D2I

Integrazione, Warehousing e Mining di sorgenti eterogenee
Programma di ricerca (cofinanziato dal MURST, esercizio 2000)

Description of the methodology for the integration of strongly heterogeneous sources

S. Bergamaschi, S. Castano, A. Ferrara, F. Grandi, F. Guerra, G. Ornetti, M. Vincini

D1.R6 **28.01.2002**

Sommario

In modern Internet-based and Intranet-based architectures, an increasing number of applications requires an integrated and uniform access to a multitude of heterogeneous and distributed datasources. In this report, we describe a comprehensive methodological framework, called *D2I-METH*, for the integration of strongly heterogeneous datasources. *D2I-METH* puts into a coherent whole the set of approaches and techniques developed as the first step of the *D2I* project and described in the D2I report D1.R1. *D2I-METH* provides guidance for integrating datasources presenting different types and levels of heterogeneity. It is based on a set of intelligent techniques and tools for automating as much as possible various activities involved in the integration process, such as for example the acquisition of interschema properties, the schema matching and reconciliation activities, and the generation of semantic mappings between the global schema and the local schemas of the involved datasources.

Tema	Tema 1: Applicazioni per basi di dati su Internet e Intranet
Codice	D1.R6
Data	28.01.2002
Tipo di prodotto	Rapporto tecnico
Numero di pagine	37
Unità responsabile	MI
Unità coinvolte	BO, MI, MO
Autore da contattare	S. Castano Dipartimento Scienze dell'Informazione Università degli Studi di Milano Via Comelico 39, 20135 Milano e-mail: castano@dsi.unimi.it

Description of the methodology for the integration of strongly heterogeneous sources

S. Bergamaschi, S. Castano, A. Ferrara, F. Grandi,
F. Guerra, G. Ornetti, M. Vincini

28.01.2002

Abstract

In modern Internet-based and Intranet-based architectures, an increasing number of applications requires an integrated and uniform access to a multitude of heterogeneous and distributed datasources. In this report, we describe a comprehensive methodological framework, called *D2I-METH*, for the integration of strongly heterogeneous datasources. *D2I-METH* puts into a coherent whole the set of approaches and techniques developed as the first step of the *D2I* project and described in the D2I report D1.R1. *D2I-METH* provides guidance for integrating datasources presenting different types and levels of heterogeneity. It is based on a set of intelligent techniques and tools for automating as much as possible various activities involved in the integration process, such as for example the acquisition of interschema properties, the schema matching and reconciliation activities, and the generation of semantic mappings between the global schema and the local schemas of the involved datasources.

1 Introduction

In modern Internet-based and Intranet-based architectures, an increasing number of applications requires an integrated and uniform access to a multitude of heterogeneous and distributed datasources. In fact, many organizations have to face the data integration problem for building warehousing, mining, Enterprise Resource Planning, as well as Web-based information system solutions. Such kind of integration projects generally require to consider a (possibly very) high number of (possibly very) large datasources, where stored data can change over time (possibly very) frequently, especially in the Web context. As a matter of fact, datasources that need to be integrated are generally highly heterogeneous, either for using different data models and structuring principles, or for the kind and nature of information contents, or for the kind of support technology for data management. A correct integration is essential for effectively and correctly exploit data available at the sources for query purposes. Integration of highly heterogeneous datasources is a very complex and time-consuming activity, and comprehensive methodological frameworks and associated tools are strongly required. A methodological framework for the integration of highly heterogeneous datasources should be based on guidance and support tools for dealing with different types of heterogeneity that can occur among the sources, and should provide intelligent techniques and tools for automating as much as possible various activities involved in the integration process, such as for example the acquisition of interschema properties, the schema matching and reconciliation activities, or the generation of semantic mappings between the global schema and the local schemas of the underlying datasources. A first type of heterogeneity to be considered when integrating strongly heterogeneous sources derives from the level of structuring of data to be integrated. Several data models and languages can be adopted for data representation and storage, ranging from non structured data, typical of file systems, to highly structured data, typical of database systems, to semistructured data, typical of Web datasources [1, 26, 40]. Another type of heterogeneity to be considered derives from the terminology, structure, and context characterizing data stored at each source, which requires to take into account data semantics and to define semantic mappings between data. These problems have been first addressed in the context of database integration [29], and more recently in the context of Web and XML data integration [32, 16, 20].

In this report, we describe a comprehensive methodological framework, called *D2I-METH*, for the integration of strongly heterogeneous datasources. *D2I-METH* puts into a coherent whole the set of approaches and techniques developed as the first step of the *D2I* project (described in the D2I report D1.R1 [22]) by the four different partners involved in the integration task of *D2I*, namely the Univ. of Calabria, the Univ. of Milano, the Univ. of Modena and Reggio Emilia, and the Univ. of Rome. The first approach is based on graphs and extends to semistructured data, the techniques developed in the DIKE system [2, 35, 34]. The second approach is object-oriented and extends to semistructured data the techniques developed in the MOMIS system [6, 4], and in the ARTEMIS tool environment [12]. Finally, the third approach is based on a two level architecture, in which a logical representation of sources in terms of the relational data model is mapped to a conceptual representation of the domain of interest, given in terms of the expressive Description Logic $\mathcal{DLR}$ [9, 10]. *D2I-METH* supports a virtual approach to integration, in that data residing at the sources are accessed during query processing by exploiting defined mappings, instead of being replicated at the global level. The methodology supports a Global-as-View (GAV) approach, to build a global schema which is expressed in terms of the datasources. Based on the common features of these approaches, main features of *D2I-METH* can be summarized as follows:

- Semantically rich representation of involved datasources through a common conceptual model which is exploited for deriving and representing the semantics of each involved datasource.
- Semiautomatic extraction of interschema properties relating schema concepts of datasources at the conceptual level (including "encapsulation" of schema versioning support by datasources). Since the number and the dimension of datasources are great and since the information they store change quite frequently over time, manual extraction of interschema properties is expensive and difficult.
- Use of an integrated and unified representation of the involved datasources, to be used for querying the integration system, and explicit representation of the mapping between the data at the sources and the integrated representation;
- Use of formalization and reasoning capabilities of Description Logics in both semantic integration and view-based query processing.

The global schema and associated mappings obtained by applying such a methodology provide support for expressing queries in terms of global views over underlying datasources, with mechanisms for the reformulation and/or answering of such queries in terms of the data stored in the sources. Such a problem is known in the literature as view-based query processing, and has been studied very actively in the recent years [41, 31, 11]. In D2I, such aspects are addressed in [18].

The paper is organized as follows. Section 2 provides an overview of the *D2I-METH* methodological framework and describes its instantiation to the ARTEMIS/MOMIS techniques. Section 3 describes wrapping of source schemas in MOMIS. Section 4 describes the extraction and representation of intra and inter schema properties, while Section 5 illustrates the techniques for matching source schemas. Section 6 presents the generation of a global integrated representation, and Section 7 describes the definition of mappings. In Section 8 the integration techniques based on temporal and schema versioning information are presented. Concluding remarks are given in section 9.

2 Overview of the proposed integration methodological framework

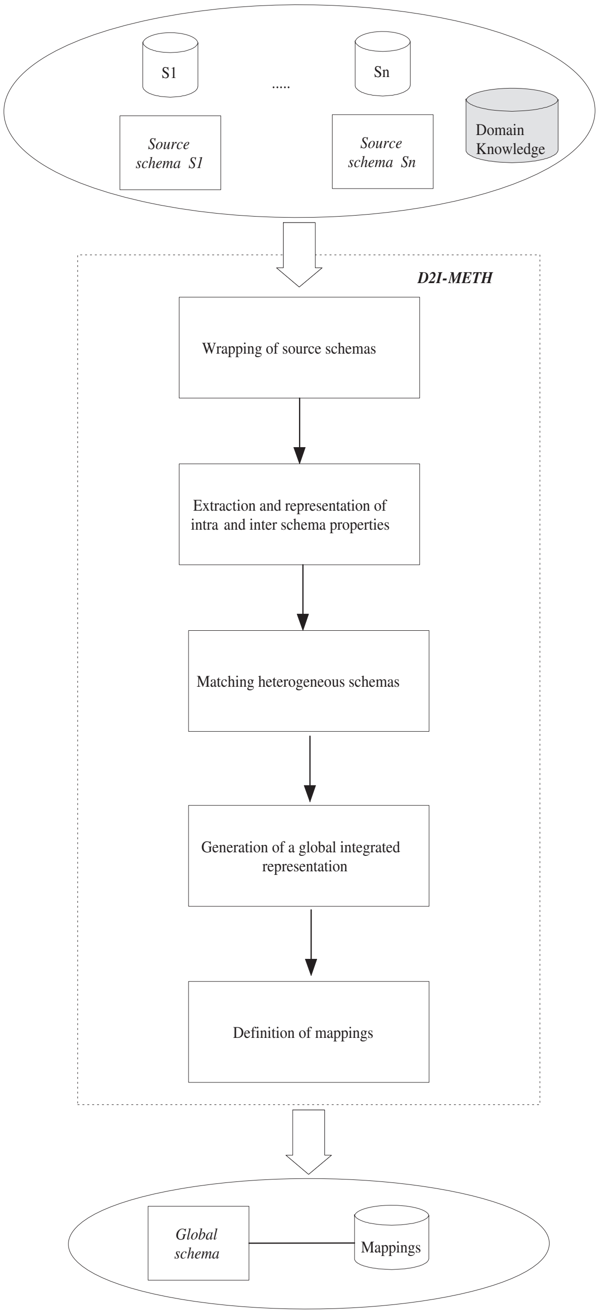

Figure 1: The D2I-METH methodological framework for the integration of highly heterogeneous data sources.

In this section, we describe the *D2I-METH* methodological framework for the integration of strongly heterogeneous datasources. The general structure of *D2I-METH* is illustrated in Figure 1. In the figure, a box represents an activity (i.e., a phase of the methodology), possibly further decomposed into sub-activities, and a link between two activities represents both an order relationship between their execution and a flow of information. *D2I-METH* generalize the approaches developed in DIKE, MOMIS/ARTEMIS, and $\mathcal{DLR}$, each of them including a variety

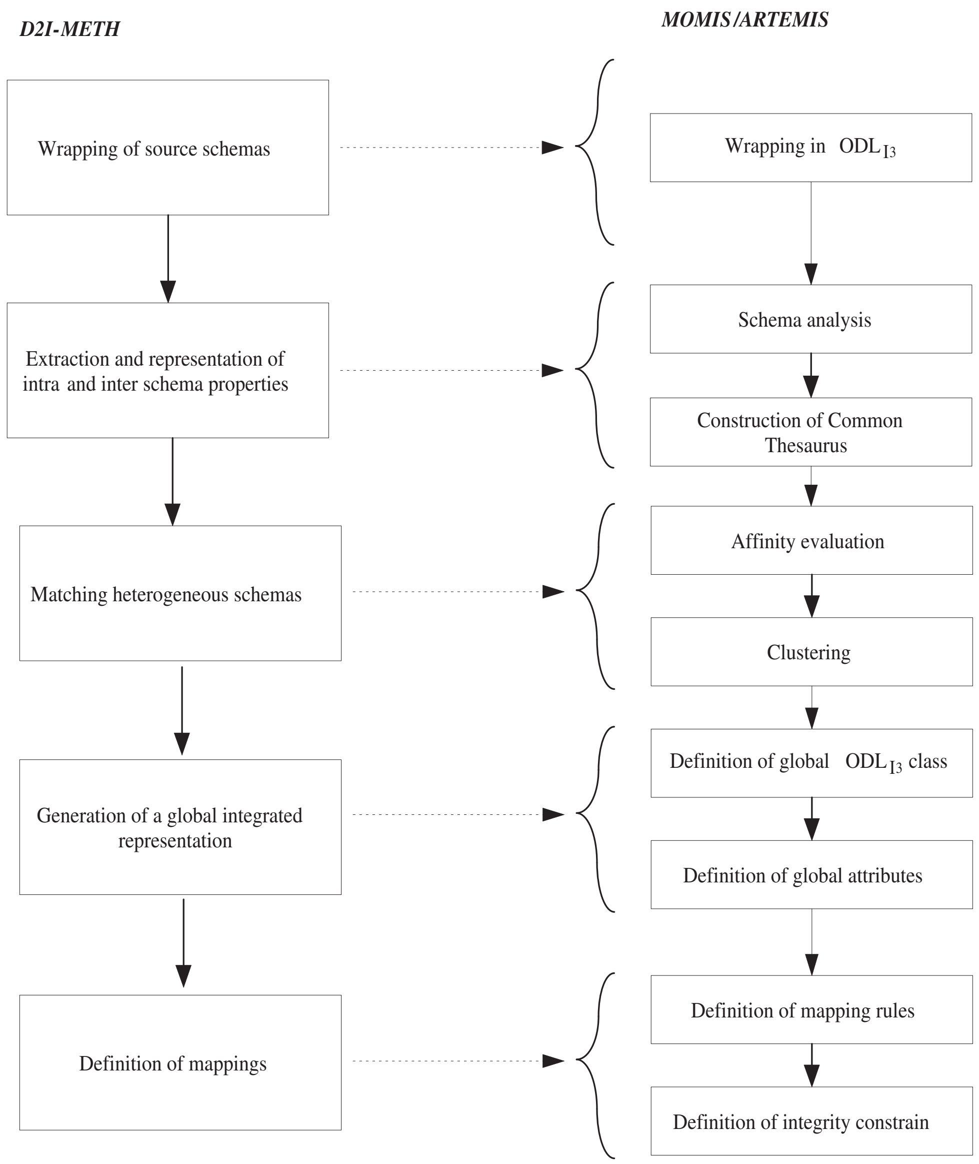

Figure 2: Instantiation of the D2I-METH methodological framework to the MOMIS/ARTEMIS approach.

of techniques to support semantic integration as well as query processing over the integrated schema. The MOMIS and the DIKE systems adopt the *global-as-view* (GAV) approach, and provide techniques for building a global schema and for defining mappings in which each global concept is defined in terms of the concepts of the source schemas, which are represented in a conceptual data model. The $\mathcal{DLR}$ system adopts the *local-as-view* (LAV) approach, in which the source schemas, expressed in the relational model, are defined in terms of the global elements. All the three approaches use a global schema expressed in a conceptual data model to provide a unified and reconciled view of the information at the sources and which can be queried by the user. *D2I-METH* is articulated in the following phases.

Wrapping of source schemas. This phase has the goal of constructing a semantically rich representation of the datasources to be integrated, by means of a common data model. In semantic approaches to integration, this task is performed by wrapper tools developed for each kind of data representation, by considering the schema of the source. For conventional structured information sources (e.g., relational databases, object-oriented databases), schema description is always available and can be directly translated into the selected common data model using wrapper tools [6]. For semistructured information sources schema description is generally not directly available in the sources. In fact, a basic characteristic of semistructured data is that they are "self-describing". This means that the information generally associated with the schema is specified directly within data [7]. For XML datasources, schema information can be given in form of DTDs or XMLSchemas [13]. In D2I, wrapping techniques have been developed both in DIKE, MOMIS and ARTEMIS for converting datasources at different level of structuring (e.g., databases, XML datasources) into selected common data models used for integration [30]. In Section 3, we describe the wrapping process in MOMIS.

Extraction and representation of intra and interschema properties. The objective of this phase is the identification of the schema properties featuring the sources to be integrated. Both intraschema and interschema properties are of interest. The techniques described in [22] support a semi-automatic derivation of such properties, by analyzing structure and context of elements in source schemas, using Description Logics. For example, MOMIS/ARTEMIS and DIKE provide techniques for the specification and/or semi-automatic extraction of interschema properties among concepts/subschemas of the source schema, while $\mathcal{DLR}$ allows the specification of interschema properties among the source elements at the conceptual level. In the presence of datasources supporting schema versioning, useful inter-schema-version properties can also be derived in this phase, as described in Sec. 4.

Matching of source schemas. The goal of the schema matching phase is to define semantic correspondences between schema elements of different datasources for subsequent integration [32]. D2I provides schema matching techniques based on interschema properties and affinity correspondences in both MOMIS/ARTEMIS and DIKE, to identify in a semiautomated way schema elements candidate to integration.

Generation of a global integrated representation. The goal of this phase is to construct a global integrated representation (also called mediator schema or global schema) for the considered datasources. The global schema generated in this phase is virtual, in that data residing at the sources are not replicated at the global level, but are accessed during query processing. The global schema provides an appropriate abstraction of all the data residing at the sources. One aspect deserving special attention is the choice of the language used to express the global schema. In D2I, we have different techniques for the construction of an integrated representation of heterogeneous datasources, as described in [22]. Such techniques provide different languages for expressing the global schema. Moreover, such techniques support the construction of a global schema based on the results of schema matching, exploiting also knowledge provided by interschema properties holding between matching elements.

Definition of mappings. Mappings express the reciprocal correspondences between the global schema and the schemas of the sources. In mapping definition, main problems to be faced are related to the resolution of heterogeneities between source data and the corresponding global element in the global schema. In D2I, we have techniques for semi-automatically defining mapping rules, integrity constraints, and conversion functions for a global schema obtained according to the GAV approach. In this case, mappings specify the rules for mapping an element of the global schema to the corresponding elements in the source schemas in a way that data residing at the sources can be correctly retrieved during query processing.

UNIVERSITY source (*UNI*)

```
Research_Staff(name,e_mail,dept_code,s_code)
School_Member(name,school,year,e_mail)
Department(dept_name,dept_code,budget)
Section(section_name,s_code,length,room_code)
Room(room_code,seats_number,notes)
```

COMPUTER_SCIENCE source (*CS*)

```
CS_Person(first_name,last_name)
Professor:CS_Person(belongs_to:Division,rank)
Student:CS_Person(year,takes:set⟨Course⟩,rank,e_mail)
Division(description,address:Location)
Location(city,street,number,country)
Course(course_name,taught_by:Professor)
```

TAX_POSITION_XML source (*TP*)

```
<!ELEMENT ListOfStudent (Student*)>
<!ELEMENT Student (name, s_code, school_name, e_mail, tax_fee)>
<!ELEMENT name (#PCDATA)>
...
```

Figure 3: Three heterogeneous University Sources

> In D2I we support also a lacal-as-view (LAV) approach. In this case, starting from a given global schema, mappings are defined for expressing the source schemas in terms of the concepts of the the global schema. Techniques for using mappings between a global schema and source schemas according to the LAV approach in query rewriting have been developed in the framework of the $\mathcal{DLR}$ approach of D2I [22, 18].

In the following, we describe the instantiation of the *D2I-METH* framework based on the use of the techniques developed in the framework of the MOMIS/ARTEMIS approach.

2.1 Instantiation of the methodology and running example

In Figure 2 we show the instantiation of the D2I-METH methodological framework to the MOMIS/ARTEMIS approach. The various phases of the methodology instantiated to MOMIS/ ARTEMIS will be described in the remaining sections of this report. In particular, we illustrate the wrapping techniques developed in MOMIS and the recent extensions of the ARTEMIS schema matching techniques described in [22] with extensional interschema properties. Moreover, we describe techniques based on the use of temporal attributes and schema versioning information for enriching the interschema knowledge available in MOMIS/ARTEMIS for integration.

As a running example, we consider three sources with different data models (see Figure 3).

The first source is a relational database, `University` (UNI), containing data about the staff and the students of a given university. The relations are: `Research_Staff`, `School_Member`, `Department`, `Section` and `Room`. For a given professor (in `Research_Staff`) his department (`dept_code`) and his section (`s_code`) are stored. In the relation `School_Member` the information `name`, `year`, `e_mail`, and `school` about students enrolled at the university are stored.

The second source `Computer_Science` (CS) is an object–oriented database containing information about people belonging to the Computer Science department of the same university, and is an object-oriented database. There are six classes: `CS_Person`, `Professor`, `Student`, `Division`, `Location` and `Course`. Information is quite similar to the first source: it stores data on professors and students, also giving the possibility to retrieve the division of a given professor. This division may be part of another department, being a logical specialization of `Department`. The class `Location` maintains the division address. With respect to students, we may know the courses they take and their enrollment year.

A third source is also considered, `Tax_Position` (TP), derived from the Registry Office. It consists of an XML file, storing information about student's tax_fees.

3 Wrapping of source schemas

In this section, we describe wrapping techniques developed in MOMIS.

The main goal of an information mediator system is to permit an integrated information accessibility from heterogeneous information sources, located in different sites and stored in different architectural platform [36]. During this integration process many problems coming from structural and implementation heterogeneity (including for example diffences in hardware platforms, DBMS, data languages) and from the semantic heterogeneity, when different names are employed to represent the same information (naming conflicts) or when different modeling constructs are used in different sources to represent the same kind of information.

To manage the implementation heterogeneity, a mediator system typically encapsules each sources by a *wrapper*, that logically converts the underlying data structure to a common information model. Therefore the wrapper architecture and interfaces are crucial, because wrappers are the focal point for managing the diversity of data sources [39].

In particular, the main tasks of a wrapper component are:

- during the integration process, the wrapper translates the schema of the source into the common data model of the mediator;
- during the query processing process, the wrapper converts queries posed over the common data model into requests suitable for the source, and it converts data returned by the source into the common data model.

For conventional structured information sources (e.g. relational databases, object oriented databases), schema description is always available and can be directly translated into the selected common data model. For example, for flat files and object oriented databases wrappers perform a syntactic translation, while for the relational databases the translation is based on *transformation rule-sets,* as described in [23] for relational to ODMG schema conversion. For semistructured information sources (e.g., Web data sources), schema description is generally not directly available in the sources. In fact, a basic characteristic of semistructured data is that they are "self-describing". This means that the information generally associated with the schema is specified directly within data [7].

In this section we describe the wrapper strategy and implementation within the MOMIS system. In MOMIS, we represent source schemas by using an object-oriented language, called ODM_{I^3}. According to the recommendations of ODMG and to the diffusion of I^3/POB [28, 15], the object data model ODM_{I^3} is very close to the ODL language. ODM_{I^3} is a source independent language used for information extraction to describe heterogeneous schemas of structured and semistructured data sources in a common way. A wrapper translates the schema representation of the source into ODM_{I^3} descriptions

We developed two kind of wrappers: the former is a wrapper to manage relational databases, founded on JDBC technology [24] to connect and to query the sources. The latter is a wrapper to manage semistructured data files. In this way we are also able to integrate XML sources. We are trying to improve the capabilities of our wrapper in order to manage HTML files, and other XML based languages.

3.1 Wrapping relational sources

Within the MOMIS system, the wrapper that manages relational databases is implemented with the JDBC technology and lets the mediator system access virtually any relational data source by using the interfaces provided by the Java programming language. In fact, this technology permits the use of the standard interface SQL to query a wide range of relational databases. In this way, the schema structure of the source is available from a standard language.

The translation into ODM_{I^3}, on the basis of the ODM_{I^3} syntax (see Appendix B) and of the schema definition is performed by the wrapper as follows. Given a relation of a relational source, translation involves the following steps:

i) an ODM_{I^3} class name corresponds to the relation name

ii) for each relation attribute an attribute is defined in the corresponding ODM_{I^3} class.

Furthermore, attribute domains are extracted.

Our approach aims to obtain source schemas by adopting standard technologies. Nevertheless each wrapper has to be customized in relation of the specific DBMS the mediator system needs to be connected. In particular, different DBMS platforms do not store information about primary and foreign key in a standard way and using standard catalogs. For this reason, we have developed one customized wrapper for each database platform we want to interact to.

3.2 Wrapping semistructured sources

Research on semistructured data aims at extending database management techniques to data with irregular, unknown or often changing structure. To integrate this sources, two issues are to be faced: the former is an automatic way to derive and explicitly represent the schema of the source. The latter is to communicate to the mediator when and how a source was modified, and to develop methodologies to modify automatically the global schema.

Moreover, in order to integrate semistructured information sources, we have to take into account that several data models represent this kind of source have been proposed in literature. In particular the OEM model (Object Exchange Model) may be thought as the de facto model to represent semistructured data. It was originally introduced for the Tsimmis data integration project [37]. In [33, 8] they are proposed methodologies to extract structures from semistructured data files. But, in order to query this kind of sources, you have to consider that no standard languages to query semistructured information sources were developed. The work to realize a language to perform queries over XML files is developing at the Word Wide Web Consortium (now only working drafts have been issued).

According to the models proposed in literature for semistructured information sources [7, 42], MOMIS represents semistructured information sources as rooted, labeled graph with the semistructured data (e.g., an image or free-form text) as nodes and labels on edges.

A semistructured object (object, for short) can be viewed as a triple of the form $\langle \mathtt{id}, \mathtt{label}, \mathtt{value}\rangle$, where `id` is the object identifier, `label` is a string describing what the object represents, and `value` is the value, that can be atomic or complex. The atomic value can be integer, real, string, image, while the complex value is a set of semistructured objects, that is, a set of pairs (*id,label*). A complex object can be thought as the parent of all the objects that form its value (children objects). A given object can have one or more parents. We denote the fact that an object so' is a child object of another object so by $so \to so'$ and use notation $label(so)$ to denote the label of so. In semistructured data models, labels are descriptive as much as possible. Generally, the same label is assigned to all objects describing the same concept in a given source. To represent the schema of a semistructured source S, we introduce the notion of *object pattern*. All objects so of S are partitioned into disjoint sets, denoted set_l, such that all objects belonging to the same set have the same label l. An object pattern is then extracted from each set to represent all the objects in the set. Formally, an object pattern is defined as follows.

[Object pattern] Let set_l be a set of objects in a semistructured source S having the same label l. The *object pattern* of set_l is a pair of the form $\langle$l,A$\rangle$, where l is the label of the objects belonging to set_l, and $A = \bigcup \text{label}(so')$ such that there exists at least one object $so \in \text{set}_l$ with $so \to so'$.

From this definition, an object pattern is representative of all different objects that describe the same concept in a given semistructured source. In particular, l denotes the concept and set A the properties (or attributes) characterizing the concept in the source. Since semistructured objects can be heterogeneous, labels in A can be defined only for some of the objects in set_l, but not for all. We call such kind of labels "optional" and denote them with symbol "?".

An object pattern description follows an *open world semantics* typical of the Description Logics approach [43]. Objects conforming to a pattern share a common minimal structure represented by non optional properties, but can have further additional (i.e., optional) properties. In this way, objects in a semistructured data source can evolve and add new properties, but they will be retrieved as valid instances of the corresponding object pattern when processing a query.

According to this data model we developed a wrapper to manage XML files. This wrapper aims to map a data model of XML file into the corresponding OEM model. This wrapper could be thought as the core of further extensions that aim to manage XML based files as RDF files or XHTML files.

3.2.1 A wrapper for XML files

The XML language [17] is a W3C recommendation and arises as a language to describe information sources by using an universal format. One of the main goals of this standard is to exchange files across the Internet. Contrary to HTML, that aims to exchange files in a "human-readable" way , XML language aims to exchange files across the Internet in a "machine-readable" way. In comparison with HTML, the fashion data-oriented of XML it is synthesizeable as follows [40]:

- The user may define personal tag names at will.
- The structure of the XML file may include tags nested to any level.
- A XML file may include a description of its structure for use by applications that need to perform structural validation. This definition is called DTD (Document Type Definition).

In this way a XML file may be thought as self-describing like a semistructured data source. The main analogies may be summarized as follows:

- OEM attribute $\longrightarrow$ XML tag
- OEM object $\longrightarrow$ XML element
- atomic value of an OEM attribute (string, real, image, ...) $\longrightarrow$ PCDATA value

By using this mapping it is possible to use the XML data model to describe semistructured information sources. This mapping implies some critical aspects that have to be analyzed [40].

- **the order**: strictly, a XML file may be thought as representing an ordered data model. An element in a DTD row is described as containing other elements with a prefixed order. The concept of order is normally not applied in semistructured data file. To foresee an element order management could be useful to obtain specific information during the query process. Actually our wrapper describes data in an unordered way.
- **the attributes**: The XML tags may contain attributes (called ATTLIST) which are semantically w.r.t. OEM attributes. For example the following extract of XML file uses an XML attribute:

  ```
  <Student country="Italy">
  <name>Philip</name>
  </Student>
  ```

Each tag may contain several couples attribute-value. Our wrapper translates this structure into an OEM compliant structure without lack of semantics. This translation generates a new tag for each couple attribute-value as follows:

```
<Student>
<name>Philip</name>
<country>Italy</country>
</Student>
```

- **the references/links**: Our wrapper does not support link inter files. Xlink, the W3C recommendation which allows elements to be inserted into XML documents in order to create and describe links between resources, is not supported in our software. Nevertheless ID-IDREF(S) couples, defined in XML language to express references intra-schema, are supported. In order to manage this kind of information our wrappers have to interact with the user. In fact, according to the XML syntax, we do not have to explicitly express which tag an IDREF attribute refers to. To avoid its loss during the translation process, the user is asked for some further information by a graphical interface.

3.2.2 Example of translation rule from XML to ODM$_{I^3}$

In the following we give the list of the translation rules from a data source defined by a DTD within a XML file into the common data model expressed in ODM$_{I^3}$.

The translation process consists in the parsing of the XML file in order to build the tree structure associated. Starting from the obtained tree, the wrapper translates each XML tag by following an event-driven process.

Element translation

The XML1.0 language specification provides four different types of nodes: children, mixed, any, empty. In the following we divide the mixed category in two section: the former including only the PCDATA value and the latter the other mixed data nodes. In this way, in analogy with XML-Schema, elements of a XML tree may be divided in two blocks: the first one including only **simple** element, i.e. the elements that represent the leafs of the tree (PCDATA and empty); the second one that represent the **complex** element, i.e. intermediate nodes or terminal nodes with attributes (mixed and children). An any tag may be contained into either the categories.

Later in this section we give the translation-rules by examples of translation both of elements and of attributes:

- Simple type elements

DTD	ODM$_{I^3}$

`<!ELEMENT element1 (#PCDATA)>`
`<!ELEMENT element2 EMPTY>`
`<!ELEMENT element3 ANY>`
	`attribute string element1;`
	`const string element2="" ;`
	`attribute string element3;`

- Complex elements:

  - Children type elements with a simple type child

DTD	ODM$_{I^3}$

<!ELEMENT elem1 (cspec1,cspec2)>
<!ELEMENT cspec1 (#PCDATA)>
<!ELEMENT cspec2 (#PCDATA)>
	Interface elem1
	(...)
	{
	attribute string cspec1;
	attribute string cspec2;
	};

- Mixed type elements with a simple type child

DTD	ODM$_{I^3}$

<!ELEMENT elem1 (PCDATA|cspec1)>
<!ELEMENT cspec1 (#PCDATA)>
	Interface elem1
	(...)
	{
	attribute string PCDATA_NODE;
	};
	union elem 11
	{
	attribute string cspec1;
	};

- Children type elements with a complex type child

DTD	ODM$_{I^3}$

<!ELEMENT elem1 (csp1)>
<!ELEMENT csp1 (child1,child2)>
	Interface elem1
	(...)
	{
	attribute csp1 csp1;
	};

Instances management

In a DTD specification, three keywords are used to manage the instances: '*' to indicate 0-n allowed instances, '+' to indicate 1-n allowed instances, and '?' to indicate the optional element. If none of this keywords are present, then a single, mandatory instance of element are due.

In the ODM$_{I^3}$ language does not exist an operator containing the same semantics of '+' operator. For this reason we translate '*' and '+' operators with the same rule without lost of semantics.

DTD	ODM$_{I^3}$

<!ELEMENT elem1 (cs1*,cs2+,cs3?,cs4)>
<!ELEMENT cs1 (#PCDATA)>
<!ELEMENT cs2 (#PCDATA)>
<!ELEMENT cs3 (#PCDATA)>
<!ELEMENT cs4 (#PCDATA)>
	Interface elem1
	(...)
	{
	attribute set<string> cs1;
	attribute set<string> cs2;
	attribute string cs3?;
	attribute string cs4;
	};

Translation of attributes

In order to translate the attributes, it can be considered both the type of attribute (CDATA, ID, IDREF,...), and the default declaration.

- Translation of CDATA attributes.

DTD	ODM$_{I^3}$

<!ELEMENT elem1 (cs1,cs2)>
<!ATTLIST elem1 a1 CDATA #REQUIRED>
<!ELEMENT cs1 (#PCDATA)>
<!ATTLIST cs1 acs1 CDATA #REQUIRED>
<!ELEMENT cs2 (chs1,chs2)>
<!ATTLIST cs2 acs2 CDATA #REQUIRED>
	Interface elem1
	(...)
	{
	attribute cs1 cs1;
	attribute cs2 cs2;
	attribute string elem1_a1;
	};
	
	Interface cs1
	(...)
	{
	attribute string PCDATA_NODE;
	attribute string cs1_acs1;
	};

- Translation of different attribute types

DTD	ODM$_{I^3}$
<!ELEMENT el1 (cs1)> <!ATTLIST el1 atl1 CDATA #REQUIRED> atl2 ID #REQUIRED> atl3 IDREF #REQUIRED> atl4 ENTITY #REQUIRED> atl5 NMTOKEN #REQUIRED> atl6 (one|two) "one"> <!ELEMENT cs1 (#PCDATA)>	Interface el1 (...) { attribute string cs1; attribute string el1_atl1; attribute string el1_atl2; attribute string el1_atl3; attribute string el1_atl4; attribute string el1_atl5; attribute enum {one.two} el1_atl6;};

4 Extraction and representation of intra and interschema properties

In MOMIS/ARTEMIS, interschema knowledge is expressed through *intensional properties* and *extensional properties*.

Intensional properties are *terminological relationships* expressing inter-schema knowledge at the intensional level for the source schemas. Intensional relationships are defined between classes and attributes, and are specified by considering class/attribute names, called terms. The following relationships can be specified in ODM$_{I^3}$:

- SYN (Synonym-of), defined between two terms t_i and t_j, with $t_i \neq t_j$, that are considered synonyms in every considered source (i.e., t_i and t_j can be indifferently used in every source to denote a certain concept). An example of SYN relationship in our example is ⟨`CS.Course` SYN `UNI.Section`⟩.

- 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. BT relationship is not symmetric. The opposite of BT is NT (Narrower Terms), or hyponymy. For instance, a NT relationship in our example is ⟨`UNI.School_Member` NT `CS.CS_Person`⟩.

- RT (Related Terms), or positive association, defined between two terms t_i and t_j that are generally used together in the same context in the considered sources. For example, we can have the following relationship ⟨`CS.Course` RT `UNI.School_Member`⟩.

An intensional relationships is only a terminological relationship, with no implications on the extension/compatibility of the structure (domain) of the two involved classes (attributes). For capturing this kind of knowledge, estensional properties are used.

Extensional properties express interschema knowledge at the estensional level.

The intensional relationships SYN, BT, NT and RT between two classes C_1 and C_2 may be "strengthened" by establishing that they are also *extensional* relationships [14]. Consequently, the following extensional relationships can be defined in ODM$_{I^3}$:

- C_1 SYN$_{ext}$ C_2: the instances of C_1 and C_2 are the same.

- C_1 BT$_{ext}$ C_2: the instances of C_1 are a superset of the instances of C_2.

- C_1 NT$_{ext}$ C_2: the instances of C_1 are a subset of the instances of C_2.

- C_1 RT$_{ext}$ C_2: the instances of C_1 overlap the instances of C_2.

- C_1 DISJ$_{ext}$ C_2: the instances of C_1 are disjoint from the instances of C_2.

Moreover, extensional relationships "constrain" the structure of the two classes C_1 and C_2. If an extensional relationship C_1 NT$_{ext}$ C_2 is issued, we have that:

- strict inheritance between C_1 and C_2 is enforced for the common attributes;

- both C_1 and C_2 may have further attributes as we adopt usual description logics semantics (i.e. open world semantics).

Intensional and extensional properties can be partially automatically extracted and partially explicitly declared by the integration designer. A *Common Thesaurus* of terminological and extensional relationships is constructed, describing interschema knowledge about ODL$_{I^3}$ classes and attributes of source schemas.

The Common Thesaurus is built through an incremental process during which relationships are added in the following order: *i) schema-derived relationships*: intensional and extensional relationships holding at intraschema level are extracted by analyzing each ODL$_{I^3}$ schema separately; *ii) lexical-derived relationships*: intensional relationships holding at interschema level are extracted by analyzing different sources ODL$_{I^3}$ schemas together according to the Wordnet supplied ontology; *iii) designer-supplied relationships*: additional intensional and extensional relationships are supplied directly by the designer, to capture domain knowledge about the source schemas. Supplied relationships are validated with ODB-Tools [5]; *iv) inferred relationships*: a new set of relationships (i.e. new generalization and aggregation properties) is inferred by ODB-Tools by reasoning over the union of the local schemas enriched with currently available interscheme properties. With reference to our running example, the Common Thesaurus contains the terminological relationships reported in Appendix A. Moreover,the Common Thesaurus contains the following extensional relationships:

Table 1: Affinity coefficients definition summary

Coefficient	Value	Description
$NA(c,c')$	$A(n_c, n_{c'})$	$NA(c,c')$ is the measure of the affinity between the names of two classes, if this measure exceeds a specified threshold.
$SA(c,c')$	$\frac{\lvert\{a_t \lvert a_t \in A(c), a_q \in A(c'), n_t \sim n_q\}\rvert + \lvert\{a_q \lvert a_t \in A(c), a_q \in A(c'), n_t \sim n_q\}\rvert}{\lvert A(c)\rvert + \lvert A(c')\rvert} \cdot F_c$	$SA(c,c')$ is the measure of the level of structural matching of the two classes, proportional to the number of their attributes whose names have affinity and whose domains are compatible.
$GIA(c,c')$	$w_{NA} \cdot NA(c,c') + w_{SA} \cdot SA(c,c')$	$GIA(c,c')$ is a global measure of the affinity of two classes based on intensional properties only, computed as the linear combination of their name and structural affinity, respectively.
$GA(c,c')$	$GIA(c,c')$+(1-$GIA(c,c')$ $\cdot$ $\sigma_{\Re_{ext}}(c,c')$	$GA(c,c')$ is a comprehensive measure of the affinity of two classes, which takes into account also the extensional properties holding between the two classes.

Legend:
$A(c)$ and $A(c')$ are the sets of attributes of c and c', respectively
$F_c = \frac{\lvert\{x \in C \mid flag(x)=1\}\rvert}{\lvert C\rvert}$, control factor where $flag(x) = 1$ stands for a domain compatibility
$C = \{(a_t, a_q) \mid a_t \in A(c), a_q \in A(c'), n_t \sim n_q\}$

1. `UNI.School_Member` SYN$_{ext}$ `TP.Student`
2. `CS.Student` NT$_{ext}$`UNI.School_Member`
3. `CS.Professor` NT$_{ext}$ `UNI.Research_Staff`
4. `UNI.Section` RT$_{ext}$`CS.Division`
5. `CS.Student` NT$_{ext}$ `CS.CS_Person`
6. `CS.Professor` NT$_{ext}$`CS.CS_Person`
7. `CS.Student` NT$_{ext}$`TP.Student`

Relationships from 1 to 4 are supplied by the designer; relationships 5 and 6 are intra–schema extensional relationships automatically extracted by the MOMIS system from the is–a hierarchies of the `COMPUTER_SCIENCE` source; relationship 7 is inferred from 1 and 2. Finally, we assume that a DISJ$_{ext}$ relationship holds when no other extensional relationship between two classes is specified.

5 Matching heterogeneous source schemas

Goal of this phase of the integration methodology in MOMIS/ARTEMIS is to identify ODL$_{I3}$ classes candidate to integration, that is, classes that describe the same or semantically related information in different source schemas. To this end, *affinity coefficients* (i.e., numerical values in the range $[0,1]$) are evaluated for all possible pairs of ODL$_{I^3}$ classes, based on the relationships in the Common Thesaurus properly strengthened. This activity is performed by the ARTEMIS tool environment [12]. Affinity coefficients determine the degree of matching of two ODL$_{I^3}$ classes c and c' based on their names (*Name Affinity* coefficient) and their attributes (*Structural Affinity* coefficient). *Global Intensional Affinity* coefficient is then determined as the linear combination of the Name and Structural Affinity coefficients. To evaluate such coefficients, an affinity function $A()$ is defined on top of the Common Thesaurus to evaluate the affinity of two terms. The affinity $A(t,t')$ of two terms t and t' is equal to the highest-strength path of terminological relationships (i.e., intensional) between them, if at least one path exist, and is zero otherwise. Given two terms t and t' and a path of terminological relationships between them, the strength of this path is computed by multiplying the strengths of all terminological relationships involved in it. $A(t,t')$ coincides with the strength of the highest-strength path between t and t', denoted by $\rightarrow^m$, that is, $A(t,t') = \sigma_{12_{\Re}} \cdot \sigma_{23_{\Re}} \cdot \ldots \cdot \sigma_{(m-1)m}$. In the following, we use the symbol $\sim$ to denote the fact that two terms have affinity in the Common Thesaurus. In Table 1, we report the definition of the affinity coefficients and a brief description. For a detailed description of the affinity coefficient evaluation process, the reader can refer to [22]. The Name Affinity, Structural Affinity, and Global Affinity coefficients are evaluated based only on terminological relationships holding between the classes of the source schemas to be integrated.In this report, we describe the extension of affinity techniques to take into account also the knowledge provided by extensional properties.

To assess the level of affinity of two ODL_{I3} classes in a comprehensive way, the Global Affinity coefficient needs to take into account also the knowledge provided by extensional properties. In the following, we refer to the Global Affinity coefficient obtained based only on intensional inter-schema properties, as Global Intensional Affinity (GIA). We now extend the Global Affinity coefficient, to take into account the knowledge provided by extensional properties. We denote by $\Re_{ext}$, with $\Re_{ext} \in \{SYN_{ext}, BT_{ext}, NT_{ext}, RT_{ext}, DJS_{ext}\}$ an inter-schema extensional relationship. For affinity evaluation, each extensional relationship $\Re_{ext}$ is associated with a strength $\sigma_{\Re_{ext}} \in [0,1]$, expressing the implication of the relationship for the computation of affinity. Following what has been done in assessing the strengths of the terminological relationships in ARTEMIS, we consider equivalence (SYN_{ext}) as the strongest indicator for affinity; moreover we consider inclusion (BT_{ext}/NT_{ext}) stronger than overlapping (RT_{ext}). Consequently, we have $\sigma_{SYN_{ext}} \geq \sigma_{BT_{ext}/NT_{ext}} \geq \sigma_{RT_{ext}} \geq \sigma_{DJS_{ext}}$, with the following default strengths $\sigma_{SYN_{ext}}$=1, $\sigma_{BT/NT_{ext}} = \sigma_{RT_{ext}}$=0.8 and $\sigma_{DJS_{ext}}$=0, respectively. The designer can modify such values if necessary.

Global Affinity coefficient. Given two ODL$_{I^3}$ classes c and c', their Global Affinity coefficient, denoted $GA(c,c')$, is the measure of their intensional and extensional affinity computed by taking into account both intensional and extensional properties holding between them, as follows:

$$\text{GA}(c,c') = \text{GIA}(c,c') + (1\text{-GIA}(c,c')) \cdot \sigma_{\Re_{ext}}(c,c')$$

where $\sigma_{\Re_{ext}}(c,c')$ denotes the strength of the extensional relationship holding between c and c'.

The Global Affinity coefficient of c and c' is combination of their Global Intensional Affinity coefficient and of a factor (1-GIA(c,c')) $\cdot$ $\sigma_{\Re_{ext}}(c,c')$ espressing the impact of extensional knowledge on the affinity of c and c'. The use of this factor has a twofold motivation: i) on one side the GIA coefficient obtained using only intensional relationships remains unalterated when c and c' are disjoint, and ii) on the other side, the affinity of two classes augments if they have common instances, in a way proportional to the kind of the extensional property and its relevance for affinity evaluation.

ODL_{I3} **class** (c)	ODL_{I3} **class** (c')	$NA(c,c')$	$SA(c,c')$	$GIA(c,c')$	$GA(c,c')$
$CS.Division$	$CS.Location$	0.5	0.0	0.25	0.25
$CS.Division$	$CS.Professor$	0.5	0.0	0.25	0.25
$CS.Division$	$UNI.Section$	0.0	0.0	0.0	0.5
$CS.Student$	$CS.Location$	0.64	0.8	0.72	0.72
$CS.Student$	$CS.CS\_Person$	0.8	0.5	0.65	0.93
$CS.Student$	$CS.Course$	0.5	0.0	0.25	0.25
$CS.Student$	$CS.Professor$	0.64	0.6	0.62	0.62
$CS.Student$	$TP.ListOfStudent$	0.5	0.0	0.25	0.25
$CS.Student$	$TP.Student$	1.0	0.545	0.773	0.9546
$CS.Student$	$UNI.School\_Member$	1.0	0.8	0.9	0.98
$CS.Student$	$UNI.Section$	0.5	0.0	0.25	0.25
$CS.Professor$	$CS.CS\_Person$	0.8	0.667	0.734	0.9468
$CS.Professor$	$CS.Course$	0.5	0.0	0.25	0.25
$CS.Professor$	$TP.Student$	0.64	0.444	0.542	0.542
$CS.Professor$	$UNI.School\_Member$	0.64	0.5	0.57	0.57
$CS.Professor$	$UNI.Research\_Staff$	0.0	0.0	0.0	0.8
$CS.Professor$	$UNI.Section$	0.5	0.0	0.25	0.25
$CS.CS\_Person$	$TP.Student$	0.8	0.571	0.686	0.686
$CS.CS\_Person$	$UNI.School\_Member$	0.8	0.667	0.734	0.734
$CS.Course$	$TP.Student$	0.5	0.286	0.393	0.393
$CS.Course$	$UNI.School\_Member$	0.5	0.0	0.25	0.25
$CS.Course$	$UNI.Room$	0.5	0.4	0.45	0.45
$CS.Course$	$UNI.Research\_Staff$	0.5	0.0	0.25	0.25
$CS.Course$	$UNI.Section$	1.0	0.333	0.667	0.9334
$TP.ListOfStudent$	$TP.Student$	0.5	0.333	0.417	0.417
$TP.ListOfStudent$	$UNI.School\_Member$	0.5	0.0	0.25	0.25
$TP.Student$	$UNI.School\_Member$	1.0	0.667	0.834	1.0
$TP.Student$	$UNI.Section$	0.5	0.222	0.361	0.361
$UNI.School\_Member$	$UNI.Section$	0.5	0.0	0.25	0.25
$UNI.Room$	$UNI.Section$	0.5	0.286	0.393	0.393
$UNI.Research\_Staff$	$UNI.Section$	0.5	0.286	0.393	0.393
$UNI.Research\_Staff$	$UNI.Department$	0.5	0.333	0.417	0.417

Table 2: Affinity coefficients for the running example

The Name Affinity, Structural Affinity, Global Intensional Affinity, and Global Affinity coefficients calculated for the source schemas of the running example are reported in Table 2. For the sake of clarity, in this table we report only the classes having a not null GIA or GA coefficient, respectively.

Global affinity coefficients are then used by a hierarchical clustering algorithm, to classify ODL_{I^3} classes according to their degree of affinity. The output of the clustering procedure is an affinity tree, where ODL_{I^3} classes are the leaves and intermediate nodes have an associated affinity value, holding for the classes in the corresponding cluster. Clusters for integration (candidate clusters) are interactively selected from the affinity tree using a threshold based mechanism.

Candidate clusters are characterized with respect to *size* and *level of homogeneity*. The size of a candidate cluster is the number of ODL_{I3} classes contained. The level of homogeneity of a candidate cluster is given by the threshold value, and conseguently the affinity value associated with the cluster in the affinity tree (i.e., a high value of affinity corresponds to a high level of homogeneity). The number, size, and level of homogeneity of candidate clusters is determined by the designer that can set the most suitable value for the threshold γ. By setting γ to high values (e.g., $\gamma \in [0.7, 1]$), a large number of small-size candidate clusters is selected, containing highly homogeneous classes. By decreasing the value of γ, the number of candidate clusters decreases. The size of candidate clusters increases as well, while their level of homogeneity decreases. For a given value of γ, when more than one cluster turns out to be candidate in the affinity tree,

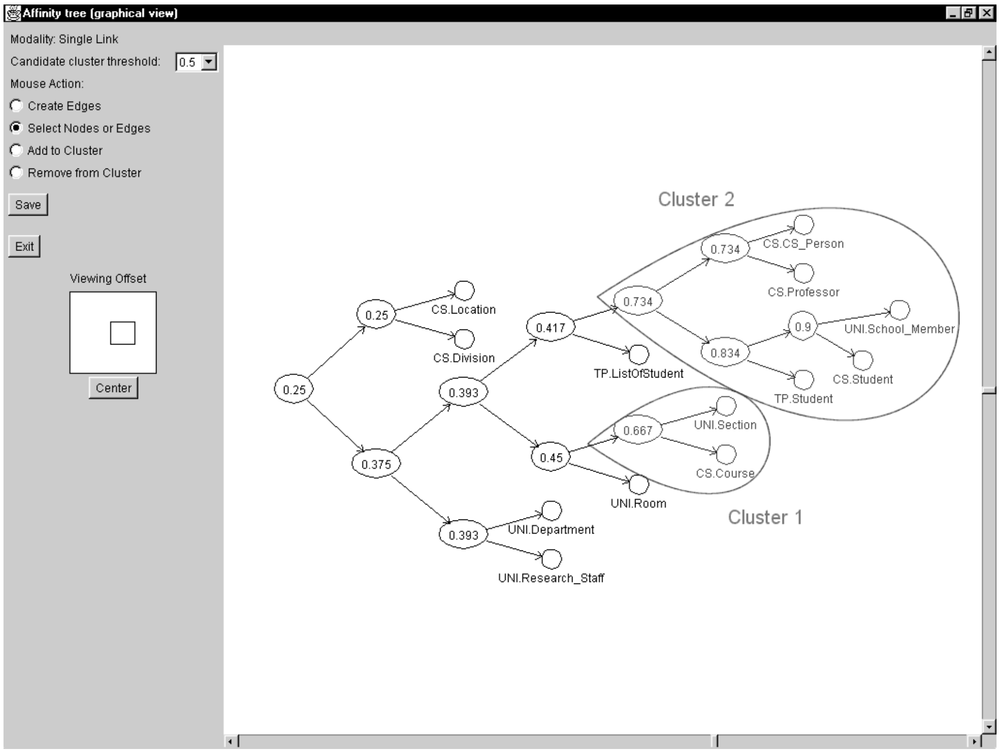

Figure 4: Candidate clusters

the maximum-size candidate cluster is selected.

In Figure 4, the affinity tree computed by considering only the intensional relationships (i.e. GIA coefficients) is shown, emphasizing the candidate clusters obtained using a threshold value $\gamma = 0.5$. There are two candidate clusters: the first one has size 5 and contains `UNI.School_member`, `CS.Student`, `TP.Student`, `CS.CS_Person` and `CS.Professors`. The second one comprises the `UNI.Section` and `CS.Course` classes (size 2).

In Figure 5, we show the affinity tree computed by considering both intensional and extensional relationships (i.e. the GA coefficients). Also in this case there are two candidate clusters: the first one is composed by ODL$_{I^3}$ classes describing person data; the second one contains the classes describing the organization of university.

The main difference between the two affinity trees is due to the different position of the `UNI.Research_Staff` and `CS.Division` classes. Considering only intensional relationships, `UNI.Research_Staff` class is not clustered together with other classes describing university people and `CS.Division` class is not clustered together with other classes describing university organization. Consider both intensional and extensional properties in affinity evaluation, such classes are included in the respective clusters. This result shows the usefulness of the extensional properties to enrich the knowledge featuring the source and to make the candidate clusters more complete and increase their level of homogeneity.

6 Generation of a global integrated representation

For each selected cluster in the tree, a global ODM$_{I^3}$ class providing the unified view of all the classes of the cluster is defined in MOMIS/ARTEMIS. The generation of global classes is interactive with the designer.

Synthesis of clusters of ODM$_{I^3}$ classes requires to take into account semantic heterogeneity, which has to be treated properly to come up with an integrated and uniform representation

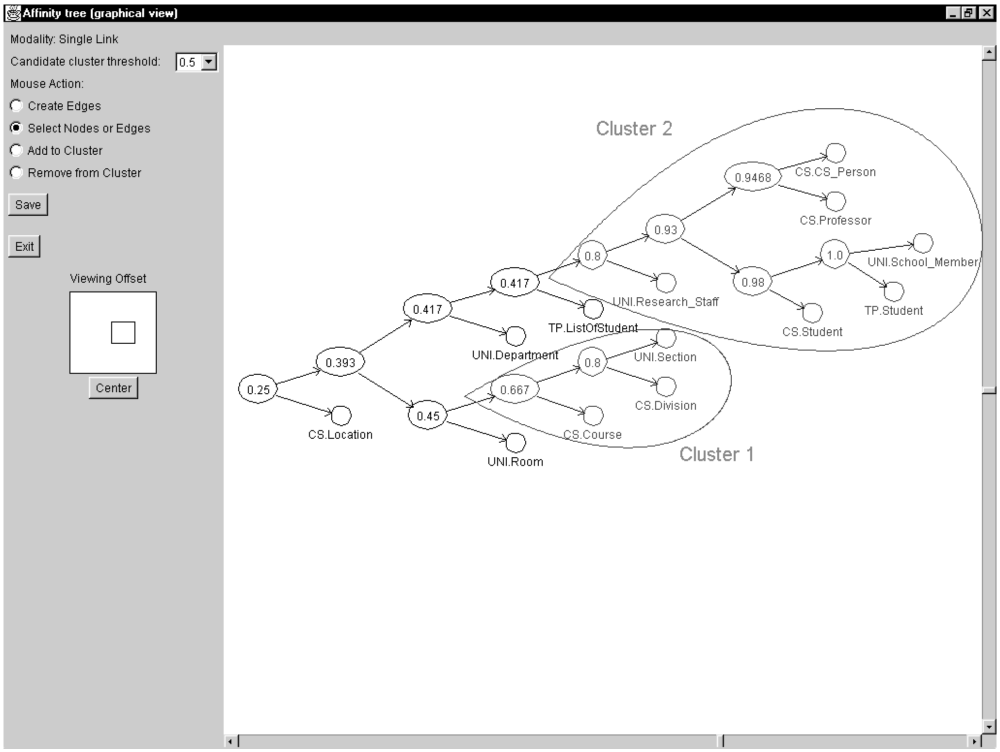

Figure 5: Candidate clusters with extensional properties

at the global level. Let Cl_i be a selected cluster in the affinity tree and $global\_class_i$ the global ODM$_{I^3}$ class to be defined for Cl_i. First, we associate with $global\_class_i$ a set of global attributes, corresponding to the union of the attributes of the classes belonging to Cl_i. The attributes having a valid terminological relationship are unified into a unique global attribute in $global\_class_i$. The attribute unification process is performed automatically for what concerns names according to the following rules:

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

In Figure 6, an example of global ODL$_{I^3}$ class defined for for cluster 1 of Figure 5 is shown.

7 Definition of mappings

To complete global class definition, information on attribute mappings and default values is provided by the designer in the form of *mapping rules*. For each global ODM$_{I^3}$ class a persistent *mapping-table* storing all the mappings is generated; it is a table whose rows represent the set of the local classes belonging to the cluster and whose columns represent the global attributes. An element $MT[L][ag]$ represents the set of attributes of the local class L which are mapped into the global attribute ag: the value of the ag attribute is a function of the values assumed by the set of attributes $MT[L][ag]$. Some simple and frequent cases of such function are the following (see Figure 7 as an example):

```
interface University_Person
{ attribute name
    mapping_rule UNI.Research_Staff.name,
                 UNI.School_Member.name,
                 (CS.CS_Person.first_name and CS.CS_Person.last_name),
                 (CS.Student.first_name and CS.Student.last_name),
                 (CS.Professor.first_name and CS.Professor.last_name),
                 TP.Student.name;
  attribute dept
    mapping_rule UNI.Research_Staff.dept_code;
  attribute e_mail
    mapping_rule UNI.Research_Staff.e_mail,
                 UNI.School_Member.e_mail,
                 CS.Student.e_mail,
                 TP.Student.e_mail;
  attribute section
    mapping_rule UNI.Research_Staff.s_code;
  attribute school
    mapping_rule UNI.School_Member.school,
                 CS.CS_Person = 'cs',
                 CS.Student = 'cs',
                 CS.Professor = 'cs',
                 TP.Student.school_name;
  attribute year
    mapping_rule UNI.School_Member.year,
                 CS.Student.year;
  attribute belong_to
    mapping_rule CS.Professor = 'belong_to';
  attribute takes
    mapping_rule CS.Student.takes;
  attribute rank
    mapping_rule UNI.Research_Staff = 'professor',
                 UNI.School_Member = 'student',
                 CS.Student.rank,
                 CS.Professor.rank,
                 TP.Student = 'student';
  attribute s_code
    mapping_rule TP.Student.s_code;
  attribute tax_fee
    mapping_rule TP.Student.tax_fee;
}
```

Figure 6: Example of global ODL_{I^3} class `University_Person`

University_Person	name	dept	e_mail	section	school	year	belong_to	takes	rank	s_code	tax_fee
UNI.Research_Staff	name	dept_code	e_mail	s_code	null	null	null	null	"professor"	null	null
UNI.School_Member	name	null	e_mail	null	school	year	null	null	"student"	null	null
CS.CS_Person	first_name and last_name	null	null	null	"cs"	null	null	null	null	null	null
CS.Student	first_name and last_name	null	e_mail	null	"cs"	year	null	takes	rank	null	null
CS.Professor	first_name and last_name	null	null	null	"cs"	null	"belong_to"	null	rank	null	null
TP.Student	name	null	e_mail	null	school_name	null	null	null	"student"	s_code	tax_fee

Figure 7: Mapping Table for the global class `University_Person` of Figure 5

- *identity* : the ag value *is equal to* the la value; we denote this case as $MT[L][ag] = la$.
- *concatenation* : the ag value is obtained as a concatenation of the values assumed by a set of local attributes la_i of the local class L; we denote this case as $MT[L][ag] = la_1$ **and** $\ldots$ **and** la_n (see MT[`CS.Student`][`name`]).

When the ag has no correspondence with any attribute of the local class L, the possible choices are:

- *constant* : ag assumes into the local class L a costant value set by the designer; we denote this case by $MT[L][ag] =$ `const` (see the `rank` attribute).
- *undefined* : ag is set undefined into the local class L; we denote this case by $MT[L][ag] =$ `null`.

8 Integration of sources supporting schema versioning

In this section, we discuss how schema version information can be taken into account during the integration process. In particular we show how new interschema properties can be derived that would not be identified otherwise. We refer to the running example and to the MOMIS approach to interschema property extraction and representation. We briefly recall the difference between a *single-pool* and a *multi-pool* solution for the management of extensional data in a database supporting schema versioning [19]:

multi-pool : each schema version is associated with a "private" extensional data pool; different data pools may contain the same objects having (possibly) completely independent representations and evolutions;

single-pool : all the schema versions are associated with a unique, shared, extensional repository, so that the the same objects cannot have different values for the same properties when "viewed" through different schema versions.

Although the multi-pool solution is more flexible and appealing from an application perspective, a single-pool solution has usually considered satisfactory for implementation (e.g. by means of the so-called *completed schema* technique [38]) as it limits the storage space overhead due to multiple schemas. Hence, when we consider in the following a single-pool implementation, we keep in mind that this should be interpreted as a potentially poor trade-off approximation of a logically correct (i.e. multi-pool) implementation solution.

Schema-version-derived relationships

In this step, we extract terminological and extensional relationships holding at intra-schema level (but both at intra-schema-version and inter-schema-version levels) by analyzing the schema of each source supporting schema versioning. In particular, useful relationships can be automatically derived and maintained at schema-change execution time or can be *a posteriori* derived by analyzing schema-change logs[1].

With reference to the running example, we consider an "upgrade" of the **UNIVERSITY** source with schema versioning support. We assume the schema depicted in Fig. 2.1 to represent an initial schema version, that we call $UNI1$. As a sample schema change, we further assume to rename the table `Research_Staff` into `Faculty` via the schema change:

<u>ChangeClassName</u> `Research_Staff`, `Faculty`

producing a new schema version $UNI2$. The semantics of this schema change induces, between $UNI1$ and $UNI2$, several obvious synonymities concerning schema items which have not been touched by the schema change, and a *non-trivial* synonymity between `Research_Staff` in $UNI1$ and `Faculty` in $UNI2$. In a multi-pool system, such terminological relationships are purely intensional, whereas they are also extensional in a single-pool system.

Assuming, for example, an ODL_{I^3} approach (but similar procedures can be developed also for the SDR-Network or $\mathcal{DLR}$ approaches), the non-trivial relationship corresponds to automatically add to the Common Thesaurus the following constraints:

⟨ `UNI1.School_Member` SYN `UNI2.School_Member` ⟩
...
⟨ `UNI1.Research_Staff.name` SYN `UNI2.Faculty.name` ⟩
...
⟨ `UNI1.School_Member.name` SYN `UNI2.School_Member.name` ⟩
...
⟨ `UNI1.Research_Staff` SYN `UNI2.Faculty` ⟩

where the last relationship is the *non-trivial* synonymity. We can safely write such constraints since the data pools associated with different schema versions can be modeled as independent sources. Actually, with the multi-pool they *must* be modeled as independent sources.

Notice that, whereas the "obvious" synonymities could also be derived later as lexicon-derived inter-schema relationships, the non-trivial one can, in general, be correctly derived only owing to the semantics of the schema change. For instance, consider an incremental design scenario (where schema versioning is likely to be highly useful), where some classes/properties were initially assigned *generic names* which have been turned into meaningful names only in a second phase of the database design, by means of schema changes like:

<u>ChangeClassName</u> `Class_A`, `Research_Staff`

where "`Class_A`" was the generic name assigned to the class containing staff data in schema version $UNI0$. In this case, the induced relationship ⟨ `UNI0.Class_A` SYN `UNI1.Research_Staff` ⟩ could not be derived in another way, as there is no semantic connection between the word "Faculty" and the generic term "Class_A" that could be suggested by any Thesaurus.

Therefore, if the schema-versioning-aware automatic generation of relationships as such is not enforced, *useful synonymities are lost and all the advantages of schema versioning support are also lost in the integration environment.* On the other hand, once such synonymities are correctly dealt with, the integration of data extracted from different schema versions of a single

[1] In case the $\mathcal{CVM}$ data model described in report D1.R4 [27] is used for the formalization of a schema-versioned source, such analysis reduces to simple queries issued on the $\mathcal{CVM}$ instance.

source does not present further difficulties with respect to the general problem of integration from independent sources and, thus, can be handled with the standard techniques.

For example, extensional relationships valid also in the presence of a multi-pool solution (i.e. when representations of the same objects in different data pools are maintained synchronized) can be detected by means of the "normal" inference/validation techniques (in the *Matching heterogeneous source schemas* phase) described in reports D1.R1 [22] and D1.R2 [3].

The same happens, also with a single-pool, for *change-type* schema changes. If the same class or property is associated to different types in different schema versions as a consequence of schema changes, intensional BT/NT inclusion relationships may hold or not depending on the compatibility of types before and after the schema change. Furthermore, if intensional relationships hold, also extensional BT/NT relationships may hold or not depending on the synchronization of data objects[2]. In any case, the standard inference/validation procedure described in reports D1.R1 [22] and D1.R2 [3] can be used to handle such cases, and must be used: information concerning schema change semantics would be of no use, since the derivable relationships do not *necessarily* hold.

On the other hand, certain intensional relationshiphs are induced and, thus, can be automatically added to the Common Thesaurus in this phase, for add/delete-attribute schema changes, since the addition/deletion of an attribute to a record type produces a sub/super-type, respectively. The corresponding extensional relationships instead need validation in the multi-pool case. A similar case is represented by schema changes AddSuperClass and DeleteSuperClass, available in object-oriented sources, explicitly affecting the type hierarchy.

We summarize now the derivable relationships by addressing the general case. Consider a schema change M_{ij} which applied to an existing schema SV_i produces a new schema version SV_j. In general, the application of M_{ij} produces, as a result, inter-schema dependencies between SV_i and SV_j [25]. In our case, the complete set of relationships induced by M_{ij} includes:

- several (obvious) inter-schema-version synonymities ⟨ `SVi.X` SYN `SVj.X` ⟩ between schema items X non affected by M_{ij};
- the (obvious) replication in SV_j of all intra-schema-version relationships holding in SV_i and concerning non affected schema items;
- some *non-trivial* inter-schema-version relationships between SV_i and SV_j items affected by M_{ij} possibly induced by the schema change semantics.

Notice that the second group of (obvious) relationships does not need to be explicitly derived at this point and memorized, as they can always be deduced from the intra-schema-version relationships holding in SV_i and the SV_i–SV_j (obvious) synonymities. Actually, owing to transitivity of synonymy, only the intra-schema-version relationships holding in the initial schema SV_0 need to be explicitly represented. However, an explicit bookkeeping of all the inter-schema-version relationships, which come for free during the schema evolution process, saves the computational burden of their (repeated) deduction during the integration/reasoning phases.

Table 3 shows the non-trivial schema-version-derived relationships which are induced by the schema changes, both for the multi-pool and the single-pool solution. We considered all the schema changes usually taken into account in the literature on schema evolution and versioning (where an object-oriented data model [25] is the most common choice).

In conclusion, in the presence of data sources with schema versioning support, we slightly modify our integration methodology by modeling a versioned source as a collection of independent data sources, which correspond to individual data pools associated to the different schema

[2]Notice that, in a single-pool system based on the *completed schema* [19], the notion of type compatibility is relaxed as far as "union types" are actually always implicitly used at physical level, when schema changes produce incompatible types: in this case, the completed schema will contain both "versions" of the concept/property and the schema versions can simply be defined as *views* on the completed schema.

Applied Schema Change	**Induced Relationships** in multi-pool	in single-pool
<u>AddAttr</u> C, A:T	⟨ `SVi.C` BT `SVj.C` ⟩ ⟨ `SVi.C` BT_{ext} `SVj.C` ⟩ (∗)	⟨ `SVi.C` BT_{ext} `SVj.C` ⟩
<u>DeleteAttr</u> C, A	⟨ `SVi.C` NT `SVj.C` ⟩ ⟨ `SVi.C` NT_{ext} `SVj.C` ⟩ (∗)	⟨ `SVi.C` NT_{ext} `SVj.C` ⟩
<u>ChangeAttrName</u> C, A, B	⟨ `SVi.C.A` SYN `SVj.C.B` ⟩	⟨ `SVi.C.A` SYN_{ext} `SVj.C.B` ⟩
<u>ChangeAttrType</u> C, A, T	⟨ `SVi.C.A` BT `SVj.C.A` ⟩ (∗) ⟨ `SVi.C.A` NT `SVj.C.A` ⟩ (∗)	⟨ `SVi.C.A` BT_{ext} `SVj.C.A` ⟩ (∗) ⟨ `SVi.C.A` NT_{ext} `SVj.C.A` ⟩ (∗)
<u>AddClass</u> C	(no relationship)	(no relationship)
<u>DeleteClass</u> C	(no relationship)	(no relationship)
<u>ChangeClassName</u> C, D	⟨ `SVi.C` SYN `SVj.D` ⟩	⟨ `SVi.C` SYN_{ext} `SVj.D` ⟩
<u>ChangeClassType</u> C, T	⟨ `SVi.C` BT `SVj.C` ⟩ (∗) ⟨ `SVi.C` NT `SVj.C` ⟩ (∗)	⟨ `SVi.C` BT_{ext} `SVj.C` ⟩ (∗) ⟨ `SVi.C` NT_{ext} `SVj.C` ⟩ (∗)
<u>AddSuperclass</u> C, D	⟨ `SVj.D` BT_{ext} `SVj.C` ⟩	⟨ `SVj.D` BT_{ext} `SVj.C` ⟩ ⟨ `SVi.D` BT_{ext} `SVi.C` ⟩
<u>DeleteSuperclass</u> C, D	(no relationship)	(no relationship)

Table 3: Non-trivial relationships induced by basic schema changes (creating SV_j from SV_i). The symbol (∗) marks the relationships that not necessarily hold and, thus, need validation.

versions. Once the *non-trivial* schema-version-derived relationships have been introduced in the *Extraction and representation of intra and interschema properties* phase, according to Tab. 3 (or similar, if the SDR-Network or $\mathcal{DLR}$ approaches are followed), the presence of schema versioning does not present further peculiar integration difficulties, that cannot be handled with the standard methodology described in the rest of the paper. Hence, the integration sub-phase described here provides a sort of "encapsulation method" to make local schema versioning transparent with respect to the global integration scenario.

9 Concluding remarks

In this report, we have described the *D2I-METH* methodological framework for the integration of strongly heterogeneous datasources. *D2I-METH* generalizes the integration approaches and techniques developed as the first step of the *D2I* project by the four different partners involved in the integration task of *D2I*. *D2I-METH* supports a virtual approach to integration, in that data residing at the sources are accessed during query processing by exploiting defined mappings, instead of being replicated at the global level. Moreover, a GAV approach is followed to build a global schema which is expressed in terms of the datasources. The global schema and associated mappings obtained by applying such a methodology provide support for expressing queries in terms of global views over underlying datasources, with mechanisms for the reformulation and/or answering of such queries in terms of the data stored in the sources. An important feature of the framework is that the underlying approaches use of formalization and reasoning capabilities of Description Logics in both semantic integration and view-based query processing. The application of the *D2I-METH* framework requires the instantiation to one of the three approaches supported by the framework (i.e., MOMIS/ARTEMIS, DIKE, and

$\mathcal{DLR}$). In the report, we have described the instantiation to the MOMIS/ARTEMIS integration approach, by describing how each phase of *D2I-METH* is performed in this approach. The instantiation has shown the activites that are performed and the techniques that are adopted in MOMIS/ARTEMIS to accomplish each general activi y of *D2I-METH*. Furthermore, we have discussed how schema versioning information can provide additional interschema knowledge that can be exploited during integration process.

Besides the instantiation to a given integration approach, for each phase of *D2I-METH*, several techniques can be available in the different underlying integration approaches for accomplishing the activity considered in the phase, such as for example for the inter-schema property extraction and representation phase or for the schema matching phase.

For this purpose, a metadata repository providing a centralized and unified representation of the metadata documenting all activities related to integration applications has been defined in D2I. Thus, it will be used as a common basis by the various tools developed within the D2I project aiming at supporting the various phases of the integration methodology. A metadata schema describing the metadata maintained during the integration activities has been defined for each approach [21]. A common *metadata repository* architecture, which constitutes a unification layer for the various approaches has also been defined to allow the migration to the common framework of the various techniques developed independently in the three approaches. The availability of such a common architecture will be the basis for supporting the use of tools implementing alternative techniques in a single phase of the methodology, giving rise to an instantiation of the methodology involving several techniques of the different approaches developed in D2I.

References

[1] S. Abiteboul. Querying semi-structured data. In *Proc. of International Conference on Database Theory (ICDT'97)*, pages 1–18, Delphi, Greece, 1997. Lecture Notes in Computer Science, Springer-Verlag.

[2] C. Batini and M. Lenzerini. A methodology for data schema integration in the entity relationship model. *IEEE Transactions on Software Engineering*, 10(6):650–664, 1984.

[3] D. Beneventano, S. Bergamaschi, F. Guerra, S. Castano, and M. Vincini. Use of Ontologies and Extensional Inter-Schema Properties for Integration. Technical Report D1.R2, D2I Project, 2001.

[4] D. Beneventano, S. Bergamaschi, F. Guerra, and M. Vincini. The MOMIS approach to information integration. In *AAAI International Conference on Enterprise Information Systems (ICEIS 2001)*, 2001.

[5] D. Beneventano, S. Bergamaschi, C. Sartori, and M. Vincini. ODB-QOPTIMIZER: A tool for semantic query optimization in oodb. In *Int. Conference on Data Engineering (ICDE'97)*, 1997.

[6] S. Bergamaschi, S. Castano, D. Beneventano, and M. Vincini. Semantic integration of heterogenous information sources. *Journal of Data and Knowledge Engineering*, 36(3):215–249, 2001.

[7] P. Buneman. Semistructured data. In *Proc. of Symposium on Principles of Database Systems, (PODS'97)*, pages 117–121, Tucson, Arizona, USA, 1997. ACM Press.

[8] P. Buneman, S. Davidson, M. Fernandez, and D. Suciu. Adding structure to unstructured data. In *Proc. of ICDT 1997*, pages 336–350, Delphi, Greece, 1997.

[9] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. On the decidability of query containment under constraints. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98)*, pages 149–158, 1998.

[10] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Description logic framework for information integration. In *Proc. of the 6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 2–13, 1998.

[11] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. What is query rewriting? In *Proc. of the 7th Int. Workshop on Knowledge Representation meets Databases (KRDB 2000)*, pages 17–27, 2000.

[12] S. Castano, V. De Antonellis, and S. De Capitani di Vimercati. Global viewing of heterogeneous data sources. *Transactions on Data and Knowledge Engineering*, 13(2), 2001.

[13] S. Castano, V. De Antonellis, S. De Capitani di Vimercati, and M. Melchiori. An XML-based framework for information integration over the Web. In *Proc. of Int. Workshop on Information Integration and Web-based Applications & Services 2000*, Yogyakarta, Indonesia, 2000.

[14] T. Catarci and M. Lenzerini. Representing and using interschema knowledge in cooperative information systems. *Journal of Intelligent and Cooperative Information Systems*, 2(4):375–398, 1993.

[15] R. G. G. Cattell, editor. *The Object Database Standard: ODMG93.* Morgan Kaufmann Publishers, San Mateo, CA, 1994.

[16] S. Cluet, P. Veltri, and D. Vodislav. Views in a large scale xml repository. In *Proc. of the 27th VLDB Conference*, Roma, Italy, 2001.

[17] World Wide Web Consortium. Extensible markup language (XML) 1.0. *http://www.w3.org/TR/REC-xml*, 1998.

[18] M. Lenzerini D. Calvanese, D. Lembo. Survey on methods for query rewriting and query answering using views. Technical Report D1.R5, D2I Project – Integration, Warehousing, and Mining of Heterogeneous Data Sources, 2001.

[19] C. De Castro, F. Grandi, and M. R. Scalas. Schema Versioning for Multitemporal Relational Databases. *Information Systems*, 22(5):249–290, 1997.

[20] A. Doan, P. Domingos, and A. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *Proc. of Int. ACM SIGMOD*, Santa Barbara, California, USA, 2001.

[21] D. Beneventano et.al. Specification for the metadata repository. Technical Report D0.R1, D2I Project, 2001.

[22] I. Benetti et.al. Methodologies and techniques for the extraction, the representation and the integration of structured and semi-structured information sources. Technical Report D1.R1, D2I Project, 2001.

[23] C. Fahrner and G. Vossen. Transforming relational database schemas into object oriented schemas according to odmg-93. In *Proc. Int. Conf. Deductive and Object-Oriented Databases DOOD-95*, Lecture Notes in Computer Science, Vol. 1013, Springer, 1995) 429-446, 1995.

[24] Maydene Fisher. Jdbc(tm) database access). *"http://java.sun.com/docs/books/ tutorial/jdbc/index.html"*.

[25] E. Franconi, F. Grandi, and F. Mandreoli. A Semantic Approach for Schema Evolution and Versioning in Object-Oriented Databases. In *Proc. Intl' Conf. on Deductive and Object-Oriented Databases (DOOD 2000)*, 2000.

[26] R. Goldman, J. McHugh, and J. Widom. From semistructured data to XML: Migrating the lore data model and query languages. In *Proc. of International Workshop on the Web and Databases (WebDB'99)*, pages 25–30, Philadelphia, Pennsylvania, USA, 1999.

[27] F. Grandi. A Generalized Schema Versioning Model for Object-Oriented and Semi-Structured Data. Technical Report D1.R4, D2I Project, 2001.

[28] R. Hull and R. King et al. Arpa i^3 reference architecture, 1995. Available at http://www.isse.gmu.edu/I3_Arch/index.html.

[29] Richard Hull. Managing semantic heterogeneity in databases: A theoretical perspective. In *Proc. of the 16th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'97)*, 1997.

[30] G. Terracina D. Ursino L. Palopoli, D. Rosaci. Algorithms for data translation from a source format to a target one. Technical Report D1.R9, D2I Project, 2001.

[31] A.Y. Levy. Answering queries using views: A survey. Technical report, University of Washinghton, 1999.

[32] J. Madhavan, P.A. Bernstein, and E. Rahm. Generic schema matching with cupid. In *Proc. of the 27th VLDB Conference*, Roma, Italy, 2001.

[33] S. Nestorov, S. Abiteboul, and R. Motwani. Inferring structure in semistructured data. *SIGMOD Record*, 26(4):39–43, 1997.

[34] L. Palopoli, G. Terracina, and D. Ursino. The system dike: Towards the semi-automatic synthesis of cooperative information systems and data warehouses. In *Proc. of Challenges of Symposium on Advances in Databases and Information Systems (ADBIS-DASFAA 2000)*, pages 108–117, Prague, Czech Republic, 2000. Matfyzpress.

[35] L. Palopoli, G. Terracina, and D. Ursino. A graph-based approach for extracting terminological properties of elements of XML documents. In *Proc. of International Conference on Data Engineering (ICDE 2001)*, pages 330–340, Heidelberg, Germany, 2001. IEEE Computer Society.

[36] Y. Papakonstantinou, S. Abiteboul, and H. Garcia-Molina. Object fusion in mediator systems. In *VLDB Int. Conf.*, Bombay, India, September 1996.

[37] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In *Proc. of International Conference on Data Engineering (ICDE'95)*, pages 251–260, Taipei, Taiwan, 1995. IEEE Computer Society.

[38] J. F. Roddick and R. T. Snodgrass. Schema Versioning. In *The TSQL2 Temporal Query Language*, pages 427–449. Kluwer Academic Publishers, 1995.

[39] Mary Tork Roth and Peter M. Schwarz. Don't scrap it, wrap it! a wrapper architecture for legacy data sources. In Matthias Jarke, Michael J. Carey, Klaus R. Dittrich, Frederick H. Lochovsky, Pericles Loucopoulos, and Manfred A. Jeusfeld, editors, *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, pages 266–275. Morgan Kaufmann, 1997.

[40] D. Suciu. Semistructured data and XML. In *Proc. of International Conference on Foundations of Data Organization (FODO'98)*, Kobe, Japan, 1998.

[41] Jeffrey D. Ullman. Information integration using logical views. In *Proc. of the 6th Int. Conf. on Database Theory (ICDT'97)*, volume 1186 of *Lecture Notes in Computer Science*, pages 19–40. Springer-Verlag, 1997.

[42] J. Widom. Research problems in data warehousing. In *Proc. of International Conference on Information and Knowledge Management (CIKM'95)*, pages 25–30, Baltimore, Maryland, USA, 1995. ACM Press.

[43] W.A. Woods and J.G. Schmolze. The KL-ONE family. *Computer & Mathematics with Applications*, 23(2–5):133–177, 1991.

A Appendix A: Intensional relationships in the Common Thesaurus

Relationships between classes

```
University Department RT University Research_Staff
University Section RT University Research_Staff
University Room RT University Section
University School_Member NT Computer_Science CS_Person
University Room RT Computer_Science Course
University Research_Staff RT Computer_Science Course
Computer_Science Student NT Computer_Science CS_Person
Computer_Science Professor NT Computer_Science CS_Person
Computer_Science Course RT tax_position_xml Student
Computer_Science Location RT Computer_Science Division
Computer_Science Course RT Computer_Science Student
Computer_Science Division RT Computer_Science Professor
Computer_Science Professor RT Computer_Science Course
Computer_Science Course SYN University Section
Computer_Science Student SYN tax_position_xml Student
Computer_Science Student SYN University School_Member
Computer_Science Course RT University School_Member
Computer_Science Student RT University Section
Computer_Science Professor RT University Section
tax_position_xml Student SYN University School_Member
tax_position_xml Student NT Computer_Science CS_Person
tax_position_xml Student RT tax_position_xml ListOfStudent
tax_position_xml ListOfStudent RT University School_Member
tax_position_xml ListOfStudent RT Computer_Science Student
```

Relationships between attributes or between class and attribute

```
University s_code SYN University section_code
University Section SYN Computer_Science course_name
University Room SYN University room_code
University School_Member SYN tax_position_xml s_code
University Department SYN University dept_name
Computer_Science year SYN University year
Computer_Science first_name NT tax_position_xml name
Computer_Science first_name NT University name
Computer_Science first_name RT Computer_Science last_name
Computer_Science Location BT Computer_Science city
Computer_Science Professor RT tax_position_xml school_name
Computer_Science Professor RT University school
Computer_Science last_name NT tax_position_xml name
Computer_Science last_name NT University name
Computer_Science Location BT Computer_Science country
Computer_Science Course SYN Computer_Science course_name
Computer_Science email SYN tax_position_xml email
Computer_Science email SYN University e_mail
Computer_Science Student SYN tax_position_xml s_code
Computer_Science CS_Person BT tax_position_xml s_code
```

```
tax_position_xml Student SYN tax_position_xml s_code
tax_position_xml school_name SYN University School
tax_position_xml name SYN University name
tax_position_xml email SYN University e_mail
```

B Appendix B: Syntax generated by parseYOut

⟨OdlSpecification⟩	::=	⟨Definition⟩ .
	|	⟨Definition⟩ ;
	|	⟨Definition⟩ ; ⟨OdlSpecification⟩
⟨Definition⟩	::=	⟨TypeDcl⟩
	|	⟨ConstDcl⟩
	|	⟨ExceptDcl⟩
	|	⟨Interface⟩
	|	⟨RuleDcl⟩
	|	⟨ExtRuleDcl⟩
	|	⟨ThesaurusRelation⟩
	|	⟨Module⟩
	|	⟨error⟩
⟨TypeDcl⟩	::=	**typedef** ⟨TypeDeclarator⟩
	|	⟨ConstrTypeSpec⟩
⟨TypeDeclarator⟩	::=	⟨TypeSpec⟩ ⟨Declarators⟩
⟨Declarators⟩	::=	⟨Declarator⟩
	|	⟨Declarators⟩ , ⟨Declarator⟩
⟨Declarator⟩	::=	⟨Identifier⟩
	|	⟨Identifier⟩ ⟨ArraySizeList⟩
⟨ArraySizeList⟩	::=	⟨FixedArraySize⟩
	|	⟨ArraySizeList⟩ ⟨FixedArraySize⟩
⟨TypeSpec⟩	::=	⟨SimpleTypeSpec⟩
	|	⟨ConstrTypeSpec⟩
⟨SimpleTypeSpec⟩	::=	⟨BaseTypeSpec⟩
	|	⟨TemplateTypeSpec⟩
	|	⟨Identifier⟩

⟨BaseTypeSpec⟩ ::= ⟨FloatingPtType⟩
| ⟨IntegerType⟩
| ⟨CharType⟩
| ⟨BooleanType⟩
| ⟨OctetType⟩
| ⟨RangeType⟩
| ⟨AnyType⟩

⟨FloatingPtType⟩ ::= **float**
| **double**

⟨IntegerType⟩ ::= ⟨LongIntType⟩
| **short**
| **unsigned long**
| **unsigned short**

⟨LongIntType⟩ ::= **integer**
| **int**
| **long**

⟨CharType⟩ ::= **char**

⟨BooleanType⟩ ::= **boolean**

⟨OctetType⟩ ::= **octet**

⟨RangeType⟩ ::= **range** { ⟨RangeSpecifier⟩ }

⟨RangeSpecifier⟩ ::= ⟨SignedIntegerLiteral⟩ , ⟨SignedIntegerLiteral⟩
| ⟨SignedIntegerLiteral⟩ , + **infinite**
| **- infinite** , ⟨SignedIntegerLiteral⟩

⟨AnyType⟩ ::= **any**

⟨TemplateTypeSpec⟩ ::= ⟨ArrayType⟩
| ⟨StringType⟩
| ⟨CollectionType⟩

⟨ArrayType⟩ ::= **array** < ⟨SimpleTypeSpec⟩ , ⟨IntegerLiteral⟩ >

⟨$$1⟩ ::=

⟨ArrayType⟩ ::= **array** < ⟨SimpleTypeSpec⟩ > ⟨$$1⟩ **sequence** < ⟨SimpleTypeSpec⟩ , ⟨IntegerLiteral⟩ >

⟨StringType⟩ ::= **string** < ⟨IntegerLiteral⟩ >
| **string**

⟨CollectionType⟩ ::= ⟨AttrCollectionSpecifier⟩ < ⟨SimpleTypeSpec⟩ >

⟨AttrCollectionSpecifier⟩ ::= **set**
| **list**
| **bag**

⟨ConstrTypeSpec⟩ ::= ⟨StructType⟩
| ⟨UnionType⟩
| ⟨EnumType⟩

⟨StructType⟩ ::= **struct** ⟨Identifier⟩ { ⟨MemberList⟩ }

⟨MemberList⟩ ::= ⟨Member⟩
| ⟨MemberList⟩ ⟨Member⟩

⟨Member⟩ ::= ⟨TypeSpec⟩ ⟨Declarators⟩ ;

⟨UnionType⟩ ::= **union** ⟨Identifier⟩ **switch** (⟨SwitchTypeSpec⟩) { ⟨SwitchBody⟩ }

⟨SwitchTypeSpec⟩ ::= ⟨IntegerType⟩
| ⟨CharType⟩
| ⟨BooleanType⟩
| ⟨EnumType⟩
| ⟨ScopedName⟩
| ⟨RangeType⟩

⟨SwitchBody⟩ ::= ⟨Case⟩
| ⟨Case⟩ ⟨SwitchBody⟩

⟨Case⟩ ::= ⟨CaseLabelList⟩ ⟨ElementSpec⟩ ;

⟨CaseLabelList⟩ ::= ⟨CaseLabel⟩
| ⟨CaseLabel⟩ ⟨CaseLabelList⟩

⟨CaseLabel⟩ ::= **case** ⟨ConstExp⟩ **:**
| **default :**

⟨ElementSpec⟩ ::= ⟨TypeSpec⟩ ⟨Declarator⟩

⟨EnumType⟩ ::= **enum** ⟨Identifier⟩ { ⟨EnumeratorList⟩ }

⟨EnumeratorList⟩ ::= ⟨Enumerator⟩
| ⟨EnumeratorList⟩ , ⟨Enumerator⟩

⟨Enumerator⟩ ::= ⟨Identifier⟩

⟨ConstExp⟩ ::= ⟨OrExpr⟩

⟨OrExpr⟩ ::= ⟨XOrExpr⟩
| ⟨OrExpr⟩ | ⟨XOrExpr⟩

⟨XOrExpr⟩ ::= ⟨AndExpr⟩
| ⟨XOrExpr⟩ ^ ⟨AndExpr⟩

⟨AndExpr⟩ ::= ⟨ShiftExpr⟩
| ⟨AndExpr⟩ & ⟨ShiftExpr⟩

⟨ShiftExpr⟩ ::= ⟨AddExpr⟩
| ⟨ShiftExpr⟩ >> ⟨AddExpr⟩
| ⟨ShiftExpr⟩ << ⟨AddExpr⟩

⟨AddExpr⟩	::=	⟨MultExpr⟩
	|	⟨AddExpr⟩ + ⟨MultExpr⟩
	|	⟨AddExpr⟩ - ⟨MultExpr⟩
⟨MultExpr⟩	::=	⟨UnaryExpr⟩
	|	⟨MultExpr⟩ * ⟨UnaryExpr⟩
	|	⟨MultExpr⟩ / ⟨UnaryExpr⟩
	|	⟨MultExpr⟩ \% ⟨UnaryExpr⟩
⟨UnaryExpr⟩	::=	⟨Signes⟩ ⟨PrimaryExpr⟩
	|	⟨PrimaryExpr⟩
⟨PrimaryExpr⟩	::=	⟨Identifier⟩
	|	⟨IntegerLiteral⟩
	|	⟨FloatingPtLiteral⟩
	|	(⟨OrExpr⟩)
	|	(⟨error⟩)
⟨ConstDcl⟩	::=	**const** ⟨StringType⟩ ⟨Identifier⟩ = ⟨StringLiteral⟩
	|	**const** ⟨CharType⟩ ⟨Identifier⟩ = ⟨CharacterLiteral⟩
	|	**const** ⟨IntegerType⟩ ⟨Identifier⟩ = ⟨ConstExp⟩
	|	**const** ⟨FloatingPtType⟩ ⟨Identifier⟩ = ⟨ConstExp⟩
⟨SignedIntegerLiteral⟩	::=	⟨IntegerLiteral⟩
	|	⟨Signes⟩ ⟨IntegerLiteral⟩
⟨SignedFloatingPtLiteral⟩	::=	⟨Signes⟩ ⟨FloatingPtLiteral⟩
	|	⟨FloatingPtLiteral⟩
⟨Signes⟩	::=	-
	|	+
⟨ExceptDcl⟩	::=	**exception** ⟨Identifier⟩ { ⟨OptMemberList⟩ }
⟨OptMemberList⟩	::=	
	|	⟨MemberList⟩
⟨ScopedName⟩	::=	⟨Identifier⟩
	|	**::** ⟨Identifier⟩
	|	⟨ScopedName⟩ **::** ⟨Identifier⟩
⟨Interface⟩	::=	⟨InterfaceDcl⟩
⟨IntView⟩	::=	**interface**
	|	**view**

⟨InterfaceDcl⟩	::=	⟨IntView⟩ ⟨Identifier⟩ **:** ⟨InheritanceSpec⟩ ⟨OptTypePropertyList⟩ ⟨OptPersistenceDcl⟩ ⟨SingleInterfaceBody⟩
	|	⟨IntView⟩ ⟨Identifier⟩ **:** ⟨InheritanceSpec⟩ ⟨OptTypePropertyList⟩ ⟨OptPersistenceDcl⟩ ⟨SingleInterfaceBody⟩ ⟨InterfaceBodyUnionList⟩
	|	⟨IntView⟩ ⟨Identifier⟩ ⟨OptTypePropertyList⟩ ⟨OptPersistenceDcl⟩ ⟨SingleInterfaceBody⟩
	|	⟨IntView⟩ ⟨Identifier⟩ ⟨OptTypePropertyList⟩ ⟨OptPersistenceDcl⟩ ⟨SingleInterfaceBody⟩ ⟨InterfaceBodyUnionList⟩
⟨InheritanceSpec⟩	::=	⟨Identifier⟩
	|	⟨InheritanceSpec⟩ **,** ⟨Identifier⟩
⟨OptTypePropertyList⟩	::=	
	|	**(** ⟨OptSourceSpec⟩ ⟨OptExtentSpec⟩ ⟨OptKeySpec⟩ ⟨OptCandKeySpec⟩ ⟨OptForKeySpec⟩ **)**
⟨OptSourceSpec⟩	::=	
	|	**source** ⟨SourceType⟩ ⟨Identifier⟩
⟨SourceType⟩	::=	**relational**
	|	**nfrelational**
	|	**object**
	|	**file**
	|	**semistructured**
⟨OptExtentSpec⟩	::=	
	|	**extent** ⟨ListExtent⟩
⟨ListExtent⟩	::=	⟨Identifier⟩
	|	⟨ListExtent⟩ **,** ⟨Identifier⟩
⟨OptKeySpec⟩	::=	
	|	**key** ⟨Key⟩
⟨OptCandKeySpec⟩	::=	
	|	⟨CandKeySpecList⟩
⟨CandKeySpecList⟩	::=	⟨CandKeySpec⟩
	|	⟨CandKeySpecList⟩ ⟨CandKeySpec⟩
⟨CandKeySpec⟩	::=	**candidate_key** ⟨Identifier⟩ ⟨Key⟩
⟨OptForKeySpec⟩	::=	
	|	⟨ForKeySpecList⟩
⟨ForKeySpecList⟩	::=	⟨ForKeySpec⟩
	|	⟨ForKeySpecList⟩ ⟨ForKeySpec⟩
⟨ForKeySpec⟩	::=	**foreign_key** **(** ⟨ForeignKeyList⟩ **)** **references** ⟨Identifier⟩ ⟨OptRefKeyList⟩
⟨OptRefKeyList⟩	::=	
	|	**(** ⟨ForeignKeyList⟩ **)**

⟨ForeignKeyList⟩	::=	⟨Identifier⟩
	|	⟨ForeignKeyList⟩ **,** ⟨Identifier⟩
⟨Key⟩	::=	**(** ⟨PropertyList⟩ **)**
⟨PropertyList⟩	::=	⟨PropertyName⟩
	|	⟨PropertyList⟩ **,** ⟨PropertyName⟩
⟨PropertyName⟩	::=	⟨Identifier⟩
⟨OptPersistenceDcl⟩	::=	
	|	**persistent**
	|	**transient**
⟨InterfaceBodyUnionList⟩	::=	⟨InterfaceBodyUnion⟩
	|	⟨InterfaceBodyUnionList⟩ ⟨InterfaceBodyUnion⟩
⟨InterfaceBodyUnion⟩	::=	**union** ⟨Identifier⟩ ⟨SingleInterfaceBody⟩
⟨SingleInterfaceBody⟩	::=	**{** ⟨InterfaceBody⟩ **}**
⟨InterfaceBody⟩	::=	⟨Export⟩ **;**
	|	⟨Export⟩ **;** ⟨InterfaceBody⟩
⟨Export⟩	::=	⟨TypeDcl⟩
	|	⟨ConstDcl⟩
	|	⟨ExceptDcl⟩
	|	⟨AttrDcl⟩
	|	⟨RelDcl⟩
	|	⟨OpDcl⟩
⟨AttrDcl⟩	::=	⟨OptReadonly⟩ **attribute** ⟨DomainType⟩ ⟨AttributeName⟩ ⟨OptFixedArraySize⟩ ⟨OptMappingRuleDcl⟩
⟨OptReadonly⟩	::=	
	|	**readonly**
⟨AttributeName⟩	::=	⟨Identifier⟩
	|	⟨Identifier⟩ **?**
⟨DomainType⟩	::=	⟨SimpleTypeSpec⟩
	|	⟨StructType⟩
	|	⟨EnumType⟩
⟨OptFixedArraySize⟩	::=	
	|	⟨FixedArraySize⟩
⟨FixedArraySize⟩	::=	**[** ⟨IntegerLiteral⟩ **]**
⟨OptMappingRuleDcl⟩	::=	
	|	⟨MappingRuleDcl⟩
⟨MappingRuleDcl⟩	::=	**mapping rule** ⟨MapRuleList⟩
⟨MapRuleList⟩	::=	⟨MapRule⟩
	|	⟨MapRuleList⟩ **,** ⟨MapRule⟩

⟨MapRule⟩	::=	⟨LocalAttributeName⟩
	|	⟨DefaultValue⟩
	|	⟨MapAndExpression⟩
	|	⟨MapUnionExpression⟩
⟨LocalAttributeName⟩	::=	⟨LocalClassName⟩ **.** ⟨Identifier⟩
⟨LocalClassName⟩	::=	⟨Identifier⟩ **.** ⟨Identifier⟩
⟨DefaultValue⟩	::=	⟨Identifier⟩ **.** ⟨Identifier⟩ = ⟨StringLiteral⟩
⟨MapAndExpression⟩	::=	(⟨MapAndList⟩ **and** ⟨LocalAttributeName⟩)
⟨MapAndList⟩	::=	⟨LocalAttributeName⟩
	|	⟨MapAndList⟩ **and** ⟨LocalAttributeName⟩
⟨MapUnionExpression⟩	::=	(⟨MapUnionList⟩ **union** ⟨LocalAttributeName⟩ **on** ⟨Identifier⟩)
⟨MapUnionList⟩	::=	⟨LocalAttributeName⟩
	|	⟨MapUnionList⟩ **union** ⟨LocalAttributeName⟩
⟨RelDcl⟩	::=	**relationship** ⟨TargetOfPath⟩ ⟨Identifier⟩ **inverse** ⟨InverseTraversalPath⟩ ⟨OptOrderBy⟩
⟨TargetOfPath⟩	::=	⟨Identifier⟩
	|	⟨RelCollectionType⟩ < ⟨Identifier⟩ >
⟨RelCollectionType⟩	::=	**set**
	|	**list**
⟨InverseTraversalPath⟩	::=	⟨Identifier⟩ **::** ⟨Identifier⟩
⟨OptOrderBy⟩	::=	
	|	{ **order_by** ⟨ScopedNameList⟩ }
⟨ScopedNameList⟩	::=	⟨ScopedName⟩
	|	⟨ScopedNameList⟩ **,** ⟨ScopedName⟩
⟨OpDcl⟩	::=	⟨OptOneway⟩ ⟨OperTypeSpec⟩ ⟨Identifier⟩ ⟨ParameterDcls⟩ ⟨OptRaisesExpr⟩ ⟨OptContextExpr⟩
⟨OptOneway⟩	::=	
	|	**oneway**
⟨OperTypeSpec⟩	::=	⟨SimpleTypeSpec⟩
	|	**void**
⟨ParameterDcls⟩	::=	(⟨ParamDclList⟩)
	|	( )
⟨ParamDclList⟩	::=	⟨ParamDcl⟩
	|	⟨ParamDclList⟩ **,** ⟨ParamDcl⟩
⟨ParamDcl⟩	::=	⟨ParamAttribute⟩ ⟨SimpleTypeSpec⟩ ⟨Declarator⟩

⟨ParamAttribute⟩	::=	
	|	**in**
	|	**out**
	|	**inout**
⟨OptRaisesExpr⟩	::=	
	|	**raises** (⟨ScopedNameList⟩)
⟨OptContextExpr⟩	::=	
	|	**context** (⟨StringLiteralList⟩)
⟨StringLiteralList⟩	::=	⟨StringLiteral⟩
	|	⟨StringLiteralList⟩ **,** ⟨StringLiteral⟩
⟨ExtRuleDcl⟩	::=	**extrule** ⟨Identifier⟩ ⟨ExtRuleSpec⟩
⟨ExtRuleSpec⟩	::=	⟨ForAll⟩ ⟨Identifier⟩ **in** ⟨ExtRuleType⟩
⟨ExtRuleType⟩	::=	⟨ExtRuleIsa⟩
	|	⟨ExtRuleBottom⟩
⟨ExtRuleBottom⟩	::=	(⟨LocalClassName⟩ **and** ⟨LocalClassName⟩) **then** ⟨Identifier⟩ **in bottom**
⟨ExtRuleIsa⟩	::=	⟨LocalClassName⟩ **then** ⟨Identifier⟩ **in** ⟨LocalClassName⟩
⟨RuleDcl⟩	::=	**rule** ⟨Identifier⟩ ⟨RuleSpec⟩
⟨RuleSpec⟩	::=	⟨ForAll⟩ ⟨Identifier⟩ **in** ⟨Identifier⟩ **:** ⟨RuleBodyList⟩ **then** ⟨RuleBodyList⟩
	|	{ **case of** ⟨Identifier⟩ **:** ⟨CaseList⟩ }
⟨ForAll⟩	::=	**for all**
	|	**forall**
⟨RuleBodyList⟩	::=	(⟨RuleBodyList⟩)
	|	⟨RuleBody⟩
	|	⟨RuleBodyList⟩ **and** ⟨RuleBody⟩
⟨$$2⟩	::=	
⟨RuleBody⟩	::=	⟨DottedName⟩ ⟨RuleConstOp⟩ ⟨OptRuleCast⟩ ⟨LiteralValue⟩ ⟨$$2⟩ ⟨DottedName⟩ ⟨RuleConstOp⟩ ⟨OptRuleCast⟩ ⟨DottedName⟩
	|	⟨DottedName⟩ **in** ⟨DottedName⟩
	|	⟨ForAll⟩ ⟨Identifier⟩ **in** ⟨DottedName⟩ **:** ⟨RuleBodyList⟩
	|	**exists** ⟨Identifier⟩ **in** ⟨DottedName⟩ **:** ⟨RuleBodyList⟩
	|	⟨DottedName⟩ = ⟨SimpleTypeSpec⟩ ⟨Identifier⟩ (⟨DottedLiteralList⟩)
⟨DottedLiteralList⟩	::=	⟨DottedLiteral⟩
	|	⟨DottedLiteralList⟩ **,** ⟨DottedLiteral⟩
⟨DottedLiteral⟩	::=	⟨OptSimpleTypeSpec⟩ ⟨DottedName⟩
	|	⟨OptSimpleTypeSpec⟩ ⟨LiteralValue⟩

⟨OptSimpleTypeSpec⟩	::=	
	|	⟨SimpleTypeSpec⟩
⟨RuleConstOp⟩	::=	`=`
	|	`>=`
	|	`<=`
	|	`<`
	|	`>`
⟨LiteralValue⟩	::=	⟨SignedFloatingPtLiteral⟩
	|	⟨SignedIntegerLiteral⟩
	|	⟨CharacterLiteral⟩
	|	⟨StringLiteral⟩
⟨DottedName⟩	::=	⟨Identifier⟩
	|	⟨DottedName⟩ `.` ⟨Identifier⟩
⟨OptRuleCast⟩	::=	
	|	`(` ⟨SimpleTypeSpec⟩ `)`
⟨CaseList⟩	::=	⟨CaseSpec⟩
	|	⟨CaseList⟩ ⟨CaseSpec⟩
⟨CaseSpec⟩	::=	⟨LiteralValue⟩ `:` ⟨DottedName⟩
⟨ThesaurusRelation⟩	::=	⟨LocalAttributeName⟩ ⟨ThesRelType⟩ ⟨LocalAttributeName⟩
	|	⟨LocalClassName⟩ ⟨ThesRelType⟩ ⟨LocalClassName⟩
	|	⟨LocalClassName⟩ ⟨ThesRelType⟩ ⟨LocalAttributeName⟩
	|	⟨LocalAttributeName⟩ ⟨ThesRelType⟩ ⟨LocalClassName⟩
⟨ThesRelType⟩	::=	**syn**
	|	**bt**
	|	**nt**
	|	**rt**
⟨Module⟩	::=	**module** ⟨Identifier⟩ `{` ⟨OdlSpecification⟩ `}`