Extraction of informations from highly heterogeneous source of textual data*

Sonia Bergamaschi

Dipartimento di Scienze dell'Ingegneria
Universitá di Modena
CSITE-CNR Bologna
ITALY
sonia@dsi.unimo.it,sbergamaschi@deis.unibo.it

Abstract.
Extracting informations from multiple sources, highly heterogeneous, of textual data and integrating them in order to provide *true information* is a challenging research topic in the database area. In order to illustrate problems and solutions, one of the most interesting projects facing this problem, *TSIMMIS*, is presented. Furthermore, a *Description Logics* approach, able to provide interesting solutions both for data integration and data querying, is introduced.

1 Introduction

The availiability of large numbers of network informations sources (and the recent explosion of Internet) makes it possible to access to a very large amount of information sources all over the world. The increased amount of available informations has as a consequence the fact that, for a given query, the set of potentially interesting sites is very high but only very few sites are really relevant. Furthermore, informations are highly heterogeneous both in their structure and in their origin. In particular, not only data types are heterogeneus (textual data, images, sounds, etc.), but even the representation of a single data type can differ.

Even in the restricted domain of textual data, the problem of *organizing* data (often a huge amount) coming from multiple heterogeneous sources in *easily accessible structures*, in order to provide *true information*, is a challenging research topic for different research communities: database, artificial intelligence, information retrieval. Let us individuate two increasing complexity scenarios:

1. *known sources* – the sources of heterogeneous textual data are known;
2. *unknown sources* –the sources of *relevant* heterogeneous textual data must be individuated.

* This research has been partially funded by the MURST 40 % Italian Project: '*Basi di dati Evolute: Modelli, metodi e sistemi*'.

The first scenario is, at present, heavily investigated in the database area, involving many research topics and application areas: decision support systems (DSS), integration of heterogeneous databases, datawarehouse. Decision makers need informations from multiple heterogeneous sources (including databases, file systems, knowledge bases, digital libraries, information retrieval systems, and electronic mail systems), but are usually unable to get and fuse them in a timely fashion due to the difficulties of accessing the different systems and to consistently integrate them. Significant contributions about the integration of well-structured conventional databases exist (e.g. [9, 21, 22, 24, 18]). Many projects have adopted OO models to facilitate integration [9, 13, 5] and, recently, systems for the integration of sources with minimal structure have appeared [17, 23, 4]. Futhermore, the DARPA Intelligent Integration of Information (I^3) research program is devoted to this problem. However, as a consequence of the rapid development of prototype implementations in this area, the initial outcome of this program appears to have been to produce a new set of systems. While they can perform certain advanced information integration tasks, they cannot easily communicate with each other. With a view to understanding and solving this problem, recently a workshop was held on this topic at the University of Maryland in April, 1996 [12].

The second, most complex scenario, is associated to the so-called *information discovery* problem. This problem arised mainly due to the *Internet* explosion. In this scenario we have, first, to face the problem of individuating among a huge amount of sources of heterogeneous textual data a *possibly low amount* of *relevant* sources and, then, to face, if necessary, the problem of scenario 1. Research efforts devoted to face this problem come from different research areas: information retrieval, artificial intelligence, database. This scenario is out of the scope of this paper as the amount of approaches and systems very recently proposed is as large as to require a paper on its own.

In this paper I will discuss problems and solutions of the extraction of informations from multiple sources, highly heterogeneous, of textual data and of their integration in order to provide *true information*. First, one of the most interesting projects facing this problem, *TSIMMIS*, is presented. Then, a *Description Logics (DL)* approach, able to provide interesting solutions both for data modelling and data querying, is introduced.

The outline of the paper is the following. Section 2 presents an overview of the *TSIMMIS* project, including the *OEM* data model and *MSL* language adopted in the project. Section 3 describes the architecture of a *TSIMMIS wrapper*, i.e., an extractor of informations from a textual source which convert data into the *OEM* data model. Section 4 describes *TSIMMIS* approach for generating *mediators* (a mediator is an integration and refinement tool of data coming from wrappers). Section 5 introduces *Description Logics*. Section 6 sketchs a *Description Logics* approach to the problems of data modelling and data querying in heterogeneous multiple sources of textual data.

Remarks all over the paper fix relevant choices for the design of I^3 systems.

2 The TSIMMIS project: an overview

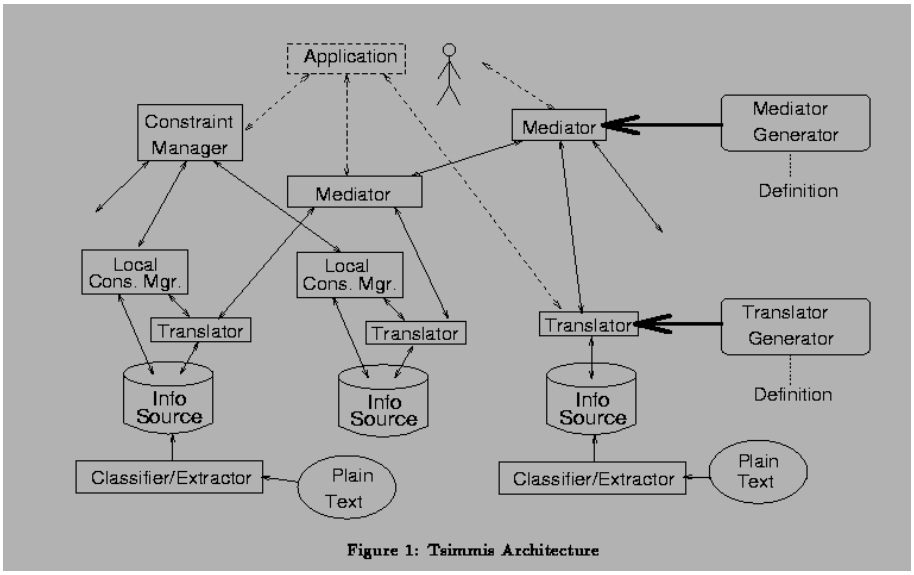

Fig. 1. Tsimmis Architecture

Let us introduce the problems and the solutions recently proposed within scenario 1, by describing one of the most interesting projects: the **TSIMMIS** (**T**he **S**tanford- **I**BM **M**anager of **M**ultiple **I**nformation **S**ources) Data Integration Project, under development at the Department of Computer Science - University of Stanford (biblio references: http://db.stanford.edu). TSIMMIS is a joint project between Stanford and the IBM Almaden Research Center whose goal is the development of tools that facilitate the rapid integration of heterogeneous textual sources that may include both structured and unstructured data [1, 2].

The TSIMMIS data-integration system provides integrated access via an architecture that is common in many other projects: *wrappers/translators* [13, 25, 5] convert data into a common model; *Mediators* combine, integrates or refines the data from the wrappers. The wrappers also provide a common query language for extracting informations. Applications can access data directly through wrappers but they can also go through *mediators* [5, 29, 4].

In Figure 1, the TSIMMIS architecture is shown: above each source is a *translator (wrapper)* that logically converts the underlying data objects to a common information model; above the translators lie the *mediators*. The translator converts queries over information in the common model into requests that the source can execute and data extracted from the source into the common model. The common model is the *OEM (Object Exchange Model)*, a *tagged* model allowing

simple nesting of objects. Furthermore, two query languages, *OEM-QL* and *MSL(Mediator Specification Language)*, for requesting *OEM* objects have been developed. *OEM-QL* is an SQL-like language, extended to deal with labels and object nesting and *MSL* is a high level language that allows the declarative specification of mediators. The possible bottlenecks of the above architecture are:

- an ad-hoc translator [2] must be developed for any information source;
- implementing a mediator can be complicated and time-consuming.

Thus, important goals of the project (and of any other project with the same aim) are:

1. **to provide translator generator** that can generate *OEM* translator based on a description of the conversion that need to take place for queries received and results returned (see *translator/generator* box in Figure 1);
2. **to automatically or semi-automatically generate mediators** from high level descriptions of the information processing they need to do (see *mediator/generator* box in Figure 1).

The solutions proposed in *TSIMMIS* for the above goals are described in Section 3 and 4.

2.1 The OEM model and the MSL language

Let us briefly introduce the *OEM* model [5]. It is a *self-describing model* [31] where each data item has an associated descriptive label and *without a strong typing system*. *OEM* is much simpler than conventional OO models: supports only *object nesting* and *object identity*, while other features, such as classes, methods and inheritance are not supported directly. An object description in *OEM* has the format:

```
<ob1: person, set,        {sub1,sub2,sub3,sub4,sub5}>
      <sub1: last_name,  str, 'Smith'>
      <sub2: first_name, str, 'John'>
      <sub3: role,       str, 'faculty'>
      <sub4: department, str, 'cs'>
      <sub5: telephone,  str, '32435465'>
```

Each *OEM* object has the following structure: an object-id, a label, a type, a value. A label is a variable-length string describing what the object represents. We have a top-level object (ob1) and five sub-objects(sub1 to sub5).

Remark 1 *A relevant feature of* OEM *is that objects sharing the same label do not follow a unique schema: for example an other object with the same label 'person' could have different sub-objects. This feature make the integration of data coming from heterogeneous sources with different schemas easier than in conventional OO data models.*

[2] translator and wrapper are synonimous in TSIMMIS.

Let us briefly introduce the *MSL* language [4] (a more detailed description is given in Section 4).

Remark 2 MSL *is a first-order logic language that allow the declarative specification of mediators; An* MSL *specification is a set of rules which define the mediator view of the data and a set of functions that are invoked to translate objects from one format to another. Each rule consists of a* head *and a* tail *separated by the symbol :-. The tail describes the pattern of the objects to be fetched from the source, while the head defines the pattern of the top-level integrated object supported by the mediator.*

3 The TSIMMIS wrapper generator

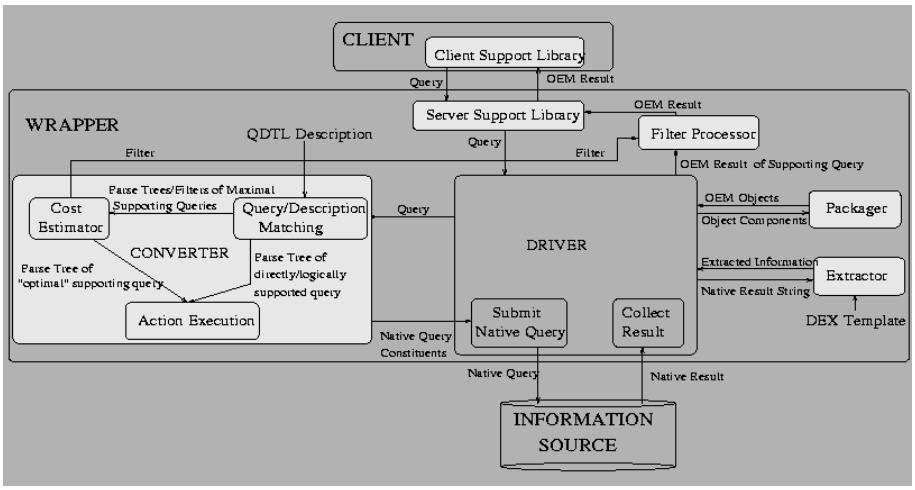

Fig. 2. TSIMMIS wrapper

TSIMMIS includes a toolkit, say *OEM Support Libraries*, to quickly implement wrappers, mediators and end-user interfaces. These libraries contain procedures that implement the exchange of *OEM* objects and queries between a server (either a translator or a mediator) and a client (either a mediator, an application or an interactive end-user) and procedures to translate queries into a suitable format.
The architecture of wrappers generated by using the toolkit is shown in Figure 2:

- the white rectangles are available in the toolkit: *CONVERTER*, *CLS (Client Support Library)*, *SSL (Server Support Library)*, *Filter Processor*, *Packager*, *Extractor*;

- The *CONVERTER* is the wrapper component which translates a query expressed in the *MSL* language into a sequence of operations executable by the information source. The translation is performed by using descriptions expressed in *QDTL (Query Description and Translation Language)*;
- *QDTL description* for the *CONVERTER* and *DEX template* for the *Extractor* must be specified;
- An architecture component, say the *DRIVER*, must be completely developed from scratch, for each wrapper, as it depends on the information source.

If we generate a wrapper exploiting the toolkit, it acts as a server in a client-server architecture (where the clients can be mediators or applications) in the following way:

- Clients use *CLS* to send queries and to receive *OEM* objects;
- by means of the *SSL*, the wrapper receives a query and send it to the *DRIVER* to be executed;
- the *DRIVER* invokes the *CONVERTER*, that, on the basis of the *QDTL description* , translates the query in a sequence of operations executable at the information source;
- the sequence of operations is sent to the *DRIVER* to be executed. The extracted data are passed by the *DRIVER* to the *Extractor*;
- by using the *DEX template*, the *Extractor* extracts the informations which will be translated into OEM objects by the *Packager*;
- if during the query conversion a filter has been generated, the *DRIVER* passes the filter and the *OEM* objects to the *Filter Processor*, then, by means of the *SSL*, the *OEM* objects are sent to the *CLIENT*.

3.1 CONVERTER and QDTL

To illustrate the *CONVERTER* functionalities and the *QDTL* syntax [3], let us refer to a university professors and students WHOIS information source. Let us suppose that this source allows only very simple retrieve operations, for example, the following:

1. retrieve persons with a given last_name: `>lookup -ln 'ss'`
2. retrieve persons with a given last_name and first_name: `>lookup -ln 'ss' -fn 'ff'`
3. retrieve all the records of the source : `>lookup`

The above operations are mapped into *QDTL* descriptions in order to make the *CONVERTER* able to decompose a *MSL* query into subqueries executable by the source. A *QDTL* description is composed by a set of templates with associated actions. The query templates for the three operations (no actions are specified for the moment) are:

[3] a full description of the *CONVERTER* and of *QDTL* is in [3]

```
D1:
    (QT1.1) Query ::= *O :- <O person {< last_name $LN>}>
    (QT1.2) Query ::= *O :- <O person {< last_name $LN>
                                       <first_name $FN>}>
    (QT1.3) Query ::= *O :- <O person V>
```

Each query template is described after the ::= symbol and is a parameterized query. Identifiers preceded by the $ symbol represent the corresponding constants of an input *MSL* query. The variables in capital letters (`V`) correspond to variables of an input *MSL* query.

The *CONVERTER* includes an algorithm able to exploit each template to describe much more queries than the ones that could be executed directly using the template. The class of supported queries is thus:

- *Directly supported queries*: queries with a syntax analogous to the template;
- *Logical supported queries*: A query q is logically supported by a template t if q is *logically equivalent* to a query q' directly supported by t (two queries are logically equivalent if they give the same answer set in the same context);
- *Indirectly supported queries*: a query q is indirectly supported by a template t if q can be decomposed in a query q' directly supported by t and a *filter* that is applied on the results of q'.

Let us consider the query:

```
(Q6) *Q :- <Q person {<last_name 'Smith'> <role 'student'>}>
```

`Q6` is not logically supported by any of the `D1` templates, but the *CONVERTER* is able to detect that `Q6` is subsumed by the query `Q7` which is directly supported:

```
(Q7) *Q :- <Q person {<last_name 'Smith'>}>
```

`Q7` contains all the informations necessary to determine `Q6` answer set; to obtain `Q6` answer set a filter, i.e. a new MSL query: *O :- <O person {<role 'student'>}> is generated which applied to `Q7` answer set gives `Q6` result.
More formally, with reference to indirectly supported queries, we define:

Query Subsumption: A query q is subsumed by a query q' if each object in the answer set of q is in the answer set of q'.

Query Equivalence: Two queries are logically equivalent if they give the same answer set in the same context.

Indirect Support: A query q is indirectly supported by a query q' if:
- q' subsumes q, and
- it exists a filter query that when applied to the answer set of q' gives the answer set of q.

Let us observe that, in general, for a given query q we can have more than one query able to support it. For example query `Q6`, besides `Q7` is supported by the

following Q11 query:

```
(Q11) O* :- <O person V>
```

which, in its turn, subsumes Q7.

Remark 3 .
The CONVERTER *should consider all the possible subsuming supporting queries in order to select the most efficient one. The lowest subsuming supporting query could be a good candidate.*

QDTL allows the use of non-terminal symbols to guarantee the support of more complex queries. Coming back to the example, we can have a combination of arguments (>`lookup -fn 'hh' -ln 'iop'`, ecc.). An exaustive list of all the possible cases is impossible, thus non-terminal symbols that allow a concise expression of a set of queries have been introduced. For example, let us consider the *QDTL* description D4:

```
D4:
    (QT4.1) Query     ::= *OP :- <OP person {_OptLN _OptFN _OptRole}>
    (NT4.2) _OptLN    ::= <Last_name $LN>
    (NT4.3) _OptLN    ::=
    (NT4.4) _OptFN    ::= <first_name $FN>
    (NT4.5) _OptFN    ::=
    (NT4.6) _OptRole  ::= <Role $R>
    (NT4.7) _OptRole  ::=
```

D4 describes all the queries that can be obtained by combining the three arguments. The non-terminal symbols are preceded by the character '_' ; each symbol has a description constituted by a set of non-terminal templates. For example, the symbol _OptRole is defined by NT4.6 and NT4.7 templates.

Direct Support: A query q is directly supported by a template t including non-terminal symbols if q is directy supported by one of the expansion of t.

An expansion is obtained by substituting each non-terminal symbol with its definition.

Actions in *QDTL* templates express the query in a format executable by the source. In the described Converter we use actions expressed in the C language. Let us refer to *D1* description to show some actions:

```
D2
   (QT2.1) Query ::= *O :- <O person {<last_name $LN>}>
   (AC2.1)           {printf (lookup_query, 'lookup -ln %s',$LN);}
   (QT2.2) Query ::= *O :- <O person {<last_name $LN>
                                      <first_name $FN>}>
   (AC2.2)           {printf (lookup_query, 'lookup -ln %s -fn %s ',
                              $LN,$FN);}
```

Furthermore, it is possible to associate non-terminal descriptions to actions:

```
(NT5.2) _OptLN ::= <last_name $LN>
(AC5.2)            {printf ($_OptLN, '-ln %s', $LN);}
```

3.2 Extractor, DEX templates and Filter Processor

A query result is often expressed in a unstructured format. The *Extractor* component uses the *DEX* templates to analyze and structure data received from the sources. *DEX* templates contain the description of the data received from a source and informations about the fields to be extracted. After the extraction of the needed informations from the source output, they are converted by the *Packager* into a set of *OEM* objects. Then, this set of objects is filtered in the *Filter Processor*. The filter to be applied to the set of objects is a *MSL* query built by the *Converter* during the translation activity of the input query into executable commands. The *Filter Processor* applies this query to the set of retrieved objects and send the subset thus obtained to the *Client*.

4 The TSIMMIS mediator generator

The **MedMaker** system [4] is the *TSIMMIS* component developed for declaratively specifying mediators. It is targeted for integration of sources with unstructured or semi-structured data and/or sources with changing schemas. MedMaker provides the high level language *MSL* that allows the declarative specification of mediators.

At run time, when the mediator receives a request for information, the *Mediator Specification Interpreter (MSI)* collects and integrates the necessary informations from the sources, according to the specification. The process is analogous to expanding a view against a conventional relational database and *MSL* can be seen as a view definition language that is targeted to the *OEM* data model and the functionality needed for integrating heterogeneous sources.

4.1 The Mediator Specification Language MSL: an example

Let us introduce an example to illustrate *MSL*. We have two input sources: a relational database with two tables

```
employee(first_name,last_name,title,report_to)
student(first_name,last_name,year)
```

and a university system 'WHOIS' with informations on students and professors. For the first source, a wrapper called **'CS'** exports the informations (some of which are shown in Figure 3), as *OEM* objects; the second source uses a wrapper called **'whois'** (some objects are shown in Figure 4).

We want to develop a mediator, called **'MED'** with objects integrating all the informations about a person, say 'Chung', of the department 'cs', coming

```
<&e1, employee, set,       {&f1,&l1,&t1, &rep1}>
      <&f1,    first_name, string,  'Joe'>
      <&l1,    last_name,  string,  'Chung'>
      <&t1,    title,      string,  'professor'>
      <&rep1,  reports_to, string,  'John Hennessy'>

<&e2, employee, set,       {&f2,&l2,&t2}>
      <&f2,    first_name, string,  'John'>
      <&l2,    last_name,  string,  'Hennessy'>
      <&t2,    title,      string,  'chairman'>
..............etc.

<&s3, student,  set,       {&f3,&l3,&y3}>
      <&f3,    first_name, string,  'Pierre'>
      <&l3,    last_name,  string,  'Huyn'>
      <&y3,    year,       integer, 3>
```

Fig. 3. CS objects in OEM

from the two wrappers. Given the objects of Figure 3 and 4, **MED** must be able to combine them to obtain the object of Figure 5.
Let us introduce the rules, expressed in in MSL, which define the mediator MED:

```
(MS1) Rules:
<cs_person {<name N> <rel R> Rest1 Rest2}>
          :- <person {<name N> <dept 'cs'> <relation R> | Rest1}>
             @whois
             AND decomp(N, LN, FN)
             AND <R {<first_name FN> <last_name LN> | Rest2}>@cs
External:
decomp(string,string,string)(bound,free,free) impl by name_to_lnfn
decomp(string,string,string)(free,bound,bound) impl by lnfn_to_name.
```

Remark 4

```
<&p1, person,   set,      {&n1, &d1, &rel1, &elem1}>
      <&n1,    name,     string, 'Joe Chung'>
      <&d1,    dept,     string, 'cs'>
      <&rel1,  relation, string, 'employee'>
      <&elem1, e_mail,   string, 'chung@cs'>
..............etc.
```

Fig. 4. whois objects in OEM

```
<&cp1, cs_person, set,       {&mn1, &mrel1, &t1, &rep1, &elm1}>
       <&mn1,    name,       string, 'Joe Chung'>
       <&mrel1,  relation,   string, 'employee'>
       <&t1,     title,      string, 'professor'>
       <&rep1,   reports_to, string, 'John Hennesy'>
       <&elem1,  e_mail,     string, 'chung@cs'>
```

Fig. 5. object exported by MED

The 'creation process' of a mediator object is a pattern matching *process: first the object extracted by the wrapper satisfying the tail are collected and their component are linked to the variables, then the bindings are used to create objects expressed in the head.*

With reference to the example, we want to search the objects of the sources 'cs' and 'WHOIS' which links to the tail expressed in rule 'MS1' (i e. top-level person object of 'whois' with sub-object name, dept='cs' and relation; top-level person object of 'cs' with FN and LN obtained from the corresponding N of whois (&e1 satisfy the model).

The decomp function executes the string transformations in order to obtain first_name and last_name of a person. When the objects satisfying the tail pattern have been obtained, the rule head is used to build the virtual object which is the union of data coming from the wrappers (&cp1 is the result of the union of &p1 e &e1).

MSL has other querying functionalities to facilitate integration of heterogeneous sources: expressing only variables in the value fields it is possible to obtain informations about the structure of an information source (e.g. after a schema changing). MSL allows 'wildcard' to search objects at any nesting level without specifying the whole path as it would be necessary with conventional OO languages.

4.2 Architecture and implementation of MSI

The Mediator Specification Interpreter *(MSI)* is the component of *MedMaker* which process a query on the basis of the rules expressed with *MSL*. It is composed of three modules: *VE&AO (View Expander* and *Algebraic Optimizer*); *cost-based optimizer*; *datamerge engine*. *VE&AO* reads a query and, on the basis of the *MSL* specification, discovers what objects have to be obtained from a source and determines the conditions that the obtained objects must satisfy; gives a result called *logical datamerge program* which is passed to the second module *cost-based optimizer*. The optimizer develops an access plane to retrieve and combine objects, i.e. , what requests to submit to the sources; the order of requests submissions; how to combine the results to obtain the requested objects.

The access plane is passed to the third component, *datamerge engine*, which executes it and gives the results. Let us consider an example of the *MSI* query processing.

Suppose that a client want to retrieve informations about 'Joe Chung'; the query expressed in *MSL* is the following:

```
(Q1) JC :- JC :< cs_person {<name 'Joe Chung'>}> @MED
```

The object pattern in the tail of the query `Q1` is matched against the structure of the objects hold in *MED*. **View expansion** - having as input the query `Q1` and the *MSL* rules, *VE&AO* substitutes the query tail with the pattern of the objects in the sources, obtaining the datamerge rule `R2`:

```
(R2) <cs_person {<name 'Joe Chung'> <rel R> Rest1 Rest2}>
                :- <person {<name 'Joe Chung'> <dept 'cs'>
                           <relation R> | Rest1}>@whois
                   AND decomp( 'Joe Chung', LN, FN)
                   AND <R {<first_name FN> <last_name LN> |
                          Rest2 }>@cs.
```

The rule obtained in this way has a head representing the query and a tail, obtained from *MS1* rule, indicating how to select the objects from the wrappers. **Execution plan** - when the *MSI* knows what objects have to be fetched from the sources, the *cost-based optimizer* build the *physical datamerge program*, that specify what query should be sent to the sources. A possible efficient plan to process query `Q1` is the following:

1. Bindings for variables `R` and `Rest1` are obtained from the source by **whois** execution of the following query:

   ```
   <bind_for_whois {<bind_for_r R> < bind_for_Rest1 Rest1>}>
            :- <person {<name 'Joe Chung'> < dept 'cs' >
                       < relation R> | Rest1 }>@whois
   ```

2. Bindings for variables `LN` and `FN` are obtained from one of the two decomp functions: `decomp (name_to_lnfn)`
3. Each bind of `R` is combined with a value obtained at step 2. and the query is submitted to **CS** to obtain the values of the variable `Rest2`.
4. the objects satisfying the head of rule `R2` can be generated (e.g. &cp1 should be an object built following these steps)[4].

5 Description Logics: overview

Description Logics languages - *DLs* [4], derived from the KL-ONE model [69], have been proposed in the 80's in the Artificial Intelligence research area. DLs

[4] *DLs* are also known as *Concept Languages* or *Terminological Logics*.

are fragments of first order logic: they enable *concepts* to be expressed, that can be viewed as logical formulas built using unary and binary predicates, and contain one free variable (to be filled with instances of the concept). They bear similarities with Complex object data models (*CODMs*), recently proposed in the database area [34, 35, 36, 39, 48, 61, 62]. CODMs are concerned with only the structural aspects of object-oriented data models proposed for Object-Oriented Databases (*OODBs*) [38, 59, 14] and represent well-known notions such as types, complex values, classes, objects with identity and inheritance. DLs too are concerned with only structural aspects; concepts roughly correspond to database classes and are organized in inheritance taxonomies. An additional feature of DLs with respect to CODMs is that concepts are differentiated in *primitive* and *defined*: a primitive concept description represents necessary conditions (thus corresponding to the usual database class semantics); a defined concept description represents necessary and sufficient conditions (thus corresponding to a database view semantics).

Remark 5 *By exploiting defined concepts semantics of DLs, and, given a* type as set *semantics to concept descriptions, it is possible to provide reasoning techniques: to compute* subsumption *relations among concepts(i.e. "isa" relationships implied by concepts descriptions) and to detect* incoherent *(i.e. always empty) concepts.*

The research on DLs has provided reasoning techniques to determine incoherence and subsumption of concepts and has assessed the complexity of these inferences for a variety of *acyclic* , i.e. not allowing recursive descriptions, DLs (see e.g.[56]).

DLs reasoning techniques are profitable for database design activities, as will be briefly argued in the following. In fact, if we map a database schema including only classes (no view) into one of the DLs supported by a system, we are able to automatically detect incoherent classes. A more active role can be performed with the introduction of views.

Remark 6 *By means of DLs reasoning techiques, a view, can be automatically* classified *(i.e., its right place in an already existing taxonomy can be found) by determining the set of its most specific subsumer views (*subsumers*) and the set of its most generalized specialization views (*subsumees*).*

Thus, besides a passive consistency check, *minimality* of the schema with respect to inheritance can easily be computed. [49] reports a fundamental work in a database environment, where well-known conceptual data models have been mapped in a suitable DL and polynomial subsumption and coherence algorithms are given.

The expressiveness of CODMs gave rise to new problems for this mapping, as many of their features were not supported by implemented DLs. For instance, most of the CODMs introduces a clear cut distinction between values and objects with identity and, thus, between object classes and value types. This distinction was not present in DLs. Further, CODMs often support additional type constructors, such as set and sequence. Mostly important, CODMs support the

representation and management of *cyclic classes*, i.e., classes which directly or indirectly refer to themselves, are allowed.

Recently, a description logics (`odl` = *Object Description Logics*) [5] overcoming the above problems, and a theoretical framework for database design based on subsumption computation and coherence detection has been proposed in [48].

Remark 7 *The* `odl` *description logics represents the structural part of OODB data models (and of the standard data model* ODM *of ODMG93 [14]) and incorporates: value-types, complex values, classes, objects with identity, and inheritance.*

The main extension of `odl`, with respect to CODMs, is the capability of expressing *base* and *virtual* classes. Base classes correspond to ordinary classes used in database systems and virtual classes (corresponding to defined concepts semantics) are similar to database *views* [6]. Cyclic classes (base and virtual) are allowed and a *greatest-fixedpoint* semantics has been adopted for cyclic virtual classes.

Note that the interpretation of tuples in `odl` implies an *open world semantics for tuple* types similar to the one adopted by Cardelli [26].

For instance, if we have the following assignments of values to objects:

$$\delta\colon \begin{cases} o_1 & \mapsto [\mathtt{a}\colon \textit{"xyz"}, \mathtt{b}\colon 5] \\ o_2 & \mapsto \langle \mathit{true}, \mathit{false} \rangle \\ & \vdots \\ o_{128} & \mapsto \{o_1, o_2\} \\ & \vdots \end{cases}$$

Adopting an *open world semantics for tuple*, it follows that

$$o_1 \in \mathcal{I}\Big[\triangle[\mathtt{a}\colon \mathtt{String}]\Big], \qquad o_1 \in \mathcal{I}\Big[\triangle[\mathtt{a}\colon \mathtt{String}, \mathtt{b}\colon \mathtt{Int}]\Big],$$
$$o_2 \in \mathcal{I}[\triangle\langle \mathtt{Bool} \rangle], \qquad o_{128} \in \mathcal{I}[\{\triangle \top_C\}].$$

Remark 8 *The adoption of an open world semantics for tuple types in* `odl` *permits an alternative formulation of the OEM feature expressed in Remark 1: objects of a class share a common minimal structure, but could have further additional properties.*

Remark 9 *Coherence checking and subsumption computation are effective for query optimization. A query has the semantics of a virtual class, as it expresses a set of necessary and sufficient conditions. If we restrict the query language to the subset of queries expressible with the schema description language we can perform incoherence detection and subsumption computation for queries.*

The choice of restricting the query language in order to have DDL=DML has been made in the some works on query optimization based on acyclic DLs such as CANDIDE [45], CLASSIC [51], BACK [66] and [52].

[5] not to be confused with the homonymous ODL language of ODMG93 [14].

[6] Views in OODB have been called virtual classes by Abiteboul and Bonner [33].

Remark 10 *Coherence checking and subsumption computation can be classified as* semantic query optimization *techniques [60, 55, 68], as they perform a transformation of a query into a semantically equivalent one, that minimizes query execution costs.*

- *if the query is detected as incoherent a null answer can immediately be returned without accessing the database;*
- *if the query is coherent, it can be temporarily classified in the schema with respect to views. As a result, either an* equivalent *view or the set of* immediate subsumers *and* immediate subsumees *is returned. In the former case, the answer set is simply the set of all objects that are instances of the view equivalent to the query; in the latter case, the union of the sets of instances of immediate subsumee views are included in the answer set and only objects that are instances of the immediate subsumer view, but not instances of the immediate subsumee views, have to be checked against the query condition.*

Usually, in database environment, query languages are more expressive than schema description languages. This holds for Relational Databases and, more recently, for OODBs, see for example the proposed standard *OQL* language [14].

Remark 11 *In the context of extraction and integration of textual heterogeneous data sources, provided that a highly expressive OO schema description language is available, we can adopt as query language the same language* [7]

As a matter of fact *recursive* [8] views and queries are relevant for OODB query languages. For instance, object-oriented recursive queries are important in engineering DBs [54]. Unfortunately, while many works have been developed in deductive relational databases [57, 37], very few proposals have been presented for OODB query languages. Recently, significant contributions in this environment have been presented. In [50] a form of recursion was introduced which corresponds to the notion of linear recursive queries; in [32] the effectiveness of incoherence detection and subsumption computation for semantic optimization of cyclic views and queries, expressed in `odl`[48], is shown by adopting different styles of semantics: *greatest fixed-point*, *least fixed-point*, and *descriptive semantics*.

A system, implementing algorithms for incoherence detection and subsumption computation in `odl` has been developed. It is composed of two modules: the `odldesigner` prototype [41] and the `DBott` prototype. `odldesigner` was first developed in Sicstus Prolog at CIOC-CNR, Bologna, as part of the *logidata* project [41] and implements DLs reasoning techniques for advanced database management systems handling complex objects data models. It is an active tool which supports automatic building of type taxonomies for complex object database systems, preserving coherence and minimality with respect to inheritance. It implements the theoretical framework of [48].

[7] the choice of a simple query language (a significative restriction of OQL) has been also recently made at the I^3 workshop on mediators language standards [12].

[8] In the following cyclic and recursive will be used as synonymous.

The actual version of `odldesigner` has been developed (in C language) at the Department of Engineering Sciences of the University of Modena and provides an ODL (ODMG93) standard interface to input a schema description. Furthermore, it includes an extension of `odl`, called `ocdl`, allowing to express *quantified path types* and *integrity constraints* 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, *quantified* paths to navigate through set types are provided. The allowed quantifications are existential and universal and they can appear more than once in the same path. Integrity constraints rules, *if then rules*, whose antecedent and consequent are ODL *virtual* types (i.e. a type description expresses a set of sufficient and necessary conditions) allows the declarative formulation of a relevant set of integrity constraints.

`DBott` prototype was developed at the Department of Engineering Sciences of the University of Modena to perform semantic optimization of OODB queries [42, 43]. It has been developed in C and provides an OQL(ODMG93) standard interface. A demo of `DBott` is available on Internet at the following address: (http://sparc20.dsi.unimo.it/).

6 A Semantic approach for Mediators: `ocdl` Description Logics

The *TSIMMIS OEM + MSL* approach towards *mediators* development is 'structural':

- OEM [5] , in fact, is a *self-describing model* [31] where each data item has an associated descriptive label and *without a strong typing system*;
- *semantic informations are effectively encoded in the MSL rules* that do the integration.

There are many projects following the 'structural approach' [10, 16, 11]. Let us introduce some fundamental arguments in favour of the 'structural approach' (considering *TSIMMIS* as a target system):

1. the power, along with the flexibility, generality and coinciseness of OEM and MSL make the 'structural' approach a good candidate for the integration of widely heterogeneous and semistructured information sources;
2. MSL and OEM can be seen as a form of first-order logic: rules are supported allowing the sharing of definitions of terms among components;
3. the schema-less nature of OEM objects is particularly useful when a client does not know in advance the labels or structure of OEM objects.
   - In traditional data models, a client must be aware of the schema in order to pose a query. With this approach, a client can discover the structure of the information as queries are posed.

– A conventional OO language breaks down in such a case, unless one defines an object class for every possible type of irregular object.

Many other projects follow a 'semantic' approach [19, 20]. This approach can be characterized as follows:

– for each source, meta-data, i.e. conceptual schema, must be available;
– semantic informations are encoded in the schema;
– a common data model as the basis for describing sharable informations must be available;
– partial or total schema unification is performed.

Let us introduce some fundamental arguments in favour of a 'semantic approach' adopting conventional OO data models:

1. in most research areas (programming languages, databases and artificial intelligence) conventional OO models with strong type systems and including: classes, aggregation and inheritance hierarchies to model structural intensional knowledge and, often, methods to model behavioural knowledge are at present adopted;
2. a relevant effort has been devoted to develop OO standards: CORBA [15] for object exchanging among heterogeneous systems; ODMG93 (including ODM model and ODL language for schema description; OQL language as query language) for object oriented databases [14];
3. the schema nature of conventional OO models together with classification aggregation and generalization abstract modelling primitives allows to organize extensional knowledge;
4. the adoption of a *type as a set* semantics for a schema permits to check consistency of instances with respect to their descriptions;
5. semantic informations encoded into a schema permit to efficiently extract informations.

By coupling a system based on description logics (`odldesigner` + `DBott`), a 'semantic approach' and some interesting features of *TSIMMIS* it is possible to devise a powerful I^3 system that conforms to the ODMG93 standard:

1. the standard ODM model and ODL language as common data model and common data language are adopted both for sources and mediators;
2. the ODL language is extended to represent rules in analogy with *MSL*;
3. the ODL language is extended to represent QDTL;
4. a *minimal core language* which is a restriction of the object oriented query language OQL such that it will accept queries for relational databases is adopted[9];
5. a system based on description logics (`odldesigner` + `DBott`) with interfaces for the above languages is adopted;
6. `ocdl` is extended to support QDTL translation.

[9] this choice is also suggested in the proposal for a standard in mediator languages [12]

A mediator can be generated with the above system by introducing the following knowledge:

- describe the schemata of the sources to be integrated and the mediator schema in the ODL language;
- describe query templates in the *minimal core language*;
- describe the mediator rules.

Having (`odldesigner` + `DBott`) available, the knowledge expressed in the standard languages above is automatically translated into `ocdl` classes and virtual classes and the the `ocdl` incoherence detection and subsumption algorithms can be exploited in the following way:

- to perform data integration by exploiting mediator rules;
- to execute a query by determining the most efficient one among the supported subsuming query.

Two final remarks: as observed in Remark 8, the adoption of an open world semantics overcomes the problems of conventional OO data models above mentioned; for sources supporting OODBMS or RDBMS, query templates are not necessary and `DBott` can be used as a powerful query optimizer for OQL queries.

References

1. Chawathe S., H. Garcia Molina, J. Hammer, K.Ireland, Y. Papakostantinou, J.Ullman, and J. Widom. The TSIMMIS Project: Integration of Heterogeneous Information Sources. In *Proceedings of IPSJ Conference*, pages. 7–18, Tokyo, Japan, October 1994. (Also available via anonymous FTP from host db.stanford.edu, file /pub/chawathe/1994/tsimmis-overview.ps.).
2. H. Garcia-Molina et al. The TSIMMIS approach to mediation: Data models and languages. In *Proceedings of 1995 NGITS workshop.* (ftp://db.stanford.edu/pub/garcia/1995/tisimmis-models-languages.ps.).
3. Y. Papakonstantinou, H. Garcia-Molina, J. Ulman and Ashish Gupta. A query Translation scheme for rapid implementation of wrappers. avaible at ftp://db.stanford.edu/pub/papakonstantinou/1995/querytran-extended.ps.
4. Y. Papakonstantinou, H. Garcia-Molina, J. Ulman, "MedMaker: A mediation system based on declarative specification", avaible at ftp://db.stanford.edu/pub/papakonstantinou/1995/medmaker.ps.
5. Y. Papakonstantinou, H. Garcia-Molina and J. Widom. Object Exchange Across Heterogeneous Information Sources. In Proceedings of IEEE International Conference on Data Engineering, pages. 251–260, Taipei, Taiwan, March 1995. (Also available via anonymous FTP from host db.stanford.edu file /pub/papakonstantinou/1994/object-exchange-heterogeneous-is.ps.)
6. H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and Jennifer Widom. Integrating and Accessing Heterogeneous Information Sources in TSIMMIS. In *Proceedings of the AAAI Symposium on Information Gathering*, pages. 61–64, Stanford, California, March 1995. (Also available via anonymous FTP from host db.stanford.edu, file /pub/garcia/1995/tsimmis-abstract-aaai.ps.)

7. A. Rajaraman, Y. Sagiv, and J. Ullman. Answering Queries Using Templates with Binding Patterns. In *Proceedings of the 14th ACM PODS*, pages. 105–112, San Jose, California, May 1995. (Also available via anonymous FTP from host db.stanford.edu, file /pub/rajarman/1994/limited-opsets.ps).
8. D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom. Querying Semistructured Heterogeneous Information. In *International Conference on Deductive and Object-Oriented Databases*, 1995. (Also available via anonymous FTP from host db.stanford.edu, file /pub/quass/1994/querying-full.ps)
9. R. Ahmed et al. The Pegasus heterogeneous multidatabase system. *IEEE Computer*, 24:19-27, 1991.
10. E. Bertino.. Integration of heterogeneous data repositories by using object-oriented views. In *Proc. Intl Workshop on Interoperability in Multidatabase Systems*, pages 22-29, Kyoto, Japan,1991.
11. Y.J. Breibart et al. Database integration in a distributed heterogeneous database system. In *Proc. 2nd Intl IEEE Conf. on Data Engineering*, Los Angeles, CA, February 1986.
12. P. Bunemann, L. Raschid, J. Ulman. Mediator Languages - a Proposal for a standard.
    Report of an I3/POB working group held at the University of Maryland, April 1996 (available as ftp://ftp.umiacs.umd.edu/pub/ONRrept/medmodel96.ps).
13. M.J. Carey et al. Towards heterogeneous multimedia information systems: the Garlic approach. Technical Report RJ 9911, IBM Almaden Research Center, 1994.
14. R. G. G. Cattel, et al. *The Object Database Standard - ODGM93. Release 1.2.* Morgan Kaufmann,1996.
15. Object Request Broker Task Force. The Common Object Request Broker: Architecture and Specification, December 1993. Revision 1.2, Draft 29.
16. U. Dayal and H. Hwuang. View definition and generalization for databse integration in a multidatabase system. In *Proc. IEEE Workshop on Object-Oriented DBMS*, Asilomar, CA, September 1986.
17. M. Freedman. WILLOW: Technical overview. Available by anonymous ftp from ftp.cac.washington.edu as the file willow/Tech-Report.ps, September 1994.
18. A. Gupta. *Integration of Information Systems: Bridging heterogeneous Databases* IEEE Press, 1989.
19. J. Hammer and D. McLeod. An approach to resolving semantic heterogeneity in a federation of autonomous, heterogeneous database systems. *Intl Journal of Intelligent and Cooperative Information Systems*, 2: 51-83, 1993.
20. M. Huhns et al. Enterprise information modeling and model integration in Carnot. Technical Report Carnot128-92, MCC,1992.
21. W. Kim et al. On resolving schematic heterogeneity in multidatabase systems. *Distributed and Parallel Databases*, 1:251-279, 1993.
22. W. Litwin, L. Mark, and N. Roussopoulos. Interoperability of multiple autonomous databases. *ACM Computing Surveys*, 22:267-293, 1990.
23. K. Shoens et al. The RUFUS system: Information organization for semistructured data. newblock In *Proc. VLDB Conference*, Dublin, Ireland, 1993.
24. G. Thomas et al. Heterogeneous distributed database systems for production use. *ACM Computing Surveys*, 22: 237-266,1990.
25. J.C. Franchitti and R. King. Amalgame: a tool for creating interoperating persistent, heterogeneous components. In *Advanced Database Systems*
26. L. A. Cardelli. Semantics of multiple inheritance. In *Semantics of Data Types*, pages 51–67. Springer-Verlag, Berlin, Heidelberg, New York, 1984.

27. D. Calvanese, G. De Giacomo and M. Lenzerini. Structured Objects: Modeling and Reasoning. In *Proceedings of International Conference on Deductive and Object-Oriented Databases*, 1995.
28. G. De Giacomo and M. Lenzerini. PDL based framework for reasoning about actions. In *Proceedings of the AI*IA'95*, LNAI 992, pages 103–114, Spriger Verlag 1995.
29. G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25:38-49,1992.
30. 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.
31. L. Mark and N. Roussopoulos. Information interchange between self-describing databases. *IEEE Data Engineering*, 10:46-52,1987.
32. D. Beneventano and S. Bergamaschi. Incoherence and Subsumption for cyclic queries and views in Object Oriented Databases. In *DKE*, january, 1997.
33. S. Abiteboul and A. Bonner. Objects and views. In *SIGMOD*, pages 238–247. ACM Press, 1991.
34. S. Abiteboul and S. Grumbach. Col: a logic-based language for complex objects. In S. Ceri, J.W. Schmidt, and M. Missikoff, editors, *EDBT '88 - Lecture Notes in Computer Science N.303*, pages 271–293. Springer-Verlag, 1988.
35. S. Abiteboul and R. Hull. IFO: A formal semantic database model. *ACM Transactions on Database Systems*, 12(4):525–565, 1987.
36. S. Abiteboul and P. Kanellakis. Object identity as a query language primitive. In *SIGMOD*, pages 159–173. ACM Press, 1989.
37. R. Agrawal. Alpha: An extension of relational algebra to express a class of recursive queries. *IEEE Transactions on Software Enginering*, 14(7), July 1988.
38. M. Atkinson et al. The object-oriented database system manifesto. In *1nd Int. Conf. on Deductive and Object-Oriented Databases*. Springer-Verlag, 1989.
39. P. Atzeni, editor. *LOGIDATA$^+$: Deductive Databases with Complex Objects*. Springer-Verlag: LNCS n. 701, Heidelberg - Germany, 1993.
40. F. Baader. Terminological cycles in KL-ONE-based knowledge representation languages. In *8th National Conference of the American Association for Artificial Intelligence*, volume 2, pages 621–626, Boston, Mass., USA, 1990.
41. J. P. Ballerini, S. Bergamaschi, and C. Sartori. The ODL-DESIGNER prototype. In P. Atzeni, editor, *LOGIDATA+: Deductive Databases with complex objects*. Springer-Verlag, 1993.
42. D. Beneventano, S. Bergamaschi and C. Sartori. Using Subsumption for Semantic query optimization in OODB. *International Workshop on Description Logics*, DFKI Technical Report D-94-10.
43. D. Beneventano, S. Bergamaschi, S. Lodi, and C. Sartori. Using subsumption in semantic query optimization. In A. Napoli, editor, *IJCAI Workshop on Object-Based Representation Systems*, Chambery - France, August 1993.
44. G. Di Battista and M. Lenzerini. Deductive entity relationship modeling. *IEEE Trans. on Knowledge and Data Engineering*, 5(3):439–450, June 1993.
45. H.W. Beck, S.K. Gala, and S.B. Navathe. Classification as a query processing technique in the CANDIDE data model. In *5th Int. Conf. on Data Engineering*, pages 572–581, Los Angeles, CA, 1989.
46. C. Beeri. Formal models for object-oriented databases. In W. Kim, J.M. Nicolas, and S. Nishio, editors, *Deductive and Object-Oriented Databases*, page 405:430. Elsevier Science Publisher, B.V. - North-Holland, 1990.

47. D. Beneventano and S. Bergamaschi. Subsumption for complex object data models. In J. Biskup and R. Hull, editors, *4th Int. Conf. on Database Theory - Berlin*, pages 357–375, Heidelberg, Germany, October 1992. Springer-Verlag.
48. S. Bergamaschi and B. Nebel. Acquisition and validation of complex object database schemata supporting multiple inheritance. *Applied Intelligence: The International Journal of Artificial Intelligence, Neural Networks and Complex Problem Solving Technologies*, 1993.
49. S. Bergamaschi and C. Sartori. On taxonomic reasoning in conceptual design. *ACM Transactions on Database Systems*, 17(3):385–422, September 1992.
50. E. Bertino, M. Negri, G. Pelagatti, and L. Sbattella. Object-oriented query languages: The notion and the issues. *IEEE Trans. Knowl. and Data Engineering*, 4(3):223–236, June 1992.
51. A. Borgida, R.J. Brachman, D.L. McGuinness, and L.A. Resnick. CLASSIC: A structural data model for objects. In *SIGMOD*, pages 58–67, Portland, Oregon, 1989.
52. M. Buchheit, M. A. Jeusfeld, W. Nutt, and M. Staudt. Subsumption between queries to object-oriented database. In *EDBT*, pages 348–353, 1994.
53. L. Cardelli. A semantics of multiple inheritance. In *Semantics of Data Types - Lecture Notes in Computer Science N. 173*, pages 51–67. Springer-Verlag, 1984.
54. R.G.G. Cattell and J. Skeen. Object operations benchmark. *ACM Transactions on Database Systems*, 17(1), 1990.
55. U. S. Chakravarthy, J. Grant, and J. Minker. Logic-based approach to semantic query optimization. *ACM Transactions on Database Systems*, 15(2):162–207, June 1990.
56. F.M. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. In J. Allen, R. Fikes, and E. Sandewall, editors, *KR '91 - 2nd Int. Conf on Principles of Knowledge Representation and Reasoning*, pages 151–162, Cambridge - MA, April 1991. Morgan Kauffmann Publishers, Inc.
57. H. Gallaire, J. Minker, and J.M. Nicholas. Logic and databases: a deductive approach. *ACM Computing Surveys*, 16(2), 1984.
58. M. Kifer, W. Kim, and Y. Sagiv. Querying object-oriented databases. In *SIGMOD '92*, pages 393–402. ACM, June 1992.
59. W. Kim and F. Lochovsky, editors. *Object-Oriented Concepts, Databases, and Applications*. Addison-Wesley, Reading (Mass.), 1989.
60. J. J. King. Quist: a system for semantic query optimization in relational databases. In *7th Int. Conf. on Very Large Databases*, pages 510–517, 1981.
61. 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.
62. C. Lécluse and P. Richard. The O_2 data model. In *Int. Conf. On Very Large Data Bases*, pages 411–422, Amsterdam, 1989.
63. J.W. Lloyd. *Logic Programming*. Springer-Verlag, Berlin, 1987.
64. B. Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43(2), 1990.
65. B. Nebel. Terminological cycles: Semantics and computational properties. In J.F. Sowa, editor, *Principles of Semantic Networks*, chapter 11, pages 331–362. Morgan Kaufmann Publishers, Inc., 1991.
66. B. Nebel and C. Peltason. Terminological reasoning and information management. Technical Report 85, Tech. Univ, Berlino, October 1990.

67. D.A. Schmidt. *Denotational Semantics: A Methodology for Language Development.* Allyn and Bacon, Boston, 1986.
68. S. Shenoy and M. Ozsoyoglu. Design and implementation of a semantic query optimizer. *IEEE Trans. Knowl. and Data Engineering*, 1(3):344–361, September 1989.
69. W.A. Woods and J.G. Schmolze. The kl-one family. In F.W. Lehman, editor, *Semantic Networks in Artificial Intelligence*, pages 133–178. Pergamon Press, 1992. Pubblished as a Special issue of *Computers & Mathematics with Applications*, Volume 23, Number 2-9.

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