3. Integration of Information from Multiple Sources of Textual Data

Sonia Bergamaschi[1,2] and Domenico Beneventano[1]

[1] Dipartimento di Scienze dell'Ingegneria Universitá di Modena, Italy.
E-Mail: sbergamaschi@deis.unibo.it
[2] CSITE - CNR, Universitá di Bologna, Italy.
E-Mail: sonia@dsi.unimo.it

3.1 Introduction

The number of information sources in the Internet is exponentially increasing. As a consequence, 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 heterogeneous (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.

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 information 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. [5, 42, 262, 342, 389, 668]). Many projects have adopted *Object Oriented* (OO) data models to facilitate integration [5, 105, 505] and, recently, systems for the integration of sources with minimal structure have appeared [223, 503, 620]. Furthermore, 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, a workshop was held on this topic at the University of Maryland in April, 1996 [98, 724].

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 chapter as the amount of approaches and systems very recently proposed is as large as to require a paper on its own.

In this chapter we will discuss problems and solutions of the extraction/integration of information from multiple sources, highly heterogeneous, of textual data and of their integration in order to provide *true information*. Main problems to be faced in integrating information coming from distributed sources are related to structural and implementation heterogeneity (including differences in hardware platforms, DBMS, data models and data languages), and to the lack of a common ontology, which leads to semantic heterogeneity. Semantic heterogeneity occurs when different names are employed to represent the same information or when different modeling constructs are used to represent the same piece of information in different sources [342]. Some approaches have been recently proposed in the literature for the extraction and integration of conventional structured databases [90] and semi-structured data [99, 111] taking into account semantic heterogeneity. Data integration architectures are usually based on *mediators* [722], where knowledge about data of multiple sources is combined to provide a global view of the underlying data. Two fundamental approaches have emerged in the literature: *structural* [230, 566] and *semantic* [18, 57, 90].

There are many projects following the "structural approach" [64, 88, 158]. This approach can be characterized as follows (considering TSIMMIS [461] as a target system):

- a *self-describing model* where each data item has an associated descriptive label and *without a strong typing system*;
- semantic information are effectively encoded in rule that do the integration.

Let us introduce some fundamental arguments in favour of the "structural approach":

1. the flexibility, generality and conciseness of a self-describing model makes the "structural approach" a good candidate for the integration of widely heterogeneous and semistructured information sources;

2. a form of first-order logic languages that allow the declarative specification of a *mediator* is provided; a mediator specification is a set of rules which defines the mediator view of the data and the set of functions that are invoked to translate objects from one format to another.
3. the schema-less nature of modelled objects is particularly useful when a client does not know in advance the labels or structure of the objects of a source.
   - 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" [17, 18, 105, 298, 271, 566]. 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 information 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. the schema nature of conventional OO models together with classification aggregation and generalization primitives allows to organize extensional knowledge and to give a high level abstraction view of information;
2. the adoption of a schema permits to check consistency of instances with respect to their descriptions, and thus to preserve the "quality" of data;
3. semantic information encoded into a schema permits to efficiently extract informations, i.e., to perform query optimization;
4. a relevant effort has been devoted to develop OO standards: CORBA [485, 481, 638] for object exchanging among heterogeneous systems; ODMG93 for object oriented databases [112];

One of the most promising line of research in this environment is the *virtual approach* [300]. It was first proposed in multidatabase models in the early 80s [389, 668]. More recently, systems have been developed based on the use of description logics [17, 377] such as CLASSIC [76]. All of the virtual approaches are based on a model of query decomposition, sending subqueries to source databases, and merging the answers that come back. Recent systems based on description logics are focused primarily on conjunctive queries (i.e., expressible using select, project and join), and have more the flavor of the *Open World Assumption* - the answer provided through an integrated view will hold a subset of the complete answer that is implied by the underlying databases. For the schema, a "top-down" approach is used: in essence a global schema encompassing all relevant information is created, and data held in the source databases is expressed as views over this global schema [680]. See [17, 376, 378, 680] for some of the algorithms used in this approach.

In order to show and discuss the "structural approach", one of the most interesting projects, TSIMMIS, is presented. Then, a virtual description logics based approach, the MOMIS project, is introduced.

The outline of the chapter is the following. Section 3.2 is devoted to present the TSIMMIS system[1] and section 3.3 is devoted to present the MOMIS project. Section 3.2 begins with an overview of the TSIMMIS project, including the *OEM* data model and the *MSL* language adopted in the project. Subsection 3.2.2 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. Subsection 3.2.3 describes the TSIMMIS approach for generating a *mediators*, i.e., an integration and refinement tool of data coming from wrappers.

Section 3.3 begins with an overview of the MOMIS project. Subsection 3.3.1 presents the MOMIS architecture, the ODL_{I^3} language and the overall approach. Subsection 3.3.2 and subsection 3.3.3 presents the generation of the mediator global schema and the Global Schema Builder, respectively. In subsection 3.3.4 an overview of description logics formalism and related inference techniques plus the obd system are presented. Subsections 3.3.5 shows the role of ODB-Tools in the MOMIS project. Section 3.4 presents some final remarks.

Remarks all over the chapter fix relevant choices which lead to the design of an I^3 systems as MOMIS.

3.2 The TSIMMIS Project

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 [461].

The TSIMMIS data-integration system provides integrated access via an architecture that is common in many other projects: *wrappers/translators* [105, 221, 505] convert data into a common model; *Mediators* combine, integrates or refines the data from the wrappers. The wrappers also provide

[1] This section is a resume taken from papers of the TSIMMIS project. The contribution of the authors of this chapter is restricted to the remarks.

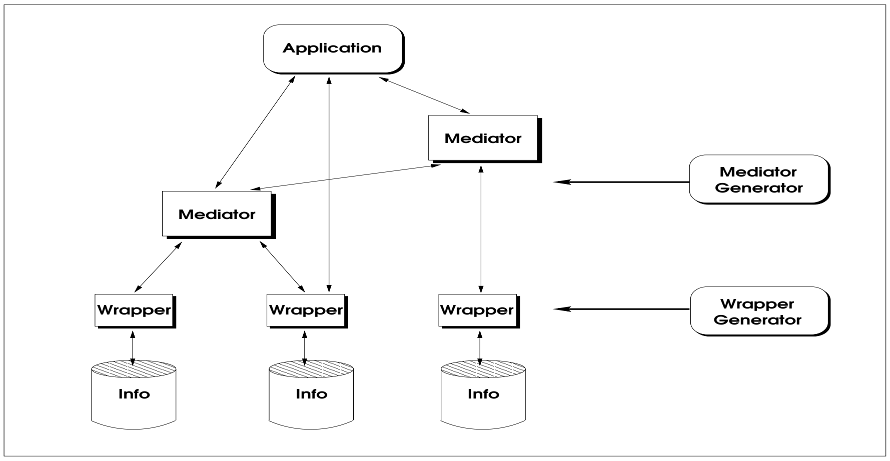

Fig. 3.1. Tsimmis Architecture

a common query language for extracting informations. Applications can access data directly through wrappers but they can also go through *mediators* [503, 505, 722].

In Figure 3.1, 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 *wrapper/generator* box in Figure 3.1);
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 3.1).

[2] translator and wrapper are synonymous in TSIMMIS.

The solutions proposed in TSIMMIS for the above goals are described in Section 3.2.2 and 3.2.3.

3.2.1 The OEM Model and the MSL Language

Let us briefly introduce the *OEM* model [505]. It is a *self-describing model* [416] 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* with a top-level object (`ob1`) and five sub-objects (`sub1` to `sub5`) 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.

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.*

Let us briefly introduce the *MSL* language [503] (a more detailed description is given in Section 3.2.3). *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.

3.2.2 The TSIMMIS Wrapper Generator

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 3.2:

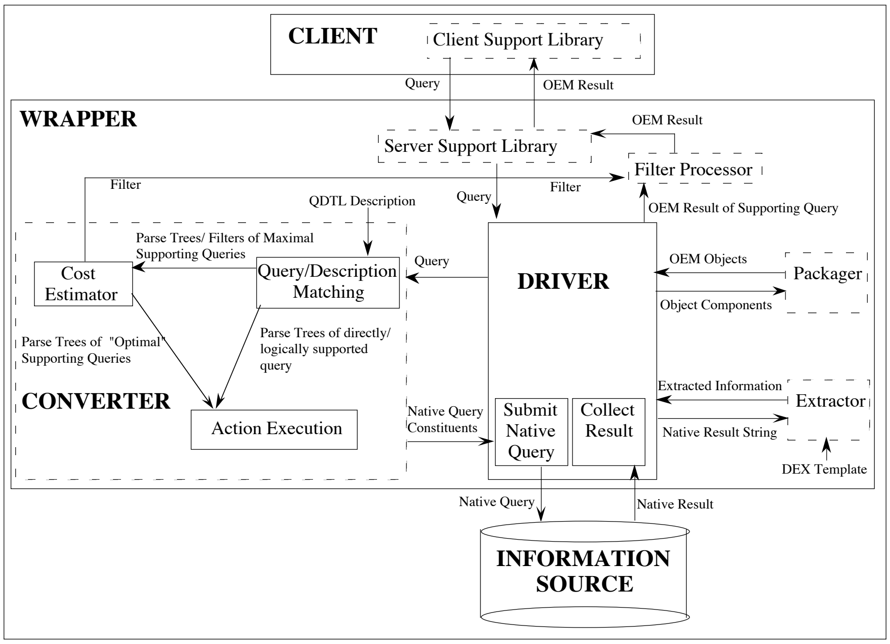

Fig. 3.2. TSIMMIS wrapper

– 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.

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] A full description of the *CONVERTER* and of *QDTL* is in [504].

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:

```
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 the following:

– *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`[4] or if it is subsumed bya a query `q'` directly supported by `t`;
– *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 as an example 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. 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 `Q8` query:

```
(Q8) O* :- <O person V>
```

which, in its turn, subsumes `Q7`.

[4] Two queries are logically equivalent if they give the same answer set in the same context.

Remark 2. *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.*

Actions in *QDTL* templates express the query in a format executable by the source. In the described Converter actions are 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);}
```

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*.

3.2.3 The TSIMMIS Mediator Generator

The **MedMaker** system [503] 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.

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)
```

```
<&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>
```

Table 3.1. CS objects in OEM

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 Table 3.1), as *OEM* objects; the second source uses a wrapper called **'WHOIS'** (some objects are shown in Table 3.2).

```
<&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.
```

Table 3.2. 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'>
```

Table 3.3. An object exported by **'MED'**

Let us suppose that a mediator, called **'MED'** with objects integrating all the informations about a person, say 'Chung', of the department **'CS'**, coming from the two wrappers has to be developed. Given the objects of Table 3.1 and 3.2, **MED** must be able to combine them to obtain the object of Table 3.3.

Let us introduce the rules of Table 3.4, expressed 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.
```

Table 3.4. Rules of **'MED'**

Remark 3. *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 satisfies 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.

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)[503].

3.3 The MOMIS Project

The goal of the MOMIS (**M**ediator envir**O**nment for **M**ultiple **I**nformation **S**ources) project[5] [58] is to provide an *integrated access* to information sources, allowing a user to pose a single query and to receive a single unified answer. The approach follows the *semantic* paradigm, in that conceptual schemata of an involved source are considered, and a common data model (ODM_{I^3}) and language (ODL_{I^3}) are adopted to describe sharable information. ODM_{I^3} and ODL_{I^3} are defined as a subset of the corresponding ODMG-93 [112] ODM and ODL. A Description Logics `olcd` (*Object Language with Complements and Descriptive cycles*) constraints [60, 52, 53]) is used as a kernel language and ODB-Tools as the supporting system [54]. An overview of description logics formalism, inference techniques and of ODB-Tools are presented in subsection 3.3.4.

With references to the classification of integration system proposed by Hull [300], MOMIS is in the category of "read-only views", i.e. systems whose task is to support an integrated, read-only, view of data that resides in multiple databases. The most similar projects are the GARLIC and SIMS project. The GARLIC project [105, 566] builds up on a complex wrapper architecture to describe the local sources with an OO language (GDL), and on the definition of Garlic Complex Objects to manually unify the local sources to define a global schema.
The SIMS project [17, 18] proposes to create a global schema definition exploiting the use of description logics (i.e. the LOOM language) for describing information sources. The use of a global schema allows both GARLIC and SIMS projects to support every possible user queries on the schema instead of a predefined subset of them.

Information integration in MOMIS is based on schemata and is performed through an *extraction and analysis* process followed by a *unification* process. The extraction and analysis process is devoted to derive a Common Thesaurus of *terminological relationships*, based on ODL_{I^3} schemata descriptions, and to the construction of clusters (by means of clustering techniques) of ODL_{I^3} classes, describing similar information in different schemata. The unification process builds an integrated global schema for the analyzed sources by integrating ODL_{I^3} classes in a given cluster.

The use of the `olcd` description logics together with hierarchical clustering techniques are the original contributions of the approach to enhance a semi-automated integration process. description logics allows us to interactively set-up the Thesaurus by deriving explicit terminological relationships from ODL_{I^3} schemata descriptions and by inferring new relationships out of the explicit ones. Moreover, optimization of the queries against the global schema is possible using description logics.

[5] developed in collaboration between the Università di Modena and the Università di Milano. Papers of the MOMIS project are available at: http://www.sparc20.dsi.unimo.it/publications.html.

Clustering techniques allow the automated identification of ODL_{I^3} classes in different source schemata that are semantically related and thus candidate to be unified in the global schema.

3.3.1 Overview, Architecture and $\mathbf{ODL}_{I^3}$ Language

In Fig. 3.3 the architecture of the MOMIS system is shown. With respect to the literature, this can be considered as an example of powerful I^3 system [98] and follows the TSIMMIS architecture [461]. Above each source lies a *wrapper* responsible for translating the structure of the data source into the common ODL_{I^3} language. In a similar way, the *wrapper* performs a translation of the query from the OQL_{I^3} language to a local request to be executed by a single source. Above the *wrapper* there is a *mediator*, a software module that combines, integrates, and refines ODL_{I^3} schemata received from the *wrappers*. In addition, the mediator generates the OQL_{I^3} queries for the *wrappers*, starting from the query request formulated with respect to the global schema. The mediator module is obtained by coupling a "semantic approach", based on a description logics component, i.e. ODB-TOOLS Engine, and an extraction and analysis component, i.e., Schema Analyzer and Classifier ARTEMIS, developed at the University of Milano [108, 109], together with a minimal ODL_{I^3} interface.

In order to easily communicate source descriptions between wrappers and mediator engine, we introduced a data description language, called ODL_{I^3}.

According to the recommendations of [98], and to the diffusion of the object data model (and its standard ODMG-93), ODL_{I^3} is very close to the ODL language and adds features to support requirements of our intelligent information integration system. ODL_{I^3} is a source independent language used by the mediator to manage the system in a common way (we suppose to deal with different source types, such as relational databases, object-oriented databases, files)[6]. The main extension, w.r.t. ODL, is the capability of expressing two kind of rules: *if then* rules, expressing in a declarative way integrity constraints intra and inter sources, and *mapping* rules between sources. It will be the wrapper task to translate the data description language of any particular source into ODL_{I^3} description, and to add information needed by the mediator, such as the source name and type.

The MOMIS approach to intelligent schema integration is articulated in the following phases:

1. *Generation of a Common Thesaurus.*
   The objective of this step is the construction of a Common Thesaurus of *terminological relationships* for schema classes belonging to different source ODL_{I^3} schemata.
   The following kinds of terminological relationships are specified: SYN

[6] The syntax of the language is included in [58].

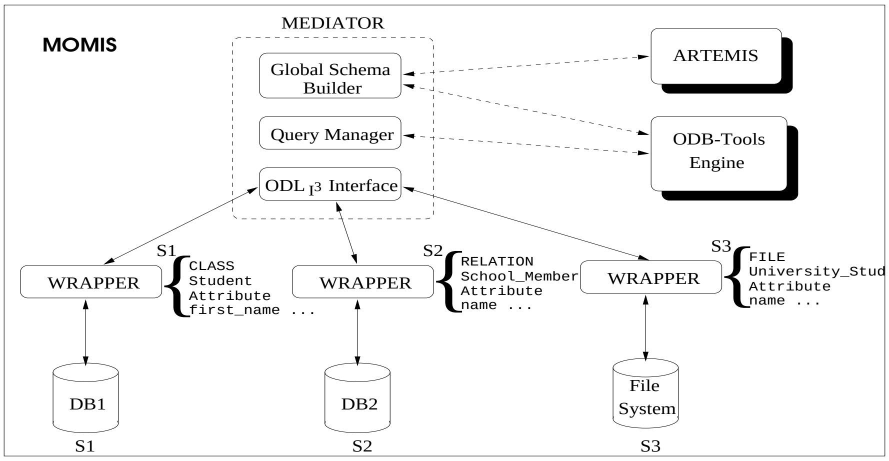

Fig. 3.3. Architecture of the MOMIS I^3 system

(Synonym-of), defined between two terms t_i and t_j, with $t_i \neq t_j$, that are considered synonyms, i.e., that can be interchangeably used in every considered source, without changes in meaning; BT (Broader Terms), or hypernymy, defined between two terms t_i and t_j such as t_i has a broader, more general meaning than t_j; RT (Related Terms), or positive association, defined between two terms t_i and t_j that are generally used together in the same context.
Terminological relationships are derived in a semi-automatic way, by analyzing the structure and context of classes in the schema, by using ODB-Tools and the description logics techniques.
2. *Affinity analysis of* ODL_{I^3} *classes.*
Terminological relationships in the Thesaurus are used to evaluate the level of *affinity* between classes intra and inter sources. The concept of affinity is introduced to formalize the kind of relationships that can occur between classes from the integration point of view. The affinity of two classes is established by means of affinity coefficients based on class names and attributes [108, 109].
3. *Clustering* ODL_{I^3} *classes* .
Classes with affinity in different sources are grouped together in clusters using hierarchical clustering techniques. The goal is to identify the classes that have to be integrated since describing the same or semantically related information.
4. *Generation of the mediator global schema.*
Unification of affinity clusters leads to the construction of the global schema of the mediator. A class is defined for each cluster, which is representative of all cluster's classes and is characterized by the union of their attributes. The global schema for the analyzed sources is composed

of all these new classes, derived from clusters, and is the basis for posing queries against the sources.

Each phase of the integration process is described in detail in [58, 59]. Once the mediator global schema has been constructed, it can be exploited by the users for posing queries. The information on the global schema is used by the *Query Manager* module of MOMIS for query reformulation and for semantic optimization using ODB-TOOLS, as discussed in [58].

3.3.2 Generation of the Mediator Global Schema

In this section we present the process which leads to the definition of the mediator global schema, that is the mediator view of data stored in local sources.

Running Example. First, we introduce an example used in this section to explain the approach (see Table 3.5). We consider three different sources. The first source is a relational database, `University` (S_1), containing information about the staff and the students of a given university. The second source `Computer_Science` (S_2) contains information about people belonging to the computer science department of the same university, and is an object-oriented database. A third source is also available, `Tax_Position` (S_3), derived from the registrar's office. It consists of a file system, storing information about student's tax_fees. For the complete source descriptions see [58]. The generation of Common Thesaurus, the Affinity analysis and the Clustering techniques applied to our example give raise to the cluster tree shown in Fig. 3.4, which classifies classes into groups at different level of affinity.

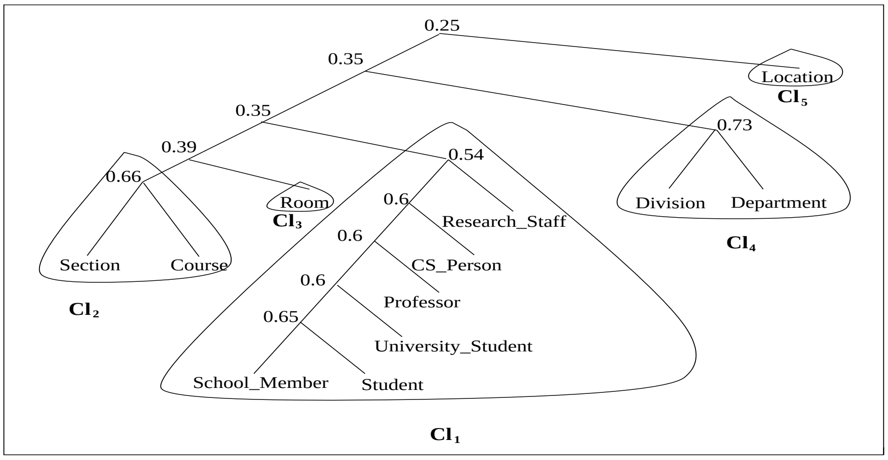

Fig. 3.4. Affinity tree of S_1, S_2, and S_3

University source ($\mathbf{S_1}$)

```
Research_Staff(first_name,last_name,relation,
email,dept_code,section_code)
School_Member(first_name,last_name,faculty,year)
Department(dept_name,dept_code,budget,dept_area)
Section(section_name,section_code,length,room_code)
Room(room_code,seats_number,notes)
```

Computer_Science source ($\mathbf{S_2}$)

```
CS_Person(name)
Professor:CS_Person(title,belongs_to:Division,rank)
Student:CS_Person(year,takes:set⟨Course⟩,rank)
Division(description,address:Location,fund,
sector,employee_nr)
Location(city,street,number,county)
Course(course_name,taught_by:Professor)
```

Tax_Position source ($\mathbf{S_3}$)

```
University_Student(name,student_code,faculty_name,
tax_fee)
```

Table 3.5. Example with three source schemata

Starting from the affinity tree produced with clustering, we define, for each cluster in the tree, a global class $global\_class_i$ representative of the classes contained in the cluster (i.e., a class providing the unified view of all the classes of the cluster). The generation of the $global\_class_i$ is interactive with the designer. Let Cl_i be a cluster in the affinity tree. First, the Global Schema Builder component of MOMIS associates to the $global\_class_i$ a set of global attributes, corresponding to the union of the attributes of the classes belonging to Cl_i, (the attributes with affinity are unified into a unique global attribute in $global\_class_i$). The attribute unification process is performed automatically for what concerns the names of attributes with affinity, according to the following rules:

- for attributes that have name affinity due to SYN relationships, only one term is selected and assigned to the corresponding global attribute in $global\_class_i$;
- for attributes that have name affinity due to BT and NT relationships, a name which is a broader term for all of them is selected and assigned to the corresponding global attribute in $global\_class_i$.

For example, the output of this attribute unification process for cluster Cl_1 of Fig. 3.4, is the following set of global attributes:

```
Cl₁ = (name, rank, title, dept_code, year,
      takes, relation,email, student_code,
      tax_fee, section_code, faculty )
```

```
interface University_Person
(extent Research_Staffers, School_Members, CS_Persons
       Professors, Students, University_Students
 key    name)
{ attribute string name
   mapping_rule (University.Research_Staff.first_name and
                University.Research_Staff.last_name),
                (University.School_Member.first_name and
                University.School_Member.last_name),
                Computer_Science.CS_Person.name,
                Computer_Science.CS_Person.last_name),
                Computer_Science.Professor.name,
                Computer_Science.Professor.last_name),
                Computer_Science.Student.name,
                Computer_Science.Student.last_name),
                Tax_Position.University_Student.name;
  attribute string rank
   mapping_rule University.Research_Staff = 'Professor',
                University.School_Member = 'Student',
   ... }
```

Table 3.6. Example of global class specification in ODL_{I^3}

In general, additional information has to be provided by the designer to complete the global class definition, thus a global class $global\_class_i$ including specification of attribute mappings and default values represented in form of *rules* must be specified in ODL_{I^3}. An example of ODL_{I^3} specification for the global class `University_Person` is shown in Table 3.6. As we can see from this figure, for each attribute, in addition to its declaration, mapping rules are defined, specifying both information on how to map the attribute on the corresponding attributes of the associated cluster and on possible default/null values defined for it on cluster classes. For example, for the global attribute `name`, the mapping rule specifies the attributes that have to be considered in each class of the cluster Cl_1. In this case, an *and* correspondence is defined for `name` for the class `University.Research_Staff` (we use dot notation for specifying the source of a given class belonging to the cluster). A mapping rule is defined for the global attribute `rank` to specify the value to be associated with `rank` for the instances of `University.Research_Staff` and `University.School_Member`.

The global schema of the mediator is composed of the global classes defined for all the clusters of the affinity tree.

3.3.3 Implementation of MOMIS: the Global Schema Builder

In this section, a brief description of the Global Schema Builder module is given, to outline the state of implementation of the MOMIS project. The

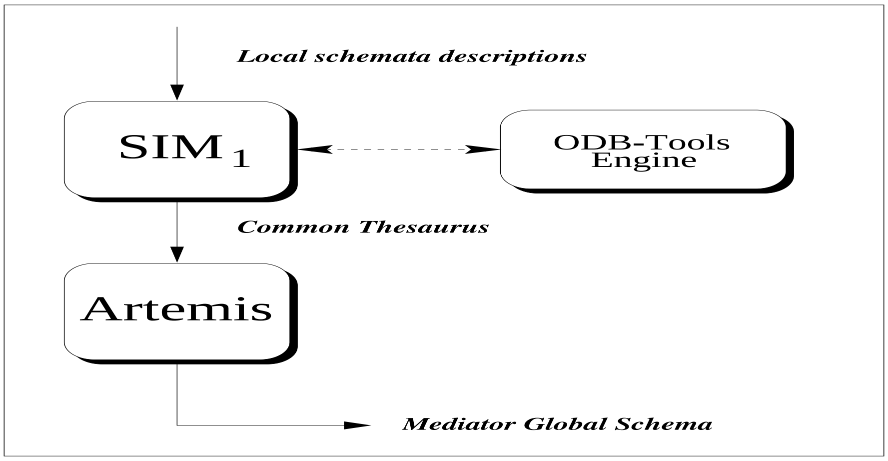

Fig. 3.5. Global Schema Builder

Global Schema Builder processes the local schemata descriptions, received from the information sources, to obtain the Mediator Global Schema, that will be the base for the user's queries. It is composed by the following components (shown in Fig. 3.5):

1. **SIM**$_1$(**S**chemata **I**ntegrator **M**odule, first version): it reads the local schemata descriptions, expressed in the ODL$_{I^3}$ language, to derive the Common Thesaurus. In particular, a set of terminological relationships is stored in the thesaurus, by interacting with the user and by using description logics (supported by ODB-TOOLS) to express the logical links existing intra and inter sources;
2. **Artemis**: starting from the relationships of the Common Thesaurus, Affinity Coefficients are computed between all the pairs of local classes to be integrated, to evaluate their level of similarity. Similar classes are grouped together using *clustering techniques*: every generated cluster will correspond to a mediator global class [108][7].

3.3.4 Description Logics and ODB-TOOLS

Description Logics languages - DLs[8], derived from the KL-ONE model [83, 738], have been proposed in the 80's in the Artificial Intelligence research area. DLs 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*),

[7] This component has been developed at the University of Milano.
[8] *DLs* are also known as *Concept Languages* or *Terminological Logics*.

recently proposed in the database area [2, 3, 4, 27, 60, 372, 373]. CODMs are concerned with only the structural aspects of object-oriented data models proposed for Object-Oriented Databases (*OODB*) [25, 112, 341] 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 the semantics of a database view or a query).

Remark 4. *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.[175]).

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 views) 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 5. *By means of DLs reasoning techniques, 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. In [61] 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.

A description logics (odl = *Object Description Logics*)[9] overcoming the above problems, and a theoretical framework for CODM database design based on subsumption computation and coherence detection has been proposed in [60].

Remark 6. *The* odl *description logics represents the structural part of OODB data models (and of the standard data model* ODM *of ODMG93 [112]) 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). Cyclic classes (base and virtual) are allowed and a *greatest-fixedpoint* semantics [462] has been adopted to uniquely assign an interpretation to cyclic virtual classes.

Furthermore the interpretation of tuples in odl implies an *open world semantics for tuple* types similar to the one adopted by Cardelli [103] and in analogy with all DLs.

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

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

Adopting an *open world semantics for tuple*, it follows that[10]

$$\begin{array}{ll} o_1 \in \mathcal{I}\Big[\triangle[\mathrm{a}\colon \mathtt{String}]\Big], & o_1 \in \mathcal{I}\Big[\triangle[\mathrm{a}\colon \mathtt{String}, \mathrm{b}: \mathtt{Int}]\Big], \\ o_2 \in \mathcal{I}[\triangle\langle\mathtt{Bool}\rangle], & o_{128} \in \mathcal{I}[\{\triangle\top_C\}]. \end{array}$$

Remark 7. *The adoption of an open world semantics for tuple types in* odl *permits an alternative formulation of the OEM capability to express semi-structured objects (see Remark 1): objects of a class share a common minimal structure, but can have further additional and different properties.*

Remark 8. *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.*

---

[9] not to be confused with the homonymous ODL language of ODMG93 [112].

[10] $\triangle$ is an object constructor operator; $\top_C$ is the top class containing any domain object.

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 [44], CLASSIC [76], BACK [401] and [95].

Remark 9. *Coherence checking and subsumption computation can be classified as* semantic query optimization *techniques [117, 343, 614], 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 OODB, see for example the proposed standard *OQL* language [112].

Remark 10. *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. 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 [98].*

A system, called ODB-TOOLS, implementing algorithms for incoherence detection and subsumption computation has been developed at the Dipartimento di Scienze dell'Ingegneria of the University of Modena[11]. ODB-TOOLS includes an extension of `odl`, called `olcd`, allowing to express *quantified path types* and *integrity constraints* rules (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, *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. IC rules, expressed as *if then* rules, whose antecedent and consequent are `olcd`

[11] ODB-TOOLS is available on Internet at the following address: (http://sparc20.dsi.unimo.it/).

virtual types, allow the declarative formulation of a relevant set of integrity constraints [53, 54].

Following the key idea of *semantics expansion* of a type in [53] integrity constraints can be used to optimize queries. The semantics expansion of a query is obtained by iterating the following transformations: if a query *implies* the antecedent of an integrity rule then the consequent of that rule can be intersected with the query; subsumption computation is used to compute logical implication. In this way new "isa" relationships can be found and it is possible *to move the query down* in a schema hierarchy.

Remark 11. *By using a description logics including* if then *integrity rules it is possible to perform semantic query optimization with respect to usual OODB schemata including only base classes.*

ODB-TOOLS is composed of two modules: ODB-DESIGNER [37] and ODB-QOPTIMIZER. ODB-DESIGNER is an active tool which provides an ODL (ODMG93) standard interface and supports the design of an OODB schema preserving coherence and minimality with respect to inheritance. It implements the theoretical framework of [60]. ODB-QOPTIMIZER performs semantic optimization of OODB queries [53]; it provides an OQL (ODMG93) standard interface.

3.3.5 On the Role of ODB-TOOLS in the MOMIS system

The key idea at the basis of the MOMIS project was that starting from a system based on description logics as ODB-TOOLS, following the "semantic approach" and some interesting features of TSIMMIS, it is easy to develop a powerful mediator of an I^3 system. In fact:

1. the standard ODM-ODMG model and ODL-ODMG language can 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 OQL-ODMG language such that it will accept queries for relational databases is adopted;
5. `olcd` is extended to support QDTL translation.

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-ODMG language;
- describe query templates in the *minimal core language*;
- describe the mediator rules in ODL_{I^3}.

Having ODB-TOOLS available, the knowledge expressed in the standard languages above is automatically translated into `olcd` classes and virtual classes and the the `olcd` 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 7, 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 ODB-Designer can be used as a powerful query optimizer for OQL queries.

3.4 Discussion and Final Remarks

In this chapter, we have presented two approaches to schema integration of heterogeneous information sources. The first one, the "structural approach", has been illustrated by means of the TSIMMIS system. The second one, the "semantic approach", has been presented by means of the MOMIS system. It is based on a description logics component (ODB-Tools) and a cluster generator module (ARTEMIS) together with an ODL_{I^3} interface module. In this way, generation of the global schema for the mediator is a semi-automated process.

It is the authors' opinion that *the schema-less* assumption of the structural approach leads to two major drawbacks w.r.t. the semantic approach:

- inefficient retrieval of data to be integrated;
- incapability to answer to not–predefined queries.

On the contrary, the use of a schema permits to support every possible user query on the schema instead of a predefined subset of them. For this reason, they strongly believe that the semantic approach is more promising.

The authors are conscious that, for lack of space, the above presentation is incomplete with respect to many topics. Among the more relevant, let us mention:

- query decomposition and optimization;
- object fusion in mediator system;
- integration of semi–structured data.

On the other hand, the authors think that the contents of the chapter includes topics sufficiently investigated by the research community whereas other topics, as the last mentioned, need more research efforts.

Acknowledgements

This research has been partially funded by the MURST 40% 97 Italian Project: '*INTERDATA*'.

We would like to thank S. Castano and S.De Capitani di Vimercati of the Università di Milano who developed the ARTEMIS component of the MOMIS system, and S. Montanari and M.Vincini of the Università di Modena who contributed to the project and development of the Global Schema Builder.

A special thank to the students of the "Laurea in Ingegneria Informatica" of the Università di Modena contributing in the software development of the MOMIS project.