Πολυτεχνείο Κρήτης

OWL & SPARQL based Mediator for Integration of OWL/RDF Information Sources

A thesis submitted to the Department of Electronic & Computer Engineering of the Technical University of Crete
In partial fulfillment of the requirements for the Diploma of Electronic and Computer Eng

By Antonia Georgaraki

Contents

To my parents and my Friends

Acknowledgements

I would like to thank my supervisor, Prof. Stavros Christodoulakis, for his encouragement and his continuous guidance and support throughout my research. I would like also to thank him for the important experiences he offered me during my stay at the Laboratory of Distributed Multimedia Information Systems and Applications (MUSIC).

I would like to express my gratitude to the readers of this thesis Mr. Antonios Deligiannakis and Mr. Michail G. Lagoudakis for the time they devoted and their critical evaluation.

My appreciation goes to Chrisa Tsinaraki and Nektarios Gioldasis for their supervision and their valuable help regarding this thesis. I am also grateful to Nikos Bikakis for being always ready to offer, as well as to the rest staff of the Laboratory for the pleasant environment they provided and for their cooperation.

Finally, I wish to thank my parents and my sister Dimitra, as well as all my friends for helping me get through the difficult times, and for all the emotional support, entertainment, and caring they provided.

Antonia Georgaraki
Technical University of Crete
July 2010

List Of Examples

List Of Tables

List of Figures

Abstract

The World Wide Web (WWW), in its current state, is a huge library of interlinked documents that are delivered over the Internet in order to be presented to the end users. It has been based on the hypertext systems principles, but the main differences are its distributed nature and the collaborative environment it offers, which allows anyone to contribute to the WWW.

The Semantic Web is a semantic-based evolution of the World Wide Web in which the web information and service semantics are clearly defined, making it possible for the web to understand and satisfy the content-related requests of both people and machines. It is based on the vision of the Web as a universal medium for data, information, and knowledge exchange, which has been expressed by the World Wide Web Consortium director Sir Tim Berners-Lee's [BerHenLas01].

At its core, the semantic web comprises a set of design principles, collaborative working groups, and a variety of enabling technologies. Some elements of the semantic web are expressed as prospective future possibilities that are yet to be implemented or realized. Other elements of the semantic web are expressed in formal specifications. Some of these include the Resource Description Framework (RDF), a variety of data interchange formats (e.g. RDF/XML), and notations such as the RDF Schema (RDFS) and the Web Ontology Language (OWL), all of which are intended to provide a formal description of concepts, terms, and relationships within a given knowledge domain.

The need for Semantic Interoperability, resolves to a point the semantic web, which allows transforming the World Wide Web into rich semantic forms with the use of semantic technologies. The most common language for semantic-based queries is SPARQL (Simple Protocol and RDF Query Language) [Be&al08], which has recently become a W3C recommendation. Finally, the need for homogeneous access to heterogeneous information based on different ontologies-vocabularies used by different communities, is nowadays bigger than ever.

Our goal is to develop a Mediation System (from now on we will refer to it as Mediator) to be used in the above mentioned situations. The Mediator is based on OWL and SPARQL for the integration of OWL/RDF based information sources, in order to achieve semantic

interoperability between heterogeneous information sources using the offered knowledge and functionality.

Key Words: SPARQL, OWL/RDF, Mediator, Ontology, Semantic Interoperability, Mappings, Articulations, Query Processing, Semantic Data Integration.

CHAPTER 1
Introduction

1.1 The Unified Information Access Challenge

Achieving uniform access to DL objects in heterogeneous digital libraries is a difficult and multi-fold problem and at its very core requires dealing with the problem of syntactic and semantic interoperability. Currently, there exist many interoperability techniques, with quite varying potential for resolving the structural and semantic heterogeneities that may exist between metadata stored in distinct repositories. In the domain of digital library information systems, interoperability is defined by [Bak02] as the potential for metadata to cross boundaries between different information contexts. Other authors from the same domain define interoperability as being able to exchange metadata between two or more systems without loss or with minimal loss of information and without any special effort on either system [Alc00], or as the ability to apply a single query syntax over descriptions expressed in multiple descriptive formats [Hun01].

In order to achieve metadata interoperability the most obvious approach is to adhere to common metadata standards. Lots of metadata standards exist either for specific application domains or domain-independent. In [Has10], a comprehensive list of metadata standards is presented. However, in real-world environments, institutions often do not adhere to the standards. Attempts to find an agreement for a standard often results in semantically weak minimum consensus schemes (e.g., the Dublin Core [Dc06]) or models with extensive and complex semantic domains (e.g. the CIDOC Conceptual Reference Model (CRM) [Iso06]). Moreover, it is not practicable for institutions to agree on a certain model or apply an existing standard because they often already have their own proprietary metadata models. Another approach to achieving interoperability is not to agree on a common model but on a common meta-model (e.g. the Meta Object Facility [Omg06]) or a common global conceptual model (e.g. CIDOC/CRM [Iso06], SUMO [Sum10], etc.).

Often, especially in settings where the incentives for an agreement on standards are weak, neither model nor meta-model agreements are suitable interoperability techniques. The digital libraries domain, for instance, is such a domain: there is no central authority that can

impose a metadata standard on all the digital libraries. Such settings require other means for reconciling heterogeneities among models.

If the metadata schemes are expressed in different schema definition languages, mappings on the language level are required. Because of the substantial structural and semantic discrepancies among the schema definition languages, translation from one language into another one causes loss of valuable semantic information (e.g. XS2OWL [Tsi07]). Model mapping for reconciling heterogeneities gains more and more attention during the last years. With the emergence of the Semantic Web, Ontologies are used to conceptually represent domain knowledge as well as different metadata models. Thus, ontology mapping and query mediation over the mapped ontologies has become a major research challenge.

Ontology mapping is the task of relating the vocabulary of two ontologies by defining a set of correspondences, in order to bridge the semantic overlap between them. The correspondences between different entities of the two ontologies are typically expressed by using some axioms expressed in a specific mapping language. The mapping discovery between two ontologies is a process which can be achieved: a) manually, defined by an expert, b)automatically, by using various ontology matching algorithms that compute the similarity between different terms [Scv05] and [Euz07], or c) semi-automatically, by using matching algorithms and techniques, as well as the user feedback. Up to now, there are many proposed strategies ([Ehr04], [Pan05], etc.), as well as a lot of systems and tools, that perform mapping discovery, several of which are compared and evaluated in [Euz07], [Kal05], [Cho06], [Euz04a]. Recently, there has been an increasing interest on schema mapping tools [Mel05] [Bon05] that can infer ever more complex mappings [Raf08]. The mappings may range from virtual transformations [STYLS], to highly complex structures [Ber07]. MAPONTO [MAPO] is such a mapping tool for inferring mappings between schemas and ontologies. The problem of ontology matching has also been studied in many research projects, e.g., Knowledge Web [KWEB] and Open Knowledge [OKW] and a relatively recent survey of the state-of-the-art can be found in [Shv05]. Other similar works have been developed in Trento [Gui05], [Ave05]. Although the automatic or semi-automatic techniques provide satisfactory results, it is unlikely to be compared with mappings which are defined manually.

Except of the mapping discovery, the mapping representation is a very important issue for a system that implements ontology mediation. A mapping representation language should

satisfy the following criteria: web compatibility, language independence, simplicity, expressiveness, purpose independence, executability, and task mediation. Although there are many languages [Bec04],[Bou03], [Hor04], [Euz04b], [Mae02], [Sch09], etc.) which can be used to represent mappings between two ontologies, very few satisfy the previous criteria. The comparison of some of these languages and formats is available in [Euz07]. An interesting approach for defining ontology mappings by using the SPARQL [Pru08] query language, and especially an extension of it, has been presented in [Pol07].

Query mediation requires SPARQL query translation and decomposition. Query translation may be used to translate SPARQL queries into other query languages. Two interesting approaches that perform SPARQL query translation into SQL queries have been proposed in [Ell09] and [Che09], resulting to complete and semantic preserving query translation. Similarly with SPARQL-to-SQL proposed methods, Bikakis et. al [Bik09] present a method and a framework which performs SPARQL query translation into XQuery queries, in order to achieve data integration by querying XML data through SPARQL queries.

Regarding query translation related to posing a query over different ontologies, Akahani et. al [Aka03] proposed approximate query reformulation for multiple ontologies, by providing a theoretical perspective. In their approach any specific context (e.g. SPARQL) is defined, while the mappings between ontologies that can be exploited, or specific algorithms for the reformulation of a query are not presented. Makris et al, [Mak09] presents a more complete approach for mediating SPARQL queries over different ontologies. In their on-going work ontology mappings are examined in the context of their executability (i.e. whether they can be used in query translation or not) and based on them specific SPARQL query translation algorithms are investigated. A similar approach is followed by Correndo et al. [Cor10] who present an algorithm for achieving RDF data mediation over linked data based on SPARQL query rewriting. However, in order to define the mappings between two ontologies, they propose to use transformations between RDF structures (i.e. graphs) and not description logic semantics.

Query decomposition in mediation frameworks refers to decomposing initial queries into sub-queries which are submitted to the individual sources. Quilitz et. al [Qui08] proposed an engine that decomposes a query into sub-queries, each of which can be answered by an individual service, giving the impression that the user queries one single RDF graph. Moreover, this engine uses query rewriting in order to speed up the query execution,

resulting to query optimization. Query decomposition is also used by an implemented system, proposed by Benslimane et. al [Ben08] in order to query heterogeneous information sources. On top of the decomposition of SPARQL queries into SPARQL sub-queries, this system seems to translate the resulting sub-queries into SQL sub-queries but any algorithm or method is provided.

A mediator architecture is a common approach in information integration systems [44]. Mediated query systems represent a uniform data access solution by providing a single point for querying access to various data sources. A mediator contains a global query processor which is used to send sub-queries to local data sources. The local query results are then combined and returned back to the query processor. Its main benefit is that the query formulation process becomes independent of the mediated data sources requiring from end-users to be aware only of their own conceptualization of the knowledge domain.

1.2 Adopted Approach

In the context of this diploma thesis, we are building a SPARQL based query Mediation System, which offers unified and integrated access to data from multiple, heterogeneous information sources.

In particular, the system provides interoperability of OWL/RDF information sources, and allows the combination of the query results received from distributed federated databases of different type, structure and data model using SPARQL queries.

The mediator uses mappings between the OWL [3] ontology of the mediator (global/main ontology) and the federated site ontologies (local/target ontologies). SPARQL [35] queries posed over the mediator, are decomposed, transformed and rewritten in order to be submitted over the federated sites. The SPARQL queries are locally evaluated and the results are returned to the mediator site. Every local source is based on its own ontology and provides a SPARQL interface. The mediator has the appropriate correspondences, more technically *term mappings* from the terms of the global ontology to the terms of the local ontologies, which have been defined from some domain experts or have been automatically created. Reclaiming all these kinds of correspondence and all the types of mappings that we

have been given, the mediation system builds and uses appropriate query translators, from the production of specified and specialized queries for each source. These queries are being sent to the distributed, autonomous information sources, in order to collect and present to the user the final results.

Finally, it is necessary to mention some of the possibilities that our system offers, in order to be original, usable and convenient to the user. The mediation system is fully user- configured and it can be parameterized in terms of:

- the main/global ontology,
- the local/target ontologies,
- the kind and the number of information sources that can be served,
- the sub-systems of SPARQL query translation and
- the level of semantics conservation of the original user query.

1.3 Main Contributions

The web of data is heterogeneous, distributed and highly structured. Querying mechanisms in this environment have to overcome the problems arising from the heterogeneous and distributed nature of data, and they have to allow the expression of complex and structured queries. The ontology mappings and SPARQL query mediation presented in this thesis aim to satisfy those requirements in the Semantic Web environment. The mediator uses mappings between the global OWL ontology of the mediator and the local ontologies of the federated knowledge bases. SPARQL queries of posed over the mediator by end users and applications, are decomposed, relaxed (if it is necessary) and rewritten in order to be submitted to the federated sources. The rewritten SPARQL queries are locally evaluated and the results are returned to the mediator.

Four aspects of this system are discussed in this thesis:

- A formal system for describing executable ontology mappings that satisfy real-world requirements and can be exploited in query translation.
- A complete set of SPARQL query rewriting and transformation algorithms that allow SPARQL queries, which are expressed based on the global ontology of the mediator, to be rewritten in terms of the local/target ontologies. These algorithms are also semantics preserving.

- A clear way of result format representation and presentation, based at first on each information source and at a second level on a unified, final result set. The final result set is extracted and presented with extra information about the kind of the mappings used, the number of variables that contain results and ranked based on the above statistics.
- A user- friendly GUI (graphical user interface) based on the system implementation of the proposed methods, for the communication of the Mediator with the end users while posing the original user queries.

Our current research focuses on the exploitation of different kind of mappings by the query rewriting process, as well as on evaluating the system performance and developing methodologies for the optimization, normalization and relaxation of the query mediation process.

1.4 Thesis Structure

The rest of this diploma thesis is structured as follows:

In Chapter 2, the related to this diploma thesis research is presented. In particular, related research and systems are presented as well as the technical parameters and the different technologies, concepts and theories in the field of mediation of heterogeneous, distributed information sources.

In Chapter 3, the Reference Architecture of the OWL/SPARQL Mediator is described, that was developed in the context of this thesis and is used to extract results from distributed OWL/RDF information Sources. Furthermore, in this chapter we introduce the fundamental concepts and definitions that concern the design and implementation of the system.

In Chapter 4, the physical system concerning the Ontology–based Mediation is analyzed.. We pay particular attention on the methods concerning the Ontology Mapping Management, the Query Normalization and Relaxation, the SPARQL query Translation and Execution and the Synthesis of the Query Results.

Chapter 5 includes the requirements analysis of the application and the system implementation. In this chapter, an architectural overview is presented and the system

components are defined and determined. Some examples of the real world application system are also presented in this chapter.

Chapter 6 provides the conclusions of our work and presents our future research directions.

CHAPTER 2
Related Work

2.1 Introduction

In this chapter, a review of the related research that has been done will be presented. In particular, concepts that have been researched during the implementation of the thesis, terms and concepts that are used in the next chapters and other environments relevant to the one implemented in the contexts of the current thesis. Also, relevant work and papers with different models will be used.

2.2 Related Technologies

In this chapter we present the standards used in this thesis, as well as the technologies used for the implementation of our SPARQL query rewriting framework. More specifically, Section 2.2.1 presents RDF/S, the standard language for representing information about resources in the World Wide Web. Section 2.2.2 presents OWL, the standard language for defining and instantiating Web ontologies. Section 2.2.3 presents SPARQL, the standard query language for RDF. Finally, Section 2.2.4 presents Jena, an open source Java framework for building Semantic Web applications.

From this point forward we consider the following namespaces:

• RDF namespace: rdf = http://www.w3.org/1999/02/22-rdf-syntax-ns#

• RDF Schema namespace: rdfs = http://www.w3.org/2000/01/rdf-schema#

• XML Schema namespace: xsd = http://www.w3.org/2001/XMLSchema#

• OWL namespace: owl = http://www.w3.org/2002/07/owl#

• Bookstore namespace (used in examples): ns = http://example.org/Bookstore#

2.2.1 Resource Description Framework and Schema Language (RDF/S)

The Resource Description Framework (RDF) [28] is the standard language for representing information about resources in the World Wide Web. RDF is based on the idea of identifying things using Web identifiers (called Uniform Resource Identifiers, or URIs). The atomic constructs of RDF are statements, which are triples (subject, predicate, object) consisting of the resource (the subject) being described, a property (the predicate), and a property value (the object).

Example 2.1. The assertion of the following RDF triples mean that the resource book1, under the namespace ns, has a property title under the same namespace, with value “Database Systems”.

```
@prefix ns: <http://example.org/Bookstore#> .
ns:book1 ns:title "Database Systems" .
```

It is worth to mention that in triple format the `@prefix keyword` associates a prefix label with a URI. A prefixed name is a prefix label and a local part, separated by a colon “:”. A prefixed name is mapped to a URI by concatenating the URI associated with the prefix and the local part.

RDF can be expressed in a variety of formats including RDF/XML. Although, in this thesis we use the triple form.

Data values in RDF are represented by so-called *literals*. The value of every literal is generally described by a sequence of characters. The interpretation of such sequences is determined based on a given datatype (XML Schema datatype mainly). In triple form, the syntax for literals is a string (enclosed in double quotes, "*: : :* "), with an optional datatype URI (introduced by ^^).

Example 2.2. The assertion of the following RDF triples mean that the resource book1 under the namespace ns, has a property price under the same namespace, with value 27.

```
@prefix ns: <http://example.org/Bookstore#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
ns:book1 ns:price "27"^^xsd:integer .
```

The `xsd:integer` part of the literal "27"8sd:integer is the XML Schema datatype for integers.
Resources in RDF may be anonymous (i.e. not identified by a URI). Such resources are called *blank nodes*. In triple form, a blank node is indicated by the label form, such as " :abc".

Example 2.3. The assertion of the following RDF triples mean that something has an author whose value is "Jeffrey D. Ullman".

```
@prefix ns: <http://example.org/Bookstore#> .
_:a ns:author "Jeffrey D. Ullman" .
```

Let ***I*** be the set of IRIs [13] (i.e. generalization of the URI), ***L*** be the set of the RDF Literals, and ***B*** be the set of the blank nodes.

Definition 2.1 (RDF Triple*).* *A triple (s,p, o) ∈ (I ∪ B) × I × (I ∪ B ∪ L) is called an RDF triple, where s, p, and o are a subject, predicate, and object, respectively.*

A collection of RDF statements (RDF triples) can be intuitively understood as a directed labeled graph, where resources are nodes and statements are arcs (from the subject node to the object node) connecting the nodes. It is worth mentioning that a relational data model is easily mapped onto this form, with a node corresponding to a table row or primitive value, and an arc corresponding to a column identifier.

Example 2.4. The assertion of the following RDF triples means that "Jeffrey D. Ullman" is an author of a book entitled "Database Systems" whose publisher is "Prentice Hall".

```
@prefix ns: <http://example.org/Bookstore#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
ns:book1 rdf:type ns:Publication .
ns:book1 ns:title "Database Systems" .
ns:book1 ns:author "Jeffrey D. Ullman" .
ns:book1 ns:publisher "Prentice Hall" .
```

Figure 2.1 shows the representation of the above RDF triples as a directed graph.

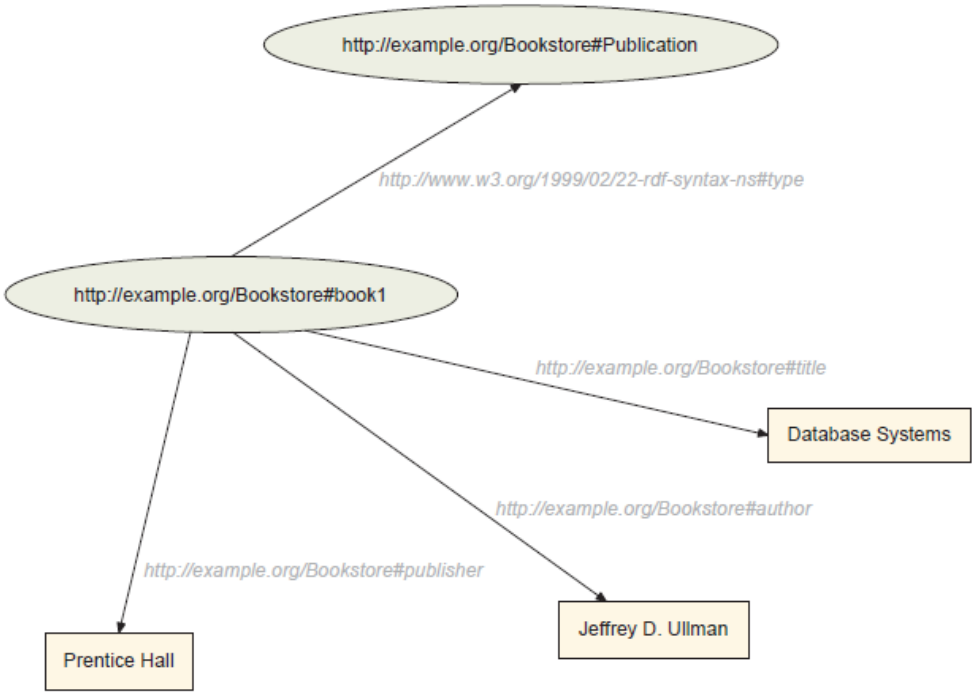


Figure 2.1: RDF graph representation example.

Definition 2.2 (RDF Graph). *An RDF graph G is a set of RDF triples.*

RDF provides a number of additional capabilities, such as built-in types and properties for representing groups of resources and simple RDF statements. These types and properties are described using a set of reserved words

(prefixed http://www.w3.org/1999/02/22-rdf-syntax-ns#) called the RDF vocabulary. However, RDF user communities also need the ability to define the vocabularies (terms) they intend to use in those statements, specifically, to indicate that they are describing specific kinds or classes of resources, and to use specific properties in describing those resources. Consequently, the RDFS vocabulary came to build on the limited vocabulary of RDF.

RDFS (RDF Schema) [28], [8] is an extension of RDF designed to describe relationships between resources and/or resources using a set of reserved words (prefixed http://www.w3.org/2000/01/rdf-schema#) called the RDFS vocabulary. It describes constructs for types of objects (classes), relating types to one another (subclasses), properties that describe objects (properties), and relationships between them (subproperty). The class system in RDFS includes a simple notion of inheritance, based on set inclusion. For example, one class is a subclass of another means that instances of the one are also instances of the other.

A class in RDFS corresponds to the generic concept of a type or category, somewhat like the notion of a class in object-oriented programming languages such as Java, and is defined using the construct rdfs:Class. RDF classes can be used to represent almost any category of thing, such as Web pages, people, document types, databases or abstract concepts. The resources that belong to a class are called its instances. Classes can be organized in a hierarchical fashion using the construct rdfs:subClassOf.

A property in RDFS is used to characterize a class/classes and is defined using the construct rdf:Property. The RDFS also provides a vocabulary used to describe how properties and classes are intended to be used together in RDF data. This kind of information is supplied by using the rdfs:domain and rdfs:range constructs. Similarly to classes, RDFS provides a way to specialize properties by using the construct rdfs:subPropertyOf.

Moreover, RDFS provides a number of other built-in properties which can be used in order to provide documentation and other information about an RDF Schema or about instances. For example, the construct rdfs:comment can be used to provide a human-readable description of a resource, while the construct rdfs:label can be used to provide a more human-readable version of a resource's name.

Finally, the semantics of RDFS is expressed through the mechanism of inferencing (i.e. the meaning of any construct in RDFS is given by the inferences that can be inferred from it).

2.2.2 Web Ontology Language (OWL)

The **Web Ontology Language** (**OWL**) is a family of knowledge representation languages for authoring ontologies endorsed by the World Wide Web Consortium. They are characterized by formal semantics and RDF/XML-based serializations for the Semantic Web. OWL has attracted academic, medical and commercial interest.

The OWL (Web Ontology Language) is a standard language for describing ontologies in world wide network and is a continuation and extension of DAML + OIL. The OWL aims to provide a full support of the linguistic knowledge representation, extending in this way expressiveness of DAML + OIL. As the complexity of results extraction is growing with the expressiveness of the language, three types (Species) of OWL were developed, which can be used as appropriate, depending on the needs of expressiveness and the limits on complexity. The three species of OWL are:

a) The **OWL Lite**, aim at low complexity conclusions, but has limited expressiveness. Learning from OWL Lite ontology has guaranteed conclusions, which are produced, at worst, in exponential time. To make this possible, OWL Lite does not support the functions of association (union) and complement, while the multiplicity constraints of the properties may have only the values 0 and 1.

b) The **OWL DL** (OWL Description Logics), extends the expressiveness of OWL Lite to narrative logic functionality and data types. The OWL DL is the more expressive than OWL Lite and guarantees the efficiency of result extraction

c) The **OWL Full** supports all the features of narrative logic and RDF. The OWL Full is the most expressive of the species of OWL, but does not guarantee efficiency of inference. O*WL Full* is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. For example, in OWL Full a class can be treated simultaneously as a collection of individuals and as an individual in its own right. OWL Full allows an ontology to augment the meaning of the predefined (RDF or OWL) vocabulary.

Each of the three species/sublanguages is an extension of previous simpler kind with respect to what canbe legally expressed and in what can be validly concluded. Thus, the following relations apply:

- Every valid (legal) OWL Lite ontology is a valid OWL-DL ontology.
- Every valid OWL-DL ontology is a valid OWL Full ontology.
- Any credible (valid) OWL Lite conclusion is reliable OWL-DL conclusion.
- Any credible OWL-DL conclusion is a valid OWL Full conclusion.

The OWL has been developed based on the example (paradigm) of narrative logic (description logic) and uses RDF (S) syntax. The components of the OWL ontology are classes, properties and individuals. The OWL classes are OWL properties, and OWL individuals identified by unique identities, set in "rdf: ID" feature. In addition, may have labels and comments in a language understood by humans, which do not affect the logical interpretation by inference tools and structures and they are defined by "rdfs: label" and "rdfs: comment" respectively.

An *instance* is an object. It corresponds to a Description Logic individual. OWL provides mechanisms in order to declare two individuals to be identical or different, by using the `owl:sameAs` and `owl:differentFrom` constructors, respectively.
A *class* (defined by using the construct `owl:Class`) is a collection of objects. It corresponds to a Description Logic (DL) concept and may contain any number of individuals, instances of the class. An instance may belong to none, one or more classes. A class may be defined to be subclass of another (using the construct `rdfs:subClassOf`), inheriting characteristics from its parent superclass. This corresponds to logical subsumption and DL concept inclusion. All classes are subclasses of `owl:Thing` (DL *top* notated T), the root class. All classes are subclassed by `owl:Nothing` (DL *bottom* notated $\perp$), the empty class. Similarly, two classes may be defined to be equivalent (using the construct `owl:equivalentClass`), indicating that these two classes have precisely the same instances. This corresponds to logical equivalence and DL concept equality.

Example 2.5. Let Product and Book be two OWL classes. Similarly, let `HalloweenAudioCD` be an instance of the `class Product` and let

DatabaseSystems be an instance of the class Book. The RDF/XML syntax used in order to define these statements is provided below.

```
<owl:Class rdf:ID="Product"/>
<owl:Class rdf:ID="Book"/>

<Product rdf:ID="HelloweenAudioCD"/>
<Book rdf:ID="DatabaseSystems"/>
```

Example 2.6. Consider the classes and the instances defined in the previous example. Moreover, let the class Book be subclass of the class Product. The RDF/XML syntax used in order to define this statement is provided below.

```
<owl:Class rdf:ID="Product"/>
<owl:Class rdf:ID="Book">
      <rdfs:subClassOf rdf:resource="#Product"/>
</owl:Class>
<Product rdf:ID="HelloweenAudioCD"/>
<Book rdf:ID="DatabaseSystems"/>
```

By defining that the class Book is a subclass of the class Product, we implicitly infer that every instance of the class Book is also an instance of the class Product. Consequently, DatabaseSystems which has been defined as an instance of the class Book, is also an instance of the class Product.

OWL provides additional constructors with which to form classes. These constructors can be used to create so-called class expressions. OWL supports the basic set operations, namely union, intersection and complement. These are named owl:unionOf, owl:intersectionOf, and owl:complementOf, respectively. Additionally, classes can be enumerated. Class extensions can be stated explicitly by means of the owl:oneOf constructor. Furthermore, it is possible to assert that class extensions must be disjoint (using the construct owl:disjointWith).

A *property* is a directed binary relation that specifies class characteristics. It corresponds to a Description Logic role. A property may be defined to be subproperty of another (using the construct rdfs:subPropertyOf), inheriting characteristics from its parent

superproperty. Similarly, two properties may be defined to be equivalent, using the construct `owl:equivalentProperty`. Two types of properties are distinguished:

- *Datatype properties* are relations between instances of classes and RDF literals or XML schema datatypes. A datatype property is defined by using the construct `owl:DatatypeProperty.`
- *Object properties* are relations between instances of two classes. An object property is defined by using the construct `owl:ObjectProperty`.

Properties may possess logical capabilities such as being transitive, symmetric, inverse and functional. Properties may also have domains and ranges. It is possible to constrain the range of a property in specific contexts in a variety of ways, by using either cardinality or value restrictions.

Example 2.7. Consider the classes and the instances defined in the previous example. Let the class `Product` have a datatype property `price`. This infers that the instances of the classes `Product` and `Book`, since Book is a subclass `of Product`, may be described using the datatype property `price`. Furthermore, let the datatype property price range over the XML Schema datatype `xsd:decimal`. The RDF/XML syntax used in order to define these statements is provided below.

```
<owl:Class rdf:ID="Product"/>
<owl:Class rdf:ID="Book">
      <rdfs:subClassOf rdf:resource="#Product"/>
</owl:Class>

<owl:DatatypeProperty rdf:ID="price">
      <rdfs:domain rdf:resource="#Product"/>
      <rdfs:range rdf:resource="&xsd;decimal"/>
</owl:DatatypeProperty>

<Product rdf:ID="HelloweenAudioCD">
      <price rdf:datatype="&xsd;decimal">15.30</price>
</Product>

<Book rdf:ID="DatabaseSystems">
```

```
      <price rdf:datatype="&xsd;decimal">57.90</price>
</Book>
```

Example 2.8. Consider the classes, the properties and the instances defined in the previous example. Let the class Book have an object property publisher. Moreover, let the object property publisher range over an OWL class Publisher. Similarly, let an object property publishes having as domain the instances of the class Publisher and as range the instances of the class Book. This infers that the object property publishes is the inverse of the object property publisher and vice versa. The RDF/XML syntax used in order to define these statements is provided below.

```
<owl:Class rdf:ID="Product"/>
<owl:Class rdf:ID="Book">
      <rdfs:subClassOf rdf:resource="#Product"/>
</owl:Class>
<owl:Class rdf:ID="Publisher"/>

<owl:DatatypeProperty rdf:ID="price">
      <rdfs:domain rdf:resource="#Product"/>
      <rdfs:range rdf:resource="&xsd;decimal"/>
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="publisher">
      <rdfs:domain rdf:resource="#Book"/>
      <rdfs:range rdf:resource="#Publisher"/>
      <owl:inverseOf rdf:resource="#publishes"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="publishes">
      <rdfs:domain rdf:resource="#Publisher"/>
      <rdfs:range rdf:resource="#Book"/>
      <owl:inverseOf rdf:resource="#publisher"/>
</owl:ObjectProperty>

<Product rdf:ID="HelloweenAudioCD">
```

```
      <price rdf:datatype="&xsd;decimal">15.30</price>
</Product>
<Book rdf:ID="DatabaseSystems">
      <price rdf:datatype="&xsd;decimal">57.90</price>
</Book>
```

2.2.3 SPARQL query language

SPARQL is an RDF query language; its name is a recursive acronym that stands for SPARQL Protocol and RDF Query Language. It was standardized by the RDF Data Access Working Group (DAWG) of the World Wide Web Consortium, and is considered a key semantic web technology. On 15 January 2008, SPARQL became an official W3C Recommendation. **SPARQL** allows for a query to consist of triple patterns, conjunctions, disjunctions, and optional patterns. Ex.: Find all the books for kids with a price less than 30€.

```
SELECT ?name
  WHERE
  {?x    rdf:type   s:Children.
   ?x    s:price    ?price.
   ?x    s:name     ?name.
 FILTER(?price<30) }
```

This section presents the syntax used by SPARQL for RDF terms and triple patterns.

Syntax for IRIs

IRIs [13] are a generalization of URIs and are fully compatible with URIs and URLs. IRI references are designated using the '<' and '>' delimiters.

The PREFIX keyword can be used in order to associate a prefix label with an IRI. A prefixed name is a prefix label and a local part, separated by a colon ":". A prefixed name is mapped to an IRI by concatenating the IRI associated with the prefix and the local part. The prefix label or the local part may be empty.

The following fragments are some of the different ways to write the same IRI:

```
1. <http://example.org/Bookstore#DatabaseSystems>
```

```
2. PREFIX ns: <http://example.org/Bookstore#>
ns:DatabaseSystems

3. PREFIX : <http://example.org/Bookstore#>
:DatabaseSystems
```

Syntax for literals

The general syntax for literals is a string (enclosed in either double quotes, "…", or single quotes, '… '), with either an optional language tag (introduced by @) or an optional datatype IRI or prefixed name (introduced by ^^).

As a convenience, integers can be written directly (without quotation marks and an explicit datatype IRI) and are interpreted as typed literals of the XML Schema datatype `xsd:integer`. Furthermore, decimal numbers for which there is '.' in the number but no exponent are interpreted as `xsd:decimal` and numbers with exponents are interpreted as `xsd:double`. Values of type `xsd:boolean` can also be written as true or false.

Examples of literal syntax in SPARQL include:

1. "chat"
2. 'chat'@fr with language tag "fr"
3. "xyz"^^<http://example.org/ns/userDatatype>
4. 1, which is the same as "1"^^xsd:integer
5. 1.3, which is the same as "1.3"^^xsd:decimal
6. 1.0e6, which is the same as "1.0e6"^^xsd:double
7. true, which is the same as "true"^^xsd:boolean
8. false, which is the same as "false"^^xsd:boolean

Syntax for query variables

Query variables in SPARQL queries have global scope. Consequently, the use of a given variable name anywhere in a query identifies the same variable. Variables are prefixed by

either "?" or "$". These two symbols are not considered part of the variable's name. In a query, $abc and ?abc identify the same variable.

Syntax for blank nodes

Blank nodes in graph patterns act as non-distinguished variables and not as references to specific blank nodes in the data being queried. Blank nodes are indicated by the label form, such as " _:abc". The same blank node label cannot be used in two different basic graph patterns in the same query.

Syntax for triple patterns

Let ***I*** be the set of IRIs, ***L*** be the set of the RDF Literals, and ***B*** be the set of the blank nodes. Assume additionally the existence of an infinite set *V* of variables disjoint from the previous sets (*I*, *B*, *L*).

Definition 2.3 (Triple pattern). *A triple* $(s, p, o) \in (I \cup L \cup V) \times (I \cup V) \times (I \cup L \cup V)$ *is called a triple pattern, where s, p, and o are a subject, predicate, and object, respectively.*

Triple Patterns are written as a whitespace-separated list of a subject, predicate and object. Some triple pattern examples are the following:

1. http://example.org/Bookstore#DatabaseSystems ns:title ?title

The above triple pattern contains the URI http://example.org/Bookstore#DatabaseSystems in its subject part, the property title under the prefix ns in its predicate part and the variable title in its object part.

2. ?x foaf:name "Alan"

The above triple pattern contains the variable x in its subject part, the property name under the prefix foaf in its predicate part and the literal Alan in its object part.

3. ?x ?y "55.30"^^xsd:decimal

The above triple pattern contains the variable x in its subject part, the variable y in its predicate part and the literal `"55.30"^^xsd:decimal` in its object part.

4. ?s ?p ?o

The above triple pattern contains the variable s in its subject part, the variable p in its predicate part and the variable o in its object part.

Graph patterns

SPARQL is based around graph pattern matching. More complex graph patterns can be formed by combining smaller patterns in various ways.

Definition 2.4 (Graph pattern). *A SPARQL graph pattern expression is defined recursively as follows:*

• *A triple pattern is a graph pattern.*
• *If P1 and P2 are graph patterns, then expressions (P1 AND P2), (P1 OPT P2), and (P1 UNION P2) are graph patterns (group graph pattern, optional graph pattern, and alternative graph pattern, respectively).*
• *If P is a graph pattern and R is a SPARQL built-in condition, then the expression (P FILTER R) is a graph pattern (a filter graph pattern).*

We note that a SPARQL built-in condition is constructed using IRIs, RDF literals, variables and constants, as well as logical connectives (&&, k, !), operators (=, ! =, >, <, ≥, ≤, +, −, ∗, =) and built-in functions (e.g. bound, isIRI, isLiteral, datatype, lang, str, regex).

Basic graph patterns

Basic graph patterns are sets of triple patterns. SPARQL graph pattern matching is defined in terms of combining the results from matching basic graph patterns. A sequence of triple patterns interrupted by a filter comprises a single basic graph pattern. Filters are constraints expressed by the keyword FILTER, which are used in order to restrict the graph pattern solutions to those for which the filter expression evaluates to true.

A filter is consisted of a SPARQL built-in condition, which is constructed using IRIs, RDF literals, variables and constants, as well as logical connectives (&&, *k*, !), operators (=, ! =, >, <, ≥, ≤, +, −, ∗, /) and built-in functions (e.g. `bound, isIRI, isLiteral, datatype, lang, str, regex`).

Definition 2.5 (Basic graph pattern). *A finite sequence of conjunctive triple patterns and possible filters is called basic graph pattern.*

Some basic graph pattern examples are the following:

```
1. ?x ns:title ?title .
    ?x ns:price "62"^^xsd:decimal .

2. ?x ns:title "Database Systems" .
    ?x ns:price ?price .
    FILTER(?price>50)
```

Group graph patterns

The group graph pattern is the most general form of a graph pattern, since it may contain every other graph pattern type. A group graph pattern is delimited with braces (*{}*). A group graph pattern example is the following:

```
{
        ?x ns:title "Database Systems" .
        ?x ns:author ?author .
}
```

The above group graph pattern is consisted of two triple patterns and is considered to be equivalent with the following:

```
{
            {?x ns:title "Database Systems" .}
             {?x ns:author ?author .}
}
```

A constraint, expressed by the keyword FILTER, is a restriction on the solutions over the whole group in which the filter appears. The following patterns all have the same solutions:

```
1. {
    ?x ns:title ?title .
    ?x ns:price ?price .
    FILTER (?price<60)
   }
2. {
    FILTER (?price<60)
```

```
      ?x ns:title ?title .
      ?x ns:price ?price .
     }
3. {
      ?x ns:title ?title .
      FILTER (?price<60)
      ?x ns:price ?price .
     }
```

Optional graph patterns

Basic graph patterns allow applications to make queries where the entire query pattern must match for there to be a solution. However, it is useful to be able to have queries that allow information to be added to the solution where the information is available, but do not reject the solution because some part of the query pattern does not match. Optional matching provides this facility: if the optional part does not match, it creates no bindings but does not eliminate the solution.

Optional parts of a graph pattern may be specified syntactically with the `OPTIONAL` keyword applied to a graph pattern. An optional graph pattern example is the following:

```
OPTIONAL {
           ?x foaf:name "Alan" .
           ?x foaf:mbox ?mbox .
         }
```

If the evaluation result of the above graph pattern is an empty set of solutions, then it will not be taken into consideration by the process of query evaluation. That is the case where there is not any resource having the value "`Alan`" for the property `foaf:name` and having any value for the property `foaf:mbox`.

Alternative graph patterns

SPARQL provides a means of combining graph patterns so that one of several alternative graph patterns may match. If more than one of the alternatives matches, all the possible

pattern solutions are found. Pattern alternatives are syntactically specified with the UNION keyword. An alternative graph pattern example is the following:

```
{
      { ?book ns1:title ?title }
      UNION
      { ?book ns2:title ?title }
}
```

The above graph pattern matches titles and books in the data, whether the property `title` is under the namespace `ns1` or `ns2.`

Query forms

SPARQL has four query forms. These query forms use the solutions from pattern matching to form result sets or RDF graphs. The query forms are:

Select

The SELECT query form returns variables and their bindings directly. The syntax `SELECT *` is an abbreviation that selects all of the variables in a query.

Construct

The `CONSTRUCT` query form returns an RDF graph constructed by substituting variables in a set of triple templates. If any instantiation produces a triple containing an unbound variable or an illegal RDF construct, such as a literal in subject or predicate position, then that triple is not included in the output RDF graph.

Ask

The ASK query form returns no information about the possible query solutions, just whether or not a solution exists.

Describe

The `DESCRIBE` query form returns a single result RDF graph containing RDF data about resources. This data is not prescribed by a SPARQL query, where the query client would need to know the structure of the RDF in the data source, but, instead, is determined by the SPARQL query processor. The DESCRIBE query form takes each of the resources identified in a solution, together with any resources directly named by using an IRI, and assembles a

single RDF graph by taking a "description" which can come from any information available including the target RDF dataset. The syntax `DESCRIBE *` is an abbreviation that describes all of the variables in a query.

Solution sequence modifiers

Query patterns generate an unordered collection of solutions. Each solution is a partial function from variables to RDF terms. These solutions are initially treated as a sequence (a solution sequence), in no specific order. In case that any sequence modifiers are applied, a new sequence is created. Finally, this latter sequence is used to generate one of the results of a SPARQL query form.

The solution sequence modifiers may be applied on `the SELECT, CONSTRUCT and DESCRIBE` query forms. In addition, the `DISTINCT and REDUCE` modifiers may be applied only on the `SELECT` query form.

Order by

The ORDER BY clause establishes the order of a solution sequence. Furthermore, it is possible to use the ASC() and DESC() modifiers in order to specify whether the order should be ascending or descending. In case that any ASC() or DESC() modifier has been specified, the order of the solution sequence is ascending.

Distinct

The DISTINCT clause eliminates duplicate solutions. For example, consider the following queries posed over the same RDF dataset.

Reduced

The REDUCED clause permits a specific number of duplicate solutions. This number is specified by the SPARQL query engine that executes the query and is at least one and not more than the cardinality of the solution set with no DISTINCT or REDUCED modifier.

Limit

The LIMIT clause puts an upper bound on the number of solutions returned. If the number of actual solutions is greater than the limit, then at most the limit number of solutions will be returned. A LIMIT of zero has no effect.

Offset

The OFFSET clause causes the solutions generated to start after the specified number of solutions. An OFFSET of zero has no effect.

Semantics of SPARQL graph pattern expressions

In this section we provide an overview of the semantics of SPARQL graph pattern expressions defined in [33], considering a function-based representation of a graph pattern evaluation over an RDF dataset.

In order not to confuse, the notation and the terminology followed in this section is differentiated in some cases, compared to the notation and terminology followed in [33]. In Table 2.1 we provide the notation which is used for defining the semantics of SPARQL graph pattern expressions.

Definition 2.6 (SPARQL graph pattern solution). *A graph pattern solution* $\omega : V \rightarrow (I \cup B \cup L)$ *is a partial function that assigns RDF terms of an RDF dataset to variables of a SPARQL graph pattern. The domain of* ω, $dom(\omega)$, *is the subset of V where* ω *is defined. The empty graph pattern solution* $\omega_{\emptyset}$ *is the graph pattern solution with empty domain. The SPARQL graph pattern evaluation result is a set* Ω *of graph pattern solutions* ω.

Two *graph pattern solutions* $\omega 1$ and $\omega 2$ are compatible when for all $x \in dom(\omega 1) \cap dom(\omega 2)$, it is the case that $\omega 1(x) = \omega 2(x)$. Furthermore, two *graph pattern solutions* with disjoint domains are always compatible, and the empty *graph pattern solution* $\omega_{\emptyset}$ is compatible with any other *graph pattern solution*.

Let Ω1 and Ω2 be sets of *graph pattern solutions* and *J* be a set of SPARQL variables. The join, union, difference, projection and left outer join operations between Ω1 and Ω2 are defined as follows:

Ω1 ⋈ Ω2 = *{ω*1 ∪ *ω*2 | *ω*1 ∈ Ω1*, ω*2 ∈ Ω2 are compatible *graph pattern solutions}*,
Ω1 ∪ Ω2 = *{ω|ω*∈ Ω1 or *ω* ∈ Ω2*}*,
Ω1 \ Ω2 = *{ω*∈ Ω1 | for all *ω'* ∈ Ω2*; ω* and *ω'* are not compatible*}*,
π J (Ω1) = *{ω| ω'* ∈ Ω1*; dom*(ω) = *dom*(*ω'*) ∩ *J* and ∀*x* ∈ *dom*(*ω*)*;ω*(*x*) = *ω'*(*x*)*}*,
Ω1 ⟕ Ω2 = (Ω1 ⋈ Ω2) ∪ (Ω1 \ Ω2)

The semantics of SPARQL graph pattern expressions is defined as a function $[[\cdot]]_D$ which takes a graph pattern expression and an RDF dataset *D* and returns a set of *graph pattern solutions* (see Definition 2.8). Refer to Definition 2.7 for the semantics of FILTER expressions, which can be part of a SPARQL graph pattern.

Definition 2.7 (SPARQL FILTER expression evaluation). *Given a graph pattern solution ω and a built-in condition R, we say that ω satisfies R, denoted by ω |= R, if:*

1. R is *bound*(?*x*) and ?*x* ∈ *dom*(*ω*);
2. R is ?*x cp c;* ?*x* ∈ *dom*(*ω*) and *ω*(?*x*) *oprt c;* where *cp* → = | ≤ | ≥ | < | >;
3. R is ?*x cp* ?*y;* ?*x* ∈ *dom*(*ω*)*;* ?*y* ∈ *dom*(*ω*) and *ω*(?*x*) *cp ω*(?*y*)*;* where *cp* → = | ≤ |≥ | < | >;
4. R is (¬*R*1)*;R*1 is a built-in condition, and it is not the case that *ω |= R*1;
5. R is (*R*1 ∨ *R*2)*;R*1 and *R*2 are built-in conditions, and *ω |= R*1 or *ω |= R*2;
6. R is (*R*1 ∧ *R*2)*;R*1 and *R*2 are built-in conditions, *ω |= R*1 and ω *|= R*2*:*

Definition 2.8 (SPARQL graph pattern evaluation). *Let D be an RDF dataset over* (*I* ∪ *B* ∪ *L*)*, t a triple pattern, P, P*1*, P*2 *graph patterns and R a built-in condition. The evaluation of a graph pattern over D, denoted by* [[·]]*D, is defined recursively as follows:*

1. [[*t*]]*D* = *{ω | dom*(*ω*) = *var*(*t*) and *ω*(*t*) ∈ *D}*
2. [[(*P*1 AND *P*2)]]*D* = [[*P*1]]*D* ⋈ [[*P*2]]*D*
3. [[(*P*1 OPT *P*2)]]*D* = [[*P*1]]*D* ⟕ [[*P*2]]*D*
4. [[(*P*1 UNION *P*2)]]*D* = [[*P*1]]*D* ∪ [[*P*2]]*D*
5. [[(*P* FILTER *R*)]]*D* = *{ω* ∈ [[*P*]]*D | ω |= R}*

For a detailed description of SPARQL semantics and for a complete set of illustrative examples, refer to [33].

Table 2.1: The notation which is used for defining the semantics of SPARQL graph pattern expressions.

Notation	Description
V	The set of variables
I	The set of IRI's
B	The set of Blank Nodes
L	The set of RDF literals
ω	A graph pattern solution ω : V → (I ∪B ∪ L).
dom(ω)	Domain of a graph pattern solution ω (subset of V).
var (t)	The variables of a triple pattern t.
ω(t)	The triple obtained by replacing the variables in triple pattern t according to a graph pattern solution ω (abusing notation).
ω |= ***R***	A graph pattern solution ω satisfies a built-in condition R.
[[·]]	Graph pattern evaluation function.
⋈	Graph pattern solution-based join.
⟕	Graph pattern solution-based left outer join.
****	Graph pattern solution-based difference.
π {…}	Graph pattern solution-based projection
∪	Graph pattern solution-based union.
∩	Set intersection.
?x, ?y	SPARQL variables.
bound	SPARQL unary predicate
AND, OPT, UNION, FILTER	SPARQL graph pattern operators.
¬, ∨, ∧	Logical not, or, and.
=, ≤, ≥, <, >	Inequality/equality operators.

2.2.4 Jena framework

Jena is a Java framework for building Semantic Web applications. It provides a programmatic environment for RDF, RDFS and OWL, SPARQL and includes a rule-based inference engine. Jena is open source and grown out of work with the HP Labs Semantic Web Programme.

The Jena Framework includes:

- A RDF API
- Reading and writing RDF in RDF/XML, N3 and N-Triples
- An OWL API
- In-memory and persistent storage
- SPARQL query engine

It provides an API to extract data from and write to RDF graphs. The graphs are represented as an abstract "model". A model can be sourced with data from files, databases, URLs or a combination of these. Jena is similar to Sesame; though, unlike Sesame, Jena provides support for OWL (Web Ontology Language). The framework has various internal reasoners and the Pellet reasoner (an open source Java OWL-DL reasoner) can be set up to work in Jena.

Jena supports serialization of RDF graphs to:

- a relational database
- RDF/XML
- Turtle
- Notation 3

2.3 Related Research

In the Semantic Web environment a number of ontology based mediator architectures have been proposed in the literature [43], [30]. In the following sections we present the most relevant research to the issues discussed in this thesis.

A cornerstone issue in the realization of the Semantic Web (SW) vision is the achievement of semantic interoperability among legacy data sources spread worldwide. In order to capture information semantics in a machine processable way, various ontology-based formalisms have been recently proposed. However, the vast majority of existing legacy data is not yet in RDF/S or any other SW language .As a matter of fact, most of the data is physically stored in relational database (RDB) systems and actually published on the Web or corporate intranets as *virtual* XML. SW applications, however, require to view data as *virtual* RDF, valid instance of a domain or application specific RDF/S schema, and to be able to manipulate them with high-level query languages, such as RQL [18] or RVL [24].

Mediation in Information Systems

The need for integrated and unified access to multiple information sources has stimulated the research on mediators. A central role for the interaction of data and knowledge is assigned to modules that *mediate* between the user workstations and the data resources. Mediators contain the administrative and technical knowledge needed to extract and/or infer the information needed for decision making. In particular, a mediator has ontology with terminology and structuring that reflects the needs of its potential users. However, it does not maintain a database of objects. Instead, the mediator has a number of articulations to other sources.

As it is clearly stated in [TziSpyCon01] the installation of high speed networks is changing the physical capabilities of information systems. These capabilities must be complemented with corresponding software systems advances to obtain a real benefit. Without smart software we will gain access to more data, but not improve access to the type and quality of information needed for decision making.

To develop the concepts needed for future information systems we model information processing as an interaction of data and knowledge. This model provides criteria for a high level functional partitioning. These partitions are mapped into information processing modules. The modules are assigned to nodes of the distributed information systems. A central role is assigned to modules that mediate between the users' workstations and data resources. So we reach at the following definition:

Definition 2.8: A **mediator** is a software module that exploits encoded knowledge about some data sets in order to extract and/or infer the information for higher-layer applications.

The making of informed decisions requires the application of a variety of knowledge to information about the state of the world. To clarify the distinction of data and knowledge in our model we restate a definition from [Wiederhold:86B]

Data describes specific instances and events. Data may be gathered automatically or clerically. The correctness of data can be verified by the real world.

Knowledge describes abstract classes. Each class typically covers many instances. Experts are needed to gather and formalize knowledge. Data can be used to disprove knowledge.

Currently there is great interest in Distributed Information Systems using mediators and facilitators following Wiederhold's original paper.
There have been some systems and models that have been implemented and are related to our work. We present some of these systems below, in order to show different ways of mediation and performance in different tasks, but also examples of data integration.

MOMIS System

The DARPA-funded Knowledge Sharing Effort (KSE) [17] aims to facilitate sharing and reuse of knowledge bases. The MOMIS (Mediator envirOnment for Multiple Information Sources) [Berg04]is a framework to perform information extraction and integration from both structured and semistructured data sources. An object-oriented language, with an underlying Description Logic, called ODL-I3, derived from the standard ODMG is introduced for information extraction. Information integration is then performed in a semi-automatic way, by exploiting the knowledge in a Common Thesaurus (defined by the framework) and ODL-I3 descriptions of source schemas with a combination of clustering techniques and

Description Logics. This integration process gives rise to a virtual integrated view of the underlying sources (the Global Schema) for which mapping rules and integrity constraints are specified to handle heterogeneity. The MOMIS system, based on a conventional wrapper/mediator architecture, provides methods and open tools for data management in Internet-based information systems by using a CORBA-2 interface.

Figure 2.2 shows the main architecture of the MOMIS System.

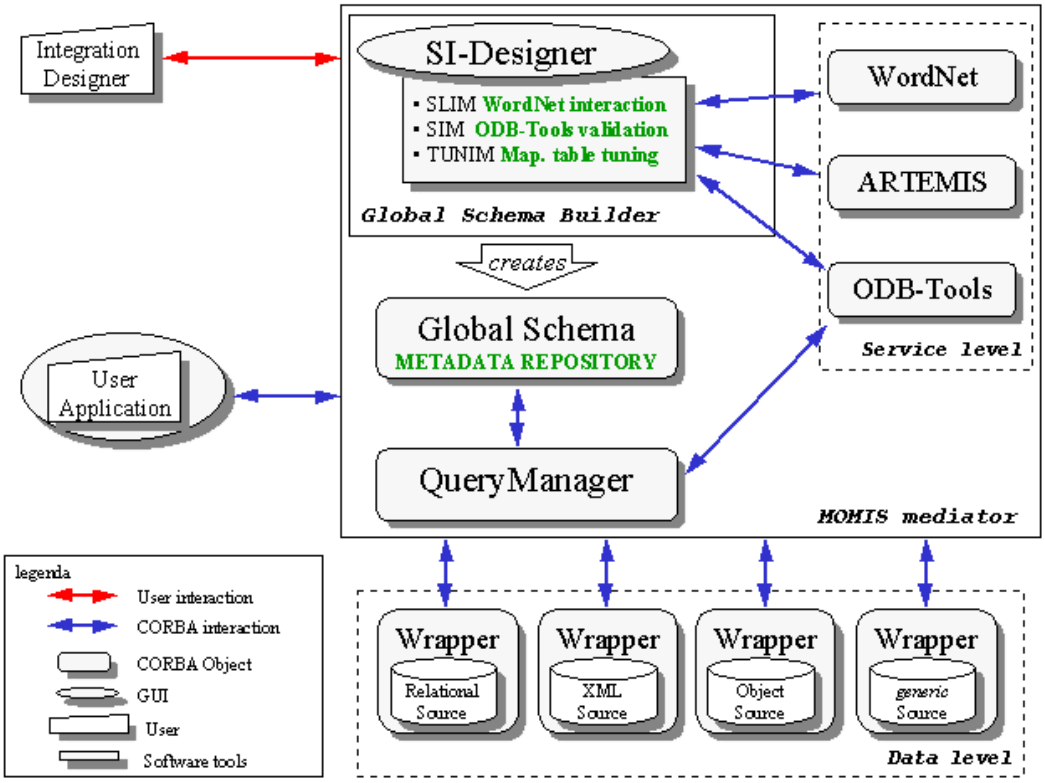


Figure 2.2: MOMIS Architecture

More specifically, in their approach data integration provides a Global Virtual View (GVV) that is a conceptualization (ontology) describing sources and allows a user to raise a query and to receive a single unified answer.

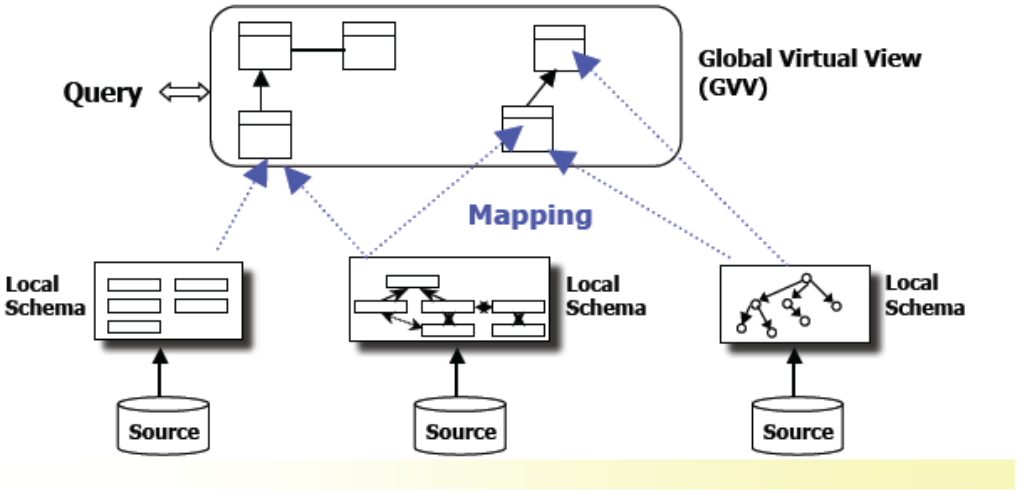


Figure 2.3: Global Virtual View

The tool supported techniques to construct the GVV are:

- Local Schema Annotation w.r.t. a common lexical ontology (WordNet)
- Semi-automatic generation of relationships between local schemata
- Clustering techniques
- Semi-automatic generation of mappings between the GVV and local schemata (Mapping Table)
- Semi-automatic GVV Annotation w.r.t. a common lexical ontology

In our approach, an effective use of data and knowledge is joined with automated information acquisition processes for decision- making activities. A framework based on OWL/RDF(contrary to ODLI3 of MOMIS SYstem), to perform information extraction and integration from structured ontologies, plus a query management environment in SPARQL, able to process incoming queries through the navigation of the mediated schema, which is based on the mappings sets defined. The local schema of each source is also available, but only to the responsible of mappings creation. Finally another difference is that in MOMIS the Semantic Query optimization is based on extensional knowledge (intensional relationships: synonymy, related term, narrower and extensional relationships)and on common Thesaurus generation and WordNet, while in our implementation we are only interested in the specification and the representation of the kind of mappings between OWL ontologies which can be exploited by a query mediation system in order to perform SPARQL query processing(transformation, relaxation and reformulation). For the above reason , we utilize a concrete set of mappings that can be used by our mediation system.

The ICS-FORTH Semantic Web Integration Middleware (SWIM)

A middleware system has been implemented [Christoph01]that can either republish XML as RDF, or publish RDB (relational databases)data directly as RDF, or - even better - be capable of doing both. Sometimes the practical solution will be to rely just on the virtual XML schema and XML query interface of an existing XML publishing system. At other times, the SW publishing middleware will be built as an alternative to the XML publishing system, taking advantage of direct access to the underlying RDB management system (RDBMS). It is also possible that the SW middleware will have to integrate data in some RDBMS with data in native XML storage.

We need to deal flexibly with all these situations in a uniform framework.

The framework allows user communities to

1. specify XML→ RDF and RDB→ RDF mappings;
2. verify that these mappings conform to the semantics of the employed SW ontologies;
3. compose RQL queries with these mappings and produce XML or RDB queries;
4. specify further levels of abstraction as RDF→ RDF views;
5. compose RQL queries with such views;
6. perform query optimizations.

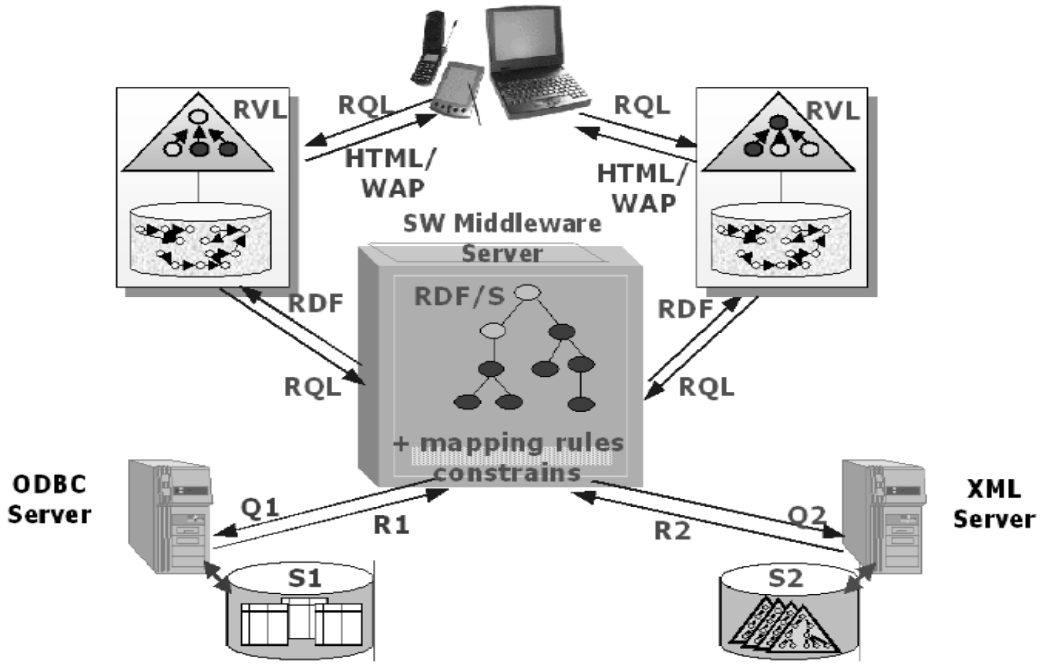


Figure 2.4: SWIM Architecture

Figure 2.4 sketches the architecture of a SWintegration middleware system that has been builded, called SWIM. The lower part of the figure depicts data sources, that could be XML repositories or RDBMS. On top of these sources, there is a domain or application ontology for a particular community, expressed, for instance, in RDF/S. Mapping rules can then be used for the integration, i.e., to translate back and forth from RDF/S to the source data models. As a result, through a SWIM server we can view the underlying sources as virtual RDF repositories and use RQL to query these sources as RDF data or even define personalized views on top using RVL.

The proposed model is a *semantic Web integration middleware* (SWIM) for *virtually* integrating relational and XML sources using RDF/S ontologies as depicted in Figure 2.5. In order to specify the data publishing services and reformulate on-the-fly mediator queries expressed against the RDF/S ontologies according to the syntax, structure, and semantics of

local, they rely on a uniform and expressive logic framework termed *semantic Web logic framework (SWLF)*.

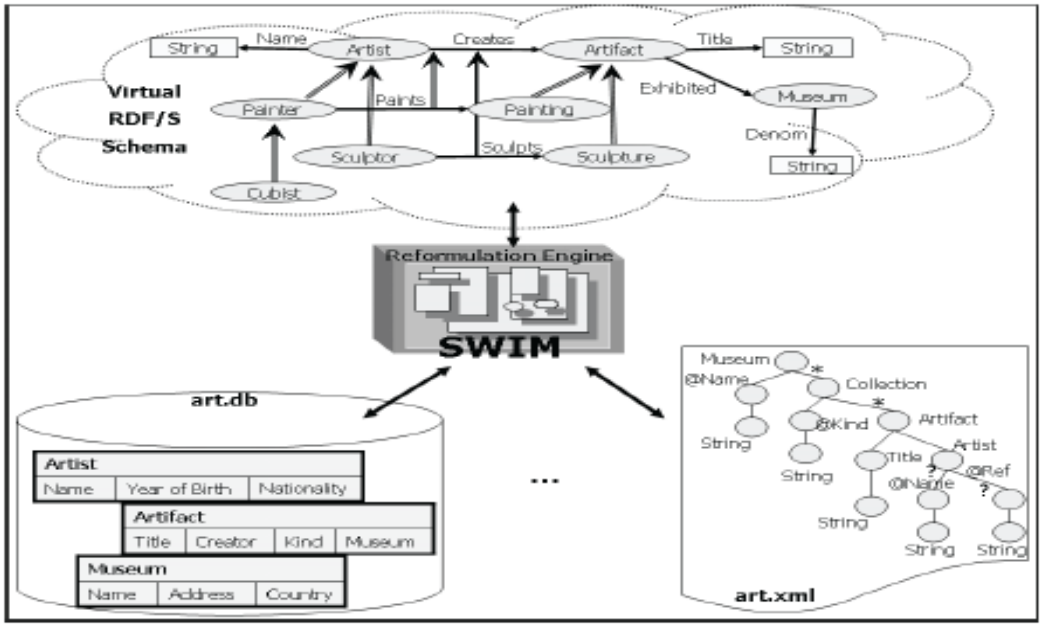


Figure 2.5: SWIM Architecture

In the context of this diploma thesis, the mediator we provide is a flexible and parameterized system, which can be easily transformed and enriched with components of language transformation (ex. XS2 OWL, SPARQL2XQuery), so it can extracts suitable results form different kind of information sources. However, until now only the Semantic Part of the Mediator is implemented. A significant difference of the two implementations is the Mappings specifications. While in our implementation we use a specific and concrete method of mapping specification between the terms of the main/global ontology and those of the local/target ontologies, defined in a specific language [Appendix], in SWIM system , internal logic framework employs first-order relations together with some first-order constraints to model RDF/S. The formal model for describing executable ontology mappings (i.e. mappings which can be used in SPARQL query rewriting) that satisfy real-world requirements and can be exploited in query processing, is definitely more reliable, in the context of our system. We present a mapping model that allows the definition of a rich set of ontology mappings.

Semantic Data Access Middleware for Grids (G-SDAM)

In this contribution [Langeg04] a system is presented, which provides access to distributed data sources using Semantic Web technology. While it was primarily designed for data sharing and scientific collaboration, it is regarded as a base technology useful for many other Semantic Web applications. The proposed system allows to retrieve data using SPARQL queries, data sources can register and abandon freely, and all RDF Schema or OWL vocabularies can be used to describe their data, as long as they are accessible on the Web. Data heterogeneity is addressed by RDF-wrappers like D2R-Server placed on top of local information systems. A query does not directly refer to actual endpoints, instead it contains graph patterns adhering to a virtual data set. A mediator finally pulls and joins RDF data from different endpoints providing a transparent on-the-fly view to the end-user.

The SPARQL protocol has been defined to enable systematic data access to remote endpoints. However, remote SPARQL queries require the explicit notion of endpoint URIs. The presented system allows users to execute queries without the need to specify target endpoints. Additionally, it is possible to execute join and union operations across different remote endpoints. The optimization of such distributed operations is a key factor concerning the performance of the overall system. Therefore, proven concepts from database research can be applied.

The main project is called *Semantic Data Access Middleware for Grids* (G-SDAM).

The mediator has a generic architecture of the mediator component, which is responsible for processing queries. The system cannot only be used for other data integration applications such as library repositories, directories, or various scientific archives and knowledge bases, it could also be used to complement Semantic Web search engines and Linked Data browsers .

The architecture of the data integration system is based on the following findings:

scientific data is usually structured, regional or globally distributed, stored in heterogeneous formats, and sometimes access is restricted to authorized people. To provide a transparent access to such kinds of data, a mediator-wrapper architecture is used. It basically consists of a mediator which is accepting queries from clients and then collects data translated by a number of wrappers attached to local data sources. These wrappers use mappings to translate data from the underlying information systems into a common schema which can be processed by the mediator. In the case of virtual data integration, mappings are used for on-the-fly translation of data during query processing.

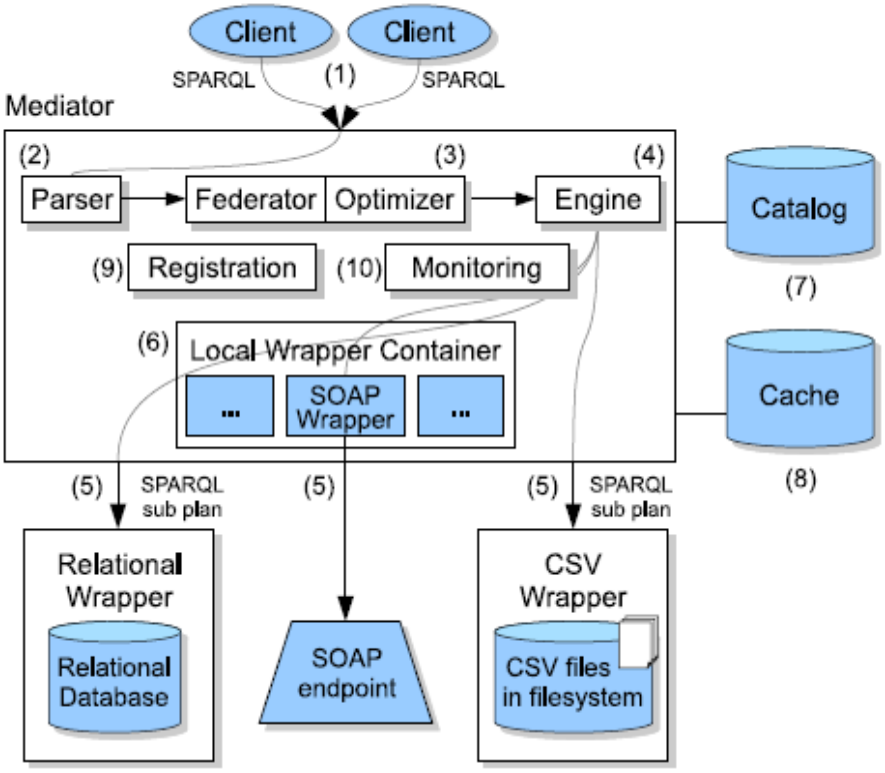


Figure 2.6: Mediator- Wrapper architecture of SemWIQ

At the implementation of SemWIQ, catalogs and monitoring are used.

The catalog is implemented as a fast in-memory Jena graph. It is persisted into a file when the mediator shuts down. Beside the endpoint URI and description, it stores statistics gathered by the monitoring component. The optimization concepts are not fully implemented in the current prototype, the statistics are rather simple. They are also used for data source selection by the federator.

In our approach, set of relations , called articulations are used instead of catalogs. Articulations, based on mapping, relate each term of the main ontology with all the relevant terms of the local ontologies, in order to have faster and more relevant results from each source. The articulations component is also used for storing data about the kind of the mappings, the number and the specification of the information sources and the relations between all the relevant terms. It is also used for the classification of the results based on the type of the answer provided.

Another difference between the two approaches is in the user-configured options about *what exactly the user wishes to be answered.* In the SemWIQ approach there are several restrictions to standard SPARQL:

– All subjects must be variables and for each subject variable its type must be explicitly or implicitly (through DL constraints) defined. A BGP like {?s :p ?o} is not allowed unless there is another triple telling the type for ?s. For instance, the BGP {?s :p ?o ; rdf:type <some-type>} is valid. In a future version, when DESCRIBE-queries are supported, it may become valid to constraint the subject term to an IRI.

– For virtual data integration it is not required to have multiple graphs. A query may only contain the default graph pattern. Furthermore, federation is done by the mediator and not explicitly by the user.

– Currently only SELECT-queries are supported, but support for DESCRIBE is planned. Supporting DESCRIBE may be useful to get further information to an already known record, however the mediator has to go through all data sources.

All other features of SPARQL like matching group graph patterns, optional graph patterns, alternative graph patterns, filters, and solution modifiers are supported but they still require further optimization.

However in our approach, all the above are answered correctly. The mediator transforms, normalizes, relaxes and reformulates the original query based on user's choices, so it can provide the best possible and relevant results from each source.

CHAPTER 3
Ontology-based Mediation

3.1 Introduction

The main objectives of the Mediation System we propose are described in this Chapter. Our goal is to present the functionality of a tool that facilitates user reaction with OWL/RDF information sources together with the use of the SPARQL query language in a systematic manner.

More specifically, we propose a model for providing integrated and unified access to multiple information sources. Each information source comprises two parts: (a) an ontology i.e. a set of terms structured by subsumption relations, and (b) a database that stores objects under the terms of the ontology. We assume that the objects of interest belong to an underlying domain that is common to all the sources (e.g. a set of bookstore web pages of interest), and that different sources may use different ontologies with terms that correspond to different natural languages or to different levels of granularity.
Information integration is obtained through a mediator comprising of two parts: (a) an ontology, and (b) a set of articulations to the information sources. Here, by articulation we mean a set of relationships between terms of the mediator ontology and related terms of every local/target ontology. Information requests (queries) are addressed to the mediator whose task is to analyze each query, expand it based on the user choices, normalize it for better results, rewrite it and translate it into queries formatted so as to be understandable by the candidate information sources and finally extract the results to answer the original query. We study the querying and answering process in such a process and present algorithms for handling the main tasks of the mediator, namely, query processing and translation between the mediator and the sources, query analysis, query normalization and relaxation, and result extraction in order to produce the final answer.

3.2 Reference Architecture

The OWL/SPARQL based Mediator for integration of OWL/RDF Information Sources presented in this thesis has been implemented as part of a Semantic Query Mediation Prototype Infrastructure developed in the TUC-MUSIC Lab.

The reference architecture of this system is shown in Figure 3.1, where many of the implementation details of the Mediator are not presented for simplicity reasons.

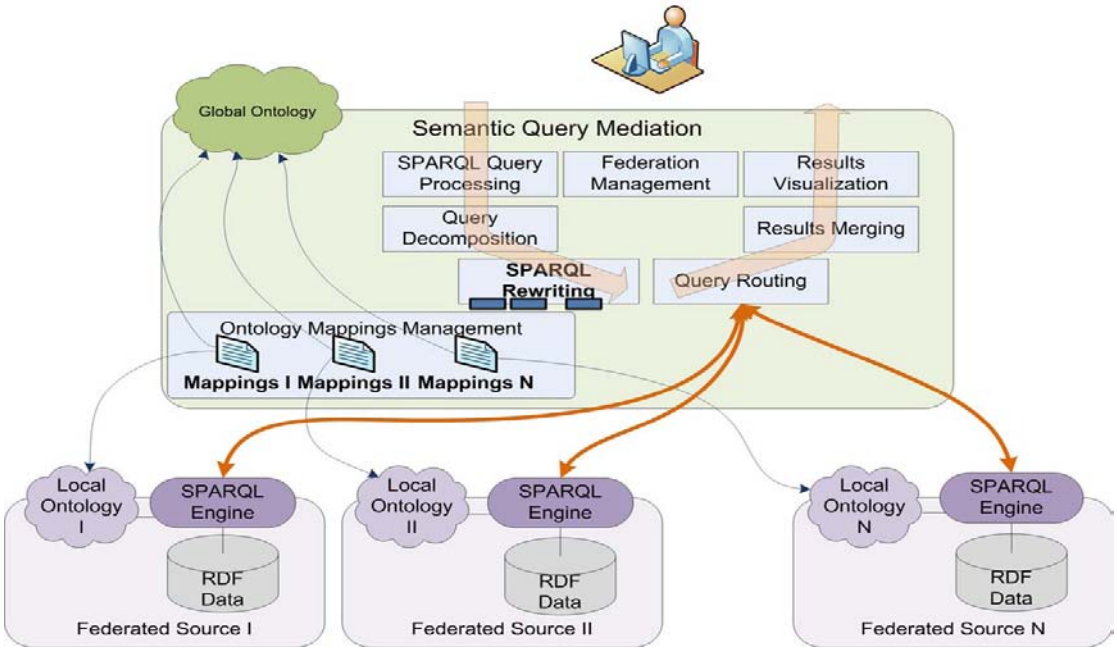


Figure 3.1: System reference architecture.

For each federated source, a dedicated Query Rewriting component is dynamically created by a Query Rewriting Factory. Such a component is able to rewrite an input SPARQL query based on some predefined mappings. As a representation language for the mappings between two overlapping ontologies we use the language presented in the Appendix.

During the system's start-up each component is initialized with the mappings between the global ontology of the mediator and the local ontologies used in the federated sources for which this component is responsible.

In this diploma thesis, we consider information *sources* over a domain consisting of a denumerable (finite) set of objects. For example, in the web environment, the domain could be the set of all the web pages and, in particular, the set of all the pointers to web pages. Each source has an *ontology*, that is, a structured set of names, or *terms*, that are familiar to

the users of the source. In particular, the ontologies that we consider in this thesis consist of a set of terms structured by subsumption relations. In addition, each source maintains a database storing objects that are of interest to its users. Specifically, each object in the database of a source is indexed under one or more terms of the ontology of that source. A user who looks for objects of interest can browse the ontology of the source until he reaches the desired terms. The source will then return the appropriate set of objects. In the environment of the web there are many examples of such sources. Specifically, the general purpose catalogs of the web, such as Yahoo! or Open Directory 1, the domain specific catalogs/ gateways (e.g. for medicine, physics, tourism), as well as the personal bookmarks of the web browsers can be considered as examples of such sources.

However, although several sources may carry information about the same domain, they usually employ different ontologies, with terms that correspond to different natural languages, or to different levels of granularity, and so on. For example, consider two sources IS1and IS2 that provide access to products of bookstores (books, CDs etc) as shown in figure 3.2 .

Suppose now that we want to provide a unified access to these two sources through a single ontology which is familiar to a specific group of users. An example of such a unifying ontology is shown in Figure 3.2 also, and constitutes part of what we call a "mediator".

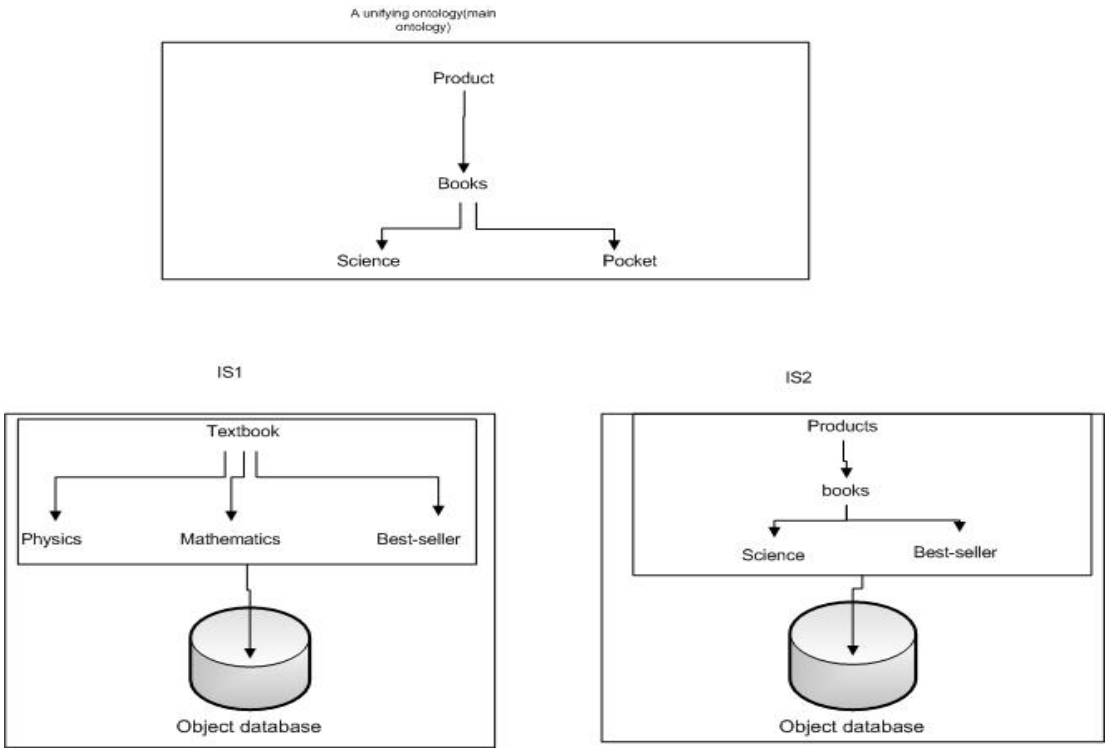


Figure 3.2 : two sources providing access to bookstores

The mediator has an ontology with terminology and structuring that reflects the needs of its potential users, but does *not* maintain a database of objects. Instead, the mediator has a number of *articulations* to the sources. An articulation to a source is a set of relationships between the terms of the mediator ontology and the terms of the source local ontology. These relationships are defined by the designer of the mediator at design time and are stored in the mediator. Figure 3.3 shows the general architecture of a mediator.

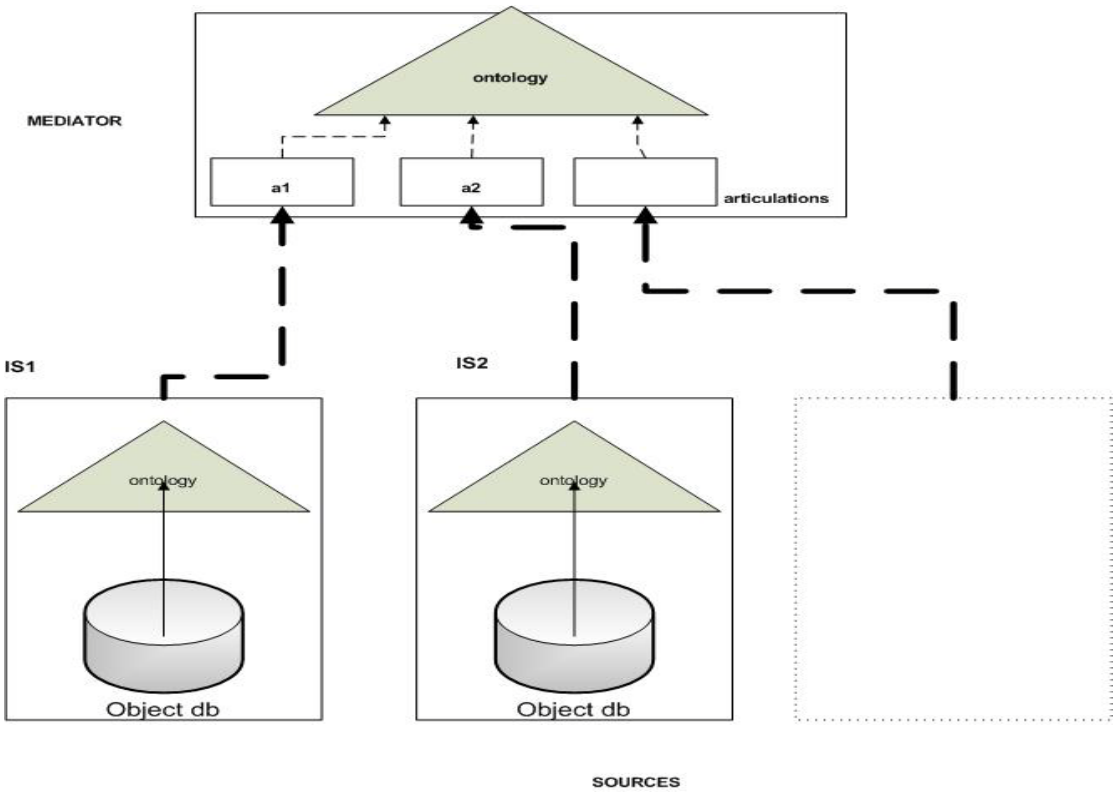


Figure 3.3: A mediator architecture

The users formulate queries over the ontology of the mediator and the mediator is responsible to choose the sources to be queried, and the query to be sent to each source. Then it is again the mediator that appropriately combines the results returned by the sources in order to produce the final result set.

Specifically, the mediator uses the articulations in order to analyze and translate queries over its own ontology to queries over the ontologies of the articulated sources. An essential feature that distinguishes our approach is that there are several options on the user's discretion, such as the selection of the variables that has posed in his query that must be answered or the selection of variables that are more relaxed and can offer results if there are available mappings for the but there will be an answer even though there are no compatible terms for this kind of variables in local/target ontologies.

More precisely through our model there are two user configured types of variables, in each query, namely, **required** and **optional variables** (the first type being suitable for variables that are mandatory by the user, who wishes exact answers for them, while the second is for variables that can add more results to the final answer, however it is not mandatory)**.** This is

crucial for the final answer concerning each information source, because it reflects also to the query expansion and relaxation.

At this point, we have to make a clear distinction between the OPTIONAL operator in SPARQL and the optional variables in our model.

Optional parts of the graph pattern in SPARQL may be specified syntactically with the OPTIONAL keyword applied to a graph pattern:

```
pattern OPTIONAL { pattern }
```

The syntactic form:

```
{ OPTIONAL { pattern } }
```

is equivalent to:

```
{ { } OPTIONAL { pattern } }
```

Grammar rule:

```
[23]   OptionalGraphPattern ::= 'OPTIONAL' GroupGraphPattern
```

The OPTIONAL keyword is left-associative :

```
pattern OPTIONAL { pattern } OPTIONAL { pattern }
```

is the same as:

```
{ pattern OPTIONAL { pattern } } OPTIONAL { pattern }
```

In an optional match, either the optional graph pattern matches a graph, thereby defining and adding bindings to one or more solutions, or it leaves a solution unchanged without adding any additional bindings.

Example 3.1 shows how optional part works.

Data:

```
@prefix foaf:       <http://xmlns.com/foaf/0.1/> .
@prefix rdf:        <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

_:a  rdf:type        foaf:Person .
```

```
_:a  foaf:name       "Alice" .
_:a  foaf:mbox       <mailto:alice@example.com> .
_:a  foaf:mbox       <mailto:alice@work.example> .

_:b  rdf:type        foaf:Person .
_:b  foaf:name       "Bob" .
```

Query:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE  { ?x foaf:name  ?name .
         OPTIONAL { ?x  foaf:mbox  ?mbox }
       }
```

Query Results

name	mbox
"Alice"	<mailto:alice@example.com>
"Alice"	<mailto:alice@work.example>
"Bob"	

There is no value of mbox in the solution where the name is "Bob".

This query finds the names of people in the data. If there is a triple with predicate mbox and the same subject, a solution will contain the object of that triple as well. In this example, only a single triple pattern is given in the optional match part of the query but, in general, the optional part may be any graph pattern. The entire optional graph pattern must match for the optional graph pattern to affect the query solution.

In our implementation, we use optional variables for convenience and for providing more options for answers. For our purpose, based on the user's options we make the appropriate transformation to the query and to the relaxation to achieve better results.

Another important feature of our implementation is the discrimination we make, depending on the user's choices about the query processing. After the selection of the required variables, the query is going through a transformation we are calling **Expanded SPARQL expression.** A distributed and expanded form of the original query is presented with clear distinction of the required and optional variables. The next step of the query processing is the **Query Normalization,** where the original query is transformed and optimized in order to

be suitable for reformulation and better result extraction. The third step is **Query Relaxation.** Through this step, we propose a user configured model consisting of four levels, namely, level1: No Relaxation, Level 2: Unsupported OPTIONAL parts, Level3: Unsupported UFGP and Level4: Unsupported Triples (the first type of relaxation being appropriate for users that focus on precision, while the rest of them for users that focus on recall).

What kind of relaxation will be used at the mediator level and what kind of answer will be requested at the source level is decided by the mediator designer at design time and/or the Mediator user at query time. Therefore, a prominent feature of our approach is that sources and mediators can operate in a variety of modes according to specific application needs. In the semantic web context our mediator can be used for providing unified access to multiple web sites.

The last stage of query processing is **Query Reformulation,** where query rewriting of SPARQL queries expressed according to the main ontology in terms of the local ontologies, is executed. The algorithms performing graph pattern rewriting are based on a set of predefined mappings.

All the above are more clearly and technically described in Chapter 4.

Finally, results from each source are extracted and formatted based on the terms of the main ontology. Each result set (from each information source) is presented separately and a unified set of results is presented at the end. The last thing our mediator does is the **ranking** of the final unified result set based on the real level of relaxation that the normalized query has been put through and the number of the asked variables that have results.

The flowchart below shows the flow of the information throughout the system, from the moment a user poses his query over the mediator until the result extraction and ranking.

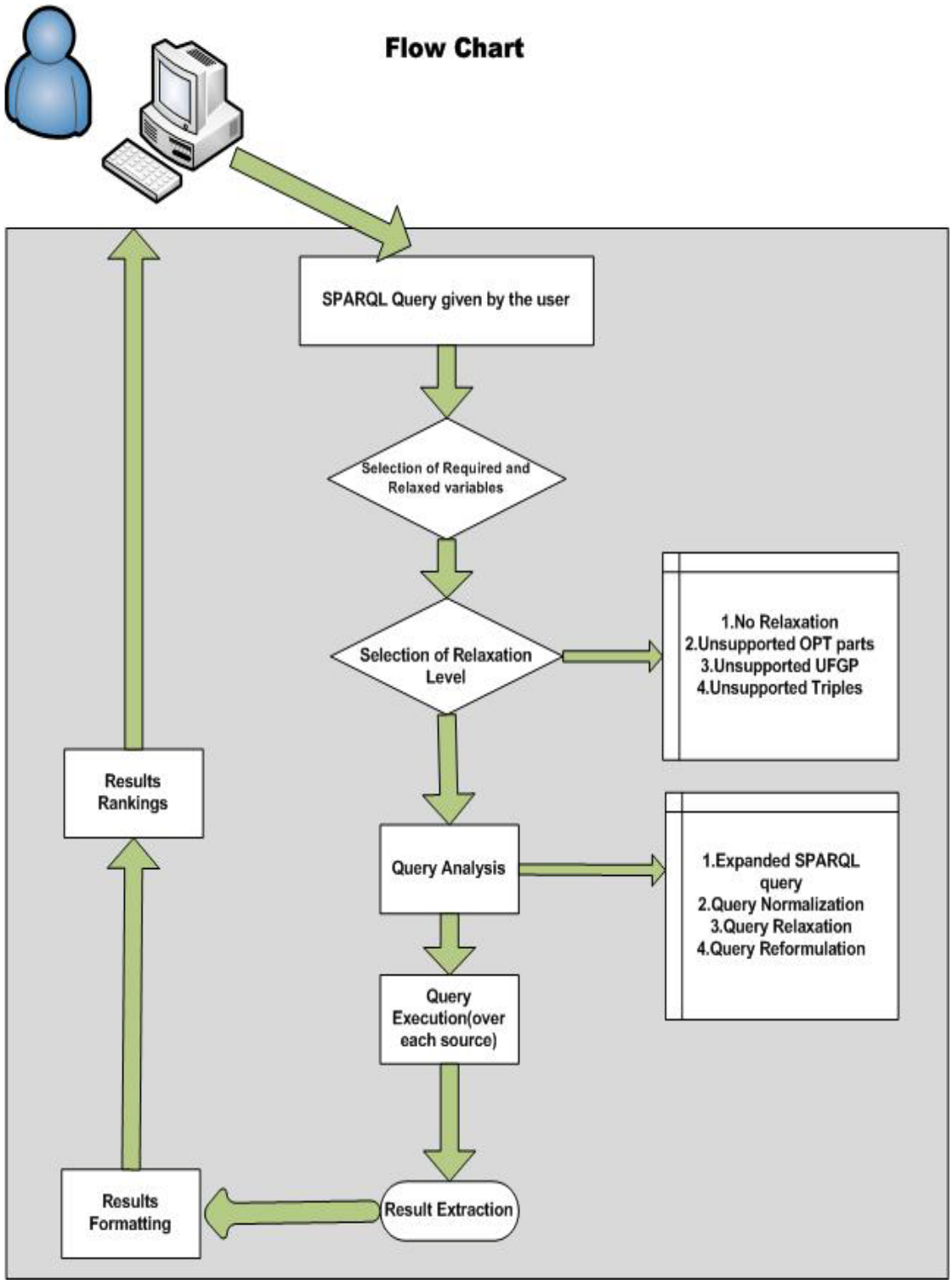


Figure 3.4 : Flow of the information

3.3 Basic Concepts

In this section we provide the formal definitions of the fundamental terms used throughout this diploma thesis:

- **Ontology**

Ontologies are considered as one of the pillars of the Semantic Web, and a well-accepted definition for an ontology is that "an ontology is a logical theory accounting for the intended meaning of a formal vocabulary, i.e., its ontological commitment to a particular conceptualization of the world" [Gua98]. Intuitively, an ontology consists of a set of words, or terms, a set of objects of interest as well as the relationships between the terms on which are based the corresponding relationships between the objects. An ontology provides a shared vocabulary, which can be used to model a domain — that is, the type of the objects and/or **terms** that exist in the domain, and their properties and relations.

Ontologies are used in artificial intelligence, the Semantic Web, software engineering, biomedical informatics, library science, and information architecture as a way of knowledge representation about the world or some part of it.

The ontologies that are used here are expressed in **OWL** syntax and describe data of the same domain but with different structures.

In this case, we consider the **main ontology** of the mediator to "represent" the **local ontologies** of each information source.

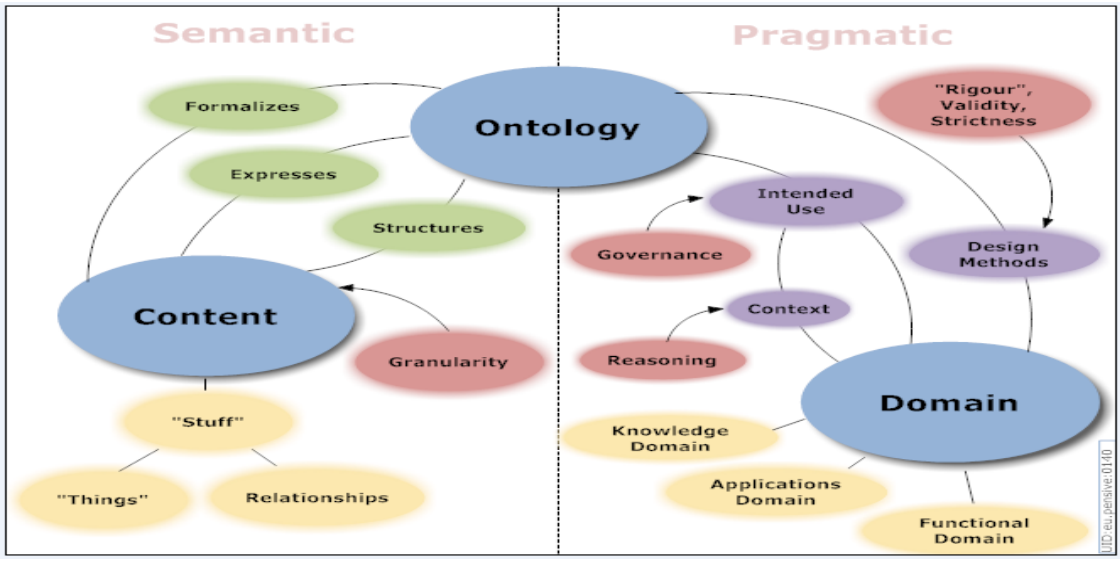


Figure 3.5: Ontology graphic representation

The above figure is a graphic representation of a general ontology. As it is shown above an ontology connects the semantic terms with the pragmatic terms.

More specific, a part of the source ontology we will use, follows:

```
<!DOCTYPE rdf:RDF [
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY owl2xml "http://www.w3.org/2006/12/owl2-xml#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
    <!ENTITY SourceOntology "http://www.owl-ontologies.com/SourceOntology.owl#" >
]>



    <!-- http://www.owl-ontologies.com/SourceOntology.owl#author -->

    <owl:ObjectProperty rdf:about="#author">
        <rdfs:domain rdf:resource="#Book"/>
        <rdfs:range rdf:resource="#Person"/>
    </owl:ObjectProperty>


  <!-- http://www.owl-ontologies.com/SourceOntology.owl#Book -->

    <owl:Class rdf:about="#Book">
        <rdfs:subClassOf rdf:resource="#Product"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#author"/>
                <owl:minCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#publisher"/>
                <owl:cardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
            </owl:Restriction>
        </rdfs:subClassOf>
        <owl:disjointWith rdf:resource="#CD"/>
        <owl:disjointWith rdf:resource="#DVD"/>
    </owl:Class>




  <!-- http://www.owl-ontologies.com/SourceOntology.owl#collection001 -->

    <Collection rdf:about="#collection001">
        <name rdf:datatype="&xsd;string">Harry Potter</name>
    </Collection>
```

Example 3.2 :Ontology OWL code representation

At the above table we give the code in OWL from a part of an ontology. It is described an object property, a class and an individual stored in the appropriate database.

In order for SPARQL queries posed over a global/main ontology to be rewritten in terms of a local ontology, mappings between the global and local ontologies should be specified.

- **Information sources**

A model is proposed for providing integrated and unified access to multiple information sources. Every information source comprises of two parts:

a) An ontology

b) A database that stores objects described by the terms of the ontology. Every object in the database of an information source is indexed under one or more terms of the ontology of that source.

For convenience reasons, from now on we shall refer to information sources, meaning the distributed local/target ontologies that represent them.

If **IS=is1…isn** , the set of all the underlying information sources then **ISq=is1…isk** is the set of the k information sources that will be queried for the query **Q. ISq** has been specified using the set of articulations **Aq** that will be used for Q.

- **Mappings**

A key aspect of semantic interoperability is the semantic mapping process itself. Traditionally, semantic mapping processes conducted by knowledge engineers have been proposed to bridge this gap. However, knowledge engineers alone are unlikely to cope with the ever increasing amount of the mapping work required, especially as mappings themselves begin to be specialized for different contexts.

The type of mappings that are allowed in the system are those that include classes **c**, object properties **op,** datatype properties **dp** and individuals ***I.***

Although, N:M cardinality mappings can be identified between two ontologies, many problems arise in the exploitation of such mapping types in SPARQL query rewriting. The

main problem is the identification of the source ontology's mapped expression inside a SPARQL query, which needs special treatment in order to be overcomed.

In the context of this diploma thesis, we have used a rich set of 1:N cardinality mapping types, in order for these mapping types to be used for the rewriting of a SPARQL query[Makris et al.2010]. Since our query rewriting methodology is generic, we will be discussing for mappings between a source and a target ontology rather than between global and local ontologies.

We consider a **class** c of the main ontology, that can be mapped to a *class expression **CE*** of a local ontology

Class Mapping: c rel CE, rel:= ≡ | ⊑ | ⊒

As class expression, we imply every complex expression between two or more classes, formed using disjunctions (⊔), conjunctions (⊓) or both. Every class that participates a in class expression may be restricted regarding one or more property values, directly or indirectly connected to the class (directly when the restriction is on a property of the same class and indirectly when the restriction is on a property of another class) using a path (with a precondition that there exists a path connecting the class with the desirable property). Such classes, are referred as *basic class expressions **BCE.*** Therefore we have the following rules:

CE::= BCE | CE ⊔ CE | CE ⊓ CE

BCE::= c | c.R

R::= R' | ¬R' | R && R | R || R

R'::= P oprd V, oprd:= != | = | ≤ | ≥ | < | >

Where

R: constraints and restrictions that can be applied on a class

R': constraints and restrictions of a path from properties P (property path) to a possible value V.

V: data value or individual. The operands that can be used for a property path is differ, according to the type of V. If it is a data values then all the operands can be used (!=, =, ≤, ≥, <, >), however if V is an individual only the operands !=, = can be used.

A property path is a sequence of **object properties** (possibly empty) that ends up in a datatype/object property. It connects the class, the individuals of which we wish to restrict, with the datatype/object property that we want to constrain. A path of object properties that ends with object property is called *object property path* ***Pop*** and the one that ends in datatype property is called *datatype property path* ***Pdp.***

$$P ::= P_{dp} \mid P_{op}$$

$$P_{dp} ::= dp \mid P_{op}.dp$$

$$P_{op} ::= \varnothing \mid op \mid P_{op}.op$$

Similarly to classes, an object property of the main ontology of the mediator can be mapped to an *object property* expression **OPE** of a local ontology.

Object Property Mapping: op rel OPE

As object property expression, we consider every complex expression between two or more object properties using disjunctions (⊔), conjunctions (⊓) or both. An object property expression may also contain the *inverse* of an object property/object property path in an object property expression. Every object property/object property path (Pop) that exists in an object property expression, can be restricted in the values of its domain and range (same as class expressions). An object property/object property path that may be constrained in the values of its domain or range, is a ***basic object property expression BOPE.***

$$OPE ::= BOPE \mid BOPE \sqcup OPE \mid BOPE \sqcap OPE$$

$$BOPE ::= BOPE' \mid BOPE' \sqcap domain(CE) \mid BOPE' \sqcap range(CE) \mid BOPE' \sqcap domain(CE) \sqcap range(CE)$$

$$BOPE' ::= P_{op} \mid inverse(P_{op})$$

Likewise, a datatype property of the main ontology can be mapped to a *datatype property expression* ***DPE*** of a local ontology.

Datatype Property Mapping: dp rel DPE

A datatype property expression is every complex expression between two or more datatype properties, using disjunctions (⊔), conjunctions (⊓) or both. Every datatype property/ property path that exists in a datatype property expression, can be restricted in the values of its domain and range. A datatype property/property property path that ends up to a datatype property Pdp (that may be constrained in the values of its domain or range) is a ***basic datatype property expression BDPE.***

$$DPE::= BDPE \mid BDPE \sqcup DPE \mid BDPE \sqcap DPE$$

$$BDPE::= P_{dp} \mid P_{dp} \sqcap domain(CE)$$

As far as the individuals are concerned, an individual of the main ontology **Im** can be mapped to an individual of a local ontology **Il.**

$$\textit{Individual Mapping:}\ i_m \equiv i_l$$

Finally, using OWL semantics, the equivalence between two different properties or between a property and a property expression implies the equivalence of their domain and range. The same holds, for the subset and superset relations.

Table 3.3 presents the notation and the semantics of the ontology mappings.

Symbol	Explanation
⊑, ⊒	subset/superset operands
&&	logical “and”
||	logical “or”
≡	equivalence operand
⊓	intersection operand
⊔	union operand
¬	Negation operand
∅	empty group
|	logical or
.	It is used to define a sequence from connected terms/roles.
domain(c)	Constraint on a domain of a property at the values of class c.
range(c)	Constraint on a range of a property at the values of class c.

inverse(p)	Inverse of a property p.

Table 3.1: Terminology for different mappings

In this table the symbols used in mappings and their meaning is provided.

- **Mapping sets**

Different mapping formats can be supported by our architecture. The representation language that is used for the description of mappings, combines an *alignment format*, a way of the algorithm results representation used in Ontology matching and the *OMWG mapping language,* a very descriptive language for the representation of ontology mapping. An example of a mapping set follows:

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
…
<rdf:RDF xmlns="http://www.omwg.org/TR/d7/ontology/alignment/" …>
  <Alignment rdf:about="http://example.com/ ">
    <xml>yes</xml>
    …
    <onto1>
      <Ontology rdf:about="&Source;"/>
      …
    </onto1>
    <onto2>
      <Ontology rdf:about="&Target;"/>
      …
    </onto2>
    <map>
      …
      <Cell>
        <entity1>
          <Class rdf:about="&Source;Science"/>
        </entity1>
        <entity2>
          <Class>
            <or rdf:parseType="Collection">
              <Class rdf:about="&Target;Physics"/>
              <Class rdf:about="&Target;Mathematics"/>
            </or>
          </Class>
        </entity2>
        <relation rdf:resource="&omwg;equivalence"/>
      </Cell>
      …
    </map>
  </Alignment>
</rdf:RDF>
```

Example 3.3: Format of mapping sets

The above example code is an example of the language that is used in mapping sets. It shows some details about the mapping of the main ontology (source) and a local ontology (target).

- **Articulations**

An **articulation** between the mediator and an information source is a set of relationships between the terms of the main/source ontology and the terms of the local/target ontologies. These relationships are defined a priori. The mediator uses the articulations in order to translate queries over its own ontology to queries over the ontologies of the articulated sources. An articulation provides information to map terms and concepts from

main ontology to those of a every local one(an articulation unifies one term of the main ontology to many mapping expressions of different local ontologies). Articulations can be used, for example, as the basis for searching across multiple domains, each with its own separate ontology. They can also be used for translating message facts from one ontology into a recipient's desired ontology. Finally, articulations enable the integration of multiple-source operational database systems into an integrated data warehouse/data mining system.

The set of articulations depend on the mapping sets that exist.

In the terms of this thesis, we construct **the set A** of all articulations based on the terms of the main ontology. Specifically for every term existing in the main ontology we declare all the relationships with terms of the locals' ontologies.

Let T be a term in the mediator ontology and m1, m2, …, mn be the mappings of T to terms of the local ontologies of the information sources IS1, IS2, …, ISn. An articulation aT of T is the set { m1, m2, …, mn } of the mappings of T to the local ontology terms. For every posed query q a different set of articulations Aq, concerning the terms of the query, will be computed.

In addition from the sets of mappings, the hashtable that represents articulations and has as key term every term of the main ontology has also as values:

- **Related Info Sources**: the related to the specific term of main ontology information sources
- **Related Terms of Info Sources**: the specific terms of the local ontologies, that correspond to the term of the main ontology
- **Relationship Identifiers**: the kind of relationship existing in the mappings for the specific terms
- **Articulation Number:** a unique number/id for each articulation that represents a term of the main ontology

The hashmap of articulations also includes two smaller hashmaps that unifies and integrates access to the terms and local/target ontologies:

- **Terms of each Source:** this hashmap connects the terms of the local ontologies to the ontology that derive from.
- **Relation of each source**: connects the type of relationship of each mapping to the source that derives from.

In the definition of articulations and during their use, we will refer to the domain and range of properties and not to their values (which may be individuals in case of object properties).

In the context of this diploma thesis, we provide the system with the grid of correlations, we are calling articulations between the local /target ontologies and the main ontology of the mediator M.O .The process is semi-automatic. This set will be:

A= a1…an, where a_i is an articulation for the i_{th} term of the main ontology.

Depending on the query **Q** the user gives, the appropriate set of articulations for **Q Aq=a1…ax** is calculated**,** which will provide the information about the information sources that will be queried.

The following example gives a simple view of an articulation.

a1:{*key*: **term of main Ontology** → Product

values: **Articulation Number** →1

Related Info Sources→IS1(Target Ontology 1),
IS2 (Target Ontology 2)

Related Terms of Info Sources → Products, Textbook

Relationship Identifiers → equivalence ≡, superset ⊇

Mappings Specification → Product(x)≡ Products(x),
Product(x)⊇ Textbook(x)

Terms of each Source→ IS1:Textbook, IS2: Products

Relation of each source → IS1: superset, IS2: equivalence

Example 3.4: Example of articulations

Above there is an example of an initiate form of articulations.

At this point we introduce the symbols presented in table 3.2, which will be used through the definition of the concepts.

Symbol	Explanation- Meaning
≡	equivalence
⊇	Superset: is a superset of A⊇ B
⊆	Subset : is a subset of A ⊆B
←	Left arrow (used in case of inverseOf), Implies the exchange of domain and range
≼	Partial ordering, the subsumer expresses a generalisation over the subsumed.

Table 3.2: Articulations Symbols

The above table shows the symbols that will be used in order to represent the articulations between the information sources and the main ontology.

Now, we can move to more generic definitions:

- **Class Extension**

For the classes we consider the set of individuals of each class, which form the class extension. Sometimes we want to emphasize the distinction between a class as an object and a class as a set containing elements. We call the set of individuals that are members of a class the ***extension*** of the class. If the combined ontologies are contradictory (all A's are B's vs. all A's are not B's) there will be no extension (no individuals and relations) that satisfies the resulting combination. OWL class extensions are sets consisting of the individuals that are members of the class. OWL provides the means to manipulate class extensions using basic set operators.

The definition of the type of the mappings (**≡, ⊇, ⊆, inverse**) that exist between the domain/range of the classes of the mediator and those of the information source ontologies. We shall refer to the classes of the target information source local ontologies with C_t, and to the classes of the main ontology of the mediator with C_s.

- $C_s \equiv C_t$: $domain_{Cs} \equiv domain_{Ct}$ and $range_{Cs} \equiv range_{Ct}$
- C_s **inverseOf** C_{Ct} : $domain_{Cs} \equiv range_{Ct}$ and $range_{Cs} \equiv domain_{Ct}$
- $C_s \supseteq C_t$: $domain_{Cs} \supseteq domain_{Ct}$ and $range_{Cs} \supseteq range_{Ct}$
- $C_s \subseteq C_t$: $domain_{Cs} \subseteq domain_{Ct}$ and $range_{Cs} \subseteq range_{Ct}$

- **Domain-range**

A very important issue that must be explained throughout is the matter of the meanings of the domain and range of the properties of ontology. For our convenience and for the usage into mappings and articulations, we have to define the meanings of the range and domain properly.

Properties: When the user defines a relationship between two properties, automatically specifies a relationship between their range and domain.

Enhanced accuracy in mappings can be achieved, by giving the possibility of restricting each class, contributing into the domain/range of the mapped object properties of the ontology target (local). In OWL properties, as domain/range we can have a single class or the union of more than one classes.

The restrictions that can be set to the domain/range of the mapped object properties of the local ontology are:

- Removal of a class (or more than one) from a domain/range
- Adding a constraint of a `ClassByAttributeValue` type in one or more of the classes on a domain/range

At first, the user will define the type of the mappings (**≡, ⊇, ⊆, inverse**) that exist between the object properties of the mediator and the information source ontologies. We to the

object properties of the target information source local ontologies with P_t, and to the object properties of the main ontology of the mediator with P_s.

- $\mathbf{P_s \equiv P_t}$: $domain_{Ps} \equiv domain_{Pt}$ and $range_{Ps} \equiv range_{Pt}$
- $\mathbf{P_s}$ **inverseOf** $\mathbf{P_{Pt}}$: $domain_{Ps} \equiv range_{Pt}$ and $range_{Ps} \equiv domain_{Pt}$
- $\mathbf{P_s \supseteq P_t}$: $domain_{Ps} \supseteq domain_{Pt}$ and $range_{Ps} \supseteq range_{Pt}$
- $\mathbf{P_s \subseteq P_t}$: $domain_{Ps} \subseteq domain_{Pt}$ and $range_{Ps} \subseteq range_{Pt}$

In case that there exists **inverseOf**, we exchange the domain with the range and vice versa. From now on the symbol ← is defined, to imply the relationship of inverseOf into articulations. In this case, if a property **a** is inverseOf a property **b**, then we write a ←b and b⊆a, and we mean that the domain of property a is the range of property b and vice versa. It is also implied that the domain of a is a subset of the range of b and the range of b is a subset of the domain of a.

From the definition of articulations it comes out that the domain/range of a property is a subset (≼) of the domain/range of another property. If the domain/range of a property **a** is a subset of the domain/range of a property **b**, and also the domain/range of **b** is a subset of the domain/range **a**, then **equivalence** exists between the domains/ranges of these properties.

Below is a table of all the abbreviations that will be used from now on:

Term	Briefing	Short Definition
Main Ontology	M.O	Ontology of the mediator
Local Ontology	L.O	Ontology for each information source
Information sources	IS={is1,…,isn}	Set of all the underlying sources of the system
Queried sources	ISq={is1,…,isk} , ISq⊆IS	Set of the sources, that will be queried, subset of the set IS of all the sources
Mappings		Functions that connect the terms of a local ontology to the main ontology
Mapping sets		Transformation of the mappings to the right form, so they can be used by articulations for each source separately.
Articulations	A ={a1,…,an}	All the relationships between the M.O and the L.Os of underlying sources
Query-Articulations	Aq={a1,…,aq} ,Aq⊆A	The set of articulations chosen for answering a user query
Initial query	Q	The initial query that the user poses to the system
Transformed queries-subqueries	Qt={q1…qx}	The initial query will be transformed and parsed appropriately for the selected sources.
Results	R={R1∪…∪Rx}	The results of each source are returned and, after the application of the union operator, the final result set is specified.

Table 3.3: Main concepts used in Mediator

3.4 Ontology Mapping Management

In order for SPARQL queries posed over a global ontology to be processed and transformed in terms of local ontologies, mappings between the global and local ontologies should be specified. In this section we present the model we are using in our mediation system, for the expression of mappings between OWL ontologies. The work presented in this section and represents the Ontology mapping model is part of the work that has been done in the TUC-MUSIC Lab [Makris et al.2010].

First, we present a motivating example for elicitating the ontology mapping requirements of the mediator framework. Since our query processing methodology is generic, we will be discussing for mappings between a source and a target ontology rather than between global and local ontologies. However, the management of the set of all eligible mappings with some more properties will be discussed later when presenting the component of articulations.

In Figure 3.6 the motivating example we are using in our implementation is presented. We consider a main/ global ontology as the ontology of the mediator that the user poses his query on. In our example, there are two distributed and heterogeneous information sources, represented by a local ontology each.

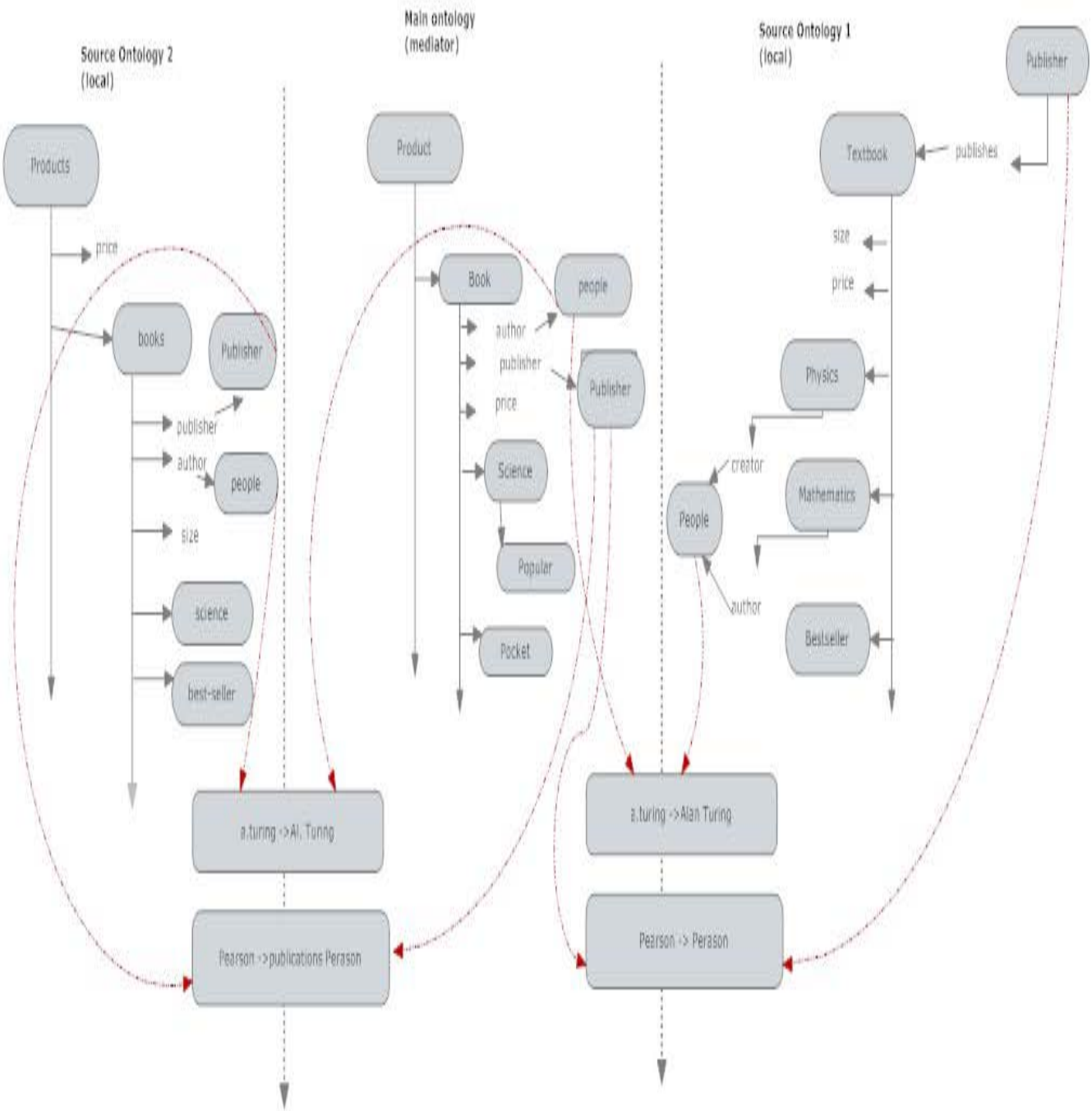


Figure 3.6 : Figure of Main example (motivating example)

The above figure represents the main figure, we are using in all the examples. It is supposed that we have two local Ontologies (sources) and one main ontology in the mediator (target). All the ontologies are consisted of terms from the same domain (in this example LIBRARIES-Books) and we try to unify and access all the stored information

In Figure 3.7, we show a part of the above structure between the main ontology (source ontology) and IS1(target ontology).

The source ontology describes a store that sells various products including books and cd's and the target ontology describes a bookstore. The rounded corner boxes represent the classes. They are followed by their properties (object and datatype). The rectangle boxes at the bottom of the figure represent individuals. The arrows represent the relationships between these basic OWL constructs.

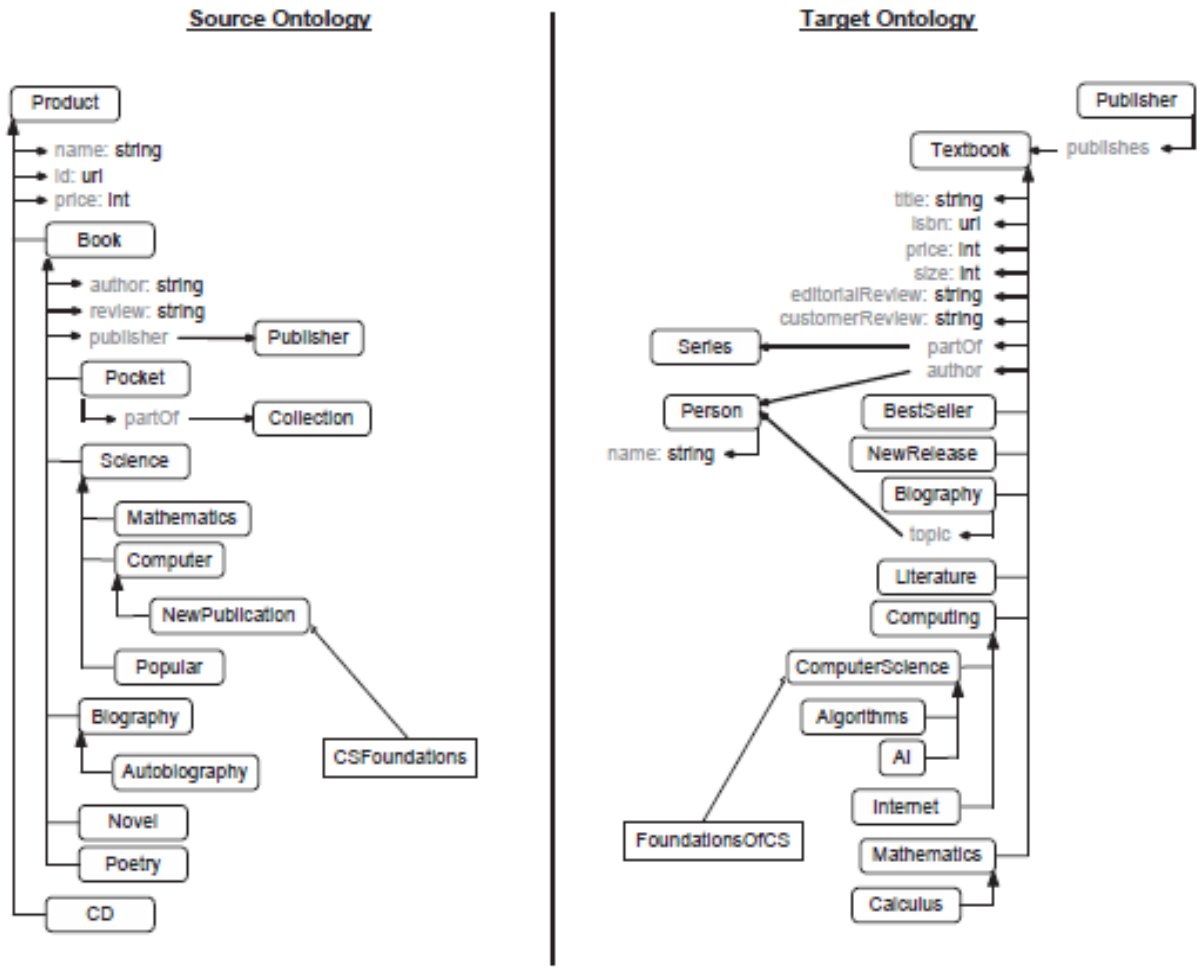


Figure 3.7: semantically overlapping ontologies

In order to map a source ontology to a target ontology, various relationship types like equivalence and subsumption can be used.

A source ontology class can be mapped to an expression between target ontology classes. The expression may involve union and intersection operations between classes.

In addition, using expressions it is possible to restrict a class on some property in order to form a correspondence.

Similarly with classes, an individual from the source ontology can be mapped to an individual from the target ontology . In this case, only the equivalence relationship can be taken into consideration since the subsumption relationship is used mainly with sets.

Accordingly, an object/datatype property from the source ontology can be mapped to an object/datatype property from the target ontology . Domain and range restrictions can be useful for mappings between properties in order to restrict the individuals that participate in these two sets. In addition, an object property from the source ontology can be mapped to the inverse of an object property from the target ontology.

Finally, a source ontology property can be mapped to an expression between target ontology properties. The expression may involve union, intersection and composition operations between properties.

Considering now the example given in figure 3.7 the mappings specified are:

A	***src : Book ≡ trg : Textbook***
B	*src : Product ⊒ trg : Textbook*
C	*src : Publisher ≡ trg : Publisher*
D	*src : Collection ⊑ trg : Series*
E	*src : Novel ⊑trg : Literature*
F	*src : Poetry ⊑ trg : Literature*
G	*src : Biography ≡ trg : Biography*
H	*src : Autobiography ≡ ∀trg : Biography:(trg : author = trg : topic)*
I	*src : NewPublication ≡ trg : Computing ⊓ trg : NewRelease*
J	*src : Science ≡ trg : ComputerScience ⊔ trg : Mathematics*
K	*src : Popular ≡ (trg : ComputerScience ⊔ trg : Mathematics) ⊓ trg : BestSeller*
L	*src : Pocket ≡ ∀trg : Textbook:(trg : size ≤ 14)*
M	*src : publisher ≡ inv(trg : publishes)*
N	*src : partOf ≡ ∀trg : partOf:domain(∀trg : Textbook:(trg : size ≤ 14))*
O	*src : name ⊒ trg : title*
P	*src : id ⊒ trg : isbn*

Q	*src : price ≡ trg : price*
R	*src : review ≡ trg : editorialReview ⊔ trg : customerReview*
S	*src: author ≡ trg : author ∘ trg : name*
T	*src : CSFoundations ≡ trg : FoundationsOfCS*

Table 3.4: Mappings based on Figure 3.7

In order to implement our framework, the need of a serializable language is of major importance. Many languages have been proposed for the task of mapping representation (C-OWL [7], SWRL [22], the Alignment Format [16], MAFRA[26], etc.). Although, the language that fulfills the majority of our requirements is EDOAL3 (Expressive and Declarative Ontology Alignment Language). Previous versions of this language have been defined in [19] and [37]. EDOAL combines the Alignment Format [16], which is used in order to represent the output of ontology matching algorithms, and the OMWG mapping language [38], which is an expressive ontology alignment language. The expressiveness, the simplicity, the Semantic Web compliance (given its RDF syntax) and the capability of using any kind of ontology language are the key features of this language.

As it is already mentioned in Section 3.3 the syntax is based on the syntax of EDOAL. The default namespace applying to the constructs in the following grammar description is oml standing for http://www.music.tuc.gr/oml#, while the namespace align is equivalent to the Alignment Format namespace which is the http://knowledgeweb.semanticweb.org/heterogeneity/alignment#.

The structure of the set of mappings between two overlapping ontologies is the same as that of the Alignment Format:

```
alignment::= <align:Alignment rdf:about="uri">
                <align:onto1> onto </align:onto1>
                <align:onto2> onto </align:onto2>
                (<align:map> cell </align:map>)*
             </align:Alignment>
```

The Ontology construct contains information about an aligned ontology:

```
onto::= <align:Ontology rdf:about="uri">
            <align:formalism> formalism </align:formalism>
      </align:Ontology>

formalism::= <align:Formalism>
            <align:uri> uri </align:uri>
            <align:name> string </align:name>
              </align:Formalism>
```

The mappings between the two overlapping ontologies are structured as a set of cells:

```
cell::= <align:Cell rdf:about="uri">
            <align:entity1> entity1 </align:entity1>
            <align:entity2> entity2 </align:entity2>
            <align:relation> relation </align:relation>
      </align:Cell>
```

The aligned construct from the source ontology can be a class, an object/datatype property, or an individual. It is worth to mention that since the current version of EDOAL provides the capability of using any kind of ontology language, it adapts a more generic vocabulary. More specifically, it refers to OWL object properties as *relations*, due to the fact that they relate class instances. Furthermore, it refers to OWL datatype properties as *properties* and to OWL individuals as *instances*. Finally, the term *attribute* is used in order to refer to both *relations* and *properties*.

Using the above syntax the mapping between the class Science from the source ontology and the union of classes Computer Science and Mathematics from the target ontology (mapping *J* of Table 3.4) can be represented as follows:

```
<align:Cell rdf:about="MappingRule_j">
      <align:entity1>
            <oml:Class rdf:about="&src;Science"/>
      </align:entity1>
      <align:entity2>
            <oml:Class>
            <oml:or rdf:parseType="Collection">
```

```
                <oml:Class rdf:about="&trg;ComputerScience"/>
                <oml:Class rdf:about="&trg;Mathematics"/>
            </oml:or>
        </oml:Class>
</align:entity2>
<align:relation>Equivalence</align:relation>
</align:Cell>
```

Example 3.4 : Alignment mappings example

After having fully described the kind of mappings and the format we support in our system, we shall refer to the general management of the mappings throughout the system's functioning with several different target/local ontologies. After the establishment of the system we have to connect all the sources given to the mediator. In this case, we must associate the databases with the local ontologies. Every ontology is associated with (at least one) data source and with the main ontology of the mediator. The main ontology is also constructed based on the information we are given. The mediator ontology is more general than the local ones, because it has to "cover" the majority of the constructs (classes, properties, individuals) of the sources ontologies. Moreover, for the association of the ontologies we use the **mappings** mentioned above**.** Ontology mapping is important when working with more than one ontologies. Typically, similarity considerations are the basis for this. The first step is the definition of the mappings between local ontologies and main ontology of the mediator.

Based on the motivating example presented in Figure 3.6, the mappings for each information source are:

- For IS1

1. ∀x, [s:Popular(x) ⇔ t:((Physics(x) ∨ Mathematics(x)) ∧ BestSeller(x))]

2. ∀x, [s:Pocket(x) ⇔t: (Textbook(x) ∧ ∃y; [size(x, y) ∧ y≤14])]

3.∀x, ∀y [s:author(x, y) ⇔t: (author(x, y) ∧ creator(x, y))

4.∀x, [s: ATuring = t:AlanTurnig]

5.∀x, [s:Pearson = t:Pearson]

6.∀x, [s:Publisher → inverseOf (t:publishes)] **OR** ∀x, ∀y [publisher(x, y) ⇔ publishes(y, x)]

7.∀x, [s: Book(x) ⇔t:Textbook(x)]

8.∀x, [s:Price(x) ⇔t:price(x)]

- For IS2

1. ∀x, [s:Popular (x) ⇔t:best-seller(x)]

2.∀x, [s:Pocket(x) ⇔ (t:books; s:size(x, y) ≤14]

3.∀x, ∀y, [s:Author(x,y) ⇔ t:author(x,y)]

4.∀x, [s: ATuring= t:Al.Turnig]

5.∀x, [s:Pearson = t:PublicationsPearson]

6.∀x, [s:Publisher(x) ⇔ t:Publisher(x)]

7.∀x, [s: Book(x) ⇔t:books(x)]

8.∀x, [s:Price(x)⇐ t:price(x)]

Example 3.5: Initial form of mappings

In the above example , the mappings between the terms of the two information sources , shown in figure 3.6, and the main ontology, are represented in an initial form.

The above mappings are translated from simple algebraic form to a format used by the mediator, using the language EDOAL3 described in *Appendix: Mapping Representation Example.*

Having now strong definitions of mappings, the set of articulations can emerge by transforming appropriately the format of the given mappings. The set of articulations is equivalent to alignment, because a relation can be decomposed into a pair of functions from some intermediate source. The defined set includes all the relationships between the M.O and the L.Os and is based on the mappings. The goal of the **Articulations Structure** is to present the set of available relationships for every term (class, property, individual) of the main ontology, that there exists in a mapping, with terms of the distributed sources.

Below a table of the data available in articulations structure is presented

Articulations	Related information sources	Term of M.O.	Terms of information Sources	Relationship Identifiers	Mappings specification
A1	M.O, IS1, IS2	Product	Products(IS2), Textbook(IS1)	equivalence ≡, superset ⊇	Product(x)≡Products(x), Product(x)⊇Textbook(x)
A2	M.O, IS1, IS2	Book	Textbook(IS1), books (IS2)	equivalence ≡	Book(x) ≡ Textbook(x), Book(x) ≡ books(x)
A3	M.O, IS1, IS2	Science	Physics(IS1),Mathematics(IS1),science(IS2)	superset ⊇, equivalence ≡	Science(x)≡Physics(x)\/ Mathematics(x), Science(x)≡science(x)
A4	M.O, IS1, IS2	Popular	Bestseller(IS1), Physics(IS1), Mathematics(IS1),bestseller(IS2)	subset ⊆,equivalence ≡	Popular(x)≡(Physics(x)\/ Mathematics(x)) /\ BestSeller(x), Popular(x)≡bestseller(x)
A5	M.O, IS1, IS2	Pocket	Textbook(IS1), books (IS2)	subset ⊆	Pocket(x)≡(Textbook(x) /\ ∃y, (size(x, y) /\ y≤14), Pocket(x)≡books(x)/\ ∃y, (size(x, y)/\ y≤14)
A6	M.O, IS1, IS2	a.Turing	AlanTuring(IS1),AI.Turing(IS2)	equivalence ≡	a.Turing=AlanTurnig, a.Turing=AI.Turing
A7	M.O, IS1, IS2	Price	price(IS1), price(IS2)	equivalence ≡	Price(x)≡price(x)
A8	M.O, IS1, IS2	Pearson	Pearson(IS1), publicationsPearson(IS2)	equivalence ≡	Pearson=Pearson,Pearson=publicationsPearson
A9	M.O, IS1, IS2	Publisher	publishes(IS1), publisher(IS2)	inverseOf ←, equivalence ≡	Publisher → inverseOf (publishes) OR ∀x, ∀y (publisher(x, y) ⇔ publishes(y, x), Publisher(x)≡publisher(x)
A10	M.O, IS1, IS2	Author	author(IS1),creator(IS1),author(IS2)	equivalence ≡	author(x, y) ≡ (author(x, y) /\ creator(x, y)), author(X) ≡author(x)

Table3.5 : Table of articulation description for all the terms of main ontology.

All the above definitions and calculations must be ready before the user poses his query. So we call all these actions **preprocessing activities.**

A graphical representation of the preprocessing activities is shown in Figure 3.8. The ontology based management is based on these activities. Using now mappings and the articulations structure, we can easily and faster exploit all the given information and extract suitable results. In real time functioning the mediator is enriched with all the above, and further action by the user is required, when posing his SPARQL query.

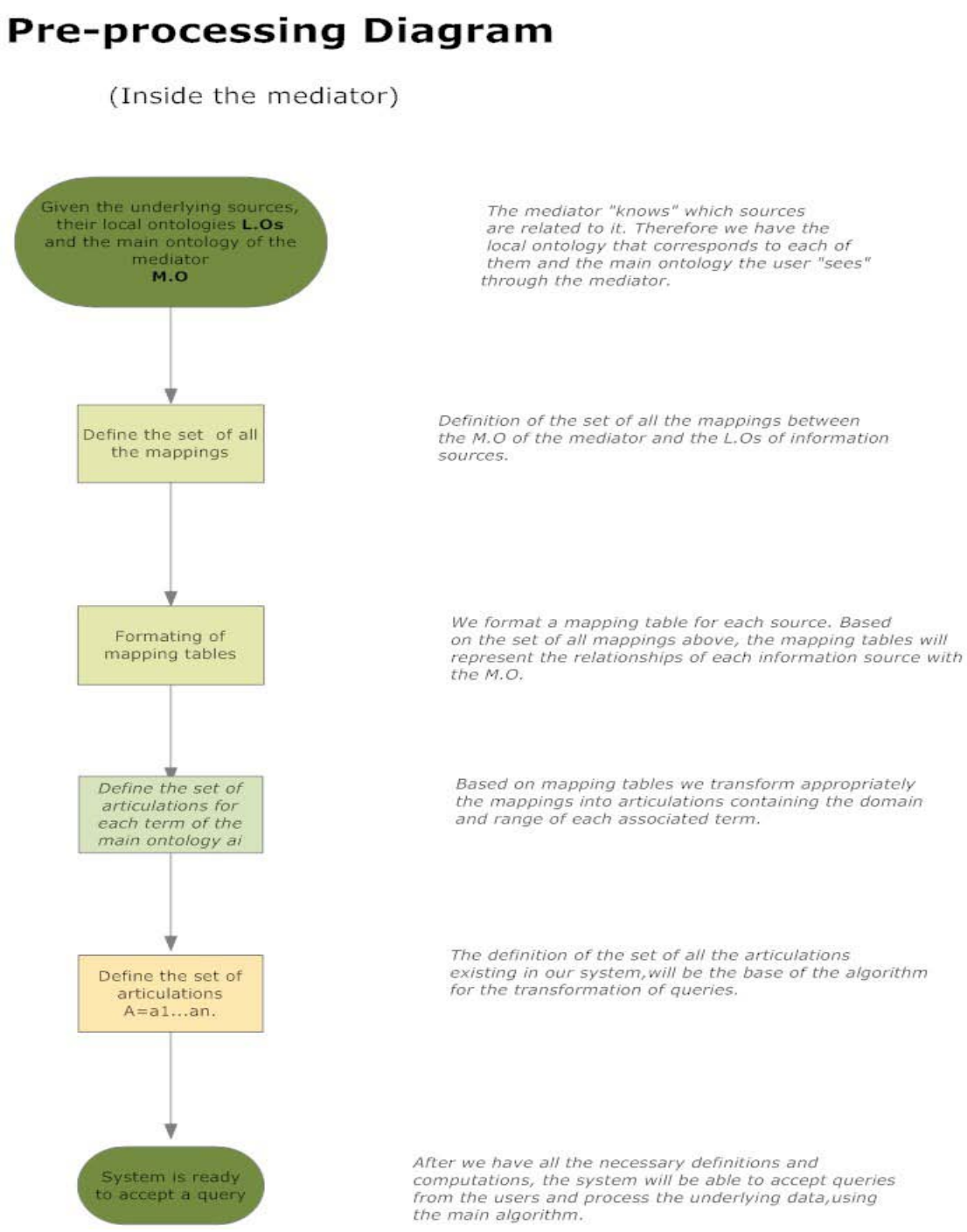


Figure 3.8: Preprocessing Activities

This figure shows the processes that are taking place before the system starts functioning.

The algorithm of the actions described above is defined in Appendix.

3.5 Query Normalization

In this Section, a core part of the query normalization throughout our Mediation System's functioning is presented. Query Normalization. The query normalization includes the Graph Pattern Normalization which makes the normalization of the pattern graph of SPARQL queries .The Normalization is intended to streamline and simplify the process of translation. Retroactive application of rules of equivalence properties of operators that appear between Patterns and graph re-writing techniques, have resulted in a normalized grammar for the pattern graph of the question. More specifically we use the descriptions and definitions given in [Perez et al.]

The most important task for the Mediator is the parsing of the queries so that we can reach every appropriate source and extract the correct results. The initial query should, in some cases, be normalized appropriately so that can be answered by an individual source.

In the context of this diploma thesis we define a SPARQL query Q as a tupple *Q = (E,DS,R).* The basis of a SPARQL query is an algebra **expression E** that is evaluated with respect to an RDF graph in a **dataset DS**. The results of the matching process are processed according to the definitions of the **result form R (SELECT, CONSTRUCT, DESCRIBE,ASK).**
The algebra expression E is built from different graph patterns and can also include solution modifiers, such as PROJECTON, DISTINCT, LIMIT, or ORDER BY.

The simplest graph pattern defined for SPARQL is the **triple pattern**. A **triple pattern t** is similar to an RDF triple but allows the usage of variables for subject, predicate, and object:

$$t \in TP = (\text{RDF-T} \cup V) \times (I \cup V) \times (\text{RDF-T} \cup V)$$

with *RDF-T being the set of RDF Terms (RDF Literals and Blank Nodes), I being the set of all the IRIs, and V the set of all the variables.*

A **basic graph pattern BGP** is defined as a set of triple patterns BGP ={t1…tn} with t1…tn ∈TP. It matches a subgraph if all the contained triple patterns match. Basic graph patterns can be mixed with **value constraints (FILTER**) and other graph patterns. The evaluation of basic graph patterns and value constraints is order independent. This means, a structure of two basic graph patterns BGP1 and BGP2 separated by a constraint C can be transformed in

an equivalent basic graph pattern followed by the constraint. We refer to a basic graph pattern followed by one or more constraints as filtered basic graph pattern(FBGP). SPARQL also defines other types of GP such as graph, union and optional.

The data we are using are in RDF format. RDF is a directed labeled graph data format and, thus, SPARQL is essentially a graph-matching query language. SPARQL queries are composed by three parts. The *pattern matching part*, includes several interesting features of pattern matching of graphs, like optional parts, union of patterns, nesting, filtering values of possible matchings, and the possibility of choosing the data source to be matched by a pattern. The *solution modifiers*, once the output of the pattern has been computed (in the form of a table of values of variables), allow to modify these values applying classical operators like projection, distinct, order, and limit. Finally, the *output* of a SPARQL query can be of different types: yes/no queries, selections of values of the variables which match the patterns, construction of new RDF data from these values, and descriptions of resources.
One of the delicate issues in the definition of semantics for SPARQL is the treatment of *optional matching* and incomplete answers. The idea behind optional matching is to allow information to be added if the information is available in the data source, instead of just failing to give an answer whenever some part of the pattern does not match. This feature of optional matching is crucial in Semantic Web applications, and more specifically in RDF data management, where it is assumed that every application has only partial knowledge about the resources being managed.

The definition of a formal semantics for SPARQL has played a key role in the standardization process of this query language. Although taken one by one the features of SPARQL are intuitive and simple to describe and understand, it turns out that the combination of them makes SPARQL into a complex language.
In the article (see Perez et al. [2006a]), it is presented one of the first formalizations of a semantics for a fragment of the language. Currently, the official specification of SPARQL [Prud'hommeaux and Seaborne 2008], endorsed by the W3C, formalizes a semantics based on work of [Perez et al. 2006a, 2006b; Arenas et al. 2007].

A formalization of a semantics for SPARQL is beneficial for several reasons, including to serve as a tool to identify and derive relations among the constructors that stay hidden in the use cases, identify redundant and contradicting notions, to drive and help the implementation of query engines, and to study the complexity, expressiveness, and further natural database questions like rewriting and optimization.

.

Well-designed patterns form a natural fragment of SPARQL that is very common in practice. Furthermore, well-designed patterns have several interesting features. We present several rewriting rules for well-designed patterns whose application may have a considerable impact in the cost of evaluating SPARQL queries, and prove the existence of a normal form for well-designed patterns based on the application of these rewriting rules.

Retroactive application of rules of equivalence properties of operators that appear between Patterns and graph re-writing techniques, have resulted in a normalized grammar for the pattern graph of the question. The process of Graph Pattern Normalization is the first procedure performed in the translation of questions, the process takes as input a SPARQL query and makes normalization of the pattern graph that contains. The normalized graph pattern is to draw then the other processes of translation. The normalized graph pattern has the form P1 union P2 union ... union Pn, where P1, P2 ... Pn Patterns are union free graphs. These results simplify the translation process as the original pattern is decomposed into n Patterns from each cluster is translated independently of the others. The application of rules of equivalence, the properties of operators that appear between Patterns and graph re-writing techniques result in the removal of Patterns writing and creating more (adjacent) basic graph patterns. The goal is to achieve the greatest possible number of consecutive triple Patterns with the operator AND among them.

Table 3.6 shows the correlation between the syntax rules of grammar and language in algebraic syntax, and examples that follow are for understanding using these syntax. Let P1 and P2 write and Patterns R SPARQL expression

SPARQL Syntax	Algebraic Syntax
{ P1 P2 }	(P1 AND P2)
{ P1 OPTIONAL { P2 }}	(P1 OPT P2)
{ P1 } UNION { P2 }	(P1 UNION P2)
{ P1 FILTER (R) }	(P1 FILTER R)

Table 3.6: SPARQL and Algebraic Syntax

The normalization is designed to optimize and simplify the process of translation.

After the user poses his query the system will be able to parse, transform and evaluate the original query, in order to extract the better possible results.

Firstly we present some significant definitions about the Syntax and the Semantics of SPARQL graph pattern expressions. We give an algebraic formalization of the core fragment of SPARQL over simple RDF, that is, RDF without RDFS vocabulary and literal rules.

Assume there are pairwise disjoint infinite sets I ,B, and L (IRIs [Durst and Suignard 2005], Blank nodes, and Literals, respectively). A triple $(s, p, o) \in (I \cup B) \times I \times (I \cup B \cup L)$ is called an *RDF triple*. In this tuple, s is the *subject*, p the *predicate*, and o the *object*. Assume additionally the existence of an infinite set V of variables disjoint from the previous sets.

***Definition* 3.1**. An *RDF graph* [Klyne et al. 2004] is a set of RDF triples. In our context, we refer to an RDF graph as an *RDF dataset*, or simply a *dataset*.

SPARQL is essentially a graph-matching query language. A SPARQL query is of the form $H \leftarrow B$, where B, the *body* of the query, is a complex RDF graph pattern expression that may include RDF triples with variables, conjunctions, disjunctions, optional parts, and constraints

over the values of the variables, and *H*, the *head* of the query, is an expression that indicates how to construct the answer to the query. The evaluation of a query *Q* against a dataset *D* is done in two steps: The body of *Q* is matched against *D* to obtain a set of bindings for the variables in the body, and then using the information on the head of *Q*, these bindings are processed applying classical relational operators (projection, distinct, etc.) to produce the answer to the query, which can have different forms, such as a yes/no answer, a table of values, or a new RDF dataset.

A SPARQL graph pattern expression is defined recursively as follows.

(1) A tuple from $(I \cup L \cup V) \times (I \cup V) \times (I \cup L \cup V)$ is a graph pattern (a *triple pattern*).
(2) If *P*1 and *P*2 are graph patterns, then expressions (*P*1 AND *P*2), (*P*1 OPT *P*2), and (*P*1 UNION *P*2) are graph patterns (*conjunction graph pattern*, *optional graph pattern*, and *union graph pattern*, respectively).
(3) If *P* is a graph pattern and *R* is a SPARQL *built-in* condition, then the expression (*P* FILTER *R*) is a graph pattern (a *filter graph pattern*).

The built-in condition in SPARQL is a Boolean combination of terms constructed by using = and bound, that is:
(1) If ?*X*, ?*Y* ∈ *V* and *c* ∈ *I* ∪ *L*, then bound(?*X*), ?*X* = *c* and ?*X* =?*Y* are built-in conditions.
(2) If *R*1 and *R*2 are built-in conditions, then (¬*R*1), (*R*1 ∨ *R*2) and (*R*1 ∧ *R*2) are built-in conditions.

To define the semantics of SPARQL graph pattern expressions, we need to introduce some terminology. A *mapping* μ from *V* to *U* is a partial function $\mu : V \rightarrow U$. Abusing notation, for a triple pattern *t* we denote by $\mu(t)$ the triple obtained by replacing the variables in *t* according to μ. The domain of μ, denoted by dom(μ), is the subset of *V* where μ is defined. Two mappings μ1 and μ2 are *compatible* when for all ?*X* ∈ dom(μ1) ∩ dom(μ2), it is the case that μ1(?*X*) = μ2(?*X*), that is, when μ1 ∪ μ2 is also a mapping. Intuitively, μ1 and μ2 are compatibles if μ1 *can be extended* with μ2 to obtain a new mapping, and vice versa. Note that two mappings with disjoint domains are always compatible, and that the empty mapping $\mu\emptyset$ (i.e., the mapping with empty domain) is compatible with any other mapping.

At this point , we shall refer to the differences between the definitions we give at this section and at the Chapter 2. The rules above are concerning mappings, but the rules of SPARQL in Chapter 2 (related work) present definitions about graph patterns solutions (which also include the projection statement in SPARQL.)

As far as the evaluation of SPARQL graph patterns, is concerned ,in this section there exists the definition of the semantics of graph pattern expressions as a function[[·]]*D* which takes a pattern expression and returns a set of mappings. We follow the approach in [Gutierrez et al. 2004] defining the semantics as the set of mappings that matches the dataset *D*.

***Definition* 3.2**. The *evaluation* of a graph pattern *P* over an RDF dataset *D*, denoted by [[*P*]]*D*, is defined recursively as follows.

(1) If *P* is a triple pattern *t*, then [[*P*]]*D* = {μ | dom(μ) = var(*t*) and μ(*t*) ∈ *D*}.

(2) If *P* is (*P*1 AND *P*2), then [[*P*]]*D* = [[*P*1]]*D* ⋈ [[*P*2]]*D*.

(3) If *P* is (*P*1 OPT *P*2), then [[*P*]]*D* = [[*P*1]]*D* ⟕ [[*P*2]]*D*.

(4) If *P* is (*P*1 UNION *P*2), then [[*P*]]*D* = [[*P*1]]*D* ∪ [[*P*2]]*D*.

The idea behind the OPT operator is to allow for *optional matching* of patterns.

Consider pattern expression (*P*1 OPT *P*2) and let μ1 be a mapping in [[*P*1]]*D*. If there exists a mapping μ2 ∈ [[*P*2]]*D* such that μ1 and μ2 are compatible, then μ1 ∪μ2 belongs to [[(*P*1 OPT *P*2)]]*D*. But if no such a mapping μ2 exists, then μ1 belongs to [[(*P*1 OPT *P*2)]]*D*. Thus, operator OPT allows information to be added to a mapping μ if the information is available, instead of just rejecting μ whenever some part of the pattern does not match. This feature of *optional matching* is crucial in semantic Web applications, and more specifically in RDF data management, where it is assumed that every application has only partial knowledge about the resources being managed.

The semantics of filter expressions goes as follows. Given a mapping μ and a built-in condition *R*, we say that μ satisfies *R*, denoted by μ |= *R*, if:

(1) *R* is bound(?*X*) and ?*X* ∈ dom(μ);

(2) *R* is ?*X* = *c*, ?*X* ∈ dom(μ) and μ(?*X*) = *c*;

(3) *R* is ?*X* =?*Y* , ?*X* ∈ dom(μ), ?*Y* ∈ dom(μ) and μ(?*X*) = μ(?*Y*);

(4) *R* is (¬*R*1), *R*1 is a built-in condition, and it is not the case that μ |= *R*1;

(5) *R* is (*R*1 ∨ *R*2), *R*1 and *R*2 are built-in conditions, and μ |= *R*1 or μ |= *R*2;

(6) *R* is (*R*1 ∧ *R*2), *R*1 and *R*2 are built-in conditions, μ |= *R*1 and μ |= *R*2.

***Definition* 3.3**. Given an RDF dataset *D* and a filter expression (*P* FILTER *R*),

[[(*P* FILTER *R*)]]*D* = {μ ∈ [[*P*]]*D* | μ |= *R*}.

Example 3.5

Consider an RDF dataset *D* storing information about our motivating example for book stores and libraries (see figure 3.7). This RDF dataset will have information about books (book (M.O), textbook(IS1), books(IS2), equivalence between these classes). We suppose that this example is on one RDF dataset.

D={ (B1 ,price, 15) (B1, author, A.Turing)

(B2, price, 20) (B2, publisher, Pearson)

(B3, price, 30) (B3, size, 14)

(B4, price, 35) (B4, publisher, Pearson)

(B4, size, 15) (B4, author, Al.turing) }

The following are graph pattern expressions and their evaluations over D according to the previous semantics.

- **P1= ((?A, publisher, ?E) OPT (?A, size, ?W)).** Then

[[P1]]D=

	?A	?E	?W
μ1	**B2**	**Pearson**	
μ2	**B4**	**Pearson**	**15**

- ***P*2 = (((?*A*, price, ?*N*) OPT (?*A*,publisher, ?*E*)) OPT (?*A*, size, ?*W*)).** Then

[[P2]]D=

	?A	?N	?E	?W
μ1	**B1**	**15**		
μ2	**B2**	**20**	**Pearson**	
μ3	**B3**	**30**		**14**
μ4	**B4**	**35**	**Pearson**	**15**

- ***P*3 = ((?*A*, price, ?*N*) OPT ((?*A*, publisher, ?*E*) OPT (?*A*, size, ?*W*))).** Then

[[P3]]D=

	?A	?N	?E	?W
μ1	**B1**	**15**		
μ2	**B2**	**20**	**Pearson**	
μ3	**B3**	**30**		
μ4	**B4**	**35**	**Pearson**	**15**

Notice the difference between [[*P*2]]*D* and [[*P*3]]*D*. These two examples show that [[((*A* OPT *B*) OPT *C*)]]*D* ≠ [[(*A* OPT (*B* OPT *C*))]]*D* in general.

- ***P*4 = ((?*A*,price, ?*N*) AND ((?*A*, publisher, ?*E*) UNION (?*A*, size, ?*W*))).** Then

?A	?N	?E	?W

	μ1	**B2**	**20**	**Pearson**	
[[P4]]D=	**μ2**	**B3**	**30**		**14**
	μ3	**B4**	**35**	**Pearson**	
	μ4	**B4**	**35**		**15**

- ***P*5 = (((?*A*, price, ?*N*) OPT (?*A*, author, ?*P*)) FILTER ?*N* = 15).** Then

[[P5]]D=		**?A**	**?N**	**?P**
	μ1	**B1**	**15**	**A.Turing**

As is customary when studying the complexity of the evaluation problem for a query language we consider its associated decision problem. We denote this problem by Evaluation and we define it as follows.

INPUT : An RDF dataset *D*, a graph pattern *P* and a mapping μ.

QUESTION : Is $\mu \in [[P]]D$?

The algorithm of evaluation for simple SPARQL queries is presented in *Appendix.*

For the better understanding of the Normalization Process, we present some simple Algebraic Properties.

We say that two graph patterns *P*1 and *P*2 are *equivalent*, denoted by *P*1 ≡ *P*2, if [[*P*1]]*D* = [[*P*2]]*D* for every RDF dataset *D*. The following lemma states some simple algebraic properties of AND and UNION operators. These properties are direct consequence of the semantics of AND and UNION, both based on set-theoretical union.

Definition 3.4 :*The operators* AND *and* UNION *are associative and commutative and the operator* AND *distributes over* UNION*. That is, if P*1*, P*2*, and P*3 *are graph patterns, then it holds that:*

- (*P*1 AND *P*2) ≡ (*P*2 AND *P*1);
- (*P*1 UNION *P*2) ≡ (*P*2 UNION *P*1);
- (*P*1 AND (*P*2 AND *P*3)) ≡ ((*P*1 AND *P*2) AND *P*3);
- (*P*1 UNION (*P*2 UNION *P*3)) ≡ ((*P*1 UNION *P*2) UNION *P*3);
- (*P*1 AND (*P*2 UNION *P*3)) ≡ ((*P*1 AND *P*2) UNION (*P*1 AND *P*3)).

The previous definition permits us to avoid parentheses when writing sequences of either AND operators or UNION operators. We use Definition 3.4 to simplify the notation in the SPARQL queries.

For the optimized evaluation of the given graph patterns, the definition of " union free well designed graph patterns" is introduced. One of the most delicate issues in the definition of a semantics for graph pattern expressions is the semantics of the OPT operator. The idea behind the OPT operator is to allow for *optional matching* of patterns, that is, to allow information to be added if it is available, instead of rejecting whenever some part of a pattern does not match.

Example 3.6

Consider the following graph patterns- query:

```
P=((?X, author, Al. turing)OPT((?Y, author, Perez)OPT(?X,
publisher, ?Z)))
```

What is unnatural about P is the fact that (?X, email, ?Z) is giving some optional information for (?X ,author, A.turing), but in P appears as giving optional information for (?Y, author, Perez). More precisely, (B2, publisher, Pearson) are triples in the dataset D of **Example 3.5** but the evaluation of P results in the set {{?X -> B2}} without giving information of the publisher of the books whose author is Al.Turing.

A careful examination of the examples that produce conflicts reveals a common pattern: A graph pattern *P* mentions an expression *P'* = (*P*1 OPT *P*2) and a variable ?*X* occurring both inside *P*2 and outside *P'* but not occurring in *P*1.
In general, graph pattern expressions satisfying this condition are not natural. To present the main definition of this section, we need to introduce some terminology.

We say that a graph pattern *Q* is ***safe*** if for every subpattern (*P* FILTER *R*) of *Q*, it holds that var(*R*) ⊆ var(*P*). This safety condition is present in many database query languages.

A UNION-free graph pattern *P* is *well designed* if *P* is safe and, for every subpattern *P'* = (*P*1 OPT *P*2) of *P* and for every variable ?*X* occurring in *P*, the following condition holds.

if ?*X* occurs both inside *P*2 and outside *P'*, then it also occurs in *P*1.

For instance, pattern P in Example 3.2.3.1 given earlier is not well designed, while all the UNION-free patterns in Example 3.2.2.1 are well designed.
We say that a pattern of the form (*P*1 UNION *P*2 UNION ・ ・ ・ UNION *Pn*) is *well designed* if every *Pi* (1 ≤ *i* ≤ *n*) is a UNION-free well-designed graph pattern.

Also well-designed patterns are suitable for reordering and optimization , demonstrating the significance of this class of queries from the practical point of view.

Now, for the complexity of evaluation and optimization of Well Designed Patterns intuitively, if we *delete* some optional parts of a pattern *P* to obtain a new pattern *P'*, the mappings in the evaluation of *P'* over a dataset *D* could not be more informative than the mappings in the evaluation of *P* over *D*. In other words, the optional matchings of a pattern must only serve to *extend* solutions with new information, but not to reject solutions if some information is not provided.
The formalization of this intuition shown below is used to develop a characterization of the evaluation of well-designed graph patterns.

We say that a mapping μ is *subsumed* by a mapping μ', denoted by $\mu \sqsubseteq \mu'$, if μ and μ' are compatible and dom(μ) ⊆ dom(μ'), that is, μ is subsumed by μ' if μ agrees with μ' in every variable for which μ is defined. For sets of mappings Ω and Ω', we write $\Omega \sqsubseteq \Omega'$ if for every mapping $\mu \in \Omega$, there exists a mapping $\mu' \in \Omega'$ such that $\mu \sqsubseteq \mu'$.

.

We say that a pattern *P'* is a *reduction* of a pattern *P*, if *P'* can be obtained from *P* by replacing a subformula (*P*1 OPT *P*2) of *P* by *P*1, that is, if *P'* is obtained by deleting some optional part of *P*. For example, *P* = (*t*1 AND (*t*2 OPT (*t*3 AND *t*4))) is a reduction of *P* = ((*t*1 OPT *t*2) AND (*t*2 OPT (*t*3 AND *t*4))) since *P'* can be obtained from *P* by replacing (*t*1 OPT *t*2) by *t*1. The reflexive and transitive closure of the reduction relation is denoted by $\trianglelefteq$. Thus, for example, if *P''* = (*t*1 AND *t*2), then *P''* $\trianglelefteq$ *P* since *P''* is a reduction of *P'* and *P'* is a reduction of *P*. We note that if *P'* $\trianglelefteq$ *P* and *P* is well designed, then *P'* is well designed. If *P'* $\trianglelefteq$ *P* and *P*≠*P*, then we say that *P'* is a proper reduction of *P*, denoted by *P'* $\triangleleft$ *P*.

Let P be a UNION*-free well-designed graph pattern, and P' a pattern such that P'* $\trianglelefteq$ *P. Then* [[*P'*]]*D* $\sqsubseteq$ [[*P*]]*D for every dataset D.*

The above does not hold for patterns that are not well designed.

Example 3.7

Consider dataset *D* = {(1, a, 1), (2, a, 2), (3, a, 3)} and nonwell-designed pattern

P = ((?*X*, a, 1) OPT ((?*Y*, a, 2) OPT (?*X*, a, 3))).

The evaluation of *P* results in the set {{?*X* → 1}}. By deleting the optional part (?*X*, a, 3) of *P*, we obtain the reduction *P'* = ((?*X*, a, 1) OPT (?*Y*, a, 2)) of *P*. The evaluation of *P'* results in the set {{?*X* → 1, ?*Y* → 2}}. Thus, we have that [[*P*]]*D* $\not\sqsubseteq$ [[*P*]]*D*.

We have mentioned that, when evaluating an optional part of a pattern, one is trying to extend mappings with optional information. Another intuition behind the OPT operator is that, when a pattern has several optional parts, one wants to extend the solutions *as much as possible*, that is, one does not want to lose information when the information is present. We formalize this intuition with the notion of *partial solution* for a pattern. Informally, a partial solution for a pattern *P* is a mapping that is an *exact match* for some *P'* such that *P'* $\trianglelefteq$ *P*. We show then, that the evaluation of a welldesigned graph pattern *P* is exactly the set of *maximal partial solutions* for *P* with respect to $\sqsubseteq$,that is, the solutions that retrieve as much information as possible. This proposition gives us an alternative characterization of the evaluation of well-designed graph patterns.

Given a pattern *P*, define $\text{and}(P)$ to be the pattern obtained from *P* by replacing every OPT operator in *P* by an AND operator. For example,if *P* = ((*t*1 OPT *t*2) AND (*t*2 OPT (*t*3 AND *t*4))), we have that $\text{and}(P)$ =((*t*1 AND *t*2) AND (*t*2 AND (*t*3 AND *t*4))). Notice that, by the semantics of the OPT operator, for every (not necessarily well designed) pattern *P* and every dataset *D*, we have that [[and(*P*)]]*D* ⊆ [[*P*]]*D*.

A mapping μ is a *partial solution* for a pattern *P* over a dataset *D* if $\mu \in$ [[and(*P*)]]*D*, for some *P'* $\trianglelefteq$ *P*. Partial solutions and the notion of subsumption of mappings give us the following characterization of the evaluation of welldesigned graph patterns.

Given a UNION*-free well-designed graph pattern P, a dataset D, and a mapping* μ*, we have that* $\mu \in$ [[*P*]]*D if and only if* μ *is a maximal (with respect to* $\sqsubseteq$*) partial solution for P over D.*

The property of being well designed has important consequences for the study of normalization and optimization for SPARQL.

*Let P*1*, P*2*, and P*3 *be graph pattern expressions and R a built-in condition. Consider the rewriting rules.*

((*P*1 OPT *P*2) FILTER *R*) $\dashrightarrow$ ((*P*1 FILTER *R*) OPT *P*2)

(*P*1 AND (*P*2 OPT *P*3)) $\dashrightarrow$ ((*P*1 AND *P*2) OPT *P*3)

((*P*1 OPT *P*2) AND *P*3) $\dashrightarrow$ ((*P*1 AND *P*3) OPT *P*2)

Let P be a UNION*-free well-designed pattern, and assume that P is a pattern obtained from P by applying the above rules. Then P is a* UNION*-free well-designed pattern equivalent to P.*

It is worth mentioning that the previous rules are not applicable to nonwell-designed graph patterns.

Example 3.8

For example, consider dataset *D* = {(1, a, 1), (2, a, 2), (3, a, 3)} and nonwell-designed pattern *P* = ((?*X*, a, 1) AND ((?*Y*, a, 2) OPT (?*X*, a, 3))).

The evaluation of *P* results in the empty set of mappings. If we apply rule (8) to *P*, we obtain pattern *P'* = (((?*X*, a, 1) AND (?*Y*, a, 2)) OPT (?*X*, a, 3)). The evaluation of *P'* results in the set {{?*X* → 1, ?*Y* → 2}} and, thus, we have that [[*P*]]*D* ≠ [[*P'*]]*D*.

We say that a UNION-free graph pattern *P* is in OPT *normal form* if either: (1) *P* is constructed by using only the AND and FILTER operators, or (2) *P* = (*O*1 OPT *O*2), with *O*1 and *O*2 patterns in OPT normal form.

Example 3.9

For example, consider a pattern:

P = ((((*t*1 AND *t*2) FILTER *R*1) OPT (*t*3 OPT ((*t*4 FILTER *R*2) AND *t*5))) OPT (*t*6 FILTER *R*3)),

where every *ti* is a triple pattern, and every *Rj* is a built-in condition. Then *P* is in OPT normal form. The following theorem shows that for every well-designed graph pattern, an equivalent pattern in OPT normal form can be efficiently obtained.

For every UNION*-free well-designed pattern P, an equivalent pattern in* OPT *normal form can be obtained after O*(|*P*|2) *applications of the rewriting rules.*

The application of the rewriting rules may have a considerable impact in the cost of evaluating graph patterns. One can measure this impact by analyzing the intermediate sizes of the sets of mappings produced when evaluating a pattern. By the semantics of the OPT operator, when evaluating an expression of the form (*P*1 OPT *P*2) over a dataset *D*, the number of mappings obtained is at least the number of mappings obtained when evaluating *P*1 over *D*. In other words, the application of the OPT operator never implies a reduction in the size of the intermediate results in the evaluation of a graph pattern expression. In contrast, it is clear that operators AND and FILTER may imply a reduction in the size of intermediate results. Thus, for optimization purposes, it would be convenient to perform all the AND and FILTER operations first, delaying the OPT operations to the last step of the evaluation. A pattern in OPT normal form has its operators ordered in a way that the bottom-up evaluation of the pattern follows exactly this strategy: All AND and FILTER operations are executed prior to the execution of the OPT operations.

To sum up the rules we use for normalization are in the context of this diploma thesis are:

Equivalence Rules

1. **Commutative**

(*P*1 AND *P*2) ≡ (*P*2 AND *P*1)*;*

(*P*1 UNION *P*2) ≡ (*P*2 UNION *P*1)*;*

Associative

(*P*1 AND (*P*2 AND *P*3)) ≡ ((*P*1 AND *P*2) AND *P*3)*;*

(*P*1 UNION (*P*2 UNION *P*3)) ≡ ((*P*1 UNION *P*2) UNION *P*3)*;*

For OPT (left associative)

P1 OPT P2 OPT P3 ≡ (P1 OPT P2)OPT P3

2. Delete all the empty graph Patterns PØ (Empty Graph Patterns)

3. (*P*1 AND (*P*2 UNION *P*3)) ≡ ((*P*1 AND *P*2) UNION (*P*1 AND *P*3))*.*

4. (P1 OPT (P2 UNION P3)) ≡ ((P1 OPT P2) UNION (P1 OPT P3))

5. ((P1 UNION P2)OPT P3) ≡ ((P1 OPT P3) UNION (P2 OPT P3))

6. 7.6 ((P1 UNION P2)FILTER R) ≡ ((P1 FILTER R) UNION (P2 FILTER R))

7. Remove unnecessary brackets {}

8. (P1 AND (P2 OPT P3)) ≡ ((P1 AND P2) OPT P3)

9. ((P1 OPT P2) OPT P3) ≡ ((P1 OPT P3) OPT P2).

10. ((P1 OPT P2) OPT P3) ≡ ((P1 OPT P3) OPT P2).

11. ((*P*1 OPT *P*2) FILTER *R*) –→ ((*P*1 FILTER *R*) OPT *P*2)

With retroactive application of rules of equivalence for "well designed" graph Patterns, the normalized following grammar for the "well designed" Patterns graph results.

Normalized Grammar

a) GP := BGP | Optional_Pattern | Union_Pattern
b) BGP := "(" ((tp "AND" tp)* Filter* (tp "AND" tp)*)* ")"
c) Optional_Pattern := BGP ("OPT" "(" Optional_Pattern Filter* ")")+
d) Union_free_Pattern := BGP | Optional_Pattern
e) Union_Pattern := "(" Union_free_Pattern ")" ("UNION" "(" Union_free_Pattern) ")" *
f) tp := Triple Pattern
g) Filter := FILTER(R)
h) R := SPARQL expression

As it is clearly, we have achieved our original purpose, namely the creation of greatest possible basic graph pattern. It has also reached best result as AND operators appear only between triple Patterns, ie. just inside the main graph patterns.

Within the context of this thesis we will try to transform the original query in a query consisted of well designed graph patterns. Forward this goal will try to transform the query firstly to a Union-free form and afterwards if this is possible to a set of well designed graph patterns or even in a OPT normal form, which is an optimized form using OPT operator.

At the rest part of this section we provide some useful examples of the Normalization Process. Having in mind the **Figure 3.6** and the corresponding set of mappings and articulations shown in example 3.5 and table 3.5. we pose the following query for answering, to the system.

Example 3.10: Example of a query given by the user

```
Select   ?price
WHERE
{{       ?x      rdf: type       s: Science
         ?x      s: price        ?price }
UNION
{        ?x      rdf: type       s: Pocket
         ?x      s: price        ?price}}
```

the first step towards the transformation of the original query, is to remodel it to a UNION free normal form (P1 UNION P2 UNION P3 UNION…. UNION Pn, where each Pi(1≤i≤n)is UNION-free).

```
Select   ?price
WHERE
{{       ?x      rdf: type       s: Science
         ?x      s: price        ?price
OPT {    ?x      s: price        ?price>15.00}
  }
UNION
{        ?x      rdf: type       s: Pocket
         ?x      s: price        ?price
OPT {    ?x      s: price        ?price>15.00}};
```

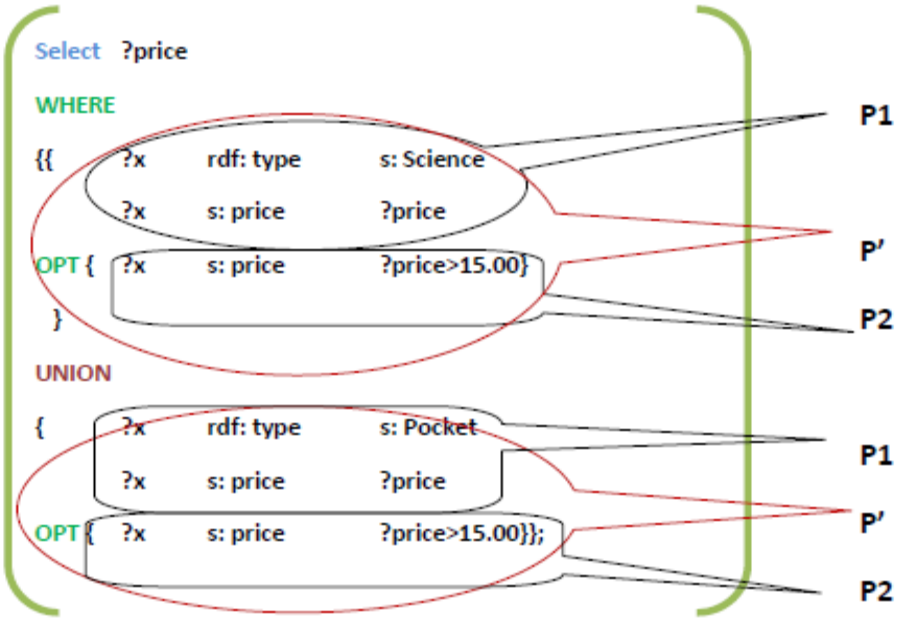
Since the optional OPT part has to be considered for both { **?x rdf: type s: Science}** and **{ ?x rdf: type s: Pocket}** the UNION free query will be:

UNION free query

Example 3.11: Example of transformation to a UNION free query given.

As we have already mentioned:A UNION-free graph pattern P is **well designed** if P is safe and, for every subpattern P' = (P1 OPT P2) of P and for every variable ?X occurring in P, the following condition holds: if ?X occurs both inside P2 and outside P', then it also occurs in P1.

In this case , we may notice that the query is already well designed.



UNION free well designed Graph pattern (?x ∈ P1,P2,P')

Example 3.12: Example of UNION free well designed query.

As far as the optimization of the graph patterns is concerned we will try to deal with this problem based on the simple algebraic rules we presented above.

Well Designed Query

```
Select   ?price        , ?author
WHERE {
{        ?x   rdf: type    s:Science.
         ?s   s: price     ?price.
OPT {    ?x   s: price     ?price >15.00}
          }
AND{     ?s   s:author     ?author}}
UNION
         {
         ?x   rdf:type     s:Pocket
         ?x   s:price      ?price}
OPT {    ?x   s:price      ?price >15.00}}
AND{     ?s   s:author     ?author}
```

Example 3.13: Example of well designed query.

Using the rule ((P1 OPT P2)AND P3)-> ((P1 AND P3) OPT P2), we obtain a new optimized Union free well designed equivalent to the original one set of graph patterns

Well Designed Optimized Query

```
Select          ?price,?author
WHERE {
{               ?x  rdf:type  s:Science.
                ?s  s:price    ?price.
AND{             ?s  s:author  ?author}}
OPT {           ?x   s:price    ?price>15.00}
                                    }
UNION
{
                ?x   rdf:type  s:Pocket
                ?x   s:price    ?price
AND{            ?s   s:author  ?author}}
OPT {           ?x   s:price    ?price >15.00}
};
```

Example 3.14: Example of well designed optimized query.

Finally, we give a last example of a possible query transformed before the evaluation to an OPT Normal Form.

```
Select          ?price,?author
WHERE {
{       ?x      rdf:type        s:Science.
        ?s      s:price         ?price.
AND{?s          s:author        ?author}}
OPT {?x         s:price         ?price>15.00}
                }
UNION           {
        ?x      rdf:type        s:Pocket
        ?x      s:price         ?price
FILTER          {?price >20.00}}
OPT {?x         s:price         ?price >15.00}};
```

Example 3.15: Example of query in OPT Normal Form.

3.6 Query Relaxation

After the initialization of the system, the Mediator will have the possibility to operate under specific circumstances and restrictions for the query to be answered. More specifically we introduce four (4) functioning levels, in order to simplify the usage of the system and to allow the users to acquire the results of their preferences.

There are many cases that a query is not in formal (union free, well designed etc) form, or there is no appropriate correspondence between the terms of the main ontology and the terms of the local ones (information sources). The malfunctioning cases we are trying to solve- or in some occasions to avoid – are better handled by the introduced Strictness *Levels*. Cases of malfunctioning are the following:

- The given query (by the user) is not in optimum form. It is supposed that for the system to work, for the reformulation of the query so it can be answered by the underlying information sources and for the extraction of the most suitable results, the query has to be in a *well designed union free form*. However, in some cases, this is extremely difficult(i.e. not well designed query posed by the user,not meaningfull).

  For example, the following graph pattern

  ```
  P=((?X, author, Al. turing)OPT((?Y, author, Perez)OPT(?X,
  publisher, ?Z))),
  ```

  Asks for extra information about another variable and not the one given in the basic GP. This has no particular meaning and makes for the system more difficult to evaluate it.

- There is no or little correspondence of the terms of the Main Ontology and the terms of the Local Ontologies. That means that the system has to extract all the possible mappings of the terms given and to use them in the best possible way. Because the user does not “knows” what are the mappings to each term of his query, the original query may not be valid or may not return results. In such cases that the posed query has some terms that are not mapped with the terms of the information sources, then there is significant difficulty in answering the query. If the system tries to answer the query as it is, then the results will be vague, incomplete or undefined.

  For example, the following Pattern

```
P = (((?A, price, ?N) OPT (?A,publisher, ?E)) OPT (?A, size, ?W))
```

May only have mappings for the term"price" , then the evaluation and the extraction of proper results is becoming more difficult.

- Based on which and what kind of mappings there exists for each source, the results that will be extracted will have important differences and discontinuities. So every query has to be treated differently for each source based on the mappings that are available.

For the above reasons the four Levels of Strictness are introduced, based on the relaxation that can be used on the SPARQL queries. The user after posing his query will be able to choose on what level the system will operate. The functions and the processes following will be bound on the chosen level

Strictness Levels

Suppose a global and one local/target ontology. The user gives his original query as it is shown below but the term ?id has no mappings. Moreover the term ?name is required variable.

Original Query	**Extended SPARQL Expression**

PREFIX src: <http://www.owl-ontologies.com/SourceOntology.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT **?x ?y ?name ?id**
WHERE{
?x rdf:type src:Product.
OPTIONAL{{?x src:id ?id.
?x src:name ?name.}
UNION { ?x src:id ?id. }}
}
	PREFIX src: <http://www.owl-ontologies.com/SourceOntology.owl#>PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	SELECT optional:?x optional:?y required:?name optional:?id
	WHERE{
	?x rdf:type src:Product.
	OPTIONAL{{?x src:id ?id.}
	?x src:name ?name.}
	UNION { ?x src:id ?id. }}
	}

Level 1- -No relaxation

In this case the user requests no relaxation. The query to be answered must have full correspondence with each information source. If one ore more variables (independently of required and optional variable) have no mappings with terms of the information source, the system puts at the result exit the warning "No Answer-Incomplete mappings for the Source ISk".

Normalized Query	**Relaxed Query**
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX src: <http://www.owl-ontologies.com/SourceOntology.owl#> SELECT ?x ?y ?name ?id WHERE { ?x rdf:type src:Product . OPTIONAL { { ?x src:id ?id ; src:name ?name . } UNION { ?x src:id ?id .} } }	***No Answer***

Level 2- -Unsupported optional parts in the SPARQL Query

2.1 exclude optional parts (completely)

2.2 exclude optional parts (unsupported UFGP)

At this level only optional variables existing in optional parts after the normalization, can be eliminated. There are two sub cases at this level: exclude unsupported union free graph patterns or exclude the OPTIONAL part completely. Before the relaxation, the system checks the normalized query, if the no mapped term exists only in a UFGP inside an optional part then only the specific UFGP is eliminated, otherwise all the OPTIONAL part is excluded. This may be desirable so that no extra information be eliminated.

- If a required variable has no mapping, even though it exists inside an OPTIONAL part of a query, the normalized query cannot be answered.

- If an optional variable with no mappings exists outside the OPTIONAL part of the query, the normalized query cannot be answered.

This level contains level 1.

Normalized Query	**Relaxed Query**
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX src: <http://www.owl-ontologies.com/SourceOntology.owl#> SELECT ?x ?y ?name ?id WHERE { ?x rdf:type src:Product . OPTIONAL { { ?x src:id ?id ; src:name ?name . } UNION { ?x src:id ?id .} } }	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX src: <http://www.owl-ontologies.com/SourceOntology.owl#> SELECT ?x ?y ?name ?id WHERE {?x rdf:type src:Product . OPTIONAL { { } UNION { }} }

Level 3- -Unsupported Union Free Graph Patterns in the SPARQL Query

3.1 exclude unsupported UFGP

At this level the system checks the normalized query per union free graph pattern, if an optional variable exists inside an optional graph pattern, then we consider the specific UFGP unsupported and we exclude it.

- If a required variable exists in a UFGP but has no mappings with terms of an information source, then the query cannot be answered.

This level contains/covers level 2 and level 1.

Normalized Query	**Relaxed Query**
PREFIXrdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX src: <http://www.owl-ontologies.com/SourceOntology.owl#> SELECT ?x ?y ?name ?id	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX src: <http://www.owl-ontologies.com/SourceOntology.owl#> SELECT ?x ?y ?name ?id

```
WHERE
 {  { ?x  src:id    ?id ;
        src:name  ?name .
      ?x  rdf:type  src:Product .
    }
   UNION
    { ?x  src:id  ?id .
      ?x  rdf:type  src:Product .
    }
 }
```

```
WHERE
 {  { }
   UNION
    { }}
```

Level 4 - -Unsupported triples

4.1 exclude unsupported triples

This level is the most relaxed. Presupposes that the user wants results, even though there will be no exact correspondence with results he wished for in his original query. At this level, we can remove triples that contain optional variables without mappings, wherever these variables exist in the WHERE clause of the query.

This level contains level3, level 2 and level 1.

Normalized Query

```
PREFIX  rdf:
<http://www.w3.org/1999/02/22-rdf-
syntax-ns#>
PREFIX  src:  <http://www.owl-
ontologies.com/SourceOntology.owl#
>
SELECT  ?x ?y ?name ?id
WHERE
 { ?x  rdf:type  src:Product .
   OPTIONAL
    {  { ?x  src:id    ?id ;
         src:name  ?name .
       }
      UNION
       { ?x  src:id  ?id .}
    }
 }
```

Relaxed Query

```
PREFIX  rdf:
<http://www.w3.org/1999/02/22-rdf-
syntax-ns#>
PREFIX  src:  <http://www.owl-
ontologies.com/SourceOntology.owl#
>
SELECT  ?x ?y ?name ?id
WHERE
 {?x  rdf:type  src:Product .
   OPTIONAL
    {  {src:name  ?name . }
     UNION
      { }}
 }
```

As it is clearly stated in the Level Description above: Levels are progressive → Relaxation Increases

As far as the returned variables are concerned, there are two different cases:

RETURN VARIABLES

A. All return variables must be bound (required):

According to the defined strictness level, relaxation Is taking place so that at least one criterion is used for the binding of each return variable.

If this relaxation cannot take place, the query is not translated for the given source.

If a return variable appears only on an optional part, then it can be eliminated.

B. Not all return variables should be bound:

Apply the relaxation rule according to the predefined strictness level.

Note: At least one return variable must be bound.

C. Users defined return variables must be bound.

We treat FILTER cases as differently as follows:

FILTERS

Level 1: Must be translated

Level 2: Eliminate expressions that refer to variables of optional parts.

Level 3: The same as in level 2 but for variables referred in UFGPs that have been eliminated. Also, expressions with OR among them can be eliminated, although at least one must be present.

Level 4: Eliminate expressions that cannot be evaluated.

3.6.1 Extended SPARQL

In this section we demonstrate the expansion of the original SPARQL query based on the required variables given by the user. When a query is posed to the system, the user has the option to choose which variables he wishes to be answered for sure. Our model gives the user the option for optional variables posed to his query.

In such a way, variables that are no mandatory may have results but in case that there are no mappings compatible with the given optional terms, this way does not disincline the query from being answered.

Example 3.16 gives an impression of how this module works.

Query

```
PREFIX src: <http://www.owl-ontologies.com/SourceOntology.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?name ?id ?Collection
WHERE{
    ?x src:name ?name.
    ?x rdf:type src:Science.
    ?x rdf:type src:Pocket.
    ?x rdf:type src:Collection
}
```

Required Var: ?name **Optional Var**: ?id, ?collection

The system function depends on the choices of the user. If the user does not set a choice for one of the available variables in the SELECT clause of the query, the system set this variable as `optional`.

Let's now suppose that the term ?id is not mapped to a term of a local ontology. That is on user discretion to choose whether he wishes the query to be answered or not, based on the relaxation level he chooses(see next chapter). If however the term ?id was required the system would not answer the query.

The expanded form of the original query will be:

Expanded SPARQL Expression

```
PREFIX src: <http://www.owl-ontologies.com/SourceOntology.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT required: ?name relaxed: ?id relaxed: ?Collection
WHERE{
    ?x src:name ?name.
    ?x rdf:type src:Science.
    ?x rdf:type src:Pocket.
    ?x rdf:type src:Collection
}
```

3.7 Query Reformulation

In this section we provide an overview of the SPARQL query reformulation process, using a predefined set of mappings that follows the different mapping types described in [Makris etal. 2010].

The SPARQL query reformulation process is based on the query's *graph pattern* reformulation and is consequently independent of the query type (*Ask, Select, Construct, Describe*). The SPARQL solution modifiers (*Limit, Offset, Order By, Distinct, Reduce*) are not taken into consideration since they do not affect the reformulation process. In order to reformulate the *graph pattern* of a SPARQL query using *1:N* mappings (mappings between a basic OWL concept of the source ontology and a complex expression, among basic OWL concepts, of the target ontology), the reformulation algorithm traverses the execution tree in a bottom up approach, taking each triple pattern of the *graph pattern* and checking for existing mappings of the subject, predicate and object part in the predefined mappings set. Finally, it reformulates the triple pattern according to those mappings.

In case of *N:M* mapping utilization (mappings between a complex expression of the source ontology and a complex expression of the target ontology), the total graph pattern must be parsed in order to discover the predefined complex mapping and then, the algorithm must produce the required combination of triple patterns, based on this mapping. The SPARQL graph pattern operators (*AND, UNION* and *OPTIONAL*), do not result in modifications during the reformulation process.

The reformulation of the *FILTER* expressions is performed by reformulating the existing *IRIs* that refer to a class, property, or individual, according to the specified mappings. The SPARQL variables, literal constants, operators (*&&, ||, !, =, !=, >, <, >=, <=, +, -, *, /*) and built-in functions (e.g. *bound, isIRI, isLiteral, datatype, lang, str, regex*) that may occur in a *FILTER* expression remain the same during the reformulation process.

Finally, in case that more than one mappings are specified for a given class or property expression of the source ontology, the reformulation algorithm chooses the one that produces the most efficient reformulated query. Moreover, for efficiency reasons, a graph pattern normalization step is applied in parallel to the reformulation process, similarly with the one that is described in our SPARQL2XQuery framework.

We briefly present in this section the SPARQL reformulation process, using a set of examples. We assume that an initial SPARQL query is posed over the source ontology presented in Fig. (motivating example) and is reformulated to a semantically equivalent query in order to be posed over the target ontology IS1 of Fig., using the mappings specified.

Example 3.17 : Consider the query posed over the source ontology: *"Return the titles of the pocket-sized scientific books"*. The SPARQL syntax of the source query and the reformulated query is shown in Fig.3. During the reformulation process the mappings

$\forall x, [Science(x) \Leftrightarrow (Physics(x) \vee Mathematics(x))]$,

$\forall x, [Pocket(x) \Leftrightarrow Textbook(x) \wedge \exists y; [size(x, y) \wedge y \leq 14])]$

and

$\forall x, \forall y\ [name(x, y) \Rightarrow title(x, y)]$ are used.

Source Query:

```
SELECT ?name
WHERE{
  ?x s:name ?name.
  ?x rdf:type s:Science.
  ?x rdf:type s:Pocket.
}
```

Reformulated - Target Query:

```
SELECT ?name
WHERE{
  {?x t:title ?name.}
  {{?x rdf:type t:Physics.}
   UNION
   {?x rdf:type t:Mathematics.}}
  {?x rdf:type t:Textbook.
   ?x t:size ?size.
   FILTER (?size ≤ 14)}
}
```

Fig. 3**.8** The source and reformulated query

Example 3.18 : Consider the query posed over the source ontology: *"Return the titles of books that belong to the poetry or novel category"*. The SPARQL syntax of the source query and the reformulated query is shown in Fig.3.9. During the reformulation process the mappings

$\forall x, [(Novel(x) \vee Poetry(x) \Leftrightarrow Literature(x))]$

And

$\forall x, \forall y\ [name(x, y) \Rightarrow title(x, y)]$ are used.

Source Query:

```
SELECT ?name
WHERE{
  {?x s:name ?name.}
  {{?x rdf:type s:Poetry.}
   UNION
   {?x rdf:type s:Novel.}}
}
```

Reformulated - Target Query:

```
SELECT ?name
WHERE{
  ?x t:title ?name.
  ?x rdf:type t:Literature.
}
```

Fig.3.9**.** The source and reformulated query

3.8 Examples of Query Processing through the Mediator

At this section we provide some examples for all the query processing: Original query, expanded SPARQL Expression, Normalized Query and Relaxed Query.

- Examples

- NO mappings: ?id, Required: ?name , Level 4

Original Query	**Expanded SPARQL Expression**

PREFIX src: <http://www.owl-ontologies.com/SourceOntology.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?id ?name
WHERE{
?x rdf:type src:Product.
?x src:id ?id.
?x src:name ?name.
}
	PREFIX src: <http://www.owl-ontologies.com/SourceOntology.owl#>
	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	SELECT relaxed:?id required:?name
	WHERE{
	?x rdf:type src:Product.
	?x src:id ?id.
	?x src:name ?name.
	}

Normalized Query	**Relaxed Query**
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>	PREFIX src: <http://www.owl-ontologies.com/SourceOntology.owl#> PREFIX rdf:

```
PREFIX  src:  <http://www.owl-
ontologies.com/SourceOntology.owl#>
SELECT  ?id ?name
WHERE
  { ?x  rdf:type  src:Product ;
        src:id    ?id ;
        src:name  ?name .
  }
```

```
<http://www.w3.org/1999/02/22-rdf-
syntax-ns#>
SELECT relaxed:?id required:?name
WHERE{
    ?x rdf:type src:Product.
    ?x src:id ?id.
    ?x src:name ?name.
}
```

- NO mappings: ?id, Required: ?name ?id , Level 2

Original Query

```
PREFIX src: <http://www.owl-
ontologies.com/SourceOntology.owl#>
PREFIX rdf:
<http://www.w3.org/1999/02/22-rdf-
syntax-ns#>
SELECT ?x  ?y ?name ?id
WHERE{
    ?x rdf:type src:Product.
     OPTIONAL{{?x src:id ?id
                              ?x
src:name ?name.}
     UNION {    ?x src:id ?id. }}
}
}
```

Expanded SPARQL Expression

```
PREFIX src: <http://www.owl-
ontologies.com/SourceOntology.owl#>
PREFIX rdf:
<http://www.w3.org/1999/02/22-rdf-
syntax-ns#>
SELECT relaxed:?x  relaxed:?y
required:?name required:?id
WHERE{
    ?x rdf:type src:Product.
     OPTIONAL{{?x src:id ?id.}
                                  ?x
src:name ?name.}
     UNION {    ?x src:id ?id. }}
}
```

Normalized Query

```
PREFIX  rdf:
<http://www.w3.org/1999/02/22-rdf-
syntax-ns#>
PREFIX  src:  <http://www.owl-
ontologies.com/SourceOntology.owl#>
SELECT  ?x ?y ?name ?id
WHERE
  { ?x  rdf:type  src:Product .
    OPTIONAL
      {   { ?x  src:id    ?id ;
            src:name  ?name .
          }
        UNION
          { ?x  src:id  ?id .}
      }
  }
```

Relaxed Query

```
PREFIX  rdf:
<http://www.w3.org/1999/02/22-rdf-
syntax-ns#>
PREFIX  src:  <http://www.owl-
ontologies.com/SourceOntology.owl#>
SELECT  ?x ?y ?name ?id
WHERE
  {?x  rdf:type  src:Product .
    OPTIONAL
      {   { }
        UNION
          { }}
  }
```

- NO mappings: ?id, Required: ?name ?id , Level 3

Original Query

```
PREFIX src: <http://www.owl-
ontologies.com/SourceOntology.owl#>
PREFIX rdf:
<http://www.w3.org/1999/02/22-rdf-
syntax-ns#>
SELECT ?id ?name
WHERE{
    ?x rdf:type src:Product.
    {{?x rdf:type src:Product.
    ?x src:id ?id.}
    UNION
    {?var1 src:name ?name.}
    OPTIONAL{?x rdf:type
src:Product.}}
}
```

Expanded SPARQL Expression

```
PREFIX src: <http://www.owl-
ontologies.com/SourceOntology.owl#>
PREFIX rdf:
<http://www.w3.org/1999/02/22-rdf-
syntax-ns#>
SELECT required:?id required:?name
WHERE{
    ?x rdf:type src:Product.
    {{?x rdf:type src:Product.
    ?x src:id ?id.}
    UNION
    {?var1 src:name ?name.}
    OPTIONAL{?x rdf:type
src:Product.}}
}
```

Normalized Query

```
PREFIX  rdf:
<http://www.w3.org/1999/02/22-rdf-
syntax-ns#>
PREFIX  src:  <http://www.owl-
ontologies.com/SourceOntology.owl#>
SELECT  ?id ?name
WHERE
  {   { ?x  rdf:type  src:Product ;
          src:id    ?id .
        ?x  rdf:type  src:Product .
        OPTIONAL
          { ?x  rdf:type  src:Product
.}
      }
    UNION
      { ?var1  src:name  ?name .
        ?x  rdf:type  src:Product .
        OPTIONAL
          { ?x  rdf:type  src:Product
.}
      }
  }
```

Relaxed Query

```
PREFIX  rdf:
<http://www.w3.org/1999/02/22-rdf-
syntax-ns#>
PREFIX  src:  <http://www.owl-
ontologies.com/SourceOntology.owl#>
SELECT  ?id ?name
WHERE
  {   { }
    UNION
      { ?var1  src:name  ?name .
        ?x     rdf:type
src:Product .
        OPTIONAL
          { ?x  rdf:type
src:Product .}
      }
  }
```

3.9 Query Execution

The Query execution is done for each source individually after all the query processes described in the previous sections. The first time the user poses his query over the ontology of the Mediator, the articulations structure is created. The same structure is used throughout the system's function and during all the queries posed afterwards, unless there is a change in the stored data, ontologies or mappings.

The query execution is meant to extract the required results from the stored local ontologies, to form them and to translate them to terms of the main ontology , i.e to terms familiar to the user. The results are provided from each information source separately. This is subject of the number and type of answers each source can provide.

After the query execution the results are joined to a unique dataset with specific details and presented to the user.

3.9 Synthesis of Query Results

Since there are numerous combinations of triples and patterns, we should arrange the types of the results too. The different occasions of corresponding mappings to the original query and the different types of answers, is the subject of this chapter.

For now we consider one information source with correspondence to the main ontology of the mediator. However there are some cases, we take under consideration, for posing and answering a query through this information source.

1. Case 1: Full Mappings

In this case there is full correspondence of the local ontology and the main ontology of the mediator. For each query posed on the M.O., there is the required mapping to the terms of the L.O. So, when a query, consisting of triples is posed to the M.O, an exact reformulation can be implemented and posed to the L.O for answering. The type of the answer (exact, possible or incomplete- combined) will be specified by the types of mappings. There will be allowed four forms of answers:

1. **Exact answer:** This type of result set will include only specific results from each information source, which can offer a complete answer for the query given. This answer type is appropriate for users that focus on precision. In this type of answer the mappings that will be used, will have only *equivalence.* This takes place because in case the user asks for only the sure- exact results from any query, the only type that answers exactly his selection is the full correspondence of the terms in M.O and the terms in L.Os.
2. **Possible answer:** This type of answer to the initial query will include answers that *may* be bound on all the criteria of the query. This type of answer is appropriate for users that focus on recall. This type will be based on mappings-articulations that are bound on supersets, so the results may serve more answers to the user but not all of them will match exactly his/her criteria. In this type of answer the mappings that will be used, will have supersets (the terms of the local ontologies will have wider/bigger domain/range than the mapped terms of the main ontology of the mediator).
3. **Incomplete- combined answer:** This type of answer will give to the user results that are incomplete but they may help him through his queries. In this type, the relevant mappings and articulations will have subsets and equivalence (if it exists).
4. **Undefined type of answer:** This type of answer will give a set of results as answer, however based on the type of mappings of the required terms the system could not decide about the right form of the answer.

2. Case 2 : No mappings

In this case there is no correspondence of the local ontology and the main ontology of the mediator. For each query posed on the M.O., there is not the required mapping to the terms of the L.O. So, when a query, consisting of triples is posed to the M.O, there is no way to reformulate the query and to send it to the L.O. for answering. In this case the system will not give an answer to the user for the specific source or if all the sources are not capable of answering the query, the system will pose an error message, which will inform the user that there is no profound answer to his question and urge him to pose a different question.

3. Case 3: Incomplete Mappings

In this case there is not full correspondence of the local ontology and the main ontology of the mediator, however there are some terms that are mapped and some that are not. For each query posed on the M.O., there some specific mappings for some terms of the L.O. So, when a query, consisting of triples is posed to the M.O, we have to review the query and try to parse and reformulate it based on how many and what kinds of mappings exist.

This is where the expanded SPARQL Query and the Relaxation Levels participate. If an information source has incomplete mappings but has results to offer based on the required variables and the relaxation level the user has chosen ,it will still offer results to the user (maybe fewer or not so specific as it has been asked) . If however the user choices are not appropriate the information source will not provide an answer. At this point, we must mention that for the reformulation process of the query, full mappings for the required variables must exist.

In every case we consider the terms into Optional and FILTER parts of the query as more relaxed , as these parts are significant for the enrichment of the results and not for the main purpose of answering the query. The first thing that always is performed is the check for the asked variables in the SELECT part of the query, at a first place for the required variables and at a second place and if this is affordable for the optional variables of the SELECT clause.

The following example gives a preview of results extraction from a single information source.

Example 3.19

Consider the example in Figure 3.7. Let's suppose that the user poses the following query

```
PREFIX src:<http://www.owl-ontologies.com/SourceOntology.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?name ?id ?Collection ?cd
WHERE{
    ?x src:name ?name.
    ?x rdf:type src:Science.
    ?x rdf:type src:Pocket.
    ?x rdf:type src:Collection
```

```
    ?x rdf:type src:cd


}
```

And the mappings with the info source IS1 presented in table 3.4.

The term ?**cd** is not mapped with any term of the local ontology. So with a first glance we see that there is no full correspondence between /full mappings between the terms of the main ontology and the terms of the local one. If the user chooses as relaxation Level ” Level 1- Full Mappings”, no results can be extracted from the specific information source. If however the term ?cd is not a required variable from the user and “Level 4- Unsupported Triples” is chosen the answer of the specific info source will be as followed(with no results for the unmapped term *cd*):

---

name	id	Collection	cd
<http://www.owl-ontologies.com/TargetOntology.owl#mathematics003>		"Calculus: Early Transcendentals"^^<http://www.w3.org/2001/XMLSchema#string>	"978-0743270783"^^<http://www.w3.org/2001/XMLSchema#string>
<http://www.owl-ontologies.com/TargetOntology.owl#mathematics003>			"978-0743270783"^^<http://www.w3.org/2001/XMLSchema#string>
<http://www.owl-ontologies.com/TargetOntology.owl#biography002>		"The Science of Leonardo: Inside the Mind of the Great Genius of the Renaissance"^^<http://www.w3.org/2001/XMLSchema#string>	"978-0743270765"^^<http://www.w3.org/2001/XMLSchema#string>
<http://www.owl-ontologies.com/TargetOntology.owl#biography002>			"978-0743270765"^^<http://www.w3.org/2001/XMLSchema#string>
<http://www.owl-ontologies.com/TargetOntology.owl#poetry001>		"Letter to My Daughter"^^<http://www.w3.org/2001/XMLSchema#string>	"978-0743270780"^^<http://www.w3.org/2001/XMLSchema#string>
<http://www.owl-ontologies.com/TargetOntology.owl#poetry001>			"978-0743270780"^^<http://www.w3.org/2001/XMLSchema#string>

| <http://www.owl-ontologies.com/TargetOntology.owl#novel002> | | "The White Tiger"^^<http://www.w3.org/2001/XMLSchema#string>
| "978-0743270779"^^<http://www.w3.org/2001/XMLSchema#string> |

| <http://www.owl-ontologies.com/TargetOntology.owl#novel002> | |
| "978-0743270779"^^<http://www.w3.org/2001/XMLSchema#string> |

| <http://www.owl-ontologies.com/TargetOntology.owl#novel001> | | "The Tempest"^^<http://www.w3.org/2001/XMLSchema#string>
| "978-0743270778"^^<http://www.w3.org/2001/XMLSchema#string> |

Example 3.19: example of results presentation

The last thing our mediator does is the **ranking** of the final unified result set based on the real level of relaxation that the normalized query has been put through and the number of the asked variables that have results. The aspect we take under consideration for the final result ranking is *how close is the query that has been answered by the mediator to the original query posed by the user.* More specifically, the ranking we proposed is based on what has been relaxed or deleted. The second aspect is *which information source has answered the most,* i.e which source has better and more results compared to the other sources. The above criterion however has as condition that the sources have been manipulated by the same Level of Relaxation.

For example, if the user chooses relaxation Level “Level 4- Unsupported Triples”, the Information Source 1 has full mappings but the Information Source 2 has incomplete mappings (with the unmapped term in a UNION free graph pattern), than no ranking can be done for these two sources. For the source however that has full mappings with the main ontology, extra information is provided about the type of the answer provided (as it is described above i.e. exact, possible, incomplete or undefined).

However, the system joins the two result sets to a unified result set and gives extra information to the user about:

1. What was the “real” relaxation level that has been used for each source, and
2. How many of the required variables from the original query have results to offer.

An example of our method is presented below.

Example 3.20

Consider the following query posed to the system by the user.

```
PREFIX src: <http://www.owl-ontologies.com/SourceOntology.owl#>

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>


SELECT ?x  ?y ?name ?id

WHERE{

    ?x rdf:type src:Product.

     OPTIONAL{{?x src:id ?id.

               ?x src:name ?name.}

      UNION {    ?x src:id ?id. }}

}
```

The above query has been given for answering to the Mediator’s main ontology. The system is provided with two information sources IS1 and IS2, as it is described in Figure 3.6.

The IS1 has full mappings, however IS2 has as no mapped term the variable *?id.* Let’s now suppose that the user specifies the variable ?name as required and the variable ?id as optional, and also chooses as relaxation level ” Level4 – Unsupported Triples”. Whilst the query execution the whole optional part of the Reformulated query for IS2 will be omitted. However the real level of relaxation for each source is: for IS1 “Level1- Full Mappings” and for IS2 “Level 2- Unsupported Optional Part”. A ranking of the results between these two sources is not possible. However the final result set will be :

FINAL RESULT SET{http://www.owl-ontologies.com/TargetOntology.owl#=----------

LEVEL OF RELAXATION: Level1: Full Mappings

TYPE OF ANSWER: Possible Answer

VARIABLES ANSWERED: ?name , id

| x | y | name | id |

==========

==========

==========

==========

| <http://www.owl-ontologies.com/TargetOntology.owl#mathematics003> | "Calculus: Early Transcendentals"^^<http://www.w3.org/2001/XMLSchema#string> | "978-0743270783"^^<http://www.w3.org/2001/XMLSchema#string> |

| <http://www.owl-ontologies.com/TargetOntology.owl#mathematics003> | | "978-0743270783"^^<http://www.w3.org/2001/XMLSchema#string> |

| <http://www.owl-ontologies.com/TargetOntology.owl#biography002> | "The Science of Leonardo: Inside the Mind of the Great Genius of the Renaissance"^^<http://www.w3.org/2001/XMLSchema#string> | "978-0743270765"^^<http://www.w3.org/2001/XMLSchema#string> |

| <http://www.owl-ontologies.com/TargetOntology.owl#biography002> | | "978-0743270765"^^<http://www.w3.org/2001/XMLSchema#string> |

| <http://www.owl-ontologies.com/TargetOntology.owl#physics001> | "Atomic Physics"^^<http://www.w3.org/2001/XMLSchema#string> | "978-0743270786"^^<http://www.w3.org/2001/XMLSchema#string> |

| <http://www.owl-ontologies.com/TargetOntology.owl#physics001> | | "978-0743270786"^^<http://www.w3.org/2001/XMLSchema#string> |

```
|        <http://www.owl-ontologies.com/TargetOntology.owl#poetry001>
|                            |         "Letter          to          My
Daughter"^^<http://www.w3.org/2001/XMLSchema#string>
| "978-0743270780"^^<http://www.w3.org/2001/XMLSchema#string> |

|        <http://www.owl-ontologies.com/TargetOntology.owl#poetry001>
|                                                                    |
| "978-0743270780"^^<http://www.w3.org/2001/XMLSchema#string> |

|   <http://www.owl-ontologies.com/TargetOntology.owl#mathematics005>
|                |     "Mathematical     Logic     for     Computer
Science"^^<http://www.w3.org/2001/XMLSchema#string>
| "978-0743270785"^^<http://www.w3.org/2001/XMLSchema#string> |

|   <http://www.owl-ontologies.com/TargetOntology.owl#mathematics005>
|                                                                    |
| "978-0743270785"^^<http://www.w3.org/2001/XMLSchema#string> |

|   <http://www.owl-ontologies.com/TargetOntology.owl#mathematics002>
|                            |         "Advanced          Engineering
Mathematics"^^<http://www.w3.org/2001/XMLSchema#string>
| "978-0743270782"^^<http://www.w3.org/2001/XMLSchema#string> |

|   <http://www.owl-ontologies.com/TargetOntology.owl#mathematics002>
|                                                                    |
| "978-0743270782"^^<http://www.w3.org/2001/XMLSchema#string> |

|       <http://www.owl-ontologies.com/TargetOntology.owl#physics003>
|                           |         "Field         and         Wave
Electromagnetics"^^<http://www.w3.org/2001/XMLSchema#string>
| "978-0743270788"^^<http://www.w3.org/2001/XMLSchema#string> |

|       <http://www.owl-ontologies.com/TargetOntology.owl#physics003>
|                                                                    |
| "978-0743270788"^^<http://www.w3.org/2001/XMLSchema#string> |

, http://www.owl-ontologies.com/TargetOntology2.owl#----------
---

LEVEL OF RELAXATION:Level2: Unsupported OPT Part

VARIABLES ANSWERED: ?name
```

```
-------------------------------------------------------------------
-----------------------------------------------

|                                                                 x
| y | name                                                  | id |

===================================================================
===================================================================

|      <http://www.owl-ontologies.com/TargetOntology2.owl#science_25>
|   | "ttttttt"^^<http://www.w3.org/2001/XMLSchema#string>    |    |

|      <http://www.owl-ontologies.com/TargetOntology2.owl#science_25>
|   |                                                         |    |

|  <http://www.owl-ontologies.com/TargetOntology2.owl#best-seller_13>
|   | "afasfdasfd"^^<http://www.w3.org/2001/XMLSchema#string> |    |

|  <http://www.owl-ontologies.com/TargetOntology2.owl#best-seller_13>
|   |                                                         |    |

|   <http://www.owl-ontologies.com/TargetOntology2.owl#best-seller_1>
|   | ""^^<http://www.w3.org/2001/XMLSchema#string>           |    |

|   <http://www.owl-ontologies.com/TargetOntology2.owl#best-seller_1>
|   |                                                         |    |

|       <http://www.owl-ontologies.com/TargetOntology2.owl#science_2>
|   | ""^^<http://www.w3.org/2001/XMLSchema#string>           |    |

|       <http://www.owl-ontologies.com/TargetOntology2.owl#science_2>
|   |                                                         |    |

-------------------------------------------------------------------
-----------------------------------------------------}
```

Example 3.20 : example of a final result set

CHAPTER 4
System Implementation

4.1 *Introduction*

The previous chapter describes the general Ontology based Mediation System we have developed for managing information sources in a systematic manner. In this chapter, the system architecture and its physical components are described. We also present the GUI (graphical user interface) we have implemented in the context of this thesis.

4.2 *Architectural Overview*

In the context of this thesis an ***Ontology-based Mediator*** has been developed. The system receives a SPARQL query as input, translates and optimizes this query in several phases into a different executable query plan for each information source, executes the query plans and combines the query results in order to provide a final result set for the query from the query results coming from every information source.

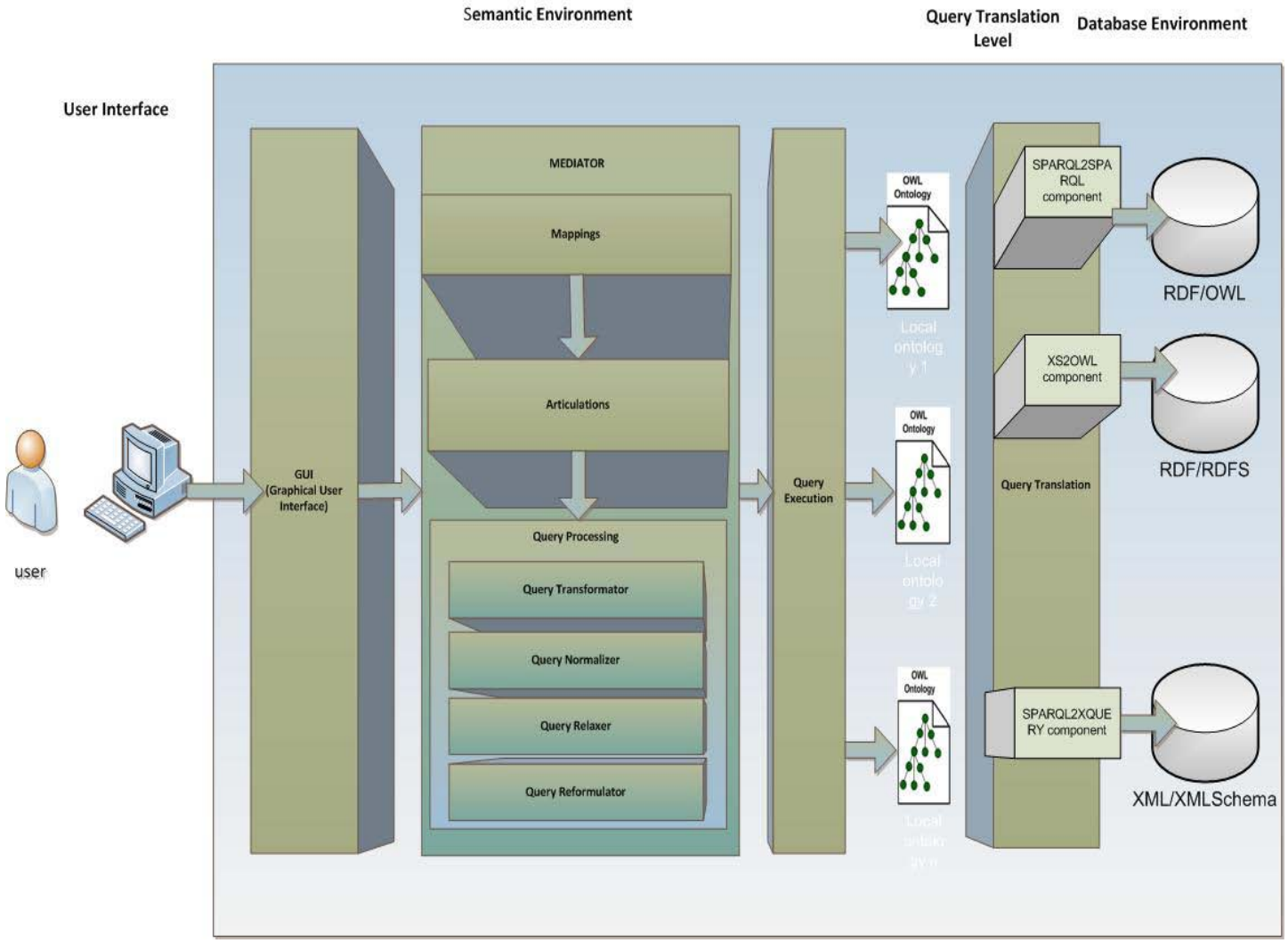
We developed a concrete system that is capable to offer information integration and interoperability to the potential users. For this purpose, we will use multiple ***information sources,*** with their ***local ontologies*** and their ***query transformation modules***. The system consists of four levels:

- 1. The first level (base) or ***database level*** includes all the information sources with their stored data. The types of information will vary, however the terminology of the terms will have a common "drift". We will be able to extract data from ***OWL/RDF*** *and* ***XML/XML Schema*** information sources. We assume that the objects of interest belong to an underlying domain that is common for all the sources, and that different sources may use different ontologies with terms expressed in different formats or in different levels of granularity.
- 2. The second level or ***query translation level*** consists of components for query translation and interpretation, based on related technologies (XS2OWL, XQuery etc.).This level will transform the accepted queries to a form appropriate for each underlying source. So, a SPARQL module may be used for an OWL/RDF data source, an

XQuery component for an XML source or we can have no use of such modules if the emerging query is in the right form.

- 3. The third level and the most important for us is the ***semantic environment.*** This level consists of the ***local ontologies*** of each source and the ***mediator.*** Every local ontology is interconnected with its own source and to the **main *ontology*** of the mediator with the use of ***mappings. Information Integration*** is obtained through a mediator comprising of the following parts:
    - A) An ontology, called ***Main Ontology***, with appropriate terms and components.
    - B) The ***Mappings,*** which imply the relationships between the sources, their ontologies and the main ontology. From these sets of mappings the set of the relations and correlations can be extracted and then the set of ***articulations*** *is formed****. A set of articulations*** to the information sources is a set of relationships between terms of the mediator and terms of the sources. These relationships are defined based on the mappings we already have, at the system's startup. The mediator uses the articulations in order to translate queries posed over the mediator ontology to queries over the ontologies of the articulated sources.
    - C) The **Query processing** component consists of query expansion, query normalization, query relaxation and query reformulation based on the mapped terms of every information source.
    - C) The transformation of the returned results, so they will be in an acceptable format for the user to read.
- 4. Last, but not least, the fourth level or the ***user interface level,*** is the interaction with the user of the system. The users submit queries expressed over the ontology of the mediator. Upon receiving a user query, the mediator has to query the underlying sources. The queries will be formatted in ***SPARQL.*** The interface is user-friendly and all the options are user-configured. In addition, this level also is responsible for the appearance of the results that the user has asked.

An overview of the architecture of the system is shown in Figure **4.1**



Fiigure 4.1: Architectural overview of the system

The figure shows the modules of the system and their union. It is also shown the basic components and the different layers/levels that will complete the final system

However in the context of this diploma thesis we mainly deal with the architecture shown in figure…, which outlines the semantic environment and the user interface:

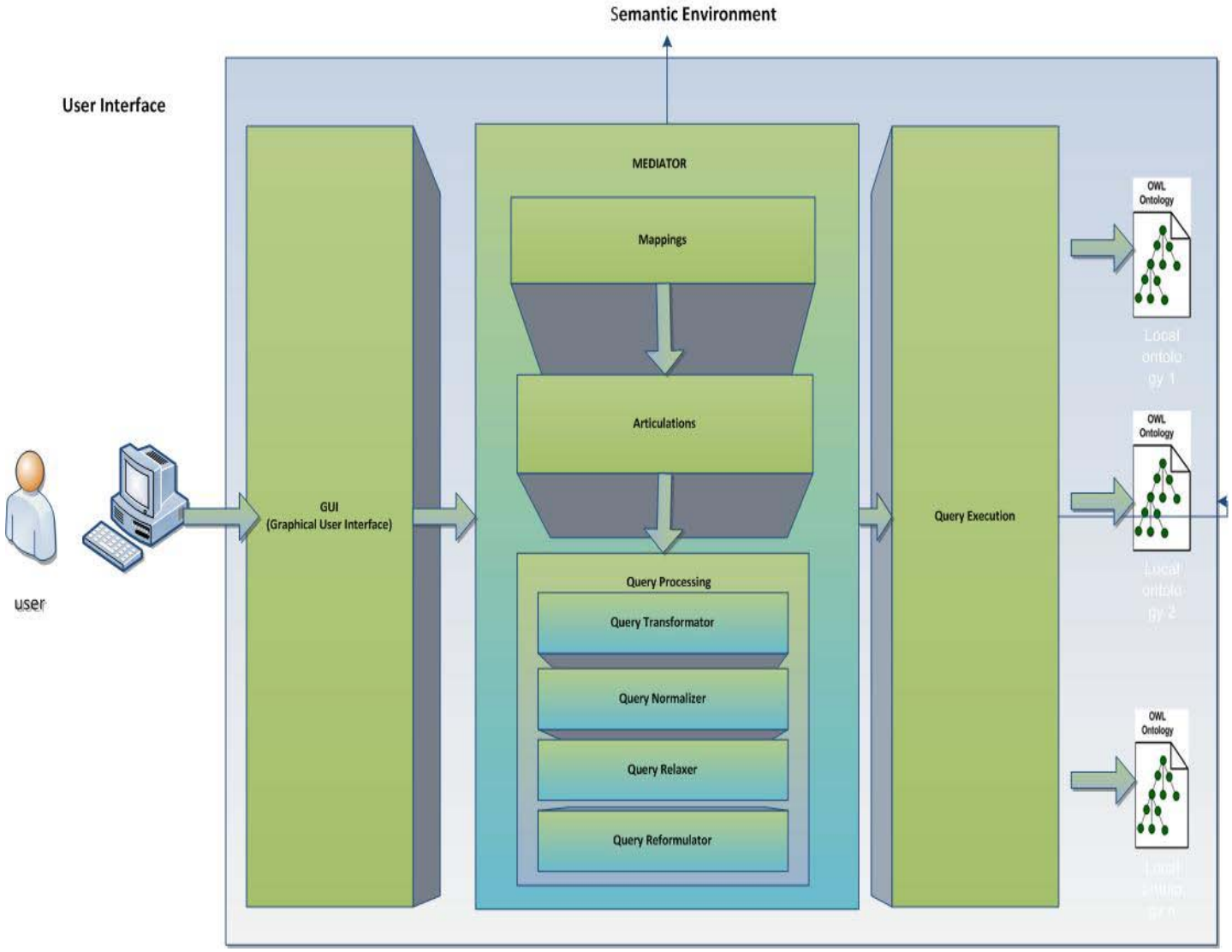


Figure 4.2: Architectural overview of our implementation

As it is clearly stated above, we pay more attention to the semantic environment and the user interface, focusing on all the possibilities that a mediation system could provide.

In **Figure 4.3** is shown the data flow through the system.

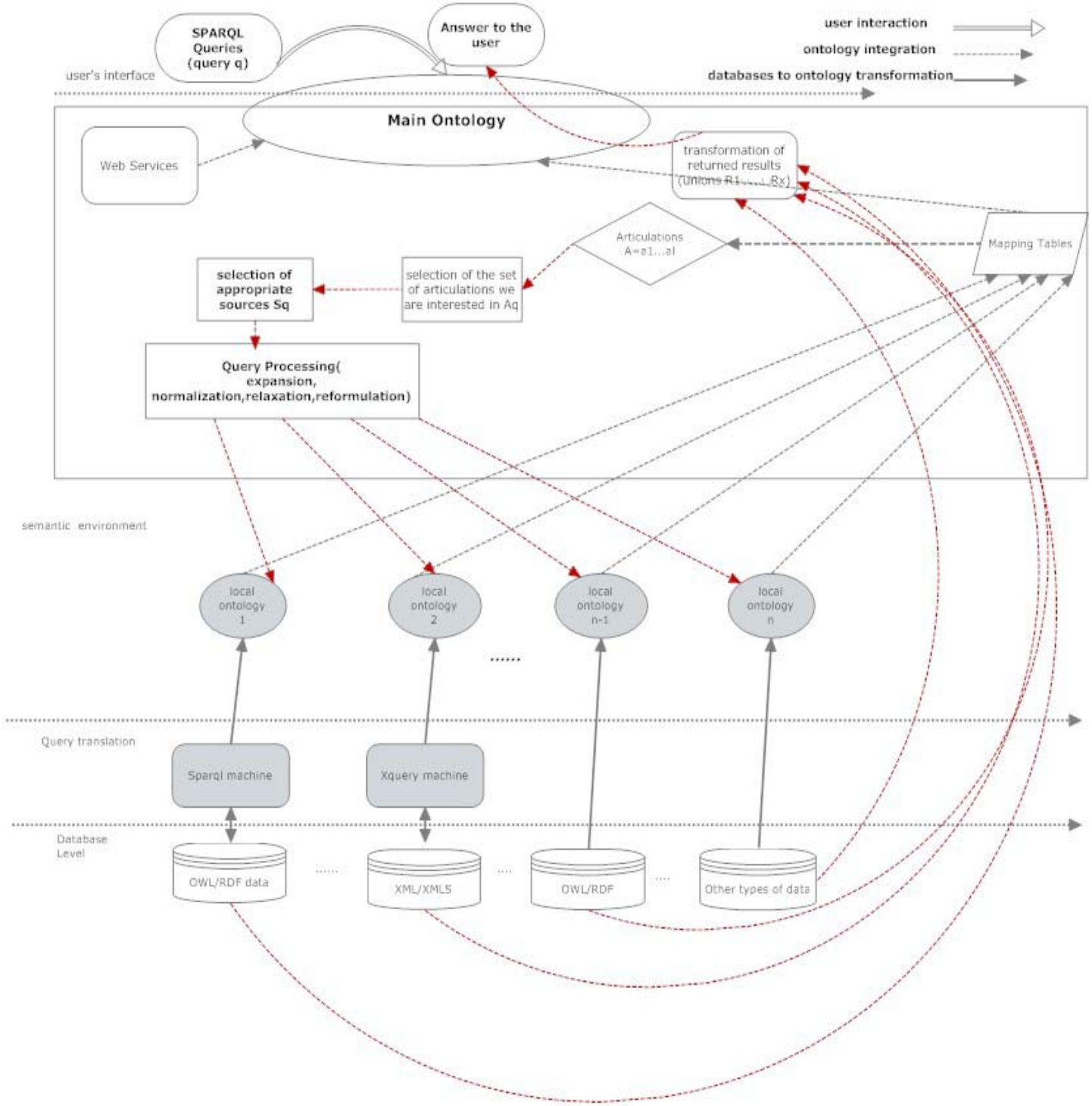


Figure 4.3: Data flow through the system

The figure shows the modules of the system and their union by using black arrows, and the dataflow from the moment a query is given, shown by the red arrows. The user interface is the most important component and it includes the terms described below.

4.3 System *Components*

At this point we are giving a more technical and descriptive explanation of the main components used in our system.

Mappings

In order for SPARQL queries posed over a global ontology to be rewritten in terms of a local ontology, mappings between the global and local ontologies should be specified.
We present some ways for elicitating the ontology mapping requirements of the mediator framework. Since our query rewriting methodology is generic, we will be discussing for mappings between a source and a target ontology rather than between global and local ontologies. The mapping types presented in this section have been selected among others because they can be used for the rewriting of a SPARQL query.

A source ontology class can be mapped to an expression between target ontology classes. The expression may involve union and intersection operations between classes. In addition, using expressions it is possible to restrict a class on some property values in order to form a correspondence. Similarly with classes, an individual from the source ontology can be mapped to an individual from the target ontology. Accordingly, an object/datatype property from the source ontology can be mapped to an object/datatype property from the target ontology. Domain and range restrictions can be useful for mappings between properties in order to restrict the individuals that participate in these two sets. In addition, an object property from the source ontology can be mapped to the inverse of an object property from the target ontology. Finally, a source ontology property can be mapped to an expression between target ontology properties. The expression may involve union, intersection and composition operations between properties.

Articulations

If we consider that a set of sources `IS1…ISn` over the *same* underlying set `Obj` of objects. Then In general, two different sources may have different terminologies either because the users of the two sources are familiar with different sets of terms, or because one source classifies objects at a different level of granularity than the other. The two sources may also have different subsumption relations as the relationships between any two given terms may be perceived differently in the two sources. Finally, two different sources may have different stored interpretations, for example some objects may be indexed by one source but not by the other. Clearly if one wants to combine or *integrate* information coming from different sources one has to cope with the above heterogeneities.

In this diploma thesis we propose the use of *mediator* as a mean for rendering all these heterogeneities transparent to users. In our approach, a mediator **M** has an ontology (main/global ontology) that reflects the needs of its potential users but has *no* stored objects. Instead, each term at the mediator is related directly or indirectly with the terms in the underlying sources. More formally, a mediator is defined as follows:

Definition 4.1 A mediator is the general system over n sources IS1…ISn and is consisted of:

1) an ontology (main/global), and
2) a set of *articulations* ai, one for each term of the main /global ontology

Roughly speaking, a mediator is just like a source but with an important difference: there is no interpretation stored at the mediator. What is stored at the mediator, instead, is the set of articulations ai For achieving integration we enrich the mediator with articulations, i.e. with relationships that relate the terms of the mediator with the terms of the sources as shown in Figure 4.4.

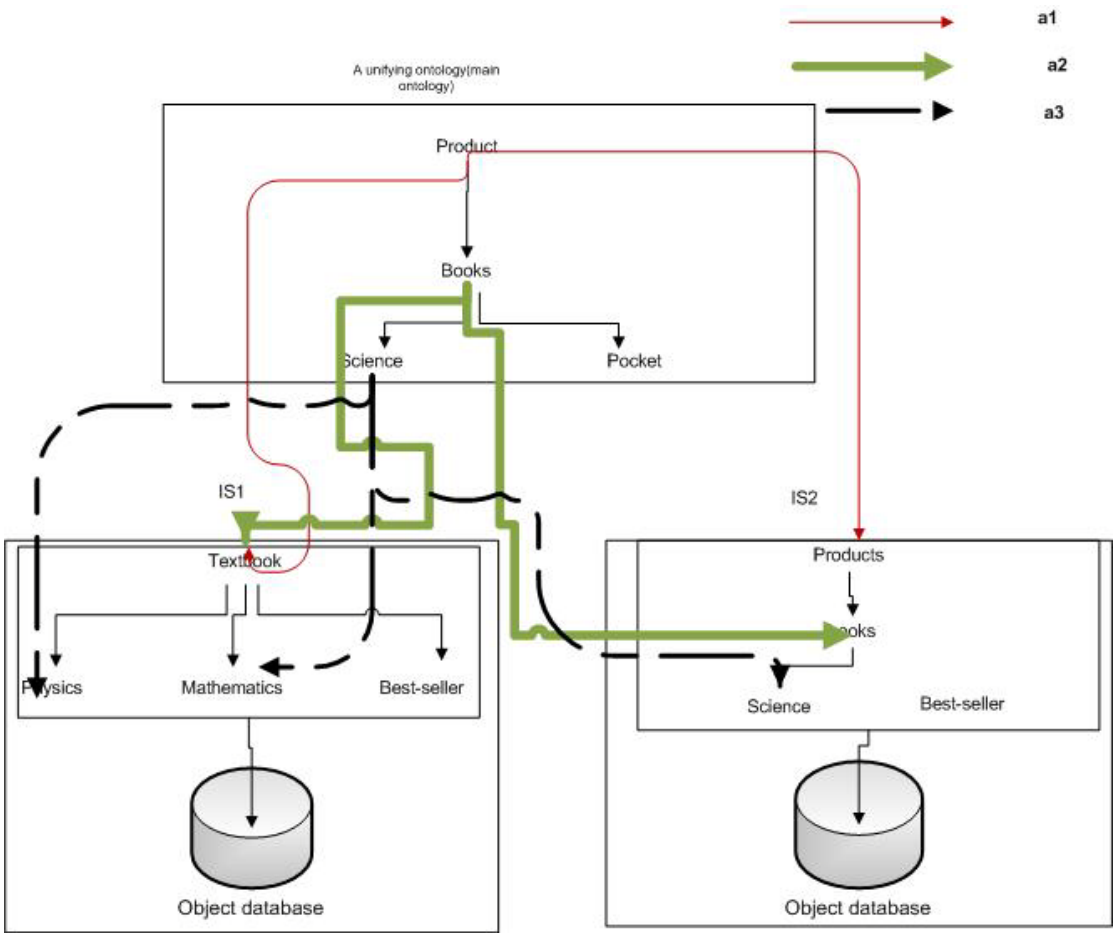


Figure 4 .4: Articulations

In the presence of several sources, one and the same term may appear in two or more sources. If the same term appears in two different sources then we consider the two appearances as two different terms, in the same articulation however.

The upper component **Query Processing** consists of the following four components:

Query Transformation Component

This component is based on the user specified required and optional variables. The goal of this transformation is to produce an expanded and distributed form of the original query based on the user's selection. The input of this component is the original query given by the user, with the specification of the variables that *must* be answered and the variables that are more relaxed. The output is an expansion of the SPARQL query with details of the wished results by the user. (See Chapter 5)

Query Normalizer

The query normalizer component is responsible for the normalization and optimization of a SPARQL query. Based on some simple Algebraic rules (See Chapter 5), rules of deleting extra brackets and parenthesis, the main objective of the SPARQL query Normalizer is to optimize each given query and to transform to an optimum UNION free well designed form. The *SPARQL Graph Pattern Normalization* activity re-writes the Graph-Pattern (*GP*) of the SPARQL query in an equivalent normal form. The SPARQL *GP* normalization is based on the *GP* expression equivalences and re-writing techniques. In particular, each *GP* can be transformed in a sequence *P1 UNION P2 UNION P3 UNION… UNION Pn*, where *Pi* ($1 \leq i \leq n$) is a Union-Free *GP* (i.e. *GPs* that do not contain Union operators). This makes the *GP* relaxation and reformulation process simpler and more efficient. As input to this component is the original query posed by the user and as output a normalized, consisted of union free GP, SPARQL query.

Query Relaxer

This component is one of the most important parts of our architecture. The user can choose on which functioning level the mediation system will work. Based now on the kind of the mappings and articulations, the relaxation is performed for each information source. Two information sources may be bound on different kind of relaxation (for example IS1: Full Mappings, IS2: Incomplete Mappings , No mapped term:?id, optional variable: id. If the user selects the first level of relaxation only IS1 will offer valid results. If however the user chooses a more relaxed level IS2 will also provide results, even though IS1 may be answered

with the Level 1(No Relaxation)) . As input to this component, the system provides the normalized query (from Query Normalizer) and as output the relaxed query for each information source.

Query Reformulation

The Query Reformulation component provides the SPARQL query reformulation process, using a the predefined set of mappings that follows the different mapping types. The SPARQL query reformulation process is based on the query's *graph pattern* reformulation and is consequently independent of the query type (*Ask, Select, Construct, Describe*). The SPARQL solution modifiers (*Limit, Offset, Order By, Distinct, Reduce*) are not taken into consideration since they do not affect the reformulation process. In general, the specific component accepts as input the relaxed query from each source and transforms it based on the terms of each local ontology, so it can be answered.

Query Execution

This component is logically present for the execution of each generated query to the appropriate source. Technically, we immediately address the reformulated query to each source to extract the available results.

4.4 Graphical User Interface for the Mediator

The OWL/SPARQL based Mediator for integration of OWL/RDF Information Sources presented in this thesis has been implemented as part of a Semantic Query Mediation Prototype Infrastructure developed in the TUC-MUSIC Lab.

The system has been implemented using Java 2SE as a software platform, and the Jena Software framework for SPARQL query parsing.

In this section, there will be a detailed description of the user interface implementation of the application. First, a description of the ten usability Heuristics principles that were written

by Jacob Nielsen (34) and how are they applied to the user interface of the application. Next, with the use of screenshots, the user interface of the application will be described in detail.

The Heuristic Principles mentioned above are:

1. *Visibility of System Status*

- The system should always keep users informed about what is going on, through appropriate feedback in reasonable time.

It is achieved by always giving accurate titles at the application dialogs and panels. Every panel has from the beginning informatory title and text inside about what he supposes to submit or what will be shown there.

2. *Match between System and the Real World*

- The system should speak in user's language with words, phrases and concepts familiar to the user, rather than system oriented terms. Follow real-world conventions, making information appear in a natural order.

The application uses terms and phrases that are commonly used in the real world and in the field of semantic web and information systems and are explanatory so the user can always know what is happening and what every term means without being necessary to be a professional or an expert of SPARQL and Semantic Web generally.

3. *User Control and Freedom*

- Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state, without having to go through an extended dialogue.

The graphical user's interface is implemented in a simple and clear way. However, if a mistake has occurred, error messages are shown as necessary, to prevent the user from an unwanted state. Also, the user can always return to the main view and pose another query. Lastly, shortcuts are being implemented for a quick exit from the system.

4. *Consistency and Standards*

- Users should not have to wonder whether different words, situations, or actions mean the same thing. Formulate platform conventions.

In every panel and dialog of the application, the same design logic (e.g. same colors, the same names in buttons that perform the same actions etc) is used. The description of every panel and text area is different and unique.

5. *Error Prevention*

- Even better than good error messages is a careful design, which prevents a problem from occurring in the first place.

The user is prevented from making mistakes, because in every step the application provides him with all the necessary actions and choices only, to prevent him from making wrong decisions and taking wrong actions.

6. *Recognition rather than recall*

- The user should not have to remember information from one dialogue to another.
  - Make objects, actions, and options visible.
  - Instructions for use of the system should be visible or easily retrievable whenever appropriate.

The user is not obligated to remember, in a given time, information from one dialog to another. In every point of the application there are sufficient elaborations in order to direct the user to do what he needs to do. The elaborations are short explanatory messages (tooltips), or even short labels that explain the process that can be followed.

7. *Flexibility and Efficiency of Use*

- Accelerators -unseen by novice users - often speed up the interaction for the expert user, such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.

The application can be used by expert users or even naïve users of SPARQL language without discrimination. In some cases the application is efficient and the calculations or the work load are not shown to the user, however every step of query processing can be viewed separately if it is wished.

8. Aesthetic and Minimalistic Design

- Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.

The look and feel of the application is the same with the operating system(i.e. Windows). That way, the application is closer to what the user is used to. Furthermore, an effort is made in order to not give unnecessary information to the user and to make the messages and labels as minimal as possible.

9. Help Users Recognize, Diagnose, and Recover from Errors

- Error messages should be expressed in plain language (no codes), precisely indicate the problem, and then constructively suggest a solution

As the principle suggests, the error messages of the application are in plain language, they precisely indicate the problem.

10. Help and Documentation

- Even though it is better if the system can be used without documentation, it may be necessary to provide and documentation. Any such information should be easy to search, focused on the user's task, list concisely the steps to be carried out, and not be too large.

The following text and view will describe in details the graphical user interface of the application and the functionality that it provides.

When the application starts running, a welcome window is displayed. In this window someone can see the main areas of the application. The main areas of the application are two, the **Query/Results** tab and the **Mediation Info** tab.. Besides the main areas, the application contains a menu bar. The functionality of each area will be described later. Figure 4.5 shows the welcome window of the application along with the different areas it contains.

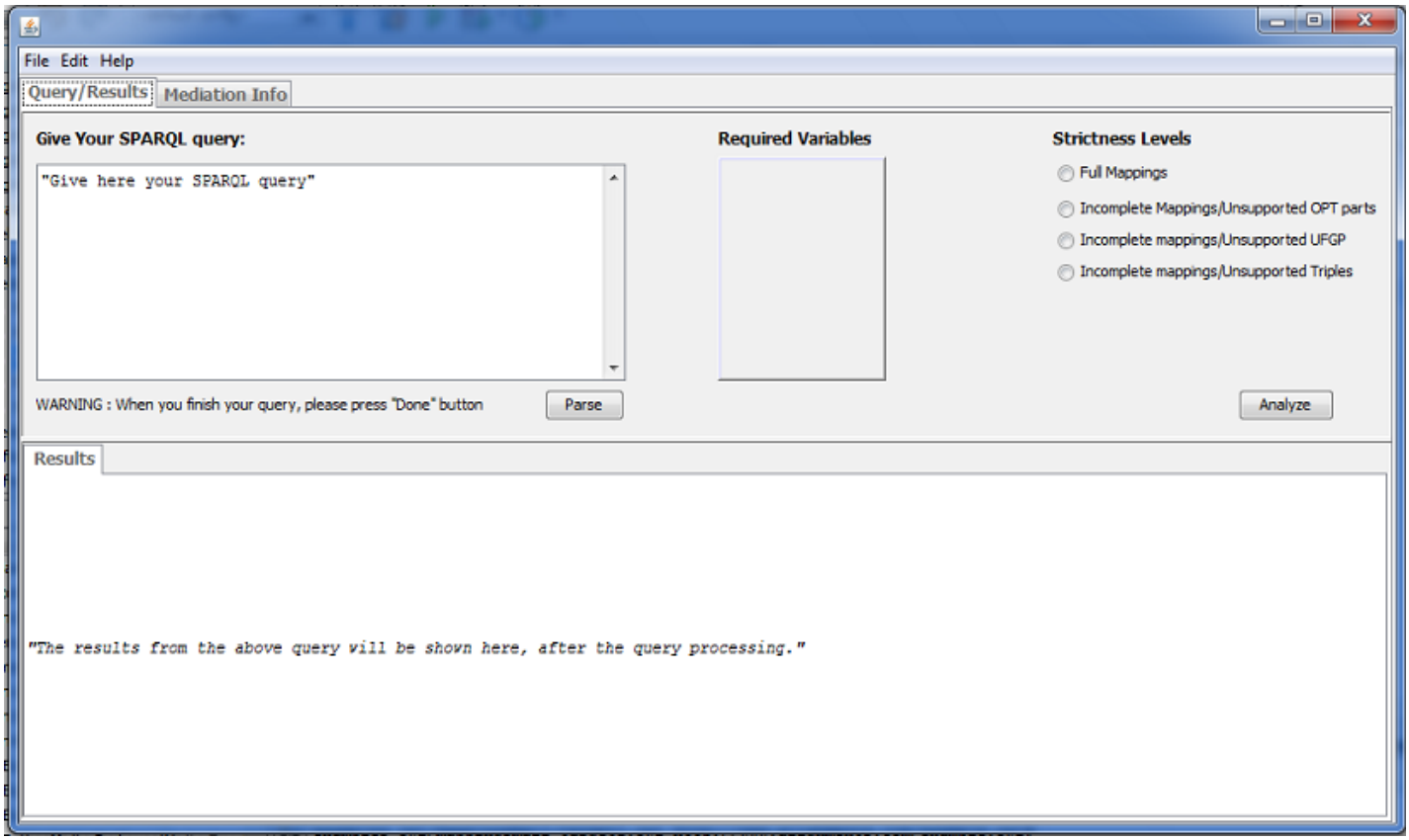


Figure 4.5 :Welcome Window

For the first time a user comes in touch with the system, the following views show the parts Query/Results tab and Mediation Info tab).

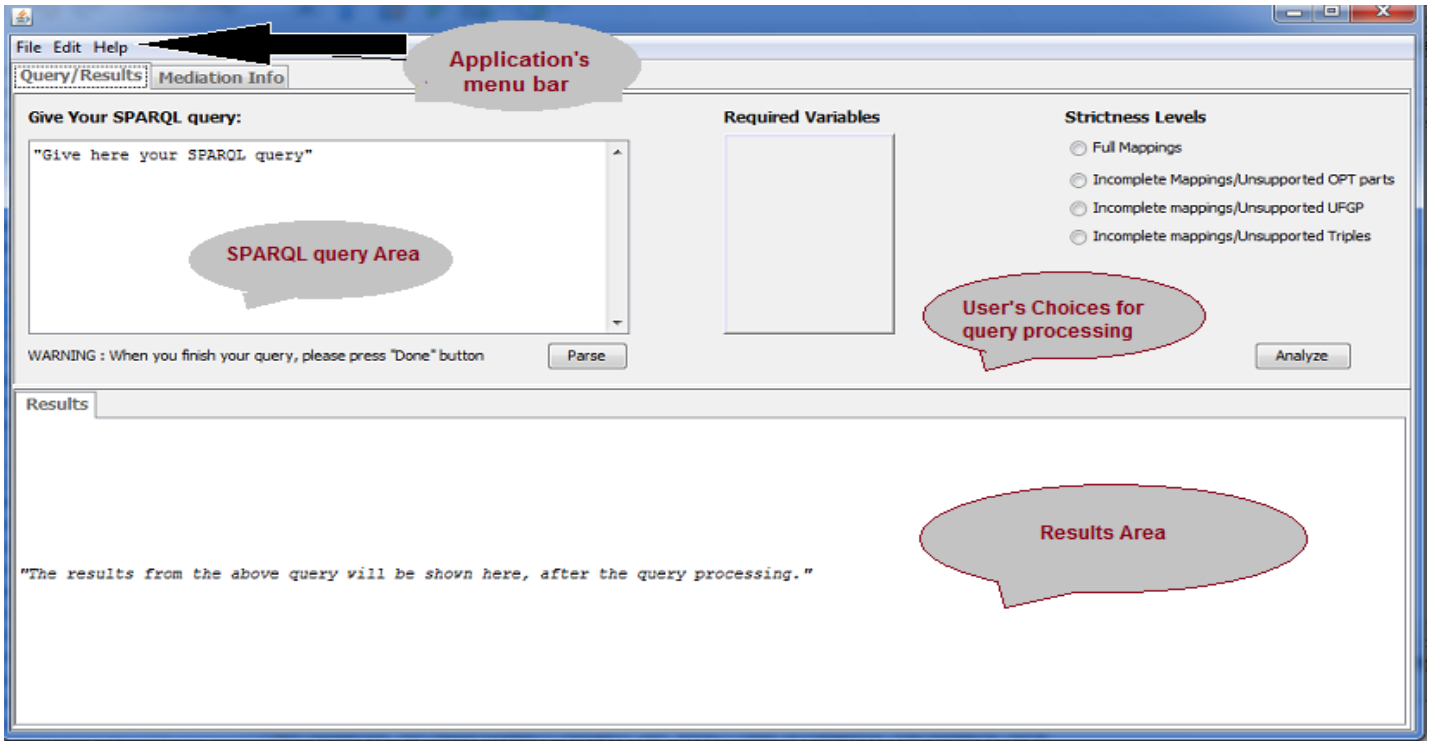


Figure 4.6 Query/Results tab

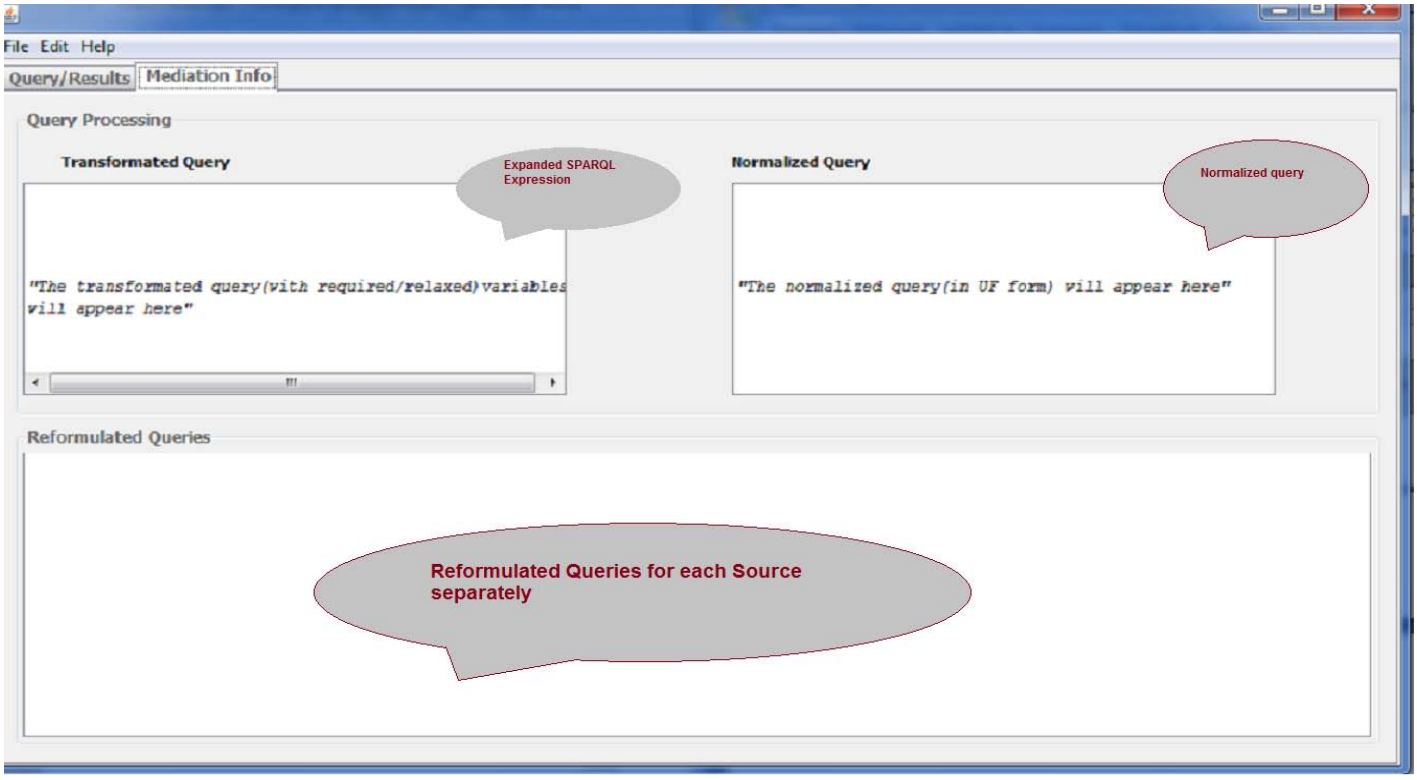


Figure 4.7 Mediation Info tab

Next follows a more detailed description of each area shown in the figure and the functionality every one of these areas provides.

- *Menu Bar*

The application defines its own menu in the menu bar, which provides the basic functionalities of the application. The standard menus provided by the Netbeans platform are also maintained.

- *SPARQL Query Area*

This area of the application is used to display the query posed by the user. The user can type his query in real time.

The next figure shows the state of the main area, when a SPARQL query is given:

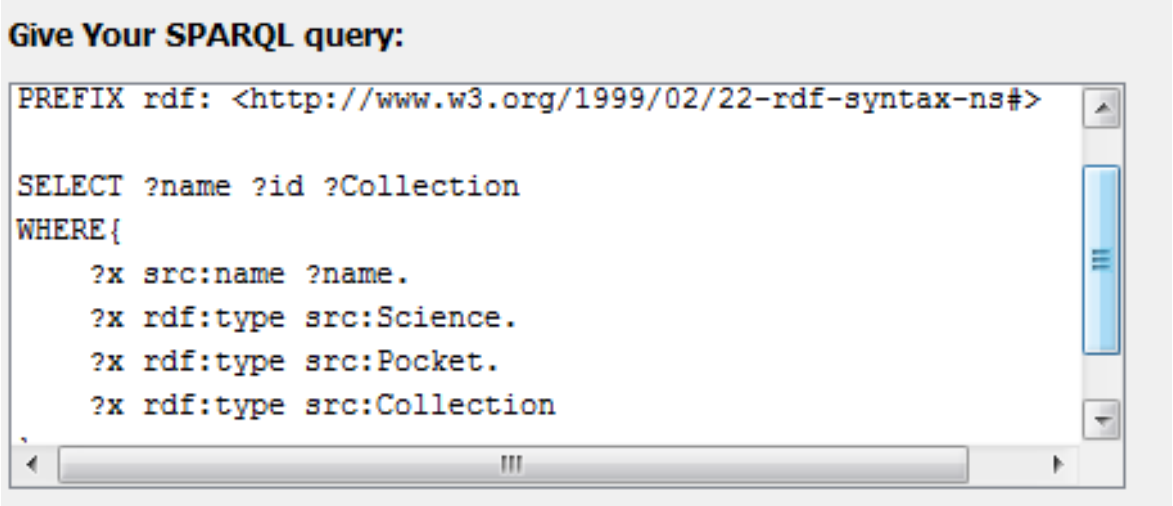


Figure 4.8: SPARQL query text area

- *User's choices of the Query Processing*

This area of the application is used to display all the information for the query that has been given to the system. The area contains two parts: **the selection of variables** part and the **relaxation levels** part. Each part is formatted after pushing the "**parse**" button and contains the choices that the user must make in order for the system to function. After these choises

the user should press the "**Analyze**" button, so for the system to proceed to query processing.

Figure 4.9 shows the format of this area after the query parsing.

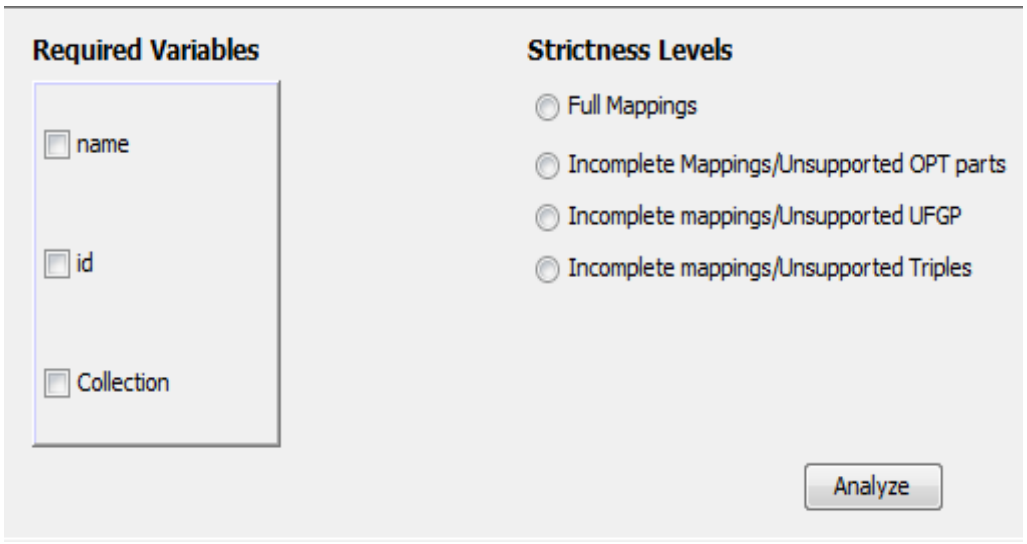


Figure 4.9 : User's choices of the Query Processing

- *Expanded SPARQL Expression Area*

After pressing the " **Analyze**" button the system continues with the query processing. The first step is to expand the query based on the required variables the user gives. At the second tab, if the user wishes to follow the query analysis , the expansion of his original query is presented.

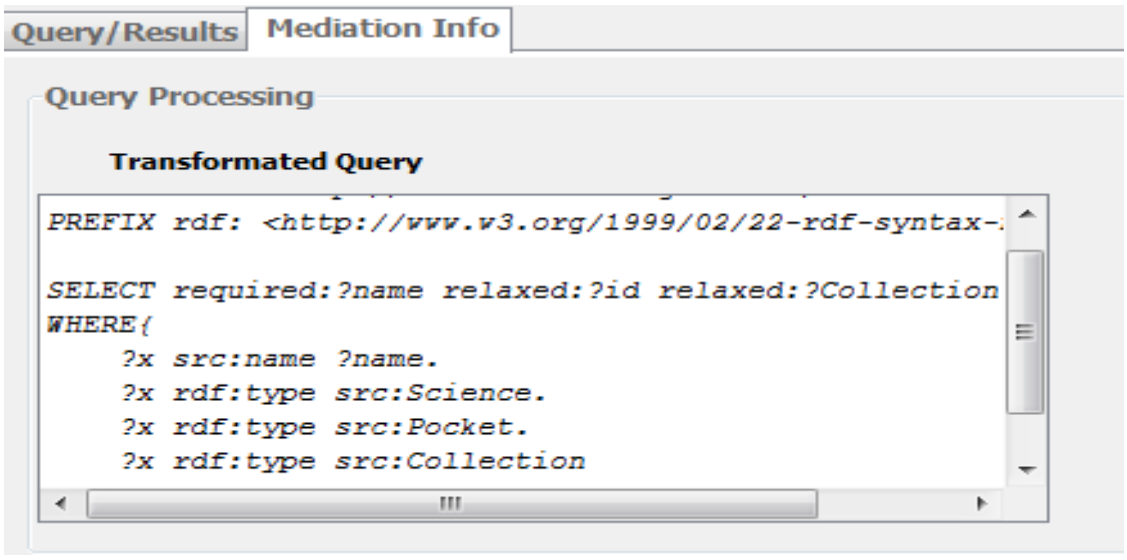


Figure 4.10: Expanded SPARQL expression

- *Normalization Area*

The next step is the presentation of the general Normalized Query. The query shown on this area is suitable for all of the information sources and is equivalent to the original one.

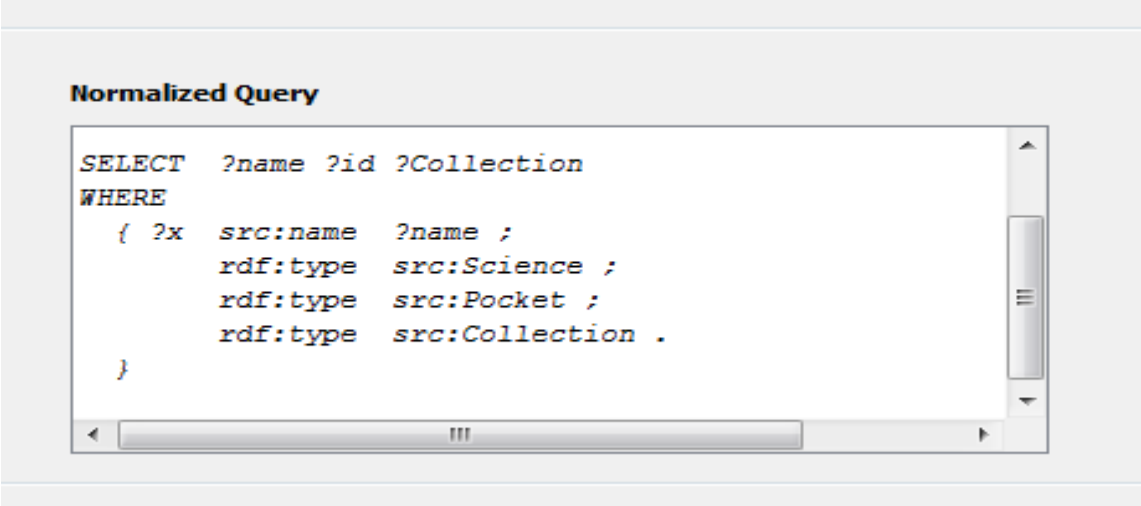


Figure 4.11: Normalized query area

- *Reformulated queries Area*

This area contains as many tabs as the information sources supported each time by the system. The modification of this area is dynamic. Each tab in this pane contains a reformulated query for each information source, based on the query produced by the proper relaxation. The Figure 4.12 shows the formatted area after the reformulation process.

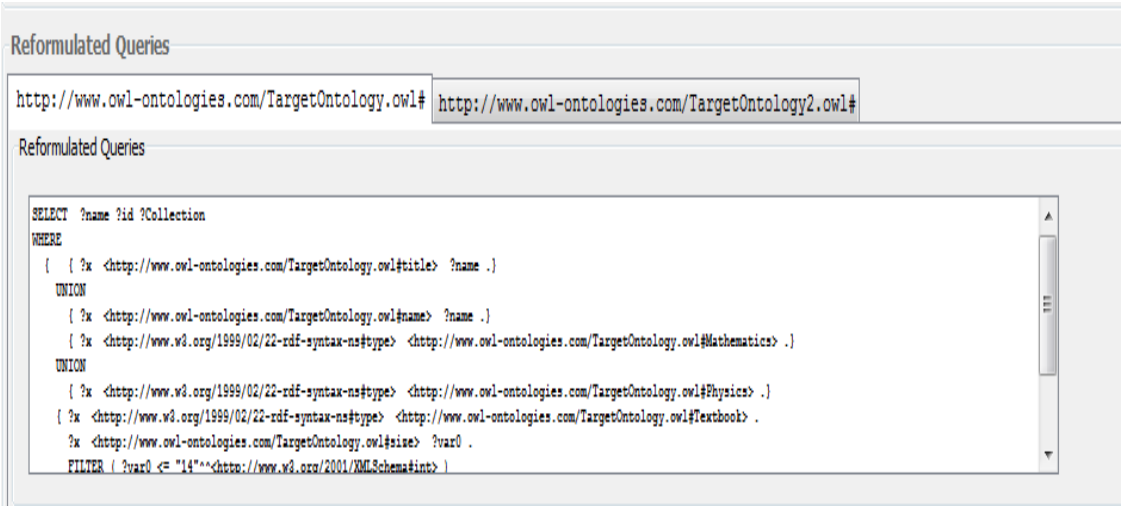


Figure 4.12: Reformulated Queries Area

- *Results Area*

After the query processing, the results from the query execution are presented at the Results Area. The results are shown for each source individually at every tab and a general result set is presented at a new tab too.

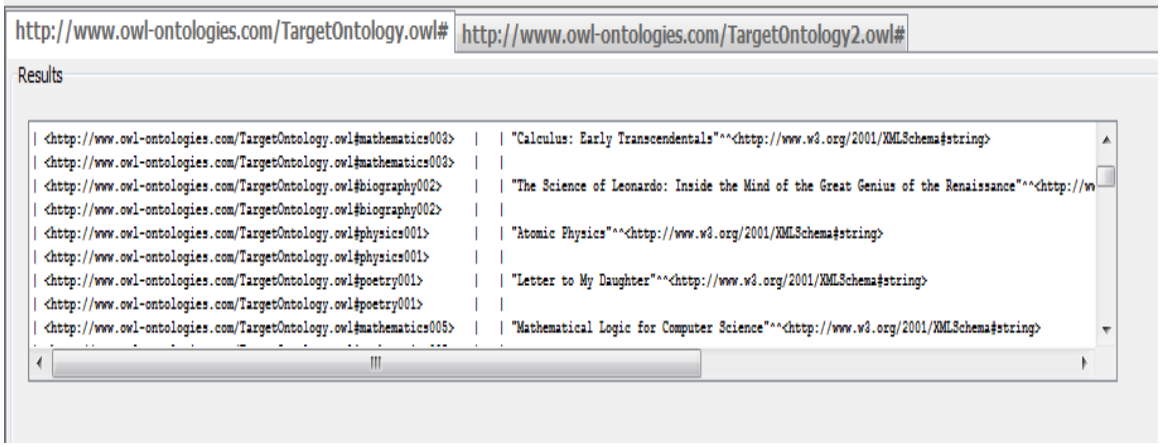


Figure 4.13: Results Area

Finally we present some total view of the system functioning.

Query/Results Tab

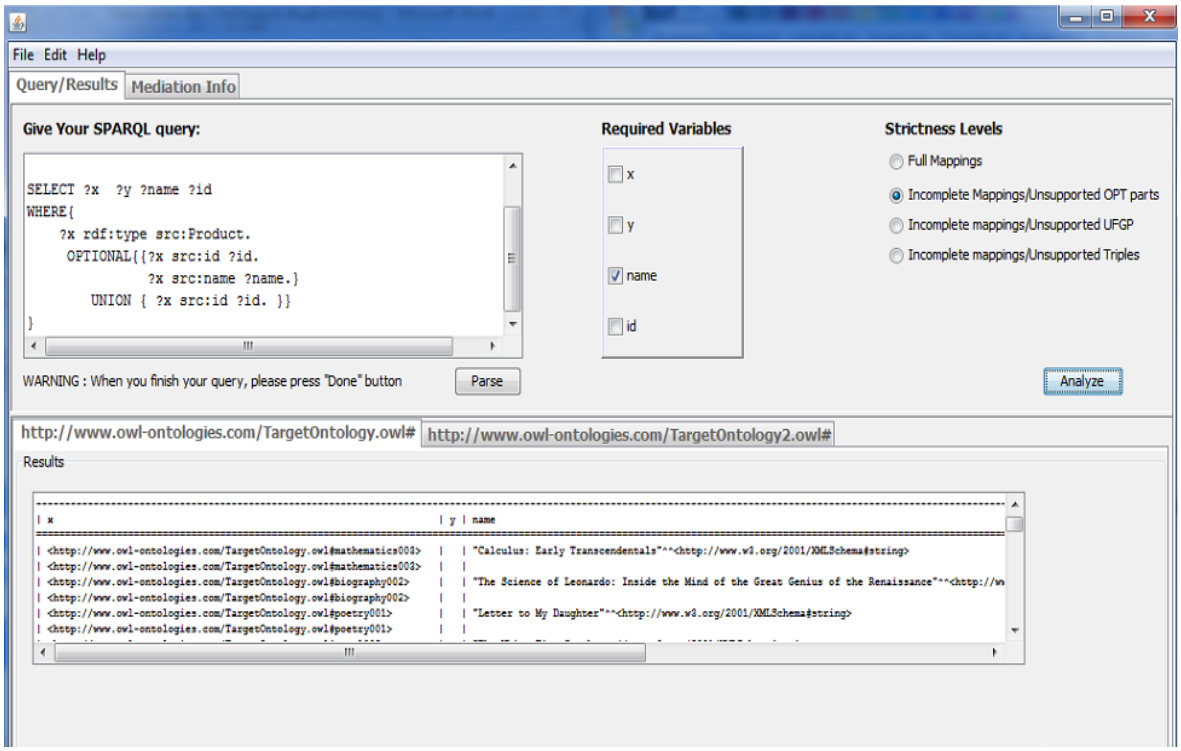


Figure 4.14: Systems Functioning(Query/Results)

Mediation Info

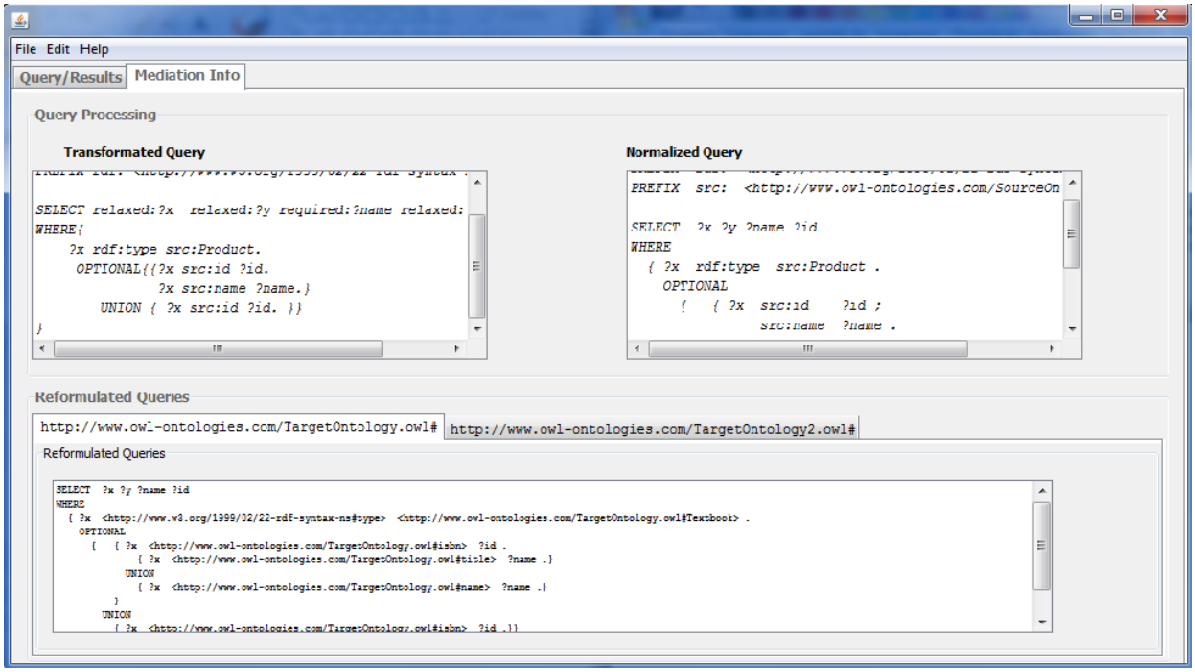


Figure 4.15: Systems Functioning(Mediation Info)

CHAPTER 5
Conclusions & Future Work

5.1 Conclusions

The web of data is heterogeneous, distributed and highly structured. Querying mechanisms in this environment have to overcome the problems arising from the heterogeneous and distributed nature of data, and they have to allow the expression of complex and structured queries. The ontology mappings and SPARQL query mediation presented in this thesis aim to satisfy those requirements in the Semantic Web environment. The mediator uses mappings between the global OWL ontology of the mediator and the local ontologies of the federated knowledge bases. SPARQL queries of end users and applications which are posed over the mediator, are processed(extended, normalized, relaxed and reformulated) in order to be submitted to the federated sources. The final SPARQL queries are locally evaluated and the results are returned to the mediator.

Mediators like the one presented in this current thesis, can be used to complement Semantic Web search engines and the web of Linked Data .
To maintain scalability, it is also possible to use multiple mediators at different locations which may synchronize catalog metadata and statistics over a peer-to-peer overlay network.

In the context of this diploma thesis an OWL/SPARQL based Mediator for Integration of OWL/ RDF Information Sources, has been designed and implemented. The mediator tries to give semantic interoperability between heterogeneous , distributed information sources.
Query federation based on SPARQL and Jena/ARQ has been demonstrated in detail and several concepts for query normalization and relaxation have been discussed.

For each federated source, supported by our system, a Query Processing component is dynamically created by a Query Factory. Such a component is able to rewrite an input SPARQL query based on some predefined mappings. As a representation language for the mappings between two overlapping ontologies we use the language presented Appendix[Schar09].
During the system's start-up each component is initialized with the mappings between the mediator's global ontology and the (local) ontology used in the federated source for which this component is responsible. The system also creates the articulations structure, which include all the relationships between the terms of the main ontology and the terms of the local ontologies.
At run time, when a SPARQL query is posed to the Mediator, it is processed, extended, normalized , relaxed and reformulated[Makris2010] for each federated. Afterwards, the final queries are submitted (routed) to the federated sites for local evaluation. Finally, the returned results from the local sources (to which queries have been routed to) are merged, and optionally visualized (taking into account which part of the initial SPARQL query was answered by each resource) for presentation to the end users.

5.2 Future Work

This current work is focused on the study of semantic interoperability of the data between OWL/RDF information Sources using the SPARQL Query Language.

Future extensions, part of which is already on schedule from Information Systems and Multimedia Applications – MUSIC Lab are:

- The possibility of supporting other types of sources and translation questions from the mediation system. The system is flexible, however adding more types of information sources (XML, RDFS, relational etc) with appropriate translation components (SPARQL2XQuery, XS2OWL etc.)
- The possibility to allow remote access to SPARQL endpoints via web services.
- Better processing and visualization of results.

CHAPTER 6
Bibliography

[1] J. Akahani, k. Hiramatsu, and T. Satoh. Approximate query reformulation for multiple ontologies in the semantic web. Technical report, 2003.

[2] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[3] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWLWeb Ontology Language Reference. Technical report, W3C, 2004.

[4] S. M. Benslimane, A. Merazi, M. Malki, and D. A. Bensaber. Ontology mapping for querying heterogeneous information sources. *INFOCOMP Journal of Computer Science*, 7(3):52–59, 2008.

[5] A. Bernstein, C. Kiefer, and M. Stocker. Optarq: A sparql optimization approach based on triple pattern selectivity estimation. Technical report, University of Zurich, 2007.

[6] N. Bikakis, N. Gioldasis, C. Tsinaraki, and S. Christodoulakis. Querying xml data with sparql. In *DEXA*, volume 5690 of *Lecture Notes in Computer Science*, pages 372–381. Springer, 2009.

[7] P. Bouquet, F. Giunchiglia, F. van Harmelen, L. Serafini, and H. Stuckenschmidt. Cowl: Contextualizing ontologies. In *International Semantic Web Conference*, volume 2870 of *Lecture Notes in Computer Science*, pages 164–179. Springer, 2003.

[8] D. Brickley and R. Guha. Rdf vocabulary description language 1.0: Rdf schema. World Wide Web Consortium, Recommendation, 2004.

[9] A. Chebotko, S. Lu, and F. Fotouhi. Semantics preserving sparql-to-sql translation. *Data Knowl. Eng.*, 68(10):973–1000, 2009.

[10] N. Choi, I.-Y. Song, and H. Han. A survey on ontology mapping. *SIGMOD Record*, 35(3):34–41, 2006.

[11] G. Correndo, M. Salvadores, I. Millard, H. Glaser, and N. Shadbolt. Sparql query rewriting for implementing data integration over linked data. In *1st International Workshop on Data Semantics (DataSem 2010)*, 2010.

[12] M. Doerr, C.-E. Ore, and S. Stead. The CIDOC conceptual reference model - A new standard for knowledge sharing. In *26th International Conference on Conceptual Modeling - ER 2007*, volume 83 of *CRPIT*, pages 51–56, 2007.

[13] M. J. D¨urst and M. Suignard. Internationalized resource identifiers (IRIs), 2005.

[14] M. Ehrig and S. Staab. Qom - quick ontology mapping. In *International Semantic Web Conference*, volume 3298 of *Lecture Notes in Computer Science*, pages 683–697.

Springer, 2004.

[15] B. Elliott, E. Cheng, C. Thomas-Ogbuji, and Z. M. Ozsoyoglu. A complete translation from sparql into efficient sql. In *IDEAS '09: Proceedings of the 2009 International Database Engineering & Applications Symposium*, pages 31–42. ACM, 2009.

[16] J. Euzenat. An api for ontology alignment. In *International Semantic Web Conference*, volume 3298 of *Lecture Notes in Computer Science*, pages 698–712. Springer, 2004.

[17] J. Euzenat et al. D2.2.3: State of the art on ontology alignment, 2004.

[18] J. Euzenat, A. Polleres, and F. Scharffe. Processing ontology alignments with sparql. In *CISIS*, pages 913–917, 2008.

[19] J. Euzenat, F. Scharffe, and A. Zimmermann. Expressive alignment language and implementation. deliverable 2.2.10, Knowledge Web NoE, 2007.

[20] J. Euzenat and P. Shvaiko. *Ontology Matching*. Springer, Heidelberg, 2007.

[21] J. Groppe, S. Groppe, and J. Kolbaum. Optimization of sparql by using coresparql. In *ICEIS (1)*, pages 107–112, 2009.

[22] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. Swrl: A semantic web rule language combining owl and ruleml. W3c member submission, World Wide Web Consortium, 2004.

[23] P. Jain, P. Z. Yeh, K. Verma, C. A. Henson, and A. P. Sheth. Sparql query re-writing using partonomy based transformation rules. In *GeoS '09: Proceedings of the 3rd International Conference on GeoSpatial Semantics*, pages 140–158. Springer-Verlag, 2009.

[24] Y. Jing, D. Jeong, and D.-K. Baik. Sparql graph pattern rewriting for owl-dl inference

queries. *Knowl. Inf. Syst.*, 20(2):243–262, 2009.

[25] Y. Kalfoglou and W. M. Schorlemmer. Ontology mapping: The state of the art. In *Semantic Interoperability and Integration*, volume 04391 of *Dagstuhl Seminar Proceedings*. IBFI, Schloss Dagstuhl, Germany, 2005.

[26] A. Maedche, B. Motik, N. Silva, and R. Volz. Mafra - a mapping framework for distributed ontologies. In *EKAW*, volume 2473 of *Lecture Notes in Computer Science*, pages 235–250. Springer, 2002.

[27] K. Makris, N. Bikakis, N. Gioldasis, C. Tsinaraki, and S. Christodoulakis. Towards a mediator based on owl and sparql. In *WSKS (1)*, volume 5736 of *Lecture Notes in Computer Science*, pages 326–335, 2009.

[28] F. Manola and E. Miller. RDF primer. World Wide Web Consortium, Recommendation, 2004.

[29] M. H. Needleman. Dublin core metadata element set. 24(3-4):131–135, 1998.

[30] N. F. Noy. Semantic integration: a survey of ontology-based approaches. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 33(4):65–70, 2004.

[31] N. F. Noy and M. A. Musen. PROMPT: Algorithm and tool for automated ontology merging and alignment. In *Proc. of AAAI/IAAI-2000: 450-455*, 2000.

[32] R. Pan, Z. Ding, Y. Yu, and Y. Peng. A bayesian network approach to ontology mapping. In *International Semantic Web Conference*, volume 3729 of *Lecture Notes in Computer Science*, pages 563–577. Springer, 2005.

[33] J. P´erez, M. Arenas, and C. Gutierrez. Semantics and complexity of sparql. *ACM Trans. Database Syst.*, 34(3), 2009.

[34] A. Polleres, F. Scharffe, and R. Schindlauer. Sparql++ for mapping between rdf vocabularies. In *OTM Conferences (1)*, volume 4803 of *Lecture Notes in Computer*

Science, pages 878–896. Springer, 2007.

[35] E. Prud'hommeaux and A. Seaborne. Sparql query language for rdf. Technical report, W3C, 2008.

[36] B. Quilitz and U. Leser. Querying distributed rdf data sources with sparql. In *ESWC*, volume 5021 of *Lecture Notes in Computer Science*, pages 524–538. Springer, 2008.

[37] F. Scharffe. *Correspondence Patterns Representation*. PhD thesis, University of Innsbruck,
2009.

[38] F. Scharffe and J. de Bruijn. A language to specify mappings between ontologies. In *SITIS*, pages 267–271, 2005.

[39] S. Schenk and S. Staab. Networked graphs: a declarative mechanism for sparql rules, sparql views and rdf data integration on the web. In *WWW*, pages 585–594. ACM, 2008.

[40] M. Schmidt, M. Meier, and G. Lausen. Foundations of sparql query optimization. *CoRR*, abs/0812.3788, 2008.

[41] P. Shvaiko and J. Euzenat. A survey of schema-based matching approaches. pages 146–171, 2005.

[42] M. Stocker, A. Seaborne, A. Bernstein, C. Kiefer, and D. Reynolds. Sparql basic graph pattern optimization using selectivity estimation. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 595–604. ACM, 2008.

[43] H. Wache, T. V¨ogele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. H¨ubner. Ontology-based integration of information - a survey of existing approaches. In *Workshop on Ontologies and Information Sharing*, pages 108–117, 2001.

[44] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, 1992.

[45] [Be&al08] Beckett D. (ed), "SPARQL Query Results XML Format". W3C recommendation, 15 January 2008, (*http://www.w3.org/TR/rdf-SPARQL-XMLres/*).

[46] [Gua98] N. Guarino, "Formal Ontology and Information Systems," Proc. First Int'l Conf. Formal Ontology in Information Systems (FOIS '98), pp. 3-15, June 1998.

[47] [IorCic08] Acta Polytechnica Hungarica Vol. 5, No. 2, 2008 Ontologies used for Competence Management Victoria Iordan, Alexandru Cicortas**,** Computer Science Department University of the West Timisoara

[48] [BerHenLas01] Tim Berners-Lee, James Hendler and Ora Lassila "The Semantic Web A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities" May 17, 2001

[49] [TziSpyCon01] Yannis Tzitzikas, Nicolas Spyratos, Panos Constantopoulos "Mediators over Ontology based Information Sources", WISE 2001, vol. 1, pp.0031, Second International Conference on Web Information Systems Engineering (WISE'01) Volume 1, 2001

[50] [GaliLEg] C. Galindo-Legaria , "Algebraic Optimization of Outerjoin Queries".

[51] [Schar09] Francois Scharffe "Correspondence Patterns Representation", dissertation, submitted to the Faculty of Mathematics, Computer Science and Physics of the University of Innsbruck, Innsbruck March 9, 2009

[52] [Euz,Schar,Zim 07] Jérôme Euzenat, François Scharffe, Antoine Zimmermann: D2.2.10: Expressive alignment language and implementation. Knowledge Web EU-IST Project deliverable 2.2.10, 2007.

[53] [Prud'hommeaux and Seaborne 2008] endorsed by the W3C, formalizes a semantics based on work of [Perez et al. 2006a, 2006b; Arenas et al. 2007].

[54] [Perez et al. 2006] Semantics and Complexity of SPARQL

Appendix

Mapping Representation Example

In this section, we provide the representation of the mappings defined in chapter 2.3 and 3.1 for the ontologies presented in Figure 3.1. As we mentioned in section 2.3, the mapping representation language that we use, has been defined in [2, 3] and as we can observe below, it is expressive, simple and Semantic Web compliant (given its RDF syntax). The most important, is that this language satisfies our requirements,
representing clearly the supported mapping types.

```
<rdf:RDF xml:base="">
<Alignment rdf:about="http://www.owl-ontologies.com/SourceOntology.owl">
<xml>yes</xml>
<dc:creator rdf:resource=""/>
<dc:date>2009/09/10</dc:date>
<level rdf:datatype="http://www.w3.org/2001/XMLSchema#int">2</level>
<type>**</type>
<onto1>
<Ontology rdf:about="http://www.owl-ontologies.com/SourceOntology.owl"/>
</onto1>
<onto2>
<Ontology rdf:about="http://www.owl-ontologies.com/TargetOntology.owl"/>
</onto2>
<map>
<Cell rdf:about="MappingRule_a">
<entity1>
<Class rdf:about="http://www.owl-ontologies.com/SourceOntology.owl#Book"/>
</entity1>
<entity2>
<Class rdf:about=" http://www.owl-ontologies.com/TargetOntology.owl#Textbook"/>
</entity2>
<relation>equivalence</relation>
</Cell>
</map>
<map>
<Cell rdf:about="MappingRule_b">
<entity1>
<Class rdf:about="http://www.owl-ontologies.com/SourceOntology.owl#Publisher"/>
</entity1>
<entity2>
<Class rdf:about="http://www.owl-ontologies.com/TargetOntology.owl#Publisher"/>
</entity2>
<relation>equivalence</relation>
</Cell>
</map>
<map>
<Cell rdf:about="MappingRule_c">
<entity1>
<Class rdf:about="http://www.owl-ontologies.com/SourceOntology.owl#Expert"/>
</entity1>
<entity2>
<Class rdf:about="http://www.owl-ontologies.com/TargetOntology.owl#Expert"/>
</entity2>
<relation>equivalence</relation>
</Cell>
</map>
```

```
<map>
<Cell rdf:about="MappingRule_d">
<entity1>
<Class rdf:about=" http://www.owl-ontologies.com/TargetOntology.owl#Textbook"/>
</entity1>
<entity2>
<Class rdf:about=" http://www.owl-ontologies.com/SourceOntology.owl#Product"/>
</entity2>
<relation>subsumption</relation>
</Cell>
</map>
<map>
<Cell rdf:about="MappingRule_e">
<entity1>
<Class rdf:about=" http://www.owl-ontologies.com/SourceOntology.owl#Collection"/>
</entity1>
<entity2>
<Class rdf:about=" http://www.owl-ontologies.com/TargetOntology.owl#Series"/>
</entity2>
<relation>subsumption</relation>
</Cell>
</map>
<map>
<Cell rdf:about="MappingRule_f">
<entity1>
<Class rdf:about=" http://www.owl-ontologies.com/SourceOntology.owl#Science"/>
</entity1>
<entity2>
<Class>
<or rdf:parseType="Collection">
<item>
<Class rdf:about="http://www.owl-ontologies.com/TargetOntology.owl#Physics"/>
</item>
<item>
<Class rdf:about="http://www.owl-ontologies.com/TargetOntology.owl#Mathematics"/>
</item>
</or>
</Class>
</entity2>
<relation>equivalence</relation>
</Cell>
</map>
<map>
<Cell rdf:about="MappingRule_g">
<entity1>
<Class rdf:about=" http://www.owl-ontologies.com/SourceOntology.owl#Popular"/>
</entity1>
<entity2>
<Class>
<and rdf:parseType="Collection">
<item>
<Class>
<or rdf:parseType="Collection">
<item>
<Class rdf:about="http://www.owl-ontologies.com/TargetOntology.owl#Physics"/>
</item>
<item>
<Class rdf:about="http://www.owl-ontologies.com/TargetOntology.owl#Mathematics"/>
</item>
</or>
</Class>
</item>
<item>
<Class rdf:about="http://www.owl-ontologies.com/TargetOntology.owl#BestSeller"/>
</item>
</and>
</Class>
</entity2>
<relation>equivalence</relation>
</Cell>
</map>
<map>
```



```
<Cell rdf:about="MappingRule_h">
```

```
<entity1>
<Class rdf:about="http://www.owl-ontologies.com/SourceOntology.owl#Pocket"/>
</entity1>
<entity2>
<Class rdf:about=" http://www.owl-ontologies.com/TargetOntology.owl#Textbook">
<attributeValueCondition>
<Restriction>
<onAttribute>
<Property rdf:about=" http://www.owl-ontologies.com/TargetOntology.owl#size"/>
</onAttribute>
<comparator>xsd:less-than-or-equal</comparator>
<value>14</value>
</Restriction>
</attributeValueCondition>
</Class>
</entity2>
<relation>equivalence</relation>
</Cell>
</map>
<map>
<Cell rdf:about="MappingRule_i">
<entity1>
<Relation rdf:about=" http://www.owl-ontologies.com/SourceOntology.owl#publisher"/>
</entity1>
<entity2>
<inverse>
<Relation rdf:about=" http://www.owl-ontologies.com/TargetOntology.owl#publishes"/>
</inverse>
</entity2>
<relation>equivalence</relation>
</Cell>
</map>
<map>
<Cell rdf:about="MappingRule_j">
<entity1>
<Relation rdf:about=" http://www.owl-ontologies.com/SourceOntology.owl#author"/>
</entity1>
<entity2>
<Relation>
<and rdf:parseType="Collection">
<item>
<Relation rdf:about=" http://www.owl-ontologies.com/TargetOntology.owl#author"/>
</item>
<item>
<Relation rdf:about=" http://www.owl-ontologies.com/TargetOntology.owl#creator"/>
</item>
</and>
</Relation>
</entity2>
<relation>equivalence</relation>
</Cell>
</map>
<map>
<Cell rdf:about="MappingRule_k">
<entity1>
<Relation rdf:about=" http://www.owl-ontologies.com/SourceOntology.owl#partOf"/>
</entity1>
<entity2>
<Relation rdf:about=" http://www.owl-ontologies.com/TargetOntology.owl#partOf"/>
<domainRestriction>
<Class ref:about=" http://www.owl-ontologies.com/TargetOntology.owl#Textbook" >
<attributeValueCondition>
<Restriction>
<onAttribute>
<Property rdf:about=" http://www.owl-ontologies.com/TargetOntology.owl#size"/>
</onAttribute>
<comparator>xsd:less-than-or-equal</comparator>
<value>14</value>
</Restriction>
</attributeValueCondition>
</Class>
</domainRestriction>
</entity2>
<relation>equivalence</relation>
```

```
</Cell>
</map>
<map>
<Cell rdf:about="MappingRule_l">
<entity1>
<Property rdf:about=" http://www.owl-ontologies.com/SourceOntology.owl#name"/>
</entity1>
<entity2>
<Property rdf:about="http://www.owl-ontologies.com/TargetOntology.owl#title"/>
</entity2>
<relation>equivalence</relation>
</Cell>
</map>
<map>
<Cell rdf:about="MappingRule_m">
<entity1>
<Property rdf:about=" http://www.owl-ontologies.com/TargetOntology.owl#isbn"/>
</entity1>
<entity2>
<Property rdf:about=" http://www.owl-ontologies.com/SourceOntology.owl#id"/>
</entity2>
<relation>subsumption</relation>
</Cell>
</map>
<map>
<Cell rdf:about="MappingRule_n">
<entity1>
<Instance rdf:about=" http://www.owl-ontologies.com/SourceOntology.owl#AlanTuring"/>
</entity1>
<entity2>
<Instance rdf:about=" http://www.owl-ontologies.com/TargetOntology.owl#ATuring"/>
</entity2>
<relation>equivalence</relation>
</Cell>
</map>
</Alignment>
</rdf:RDF>
```

Alignment Ontology

Based on the owl ontology in [Schar09] (see Appendix: Alignment Ontology), which introduces some control to the terms of the Alignment language. It gives a formal meaning in terms of OWL entities and restrict their usage using description logic restrictions.

```
<?xml version="1.0"?>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns=""
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
xmlns:align="http://knowledgeweb.semanticweb.org/heterogeneity/alignment#"
xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xml:base="http://www.omwg.org/TR/d7/ontology/alignment/">
<owl:Ontology rdf:about="http://www.omwg.org/TR/d7/ontology/alignment/"/>
<owl:Class rdf:about="Ontology">
```

<rdfs:subClassOf>
<owl:Restriction>
<owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
>1</owl:cardinality>
<owl:onProperty>
<owl:ObjectProperty rdf:about="formalism"/>
</owl:onProperty>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>Ontologies are of this type</rdfs:comment>
</owl:Class>
<owl:Class rdf:about="Step">
<owl:equivalentClass>
<owl:Class>
<owl:unionOf rdf:parseType="Collection">
<owl:Class rdf:about="Property"/>
<owl:Class rdf:about="Relation"/>
</owl:unionOf>
</owl:Class>
</owl:equivalentClass>
</owl:Class>
<owl:Class rdf:about="Property">
<rdfs:subClassOf>
<owl:Class rdf:about="Attribute"/>
</rdfs:subClassOf>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>Relation entities represent properties of the ontology which range is a datatype.</rdfs:comment>
</owl:Class>
<owl:Class rdf:about="Alignment">
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty>
<owl:DatatypeProperty rdf:about="xml"/>
</owl:onProperty>
<owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
>1</owl:maxCardinality>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty>
<owl:DatatypeProperty rdf:about="level"/>
</owl:onProperty>
<owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
>1</owl:maxCardinality>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
>1</owl:maxCardinality>
<owl:onProperty>
<owl:DatatypeProperty rdf:about="method"/>
</owl:onProperty>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty>

```
<owl:DatatypeProperty rdf:about="purpose"/>
</owl:onProperty>
<owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
>1</owl:maxCardinality>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
>1</owl:maxCardinality>
<owl:onProperty>
<owl:DatatypeProperty rdf:about="type"/>
</owl:onProperty>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty>
<owl:ObjectProperty rdf:about="onto1"/>
</owl:onProperty>
<owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
>1</owl:cardinality>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty>
<owl:ObjectProperty rdf:about="onto2"/>
</owl:onProperty>
<owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
>1</owl:cardinality>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
>1</owl:minCardinality>
<owl:onProperty>
<owl:ObjectProperty rdf:about="map"/>
</owl:onProperty>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>Alignment objects represent an alignment between two ontologies.
Each alignment is alos composed of set of cells representing the mapping rules.</rdfs:comment>
</owl:Class>
<owl:Class rdf:about="Value">
<rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
<rdfs:subClassOf>
<owl:Class>
<owl:unionOf rdf:parseType="Collection">
<owl:Restriction>
<owl:onProperty>
<owl:DatatypeProperty rdf:about="stringValue"/>
</owl:onProperty>
<owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
>1</owl:cardinality>
</owl:Restriction>
<owl:Restriction>
<owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
```

>1</owl:cardinality>
<owl:onProperty>
<owl:ObjectProperty rdf:about="instanceValue"/>
</owl:onProperty>
</owl:Restriction>
</owl:unionOf>
</owl:Class>
</rdfs:subClassOf>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>A Value object can be instanciated either with an instanceValue or a stringValue</rdfs:comment>
</owl:Class>
<owl:Class rdf:about="PoV">
<owl:equivalentClass>
<owl:Class>
<owl:unionOf rdf:parseType="Collection">
<owl:Class rdf:about="Path"/>
<owl:Class rdf:about="Value"/>
</owl:unionOf>
</owl:Class>
</owl:equivalentClass>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>A path or a value</rdfs:comment>
</owl:Class>
<owl:Class rdf:about="Instance">
<rdfs:subClassOf>
<owl:Class rdf:about="Entity"/>
</rdfs:subClassOf>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>Instance entities represent instance data of the ontology.</rdfs:comment>
</owl:Class>
<owl:Class rdf:about="Restriction">
<rdfs:subClassOf>
<owl:Restriction>
<owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
>1</owl:cardinality>
<owl:onProperty>
<owl:ObjectProperty rdf:about="onProperty"/>
</owl:onProperty>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty>
<owl:DatatypeProperty rdf:about="value"/>
</owl:onProperty>
<owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
>1</owl:cardinality>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty>
<owl:DatatypeProperty rdf:about="comparator"/>
</owl:onProperty>
<owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
>1</owl:cardinality>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>Restrictions objects are of this type</rdfs:comment>

```
</owl:Class>
<owl:Class rdf:about="Self"/>
<owl:Class rdf:about="Cell">
<rdfs:subClassOf>
<owl:Restriction>
<owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
>1</owl:cardinality>
<owl:onProperty>
<owl:ObjectProperty rdf:about="entity2"/>
</owl:onProperty>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
>1</owl:cardinality>
<owl:onProperty>
<owl:ObjectProperty rdf:about="entity1"/>
</owl:onProperty>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty>
<owl:DatatypeProperty rdf:about="relation"/>
</owl:onProperty>
<owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
>1</owl:maxCardinality>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty>
<owl:DatatypeProperty rdf:about="measure"/>
</owl:onProperty>
<owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
>1</owl:maxCardinality>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>Cells are the objects representing mapping rules</rdfs:comment>
</owl:Class>
<owl:Class rdf:about="Attribute">
<rdfs:subClassOf>
<owl:Class rdf:about="Entity"/>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty>
<owl:ObjectProperty rdf:about="condition"/>
</owl:onProperty>
<owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
>0</owl:minCardinality>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>Attribute enties represent relations between classes.
We distinguish between those having a datatype in their range (Properties)
and those having a class (Relations).</rdfs:comment>
</owl:Class>
```

```
<owl:Class rdf:about="Class">
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>Class entities represent the objects of the ontology.</rdfs:comment>
<rdfs:subClassOf>
<owl:Restriction>
<owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
>0</owl:minCardinality>
<owl:onProperty>
<owl:ObjectProperty rdf:about="condition"/>
</owl:onProperty>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Class rdf:about="Entity"/>
</rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="Collection">
<rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty>
<owl:ObjectProperty rdf:about="item"/>
</owl:onProperty>
<owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
>2</owl:minCardinality>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="Formalism">
<rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
>1</owl:minCardinality>
<owl:onProperty>
<owl:DatatypeProperty rdf:about="name"/>
</owl:onProperty>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
>1</owl:cardinality>
<owl:onProperty>
<owl:DatatypeProperty rdf:about="uri"/>
</owl:onProperty>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="Relation">
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>Relation entities represent properties of the ontology which range is a Class entity.
Only binary relations are yet considered.</rdfs:comment>
<rdfs:subClassOf rdf:resource="Attribute"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
>0</owl:minCardinality>
<owl:onProperty>
<owl:ObjectProperty rdf:about="condition"/>
</owl:onProperty>
```

```
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="Path">
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>Pathes objects represent pathes through the RDF graph in order to reach a particular set of entities.
Example: The set of persons who have a female sibbling having exactly two children.</rdfs:comment>
<rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
<rdfs:subClassOf>
<owl:Class>
<owl:intersectionOf rdf:parseType="Collection">
<owl:Restriction>
<owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
>1</owl:cardinality>
<owl:onProperty>
<owl:ObjectProperty rdf:about="next"/>
</owl:onProperty>
</owl:Restriction>
<owl:Restriction>
<owl:onProperty>
<owl:ObjectProperty rdf:about="first"/>
</owl:onProperty>
<owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
>1</owl:cardinality>
</owl:Restriction>
</owl:intersectionOf>
</owl:Class>
</rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="Entity">
<rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty>
<owl:ObjectProperty rdf:about="operator"/>
</owl:onProperty>
<owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
>1</owl:maxCardinality>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>Entities are ontological entities part of the mapping rules (cells)</rdfs:comment>
</owl:Class>
<owl:ObjectProperty rdf:about="transf">
<rdfs:domain rdf:resource="Attribute"/>
<rdfs:range rdf:resource="PoV"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>specify a transformation on a datatype property (attribute)</rdfs:comment>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="first">
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>Indicates the first element in a path</rdfs:comment>
<rdfs:domain rdf:resource="Path"/>
<rdfs:range rdf:resource="Step"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="formalism">
<rdfs:domain rdf:resource="Ontology"/>
<rdfs:range rdf:resource="Formalism"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>Indicate the formalism (ontology language) of the aligned ontologies</rdfs:comment>
</owl:ObjectProperty>
```

<owl:ObjectProperty rdf:about="or">
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>cunjunction of the entities</rdfs:comment>
<rdfs:subPropertyOf>
<owl:ObjectProperty rdf:about="operator"/>
</rdfs:subPropertyOf>
<rdfs:range rdf:resource="Collection"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="condition">
<rdfs:range rdf:resource="Restriction"/>
<rdfs:domain rdf:resource="Entity"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>Define a condition on an expression in order to restrict its scope.</rdfs:comment>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="onProperty">
<rdfs:domain rdf:resource="Restriction"/>
<rdfs:range>
<owl:Class>
<owl:unionOf rdf:parseType="Collection">
<owl:Class rdf:about="Path"/>
<owl:Class rdf:about="Step"/>
<owl:Class rdf:about="Self"/>
</owl:unionOf>
</owl:Class>
</rdfs:range>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>properties applied to pathes</rdfs:comment>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="onto1">
<rdfs:subPropertyOf>
<owl:ObjectProperty rdf:about="ontology"/>
</rdfs:subPropertyOf>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>First ontology of the alignment</rdfs:comment>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="instanceValue">
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>A value given as an instance entity</rdfs:comment>
<rdfs:range rdf:resource="Instance"/>
<rdfs:domain rdf:resource="Value"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="attributeValueCondition">
<rdfs:domain rdf:resource="Class"/>
<rdfs:subPropertyOf>
<owl:ObjectProperty rdf:about="classCondition"/>
</rdfs:subPropertyOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="entity2">
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>Second entity of a cell</rdfs:comment>
<rdfs:subPropertyOf>
<owl:ObjectProperty rdf:about="entity"/>
</rdfs:subPropertyOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="classCondition">
<rdfs:subPropertyOf rdf:resource="condition"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="operator">
<rdfs:domain rdf:resource="Entity"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>operators to combine entities</rdfs:comment>

```
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="transitive">
<rdfs:domain rdf:resource="Relation"/>
<rdfs:range rdf:resource="Relation"/>
<rdfs:subPropertyOf rdf:resource="operator"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>transitive closure of a relation</rdfs:comment>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="map">
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>Relate an alignment to each of its cells</rdfs:comment>
<rdfs:domain rdf:resource="Alignment"/>
<rdfs:range rdf:resource="Cell"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="and">
<rdfs:subPropertyOf rdf:resource="operator"/>
<rdfs:range rdf:resource="Collection"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>intersection of the entities</rdfs:comment>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="restriction">
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>A restriction specifies a the set of entities that will fulfill the related condition.</rdfs:comment>
<rdfs:range rdf:resource="Restriction"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="attributeOccurenceCondition">
<rdfs:subPropertyOf rdf:resource="classCondition"/>
<rdfs:domain rdf:resource="Class"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="attributeCondition">
<rdfs:subPropertyOf rdf:resource="condition"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="domainRestriction">
<rdfs:range rdf:resource="Class"/>
<rdfs:domain rdf:resource="Attribute"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>Restrict the domain of an Attribute or Relation to the given Class expression</rdfs:comment>
<rdfs:subPropertyOf rdf:resource="attributeCondition"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="symmetric">
<rdfs:subPropertyOf rdf:resource="operator"/>
<rdfs:domain rdf:resource="Relation"/>
<rdfs:range rdf:resource="Relation"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>Symmetric of a relation</rdfs:comment>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="next">
<rdfs:range>
<owl:Class>
<owl:unionOf rdf:parseType="Collection">
<owl:Class rdf:about="Path"/>
<owl:Class rdf:about="Step"/>
<owl:Class rdf:about="Self"/>
</owl:unionOf>
</owl:Class>
</rdfs:range>
<rdfs:domain rdf:resource="Path"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>Indicate the following path of a path, or nil to terminate a path</rdfs:comment>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="ontology">
```

<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>Represent the source and target ontologies of an alignment</rdfs:comment>
<rdfs:domain rdf:resource="Alignment"/>
<rdfs:range rdf:resource="Ontology"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="attributeTypeCondition">
<rdfs:domain rdf:resource="Class"/>
<rdfs:subPropertyOf rdf:resource="classCondition"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="service">
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>Specifies a transformation service</rdfs:comment>
<rdfs:domain rdf:resource="Attribute"/>
<rdfs:range rdf:resource="PoV"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="reflexive">
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>Reflexive of a relation</rdfs:comment>
<rdfs:range rdf:resource="Relation"/>
<rdfs:domain rdf:resource="Relation"/>
<rdfs:subPropertyOf rdf:resource="operator"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="relationDomainRestriction">
<rdfs:domain rdf:resource="Class"/>
<rdfs:range rdf:resource="Relation"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>The class with this condition is restricted
to only those instances in the domain of the given relation.</rdfs:comment>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="onto2">
<rdfs:subPropertyOf rdf:resource="ontology"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>Second ontology of the alignment</rdfs:comment>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="inverse">
<rdfs:subPropertyOf rdf:resource="operator"/>
<rdfs:range rdf:resource="Relation"/>
<rdfs:domain rdf:resource="Relation"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>Inverse of a relation</rdfs:comment>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="relationRangeRestriction">
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>The class with the condition is restricted to only those instances in the range of the given relation.</<rdfs:range
rdf:resource="Relation"/>
<rdfs:domain rdf:resource="Class"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="rangeRestriction">
<rdfs:subPropertyOf rdf:resource="attributeCondition"/>
<rdfs:range rdf:resource="Class"/>
<rdfs:domain rdf:resource="Relation"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>Restrict the range of the attribute to a given Class expression</rdfs:comment>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="item">
<rdfs:domain rdf:resource="Collection"/>
<rdfs:range rdf:resource="Entity"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="not">
<rdfs:subPropertyOf rdf:resource="operator"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"

>Negation of an entity</rdfs:comment>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="entity1">
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>First entity of a cell</rdfs:comment>
<rdfs:subPropertyOf>
<owl:ObjectProperty rdf:about="entity"/>
</rdfs:subPropertyOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="entity">
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>Entities of a Cell</rdfs:comment>
<rdfs:domain rdf:resource="Cell"/>
<rdfs:range rdf:resource="Entity"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:about="stringValue">
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>A value given as a string</rdfs:comment>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="relation">
<rdfs:range>
<owl:DataRange>
<owl:oneOf rdf:parseType="Resource">
<rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>equivalence</rdf:first>
<rdf:rest rdf:parseType="Resource">
<rdf:rest rdf:parseType="Resource">
<rdf:rest rdf:parseType="Resource">
<rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
<rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>disjoint</rdf:first>
</rdf:rest>
<rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>instanceOf</rdf:first>
</rdf:rest>
<rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>subsumption</rdf:first>
</rdf:rest>
</owl:oneOf>
</owl:DataRange>
</rdfs:range>
<rdfs:domain rdf:resource="Cell"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>relation between the aligned entities in a mapping rule.</rdfs:comment>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="value">
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>A value indicate the value compared in a condition.
It can take either the value of an Attribute by following a path, or a direct value.</rdfs:comment>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="typeRestriction">
<rdfs:domain rdf:resource="Property"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="name">
<rdfs:domain rdf:resource="Formalism"/>
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>Name of a formalism or ontology langage</rdfs:comment>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="transUri">
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>URI of a transformation function</rdfs:comment>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="purpose">
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>Describe the purpose of the alignment</rdfs:comment>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="measure">
<rdfs:domain rdf:resource="Cell"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>Measure of confidence in a mapping rule.</rdfs:comment>
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="method">
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>Indicates the method, the name of the tool or algoriithm that outputted the alignment</rdfs:comment>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="type">
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>Indicate the arity of the alignment (Deprecated, always **)</rdfs:comment>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="uri">
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
<rdfs:domain rdf:resource="Formalism"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>uri of a formalism (ontology language)</rdfs:comment>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="level">
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>Indicates the level of the alignment (deprecated?, alway 2)</rdfs:comment>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="comparator">
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>Comparators are used in conditions in order to make comparisons of attribute values.
The value of this property is an URI indicating the XSD operator to be used in the comparison.</rdfs:<rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="xml">
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>Indicate if the alignment is expressed in XML format. (deprecated?)</rdfs:comment>
<rdfs:range>
<owl:DataRange>
<owl:oneOf rdf:parseType="Resource">
<rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>yes</rdf:first>
<rdf:rest rdf:parseType="Resource">
<rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
<rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>no</rdf:first>
</rdf:rest>
</owl:oneOf>
</owl:DataRange>
</rdfs:range>
</owl:DatatypeProperty>
<Entity rdf:about="null-entity">
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"

>This entity is used to terminate a path.</rdfs:comment>
</Entity>
<rdf:Description rdf:about="http://knowledgeweb.semanticweb.org/heterogeneity/alignment">
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>RDF vocabulary for the alignment format</rdfs:comment>
</rdf:Description>
<Path rdf:about="nil">
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>Represent a null path. Used to end a path</rdfs:comment>
<first rdf:resource="null-entity"/>
<next rdf:resource="nil"/>
</Path>
</rdf:RDF>
<!-- Created with Protege (with OWL Plugin 3.4, Build 125) http://protege.stanford.edu -->

Preprocessing Algorithm

The preprocessing algorithm follows

Preprocessing Algorithm

```
Definitions: IS = {is1,…, isn},          // set of all underlying information sources
             M.O                         // Main ontology of the mediator
             L.O                         // Local ontologies
        terms t= {t1…tn}  ∈ IS(i)        //terms of local ontologies –information sources
        Terms T={T1…Tn}∈ M.O             //terms of main ontology
1:      for  each ti ∈ IS(i) do
2:           for  each Ti ∈ M.O do
3:           call map{ti, Ti} ;   // ti, Ti are input arguments of the function map
4:           Mappings= return results from call ;
5:           end for;
6:      end for;
7:      for each is_i  ∈ IS do
8:           mappingTable(is_i) =Mappings (is_i);   // filling mapping sets
9:      end for;
10:     for  mappingTable(is_i) do
11:          call relateDM(ti,Ti) ;
12:          define A(is_i)= return results from call;
13:          A=A+A(is_i);                // definition of set of articulations
14:     end for;
```

Algorithm 1.1 Preprocess algorithm*. It shows all the necessary activities before the operation of the system.*

Notes:

1. *By terms we mean all the possible words or expressions we can meet in a SPARQL query, in the local ontologies of the information sources, in the Main ontology of the mediator and therefore, in the articulations and the mappings. By **t** we imply the set of terms used in local ontologies and by **T** the set of terms in the Main ontology. The user gives his query based on the set of the terms T of the main ontology of the mediator.*
2. *From the set of all mappings between the M.O and the L.Os, we form the mapping set for each information source separately. The **map function** implies the mapping of each term of the M.O with one or more terms of the specific information source and vice versa.*
3. *We use the operator "+" to show the **addition** of a variable or of a set containing individuals of the same type, to a superset.*

4. *The **relateDM** function is an abbreviation for relate domain/range, of each term in an L.O with a term of M.O. This function is similar with **map**, because the articulations are based on the mapping sets.*

SPARQL Evaluation Algorithm

The algorithm of evaluation for simple SPARQL queries follows:

Algorithm Eval (μ: mapping, P: graph pattern, G: RDF graph)

case:

P is a triple pattern t:

if dom(μ) = var(t) and $\mu(t) \in D$ **then return** *true*

return *false*

P is a pattern of the form ($P1$ FILTER R):

if Eval(μ, $P1$, D) = *true* and $\mu \models R$ **then return** *true*

return *false*

P is a pattern of the form ($P1$ UNION $P2$):

if Eval(μ, $P1$, D) = *true* or **Eval**(μ, $P2$, D) = *true* **then return** *true*

return *false*

P is a pattern of the form ($P1$ AND $P2$):

for each pair of mappings $\mu1 \in$ pos($P1$, D) and $\mu2 \in$ pos($P2$, D)

if Eval($\mu1$, $P1$, D) = *true* and **Eval**($\mu2$, $P2$, D) = *true* and $\mu = \mu1 \cup \mu2$ **then**

return *true*

return *false*

P is a pattern of the form ($P1$ OPT $P2$):

if Eval(μ, ($P1$ AND $P2$), D) = *true* **then return** *true*

if Eval(μ, $P1$, D) = *true* **then**

for each mapping $\mu' \in$ pos($P2$, D)

if Eval(μ', $P2$, D) = *true* and μ is compatible with μ' **then return** *false*

return *true*

return *false*

Algorithm 3.2.2.1: Algorithm of evaluating simple SPARQL queries.