und kooperative Wissensevolution
22 Seiten

RR-93-05
Franz Baader, Klaus Schulz
Combination Techniques and Decision Problems for Disunification
29 pages

RR-93-04
Christoph Klauck, Johannes Schwagereit
GGD: Graph Grammar Developer for features in CAD/CAM
13 pages

RR-93-03
Franz Baader, Bernhard Hollunder, Bernhard Nebel, Hans-Jürgen Profitlich, Enrico Franconi
An Empirical Analysis of Optimization Techniques for Terminological Representation Systems
28 pages

RR-93-02
Wolfgang Wahlster, Elisabeth André, Wolfgang Finkler, Hans-Jürgen Profitlich, Thomas Rist
Plan-based Integration of Natural Language and Graphics Generation
50 pages

RR-93-01
Bernhard Hollunder
An Alternative Proof Method for Possibilistic Logic and its Application to Terminological Logics
25 pages

DFKI Technical Memos

1995

TM-95-02
Michael Sintek
FLIP: Functional-plus-Logic Programming on an Integrated Platform
106 pages

TM-95-01
Martin Buchheit, Rüdiger Klein, Werner Nutt
Constructive Problem Solving: A Model Construction Approach towards Configuration
34 pages

1994

TM-94-04
Cornelia Fischer
PAntUDE – An Anti-Unification Algorithm for Expressing Refined Generalizations
22 pages

TM-94-03
Victoria Hall
Uncertainty-Valued Horn Clauses
31 pages

TM-94-02
Rainer Bleisinger, Berthold Kröll
Representation of Non-Convex Time Intervals and Propagation of Non-Convex Relations
11 pages

TM-94-01
Rainer Bleisinger, Klaus-Peter Gores
Text Skimming as a Part in Paper Document Understanding
14 pages

1993

TM-93-05
Michael Sintek
Indexing PROLOG Procedures into DAGs by Heuristic Classification
64 pages

TM-93-04
Hans-Günther Hein
Propagation Techniques in WAM-based Architectures — The FIDO-III Approach
105 pages

TM-93-03
Harold Boley, Ulrich Buhrmann, Christof Kremer
Konzeption einer deklarativen Wissensbasis über recyclingrelevante Materialien
11 pages

TM-93-02
Pierre Sablayrolles, Achim Schupeta
Conflict Resolving Negotiation for COoperative Schedule Management Agents (COSMA)
21 pages

TM-93-01
Otto Kühn, Andreas Birk
Reconstructive Integrated Explanation of Lathe Production Plans
20 pages

RR-93-28
Hans-Ulrich Krieger, John Nerbonne, Hannes Pirker
Feature-Based Allomorphy
8 pages

RR-93-27
Hans-Ulrich Krieger
Derivation Without Lexical Rules
33 pages

RR-93-26
Jörg P. Müller, Markus Pischel
The Agent Architecture InteRRaP: Concept and Application
99 pages

RR-93-25
Klaus Fischer, Norbert Kuhn
A DAI Approach to Modeling the Transportation Domain
93 pages

RR-93-24
Rainer Hoch, Andreas Dengel
Document Highlighting — Message Classification in Printed Business Letters
17 pages

RR-93-23
Andreas Dengel, Ottmar Lutzy
Comparative Study of Connectionist Simulators
20 pages

RR-93-22
Manfred Meyer, Jörg Müller
Weak Looking-Ahead and its Application in Computer-Aided Process Planning
17 pages

RR-93-20
Franz Baader, Bernhard Hollunder
Embedding Defaults into Terminological Knowledge Representation Formalisms
34 pages

RR-93-18
Klaus Schild
Terminological Cycles and the Propositional μ-Calculus
32 pages

RR-93-17
Rolf Backofen
Regular Path Expressions in Feature Logic
37 pages

RR-93-16
Gert Smolka, Martin Henz, Jörg Würtz
Object-Oriented Concurrent Constraint Programming in Oz
17 pages

RR-93-15
Frank Berger, Thomas Fehrle, Kristof Klöckner, Volker Schölles,
Markus A. Thies, Wolfgang Wahlster
PLUS - Plan-based User Support Final Project Report
33 pages

RR-93-14
Joachim Niehren, Andreas Podelski, Ralf Treinen
Equational and Membership Constraints for Infinite Trees
33 pages

RR-93-13
Franz Baader, Karl Schlechta
A Semantics for Open Normal Defaults via a Modified Preferential Approach
25 pages

RR-93-12
Pierre Sablayrolles
A Two-Level Semantics for French Expressions of Motion
51 pages

RR-93-11
Bernhard Nebel, Hans-Jürgen Bürckert
Reasoning about Temporal Relations: A Maximal Tractable Subclass of Allen's Interval Algebra
28 pages

RR-93-10
Martin Buchheit, Francesco M. Donini, Andrea Schaerf
Decidable Reasoning in Terminological Knowledge Representation Systems
35 pages

RR-93-09
Philipp Hanschke, Jörg Würtz
Satisfiability of the Smallest Binary Program
8 pages

RR-93-08
Harold Boley, Philipp Hanschke, Knut Hinkelmann, Manfred Meyer
CoLab: A Hybrid Knowledge Representation and Compilation Laboratory
64 pages

RR-93-07
Hans-Jürgen Bürckert, Bernhard Hollunder, Armin Laux
Concept Logics with Function Symbols
36 pages

RR-93-06
Hans-Jürgen Bürckert, Bernhard Hollunder, Armin Laux
On Skolemization in Constrained Logics
40 pages

Working Notes of the KI'95 Workshop: KRDB-95 Reasoning about Structured Objects: Knowledge Representation Meets Databases

D-95-12
Document

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