Exploitation of Interschema Knowledge in a Multidatabase System

John Cardiff
Department of Computing
RTC Tallaght
Dublin 24
IRELAND
john.cardiff@rtc-tallaght.ie

Tiziana Catarci, Giuseppe Santucci
Dipartimento di Informatica e Sistemistica
Università degli Studi di Roma "La Sapienza"
Via Salaria 113
I-00198 Roma, ITALY
[catarci/santucci]@infokit.ing.uniroma1.it

Abstract

We describe a logic language to formally express interdependencies between classes belonging to different schemas, the so-called interschema knowledge. These interdependencies allow a designer of a multidatabase (MDB) to establish relationships between both the intensional definition and the set of instances of classes represented in different schemas. These assertions form the „backbone" of the MDB and we may benefit in several ways from the possibility of reasoning about them. In this paper, we present two applications of interschema knowledge, namely schema integration and global query processing and optimization.

1. Introduction

Over the past few years, our research efforts have been inspired by two different needs. On one side the number of non-expert users accessing databases is largely growing. On the other side, information systems tend to be no longer composed by a single centralised architecture, but rather by several heterogeneous component systems. In order to address such needs we have designed a new query system, having both user-oriented and multidatabase features. The system main components are an adaptive visual interface, providing the user with different and interchangeable interaction modalities, and a „translation layer", which creates and offers to the user the illusion of a single homogeneous schema out of several heterogeneous components. Both components are founded on a common ground, i.e. a formally defined and semantically rich data model, the Graph Model [Catarci Santucci Angelaccio 1993], and a minimal set of Graphical Primitives (GPs), in terms of which general query operations may be visually expressed. The Graph Model has a visual syntax, so that graphical operations can be applied on its components without unnecessary mappings, and an object-based semantics.

The generality of our approach allows the representation of a wide class of models, such as relational, semantic, and object-oriented models, in terms of the Graph Model constructs. Moreover, the GPs defined on the Graph Model can be effectively used for characterizing the semantics of the most popular visual query languages defined on both semantic and object-oriented data models. In principle, the Graph Model can be used as the internal model of any visual query system, providing a sound formalism useful to characterize the semantics of the visual operations defined on the external model. We have shown ([Catarci Santucci Cardiff 1997]) how to map databases expressed in several data models, namely relational, semantic, object-oriented, into Graph Model Databases (GMDBs). Furthermore, the translation algorithm between relational databases and GMDBs can be used for showing the relational completeness of the GPs.

This paper outlines some of our research efforts in the multidatabase environment that is „behind" the Graph Model. Specifically we describe the following aspects of our work which we believe are of most interest to the workshop audience:

- a logic language necessary for expressing assertions that hold between intensions and extensions of the component schemas,
- the application of assertions in this language for performing certain key multidatabase

Research supported by the EEC under Esprit Project 6398, „VENUS", and by the Irish Government under Forbairt Project ST/94/720 „LIOM".

Proceedings of the 4th KRDB Workshop
Athens, Greece, 30-August-1997
(F. Baader, M.A. Jeusfeld, W. Nutt, eds.)
http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-8/

functionalities, viz. optimization of multidatabase queries, and integration of component schemas.

2. Representation of Interschema Knowledge

The notion of interschema knowledge is crucial for the development of cooperating heterogeneous information systems. Recent work on interoperability points out that two individual information systems can interoperate on the basis of a mutual understanding of the information resources they provide. Obviously, in order to achieve this mutual understanding, several forms of interschema knowledge must be expressed and reasoned upon.

We have developed a logic language to express interdependencies between classes belonging to different schemas. These interdependencies allow a designer of an MDB to establish several relationships between both the intensional definition and the set of instances of classes represented in different schemas. Once we have a set of assertions of the above-mentioned kinds, we may benefit in several ways from the possibility of reasoning about them. One distinguishing feature of our work is to provide inference mechanisms that allow such reasoning to be carried out. Based on the formal semantics of the language, we can indeed devise suitable reasoning procedures that allow one to draw useful inferences on interschema knowledge. For example, one can check whether one class represented in the MDB is incoherent, i.e. it has an empty extension in every state, or can deduce that the extension of a class A in the schema S_i is always a subset of the extension of the class B in S_j, so that accessing S_i is useless if we want to retrieve all the instances (stored in any of the information systems) of the concept represented by B. One distinguishing feature of our work is to provide inference mechanisms that allow such reasoning to be carried out. Such inference mechanisms can play an important role in both checking for interschema consistency, and in providing integrated access to the MDB.

We assume that the individual information systems which are the components of the heterogeneous information system are defined in terms of the same data model, i.e. the Graph Model, which is a class-oriented model. We summarise below our efforts in developing the formal translations between virtually all data models (and their query languages) and the Graph Model, so this assumption is not a limit to the generality of our approach.

2.1 The adopted formalism

In the formalism we adopt, the basic structure of a schema is expressed as a so called Typed Graph. Moreover, a logic-based language can be used for specifying properties and constraints of the database components, by means of class and role expressions. Central to this language is the notion of class and role, which corresponds to the concept of class-node and role-node in the Typed Graph. Analogously, a role can be either functional or non-functional. Also, there is a distinction between classes and domains equivalent to the division between unprintable class-nodes and printable class-nodes in the Typed Graph. The main idea is that all the knowledge about the basic elements in the Typed Graph can be specified in terms of a set of assertions. Syntactically, an assertion is a statement of the form L_1 *isa* L_2, where L_1 and L_2 are expressions of the language. Informally, an assertion of the above form states that every instance of the class (denoted by the expression) L_1 is also an instance of the class L_2 (for a formal treatment of this subject see [Catarci Lenzerini 1993]).

Suppose we are given an alphabet *B* of symbols for a GMDB *D*, including:

- the node labels L_1, the set of elementary values D;
- two special class symbols $\perp$ and T;
- the special symbols $\cap, \cup, \exists, \forall, (,), \{, \}$.

We use the term *class expression over B* to denote any expression that is formed using the symbols in B according to some syntactic rules (see [Catarci Lenzerini 1993]). For instance, the simplest class expression is the one formed by the name of a class-node. Intuitively, the expression R:C denotes the restriction of the relationship represented by the role-node R to those pairs whose second component is an instance of the concept C. Moreover, the symbol • denotes composition of relationships, the symbol $^{-1}$ is used to denote the inverse of a relationship, and symbol * is used to denote the transitive closure of a relationship.

With regard to domains, i.e., the unprintable class-nodes, the expression $\{e_1,...,e_n\}$ denotes the concept whose interpretation is the set of values corresponding to $e_1,...,e_n$.

With regard to concept expressions, the meaning of the constructor $\neg$, $\cap$, $\cup$, is simply that of set complement, set intersection and set union, respectively. For instance, $C \cap F$ (where C and F are class-nodes) represents the set of instances of both C and F. The constructors $\exists$ and $\forall$ are used to describe concepts on the basis of their linking to relationships: intuitively, $\exists$R.C denotes the set of objects that are linked by the role-node R to at least one instance of the class-node C, whereas $\forall$R.C denotes the set of objects which are linked by R to all the objects that are instances of C. The expression $\exists$R denotes the set of objects that are linked by the role-node R to at least one object.

In order to characterize the instances of the database, the Typed Graph has an associated Interpretation. We start by considering the set of constraints to be empty. In this case, the interpretation for a Typed Graph g is a function mapping the printable class-nodes of g to a subset of the set of elementary printable values, the unprintable class-nodes of g to a subset of the set of elementary unprintable values, and the role-nodes to a subset of a set of structured objects, defined as the smallest set containing the set of elementary values and all the possible labeled tuples (of any arity). In particular, given a role-node n, its Interpretation is constitued by a set of tuples whose arity is equal to the number of class-nodes adjacent to n, and each component is labeled with the label of one adjacent class-node and takes its values in the corresponding Interpretation. When a set of assertions c is defined over the nodes of the Typed Graph g, the definition of interpretation is extended in order to satisfy such assertions. A formal semantics is defined for the class expressions, as well as a set of semantic equations which have to be satisfied in order for a certain m to be an interpretation for the couple <g, c> (see [Catarci Lenzerini 1991]). An interpretation m is called a model of a set of assertions if every assertion is satisfied by m. Recalling the correspondance between interpretations and database states, it is easy to see that the models correspond to those database states which are legal with respect to a set of integrity constraints (in our case, the set of assertions).

2.2 Assertions

The kind of assertions expressible using the logic language is a superset of those directly expressing using traditional semantic models, and we will see that this greater richness can be used once we want to represent and use interschema knowledge. It is also worth noting that while the user who wants simply to query the database does not have to be overloaded by too detailed information, the designer who builds the heterogeneous system needs to know much more about the component systems s/he has to integrate.

In fact, we can not only express the usual isa relationships between entities (e.g., Author *isa* Person), but also isa relationships on relationships. Moreover, we can use explicit negation, so stating that the set of instances of two entities are disjoint. For example, we can represent that the classes Author and Referee are disjoint by means of the assertion: Author *isa* ¬ Referee. Another example of negative information is the one stating that a certain relationship is meaningless for an entity, i.e. that the instances of the entity cannot participate in the relationship.

Several kinds of indefinite information can also be expressed by means of assertions. One of the most important kind of indefinite assertions is related to the use of disjunction. For example, one can assert that persons are either males or females by writing: Person *isa* (male ∪ female).

Finally, it is possible to combine necessary and sufficient conditions, so providing a definition of a concept in terms of other concepts. For instance, we can associate a definition to the class-node ItalianPaper as follows:

ItalianPaper def_{int} Paper
(∀WrittenBy.(Author(∀Nationality.Italian)

which means: an Italian paper is a paper whose authors are all Italians.

2.3 Interschema Knowledge

In this section we describe our approach for specifying interschema knowledge in terms of interdependencies between classes belonging to different schemas.

Suppose that the heterogeneous information system is constituted by n individual information systems, called component information systems, whose schemas are $S_1,...,S_n$ with alphabets $B_1,...,B_n$. We assume that all the schemas are expressed using the Graph Model and that some extra assertions are available for the designer. Let S_0 be a further schema (having its own alphabet B_0 and set of assertions C_0), called the common knowledge schema of the MDB. Intuitively, S_0 represents the general properties of the classes that should be considered common knowledge in the heterogeneous information system. Obviously, in those applications where such knowledge is not available, the set of assertions in C_0 will be empty.

Interschema knowledge is then specified in terms of *interschema assertions*. There are four kinds of such assertions, whose forms are:

L_1 def_{int} L_2
L_1 isa_{int} L_2
L_1 def_{ext} L_2
L_1 isa_{ext} L_2

where in every assertion, L_1 represents an S_i-class expression, and L_2 represents an S_j-class expression, in such a way that L_1 and L_2 are class expressions of the same types (either entity, domain, relationship, or attribute expressions), and $i \neq j$. Moreover, if L_1 and L_2 are relationship expressions, then there is the constraint that the set of components appearing in L_1 has the same cardinality as the set of components appearing in L_2. In other words, an interschema assertion represents a well-typed interdependency between two class expressions belonging to two different schemas.

We now discuss the intuitive meaning of the four kinds of interschema assertions that we have introduced by using some examples, and refer to [Catarci Lenzerini 1993] for their formal semantics.

The first assertion states that the expression L_1 is intensionally equivalent to L_2. Intuitively, this means that, if S_i and S_j referred to a unique set of objects in the real world, then the extension of L_1 would be the same as the extension of L_2. Therefore, the above assertion is intended to state that, although the extension of L_1 may be different from the extension of L_2, the concept represented by L_1 is in fact the same as the concept represented by L_2. As a simple example, the S_1-entity *UndergraduateStudent* can be declared intensionally equivalent to the S_2-entity *Student* $\cap$ $\neg$ *GraduateStudent*, to reflect that, even if the instances of the two expressions may be different in the various states of the MDB, the concept of *UndergraduateStudent* in the schema S_1 is fully captured by the above entity expression in the schema S_2.

The second assertion states that the entity expression L_1 of schema S_1 is intensionally less general than the entity expression L_2 (of schema S_2). This means that there is a sort of containment between L_1 and L_2, and this containment is conceptual, not necessarily being reflected at the instance level. In other words, the above intensional relationship is intended to state that, if S_1 and S_2 referred to a unique set of objects in the real world, then the extension of L_1 would be a subset of L_2. For example, *Tutor*$_2$ may be declared intensionally less general than *Teacher*$_3$, if the concept of tutor as represented in the schema S_2 is subsumed by the concept of teacher in the schema S_3.

The third assertion states that L_1 and L_2 are always extensionally equivalent. In this case we are asserting that in every state of the MDB, the set of objects that are instances of L_1 in S_1 is the same as the set of objects that are instances of L_2 in S_2.

Finally, the fourth assertion states that the extension of L_1 is always a subset of the extension of L_2. For example, if the entity Student_1 refers to the students of a University Department, and the schema S_2 refers to the whole University, then we may assert that *Student*$_1$ is extensionally less general than *Student*$_2$.

3. Applications of Interschema Knowledge

3.1 Global Query Optimization

As the user interacts with a conceptually single database, s/he is unaware of the existence of the underlying databases, and needs not be concerned with their specific storage formats or query languages. As a consequence, the onus is on the system to produce an efficient decomposition of the user's query to a number of subqueries which are to be executed on the component databases. The issue of query processing and optimization in an MDB has been considered in several research projects (see for example, [Dayal Landers Yedwab 1982], [Du Krishnamurthy Shan 1992], [Lu Ooi Goh 1992], [Florescu Raschid Valduriez 1996]). The central contribution of this paper is the development of a methodology to enhance the efficiency of MDB query decomposition by the exploitation of inter- and intra-schema knowledge. To the best of our knowledge, this has not been addressed elsewhere in the literature, which is somewhat surprising, since information describing the interrelationships of the schemas must be represented in some manner to realize an MDB.

We address this issue by augmenting the „traditional" approach to multidatabase query processing with additional methodologies to exploit the availability of semantic information that could simplify the expression of a query. The totality of schema knowledge on the multidatabase Σ is a set of schema assertions which we partition into a number of sets:

- C_G which contains assertions relating exclusively to the global schema,
- C_{IS} which contains assertions describing relationships between the component schemas [Sheth Larson 1990], and
- A set of schema assertions $\{C_{L1} \ldots C_{Ln}\}$, where C_{Li} contain assertions relating exclusively to the schema S_i, $1 \leq i \leq n$.

C_G and any of the local assertion sets C_{Li} may be empty.

Interschema assertions are exploited in global query optimization in a number of ways. We can use extensional equivalence and assertions to determine alternative data sources, ie. component databases containing replicated data, either one of which can provide the data required to answer the query. Exclusion assertions determine when no instances used in computing the result are stored by a component database, and extensional subset assertions determine if the set of instances in one database used in computing the result are a subset of those stored by another database, which may remove the need to access the latter. We have defined a comprehensive set of such transformations which can reduce the overall cost of the global query [Cardiff Catarci Santucci 1997].

We use two structures to represent the global query during the decomposition and transformation process. The first, which we call a Conceptual Global Query Tree (CGQT) is a generic representation of the decomposed global query. There is a single CGQT associated with each global query. In effect, this information describes the operations to be effected to compute the result, without explicitly stating how they will be implemented. There can be many alternative strategies for implementing a specific CGQT – an integration operation can be split over several queries and each of these can be executed at any site. We use a structure called an Implementation Global Query Tree (IGQT) to represent specific execution strategies.

The transformations summarized above vary considerably in their effect on the overall cost of the CGQTs and IGQTs, and so they cannot be applied indiscriminately to each global query tree under consideration, since this may impact the overall execution time. Instead we adopt the following tactics for application of transformations.

The global query is first considered for semantic transformations using only the global assertions, C_G. The assertions in C_G describe properties relating exclusively to the global schema $GMDB_I$, and are useful in identifying redundancy or inconsistencies in a query specification, due to user ignorance of the domain, for instance. The output of this phase is a semantically „relevant“ query, ie. one without redundancy and inconsistency. In effect, this analysis is equivalent to semantic query optimization on a centralized database.

The query is then decomposed from its specification on $GMDB_I$ to component queries on the underlying Export Schemas $GMDB_{1...n}$. Query decomposition has been well explored in the literature (see for example [Meng Yu 1995]) and is not considered in our research. The output of this phase is a Conceptual Global Query Tree (CGQT), a representation of an execution strategy for the global query on the component Graph Model databases.

There are several categories of transformations that can be applied immediately to the canonical CGQT, however we restrict application initially to the identification of redundant and alternative execution sites. This phase performs a series of initial semantic transformations on the CGQT, identifying:

- those databases which contain no data relevant to the result,
- those which need not be queried, due to the query semantics, and
- those which contain data replicated over two or more sites, any of which can be used in computing the result.

The output is a semantically transformed CGQT.

The next phase uses the CGQT to derive a number of potential execution strategies, represented as IGQTs. Since there is a large number of IGQTs that can be derived from a given CGQT, heuristics are used to narrow the search space of candidate IGQTs chosen.

Further semantic transformations are then applied to the candidate IGQTs. The set of IGQTs still represents a very large search space of potential execution strategies, and so transformations are applied selectively using heuristics which are non-exhaustive, but are designed to identify a reasonably efficient execution plan. Parallel execution is exploited where possible, the overall response time is bound by the longest sequence of subqueries (ie. that require the output of one subquery as input to the next). We apply transformations exclusively to the longest sequence, and successful transformations will shorten this sequence.

The methodology is modular, in the sense that the transformations applied at the external, inter-schema, and intra-schema levels are independent of each other and the unavailability of semantic information at any level will not affect the application of the methodology to the others. In all cases, however, our goal is to find a "near optimal" query that can be derived with minimal overhead. While the utilization of semantics in query processing has been successfully applied in centralized databases, its potential for distributed and multidatabase systems is enormous, as there is the potential to eliminate or to simplify inter-site integration operations, which are the single biggest cost factor in multidatabase query processing.

3.2. Schema Integration

Another important aspect related to the consistency of interschema knowledge is concerned with schema integration [Catarci Santucci Cardiff 1995]. If we want to integrate two or more schemas belonging to the MDB, we can benefit from the knowledge expressed in our language, in all the phases of the integration process. From one point of view, when performing the integration of two or more schemas it is possible to exploit the interschema knowledge in all the activities involved in such a process. On the other hand, if an integrated schema has been already built, the choices performed during the integration can be easily translated into interschema assertions, so speeding up the production of the interschema knowledge. Here, we summarise our work under the activities that are shared by most proposals ([Batini Lenzerini Navathe 86]), viz. Schema comparison (checking all conflicts in the representation of the same objects in different schemas), Schema conforming (aligning schemas to make them compatible for integration) and Schema restructuring (analyzing the integrated schema against three goals: completeness, minimality, and understandability).

Schema comparison

The fundamental activity in this step consists of checking all conflicts in the representation of the same objects in different schemas. Schema integration methodologies broadly distinguish two types of conflicts:

- **Naming conflicts** -Schemas in data models incorporate names for entities, attributes, and relationships. People from different application areas of the same organization are used to refer to the same data using their own different terminology and names. This results in a proliferation of names as well as a possible inconsistency among names in the component schemas. The problematic relationships among names are of two types: *homonyms*: when the same name describes two different concepts - giving rise to inconsistency unless detected, and *synonyms*: when the same concept is described by two or more names - giving rise to a proliferation of names
- **Structural conflicts -** The term structural conflicts includes conflicts that arise because of a different choice of modeling constructs or integrity constraints. Examples of conflicts mentioned in different methodologies are: *type inconsistencies*: the same concept is represented by different modeling constructs in different schemas. (E.g., the use of city as an entity in one schema and as an attribute in another one); *cardinality ratio conflicts*: a group of concepts are related among themselves with different cardinality ratios in different schemas. (E.g., Man and Woman in the relationship 'Marriage' are 1:1 in one schema, but m:n in another one, accounting for a marriage history); and *key conflicts*: different keys are assigned to the same concept in different schemas.

Schema Conforming

The goal of this activity is to conform or align schemas to make them compatible for integration. Achieving this goal amounts to resolving the conflicts, which in turn requires that schema transformations be performed. In order to resolve a conflict, the designer must investigate the reasons that caused the schemas to be diverse in order to understand the semantic relationship.

Schema Restructuring

This activity is performed by further analyzing the global schema against three goals: completeness, minimality, and understandability. This analysis usually gives rise to several transformations on the global schema. Let us examine these three goals individually:

- *Completeness* - To achieve completeness, we have to conclude the analysis and the addition of interschema properties that is usually initiated in previous design steps. Note that the variety of interschema properties is strongly related to the repertoire of schema constructs at the disposal of the data model;
- *Minimality* - The objective of minimality is to discover and eliminate redundancies
- *Understandability* - We have to analyze a schema in order to find (all) possible restructuring that can improve one's understanding of it. In general, for improved understandability, additional schema transformations are needed.

The rest of the Section is devoted to analyze each of the above activities showing how they can benefit by interschema assertions belonging to the interschema knowledge.

Concerning name conflicts, synonyms can be easily discovered by looking at the interschema assertions of the following form:

$L_1\ def_{int}\ L_2$
$L_1\ def_{ext}\ L_2$

in which L_1 and L_2 are class expressions formed by the name of a class-node. More precisely, if there exists an assertion of the form:

$L_1\ def_{ext}\ L_2$

we can conclude that L_1 and L_2 are deeply synonyms and that the integrated schema can be built using either L_1 or L_2.

On the other hand, if the assertion is of the form:

$L_1\ def_{int}\ L_2$

we can conclude that L_1 and L_2 are synonyms at the intensional level and that their instances may differ; as a consequence in the integrated schema both the instances of L_1 and L_2 must be taken into account.

Homonyms can be discovered by the <u>absence</u> of an assertion involving the two classes sharing the same name. In particular, assume that the two schemas S_1 and S_2 share the same class, say C (we denote with C_1 and C_2 the different occurrences of the class). If the interschema knowledge does not contain an assertion of the form

$C_1\ def_{int}\ C_2$

or the stronger one:

$C_1\ def_{ext}\ C_2$

we can conclude that we are in presence of a homonym and that in the integrated schema one class (or both) must be renamed.

Among the structural conflicts mentioned before, only the ones concerning type inconsistencies can be detected through the inspection of the interschema knowledge: in particular, we have to look for assertions of the form:

L_1 *def*$_{int}$ L_2
L_1 *def*$_{ext}$ L_2

in which L_1 and L_2 are attribute expressions referring to different classes. As an example, if we know that the attribute expression corresponding to the attribute *Cname* of the class *Person* is intentionally equivalent to attribute expression corresponding to the attribute *Name* of the class *City* we can argue that the same concept (City) has been modeled as an attribute in the former case and a class in the latter.

The schema conforming activity results in so-called *amended schemas*. The correspondences existing between an original schema and its amended version can be documented in terms of interschema knowledge assertions. As an example if a concept C belonging to the original schema is renamed with the new name C' in the amended version we can say that:

C *def*$_{int}$ C'

It is clear that the activity of schema conforming is just an input for the interschema knowledge.

The activity of schema restructuring can benefit from assertions of the form

L_1 *isa*$_{int}$ L_2
L_1 *isa*$_{ext}$ L_2

in which L_1 and L_2 are class expressions formed by the name of a class-node, or assertion of the form

L_1 *def* $C_1 \cup C_2 \cup ... \cup C_k$

in which L_1, C_1, C_2, ..., C_k are class expressions formed by the name of a class-node.

In all of the above cases it is possible to add to the integrated schema an isa relationship between L_1 and L_2 (first two assertions) or between L_1 and C_1, L_1 and C_2, ..., L_1 and C_k (last assertion).

If a cycle of unprintable nodes and role nodes exists, of the form u_1, r_1, u_2, r_2, ..., u_k, r_k, u_1 we can check for redundancies, looking for an assertion of the following form:

r_i *def*$_{ext}$ R_2

where r_i is a role node expression formed by the name of a role node belonging to the cycle, say r_i and R_2 is a role node expression formed by the join of all role nodes belonging to the cycle but r_i. In this case we can argue that r_i is redundant and we can remove it from the integrated schema.

4. Summary

The main goal of our approach is to allow different classes of users to access multiple, heterogeneous databases by means of an adaptive interface, offering several interaction mechanisms. This led us to design and partially implement a query system, having both user-oriented and multidatabase features. In particular, our proposal exploits a Constraint Language based on logic, used to represent and reason upon inter- and intra-schema assertions, which are proficuously used in global query optimization and schema integration.

It is worth noting that we can use the interschema knowledge in several ways. First, besides providing information on the correspondence between classes in different schemas, interschema assertions actually constitute a declarative specification of several consistency requirements over different databases. Obviously, if the interschema knowledge is itself incoherent, then no state of the MDB may exist satisfying all the interschema assertions. Therefore checking coherence is one of the basic activities for verifying the correctness of the MDB. In our approach, coherence verification corresponds to checking the multidatabase for satisfiability.

Summarising, this research has provided a strong and formal basis for the development of an adaptive user interface to heterogeneous databases. The work is still in progress. In particular, we are devising a more sophisticated schema integration technique. Starting from a set of Graph Model databases and a knowledge base containing intra- and inter-schema knowledge we will be able to automatize several phases of the schema integration. As for the multiparadigmatic interface, which has been implemented, we are presently testing it against real users.

References

[Batini Lenzerini Navathe 1986] C. Batini, M. Lenzerini, S. Navathe, A Comparative Analysis of Methodologies for database Schema Integration, ACM Computing Surveys, September 1986, pp. 323-364.

[Cardiff Catarci Santucci 1995] J. Cardiff, T. Catarci, G. Santucci, Distributed Semantic Query Processing in a Cooperative Information System, Proc. International Conference on Cooperative Information Systems, 1995 (COOPIS'95)

[Catarci Santucci Cardiff 1997] T. Catarci, G. Santucci and J. Cardiff, Graphical Interaction with Heterogeneous Databases, VLDB Journal, 6(2), 1997

[Cardiff Catarci Santucci 1997] J. Cardiff, T. Catarci, G. Santucci, Semantic Query Processing in the VENUS Environment, International Journal of Cooperative Information Systems, June 1997

[Catarci Santucci Cardiff 1995] T. Catarci, G. Santucci, J. Cardiff, Knowledge based Schema Integration in a Heterogeneous Environment, International Conference on Next Generation Database Systems, Israel, 1995

[Catarci Santucci Angelaccio 1993] T. Catarci, G. Santucci, M. Angelaccio, Fundamental Graphical Primitives for Visual Query Languages, Information Systems, Vol. 18, N.2.

[Catarci Lenzerini 1993] T. Catarci, M. Lenzerini, Representing and Using Interschema Knowledge in Cooperative Information Systems, Journal of Intelligent and Cooperative Information Systems, Vol. 2, N. 4, pp.375-398, World Scientific, 1993.

[Dayal Landers Yedwab 1982] U. Dayal, T. Landers, L. Yedwab, Global Query Optimization in Multibase, a system for Heterogeneous Distributed Databases, Technical Report TR-82-05, Computer Corporation of America

[Du Krishnamurthy Shan 1992] W. Du, R. Krishnamurthy, M.C. Shan, Query Optimization in a Heterogeneous DBMS, in Proc. of the 18th International Conference on Very Large Data Bases, Vancouver.

[Florescu Raschid Valduriez 1996] D. Florescu, L. Raschid, P. Valduriez, A Methodology for Query Reformulation in CIS using Semantic Knowledge, Int. Journal on Intelligent and Cooperative Information Systems, special issue on Formal Methods in Cooperative Information Systems: Heterogeneous Databases, 5(4), December

[Lu Ooi Goh 1992] H. Lu, B-C. Ooi, C-H. Goh, On Global Multidatabase Query Optimization, SIGMOD Record 21(4)

[Meng Yu, 1995] W. Meng, C. Yu, Query Processing in Multidatabase Systems, in Modern Database Systems, Kim ed., Addison-Wesley

[Sheth Larson 1990] A. Sheth and J. Larson, Federated Database Systems for Managing Distributed Heterogeneous, and Autonomous Databases, ACM Computing Surveys, Vol. 22, No. 3.