ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

FACOLTÀ DI SCIENZE MATEMATICHE FISICHE E NATURALI

Corso di laurea specialistica in Informatica

Materia di tesi : Ingegneria del Software

Semion: un software che implementa un metodo di trasformazione configurabile da database relazionali in linked data

Tesi di laurea di:
Andrea Giovanni Nuzzolese

Relatore:
Chiar.mo Prof. Paolo Ciancarini

Correlatori:
Dott. Aldo Gangemi
Dott.ssa Valentina Presutti

Sessione III

Anno Accademico 2008/2009

Contents

Abstract

Il Web tradizionale è pensato quasi esclusivamente per essere utilizzato e compreso da esseri umani. Infatti, i documenti che esso contiene non permettono la rappresentazione dei dati in modo da garantire anche a delle macchine di poter comprendere e gestire la semantica associata ai dati stessi.
Il problema è lagato al fatto che nel Web si pubblicano documenti espressi in un formalismo (HTML [RLHJ99]) che non può essere compreso da un macchina.
Il Web Semantico, introdotto da Berners-Lee [BLHL01], costituisce un'estensione del Web tradizionale dove i documenti pubblicati sono associati ad informazioni e dati (metadati) che ne specificano il contesto semantico in un formato adatto all'interrogazione, all'interpretazione e, più in generale, all'elaborazione automatica.
Se da un lato il Web Semantico fornisce tecnologie, come RDF [Bec04] ed OWL [HM04], per rappresentare semanticamente i dati in modo che possano essere compresi da macchine, dall'altro occorre prima trasformare sorgenti di dati non-RDF preesistenti in RDF.
Berners-Lee [BL06] ha suggerito un'insieme di buone pratiche che permettono di pubblicare e connettere tra loro nel Semantic Web sorgenti di dati non semantiche. L'utilizzo di queste buone pratiche costituisce il *Linked Data* [BL06], ossia un grafo semantico, in cui i dati sono condivisi e connessi fra loro.
In questo lavoro ci si propone di studiare, progettare ed implementare un tool basato su un metodo che permetta la trasformazione da database relazionali in grafi semantici pubblicabili nel contesto del Linked Data. Il tool si differenzia da una serie di altri tool esistenti, che permettono lo stesso processo di reingegnerizzazione, perchè fornische un

approccio configurabile dall'utente. Infatti, mentre gli altri tool trasformano il database relazionale direttamente in RDF, usando come vocabolario RDF Schema [BG04b], Semion, che è il tool studiato ed implementato, prima traduce il database in triple RDF, non associandogli uno specifica semantica derivante da uno specifico vocabolario, e successivamente permette all'utente di rifattorizzare il set di dati ottenuto ad un serie di possibili ontologie e design pattern. Di conseguenza, la semantica associata al set di dati generato dal database, non è necessariamente quella derivata dal metamodello di RDF (RDF Schema), ma quella derivata dallo specifico metamodello scelto per la rifattorizzazione (per esempio LMM [PGG08]), che può essere scelto in base allo specifico dominio del database stesso.
La trasformazione in triple RDF del database è guidata da un metamodello OWL che è stato implementato per descrivere semanticamente lo schema e i dati di database relazionali, mentre la rifattorizzazione è resa possibile da un ragionatore che riceve in input un insieme di regole di allineamento ad altre ontologie o design pattern definite dall'utente.
Il tool, in futuro, sarà esteso per gestire la trasformazione da data source eterogenei, in quanto il metodo non è limitato esclusivamente a database relazionali.

Chapter 1

Introduction

This chapter wants to be a short presentation about the traditional Web, the Semantic Web [BLHL01] and the Linked Data [BL06] that are the basic concepts to know in order to fully understand the motivations and the goals that are behind the project related to this thesis.
In chapter 2 it will be given an overview about the state of the art in Linked Data and about the existent tools that can be related to Semion, that is the software designed and implemented in this thesis.
Then, in chapter 3 is provided a description of the Semion design in its two main components that are realized by a reengineer, that transforms non-RDF data sources into RDF, and a refactor module, that aligns RDF models to other models thought as vocabularies or ontology design patterns.
In chapter 4 it is described the implentation of what was designed in chapter 3. So it is explained how the reengineer and the refactor modules were implemented.
Chapter 5 contains the evaluation of the software. This was performed by transforming three different types of data sources into RDF/OWL, namely a relational database, PGN [Wikc] files and LaTex [Lam86] documents with the skak package. The relational database was a MySQL [WA02] version of the Princeton WordNet [Mil95], while both PGN files and LaTex documents contained information about chess games.
Chapter 6 contains the conclusion and the future work.

Finally in appendix A are listed and explained some engineering detalis of the software implemented.

1.1 A concise presentation of the web architecture

In 1989 Tim Berners-Lee, an independent contractor at the European Organization for Nuclear Research (CERN) of Genevre in Switzerland, wrote a proposal for "a large scale hypertext database with typed links". In that document, referencing a previous project called ENQUIRE, he defined a system designed to help the researchers to share documents of various projects in order to prevent the so called loss of information in the "web" of the CERN organization [BL89].

Despite the scarce interest generated by the proposal Berners-Lee carried on the project finding in Robert Cailliau an enthusiastic collaborator and, by the end of 1990, they developed all the necessary tools for a working Web:

- the HyperText Transfer Protocol (HTTP)
- the HyperText Markup Language (HTML) [RLHJ99]
- the first prototype of Web Browser
- the first Web server

The turning point for the World Wide Web was the introduction of the Mosaic web browser in 1993, a graphical browser developed by a team at the National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana-Champaign (UIUC), led by Marc Andreessen.

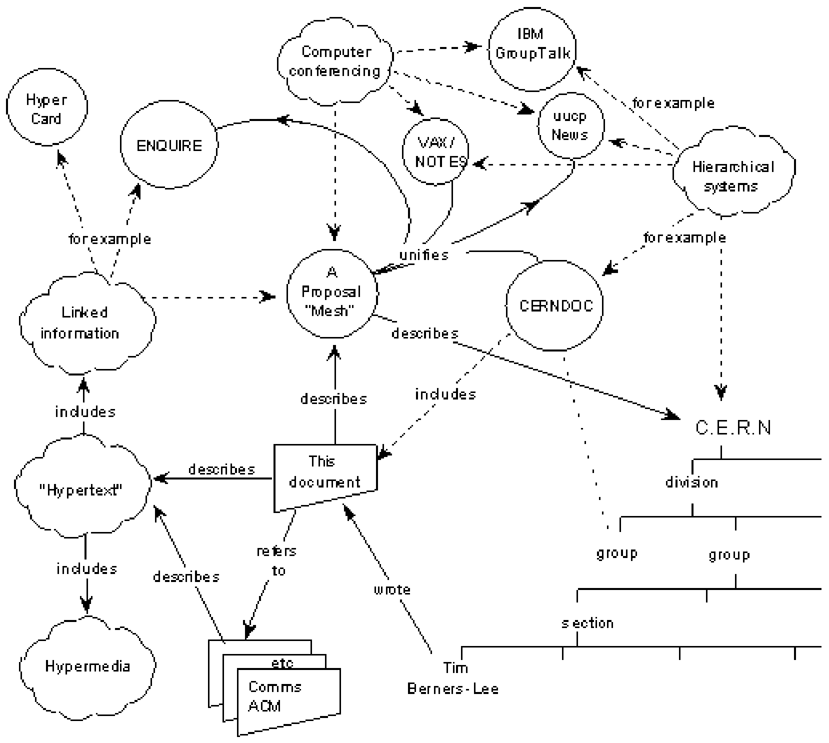


Figure 1.1: The scheme of the Berners-Lee's Proposal
[BL89]

The figure 1.1 is the first Web schema projected by Berners-Lee in order to manage on-line documents at CERN.

Passing all the other milestones in the Web history it could be stated that nowadays the Web is composed by billions of hypertext documents written in HTML. The great success of the Web is due to the extreme simplicity of the HTML language that enables almost anyone to write a document, as it is fully humane-readable and comprehensible. Moreover we cannot ignore that the web is an Internet service that allows publishing and sharing multimedia contents. HTML is just the visible part of a more complex and structured architecture. Figure 1.2 shows a modern web-based enterprise application with four layers:

- a client layer which renders web pages

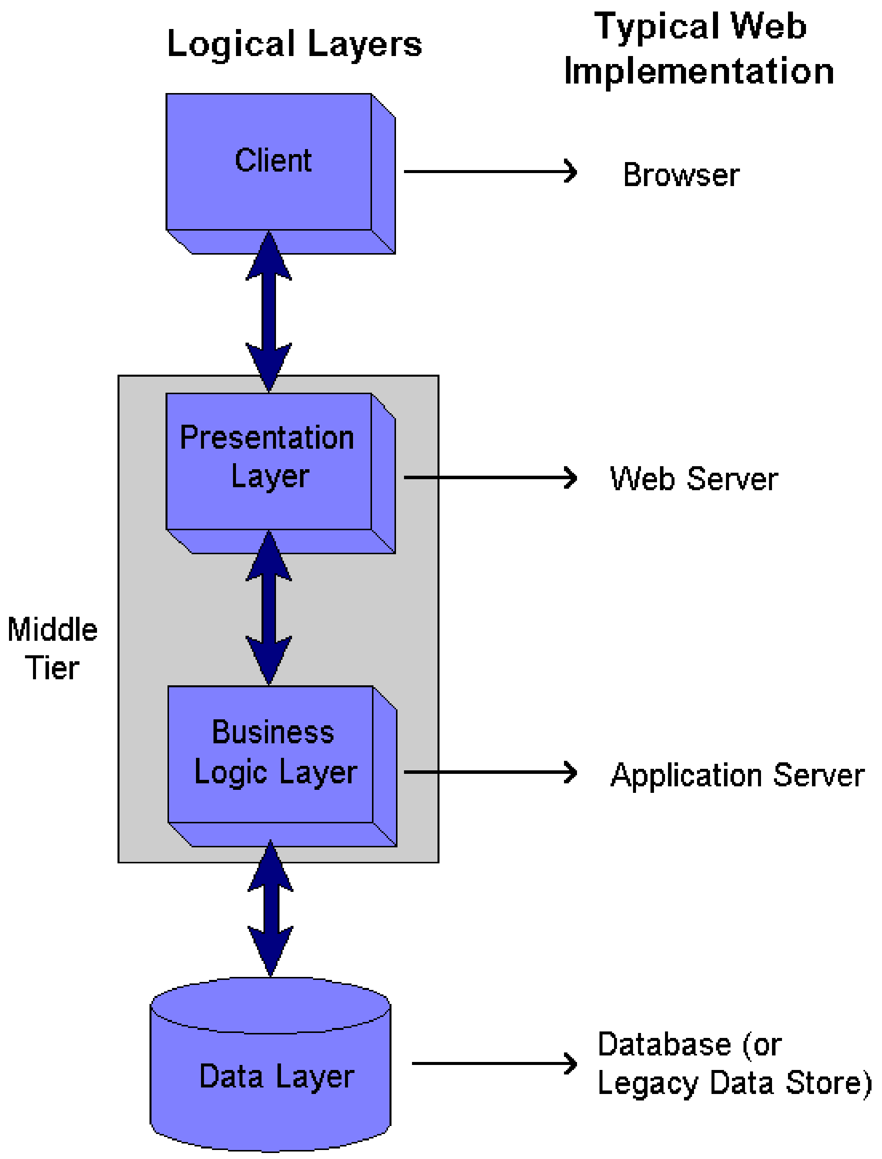


Figure 1.2: Architecture for an Enterprise Web-based Application

- a middle tier which includes:
  - a Presentation Layer which generates web pages, including their dynamic content. It interprets web pages submitted from the client

  - a Business Logic Layer which enforces validations and which handles interaction with the database

- a data layer which stores data between transactions

It is important to remark that all the information is contained in the data layer. This can be any possible data source such as a relational database, an XML [SMYM$^+$08] module, etc..., so that the user can access the information after the middle tier has transformed into a valid content for a Web browser, generally HTML. Of course the transformation performed by the middle tier and the architecture itself are completely transparent to the user so that it seems that the Web page performs every interaction.

1.2 The Semantic Web

HTML played the main role in making the Web popular thanks to its simplicity for publishing documents in the net and creating connection between documents through hyperlinks.
Although HTML's contribution in the Web growth, it cares only for structural and presentational (with CSS) aspects, while ignoring completely the meaning of the data it represents. As a consequence, the traditional Web of documents is almost completely human-oriented and much less machine-oriented, because it is not possible to state anything about the content and data hidden in HTML documents.
For this reason in 1996 Berners-Lee [BLHL01] introduced the *Semantic Web*, a dream for the Web in which computers become capable of analyzing all the data on the Web, the contents, the links, and the transactions between people and computers.
The Semantic Web brings structure to the meaningful content of Web pages, creating an environment where software agents roaming from page to page can readily carry out sophisticated tasks for users. Such an agent coming to the clinic's Web page will know not just that the page has keywords such as "treatment, medicine, physical, therapy" (as might be encoded in HTML) but also that, for example, Dr. Hartman works at this clinic on Mondays, Wednesdays and Fridays and

that the script takes a date range in yyyy-mm-dd format and returns appointment times. The Semantic Web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation.

The semantic web comprises the standards XML, XML Schema, RDF, RDF Schema and OWL that are organized in the Semantic Web Stack, as shown in figure 1.3, were:

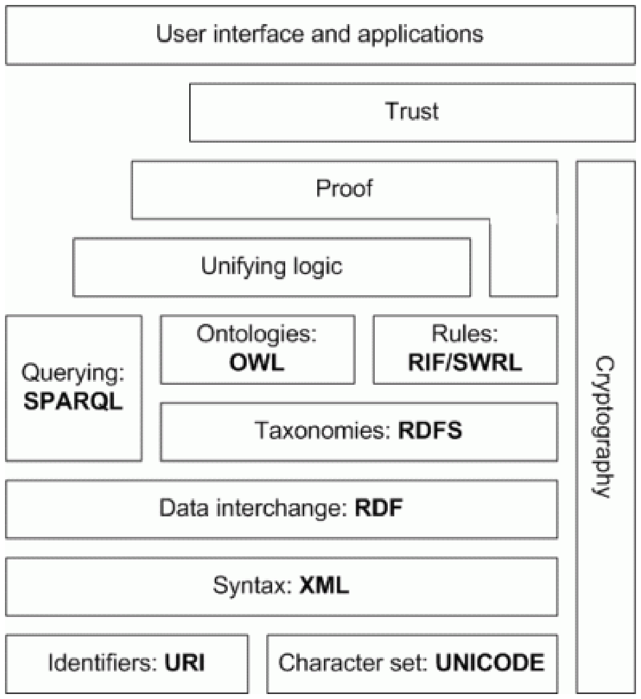


Figure 1.3: The Semantic Web stack.
[Wikd]

- **XML** provides an elemental syntax for content structure within documents, yet associates no semantics with the meaning of the content contained within;
- **XML Schema** [WF04] is a language for providing and restricting the structure and content of elements contained within XML documents;

- **RDF** [Bec04] is a simple language for expressing data models, which refer to objects ("resources") and their relationships. An RDF-based model can be represented in XML syntax;

- **RDF Schema** [BG04b] is a vocabulary for describing properties and classes of RDF-based resources, with semantics for generalized-hierarchies of such properties and classes;

- **OWL** [HM04] adds more vocabulary for describing properties and classes: among others, relations between classes (e.g. disjointness), cardinality (e.g. "exactly one"), equality, richer typing of properties, characteristics of properties (e.g. symmetry), and enumerated classes;

- **SPARQL** [PS08] is a protocol and query language for semantic web data sources.

In the next sections we will give a short description of RDF, OWL and SPARQL.

1.2.1 The Resource Description Framework: RDF

Although it is often called a "language", RDF is essentially a data model. Its basic building block is an object-attribute-value triple, called a *statement*. For example, an RDF triple can state that two people, A and B, both identified by a URI, are related by the fact that A knows B. Similarly an RDF triple may relate a person C to a scientific article D in a bibliographic database by stating that C is the author of D. Two resources linked in this way can be drawn from different data sets on the Web, allowing data in one data source to be linked to that in another, thereby creating a Web of Data. Consequently it is possible to think of RDF triples that link items in different data sets as analogous to the hypertext links that tie together the Web of documents.
The fundamental concepts of RDF are:

- **Resources** can be any kind of things we want to talk about in our domain. Resources may be authors, books, publishers, places, people, hotels, rooms, search queries, and so on [AH08]. Every resource has a an URI, an Universal Resource Identifier.

- **Properties** are a special kind of resources which describe relations between resources, (e.g. written by, age, title, and so on). Properties in RDF are also identified by URIs. This idea of using URIs to identify "things" and the relations between is quite important. This choice gives us in one stroke a global, worldwide, unique naming scheme. The use of such a scheme greatly reduces the homonym problem that has plagued distributed data representation until now

- **Statements** assert the properties of resources. A statement is an object-attribute-value triple, consisting of a resource, a property, and a value. Values can either be resources or literals. Literals are atomic values (strings), the structure of which we do not discuss further.

RDF is an universal language that lets users to describe resources using their own vocabularies. RDF does not make assumptions about any particular application domain, nor does it define the semantics of any domain. Is it up to the user to do so in RDF Schema (RDFS) [BG04b]. The main concepts of RDFS are *classes* and *properties*. A class can be thought of as a set of elements. Individual objects that belong to a class are referred to as instances of that class by using the *rdf:type* property. An important use of classes is to impose restrictions on what can be stated in an RDF document using the schema. In programming languages, typing is used to prevent nonsense from being written (such as A+1, where A is an array; we lay down that the arguments of + must be numbers). The same is needed in RDF. After all, we would like to disallow statements such as:

- Discrete Mathematics is taught by Concrete Mathematics.

- Room 16 is taught by Andrea Nuzzolese.

Once we have classes, we would also like to establish relationships between them. For example, supposing that we have classes for *staff members*, *assistant professors*, *academic staff members*, *administrative staff members*, *professors*, *technical support staff members* and *associate professors*, we can design our RDFS in order to obtain the hierarchy as the figure 1.4 shows.

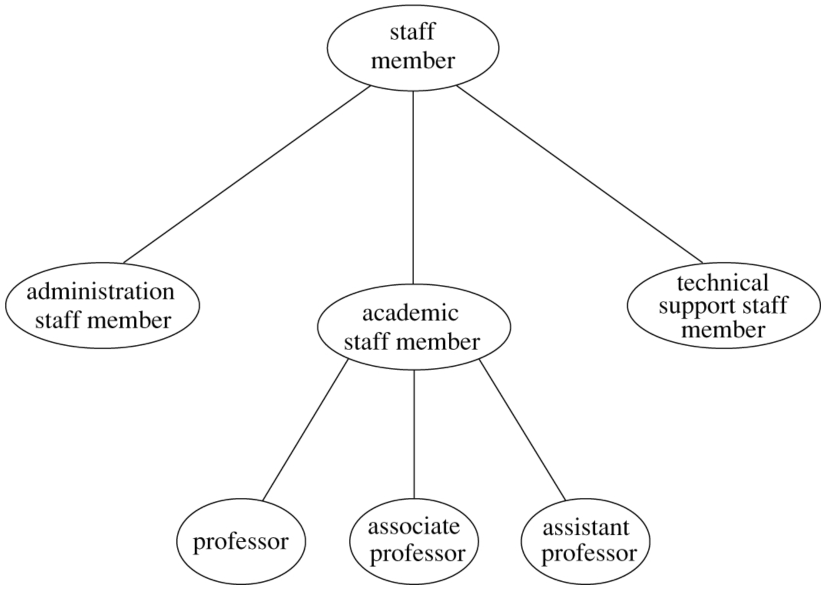


Figure 1.4: A hierarchy of classes. [AH08]

As for classes it was possible to define hierarchical relationships, the same can be done for properties. For example, “is taught by” is a *subproperty* of “involves”. If a course C is taught by an academic staff member A, then C also involves A. The converse is not necessarily true. For example, A may be the convener of the course, or a tutor who marks student homework but does not teach C. In general, P is a subproperty of Q if $Q(x, y)$ whenever $P(x, y)$.

1.2.2 The Ontology Web Language: OWL

The expressivity of RDF and RDF Schema is deliberately very limited as RDF is limited to binary ground predicates, and RDF Schema is limited to subclass and subproperty hierarchies, with domain and range definitions of these properties. Then, if machines are expected to perform useful reasoning tasks on these documents, the language must go beyond the basic semantics of RDF Schema. Thus it is needed an ontology language that is richer than RDF Schema and may offer these features [AH08]:

- Local scope of properties. *rdfs:range* defines the range of a property, say eats, for all classes. Thus in RDF Schema it is not possible to declare range restrictions that apply to some classes only. For example, it is not possible to assert that cows eat only plants, while other animals may eat meat, too.

- Disjointness of classes. Sometimes it is needed to say that classes are disjoint. For example, $male$ and $female$ are disjoint. But in RDF Schema only subclass relationships are allowed, (e.g. female is a subclass of person).

- Boolean combinations of classes. Sometimes it is needed to build new classes by combining other classes using union, intersection, and complement. An example is to define the class $person$ to be the disjoint union of the classes $male$ and $female$. RDF Schema does not allow such definitions.

- Cardinality restrictions. Sometimes it is needed to place restrictions on how many distinct values a property may or must take. For example, a person has exactly two parents, or a course is taught by at least one lecturer. Again, such restrictions are impossible to express in RDF Schema.

- Special characteristics of properties. Sometimes it is useful to say that a property is *transitive* (like "greater than"), *unique* (like "is mother of"), or the *inverse* of another property (like "eats" and "is eaten by").

OWL provides three increasingly expressive sublanguages designed for use by specific communities of implementers and users. They are:

- **OWL Full** is meant for users who need 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 pre-defined (RDF or OWL) vocabulary. It is unlikely that any reasoning software will be able to support complete reasoning for every feature of OWL Full [AH08].

- **OWL DL** supports those users who want the maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time). OWL DL includes all OWL language constructs, but they can be used only under certain restrictions (for example, while a class may be a subclass of many classes, a class cannot be an instance of another class). OWL DL is so named due to its correspondence with description logics, a field of research that has studied the logics that form the formal foundation of OWL [AH08].

- **OWL Lite** supports those users primarily needing a classification hierarchy and simple constraints. For example, while it supports cardinality constraints, it only permits cardinality values of 0 or 1. It should be simpler to provide tool support for OWL Lite than its more expressive relatives, and OWL Lite provides a quick migration path for thesauri and other taxonomies. OWL Lite also has a lower formal complexity than OWL DL [AH08].

Further details regarding the language and its abstract and concrete syntaxes can be found in [HM04].

1.2.3 The RDF query language: SPARQL

RDF models, expressed in the triple format seen in section 1.2.2, form directed, labeled graphs, that can be used to express information in the Web.
SPARQL [PS08], whose name is an acronym that stands for SPARQL Protocol and RDF Query Language, is an RDF query language that allows to users to write queries in order to retrieve the information they need from data source expressed as RDF graphs. The results of SPARQL queries can be result sets or RDF graphs.
For example with the following query

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?email
WHERE {
  ?person a foaf:Person.
  ?person foaf:name ?name.
  ?person foaf:mbox ?email.
}
```

it is possible to retrieve all the pairs of *(name, email)* of any possible person in the world from any possible data source expressed in RDF by using the FOAF [BM] vocabulary.

Most forms of SPARQL queries contain a set of triple patterns called a basic graph pattern. Triple patterns are like RDF triples except that each of the subject, predicate and object may be a variable, that are indicated by a "?" or "$" prefix($?person$, $?name$ and $?email$ in the example). A basic graph pattern matches a subgraph of the RDF data when RDF terms from that subgraph may be substituted for the variables and the result is an RDF graph equivalent to the subgraph.

SPARQL has two main query forms. The $SELECT$ query form returns variable bindings. The $CONSTRUCT$ query form returns an RDF graph. The graph is built based on a template which is used to generate RDF triples based on the results of matching the graph

pattern of the query.
For example, the following query [PS08]

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
PREFIX org:    <http://example.com/ns#>

CONSTRUCT { ?x foaf:name ?name }
WHERE  { ?x org:employeeName ?name }
```

produces the RDF triples below serialized in RDF/XML syntax.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:foaf="http://xmlns.com/foaf/0.1/">
  <rdf:Description>
    <foaf:name>Alice</foaf:name>
  </rdf:Description>
  <rdf:Description>
    <foaf:name>Bob</foaf:name>
  </rdf:Description>
</rdf:RDF>
```

Restrictions on the result sets can be realized by the construction of queries that contain the SPARQL $FILTER$ construct.
The following query retrieves the set of any title book with its price assuming that the latter is less than 30.5.

```
PREFIX  dc:  <http://purl.org/dc/elements/1.1/>
PREFIX  ns:  <http://example.org/ns#>
SELECT  ?title ?price
WHERE   { ?x ns:price ?price .
          FILTER (?price < 30.5)
          ?x dc:title ?title . }
```

1.3 Linked Data

Though the Semantic Web technologies are sufficiently good to add semantics to data in the Web, a lot of legacy and non-RDF data sources have to be transformed into RDF. Linked Data [BL06] is about using the Web to connect related data that was not previously linked, or using the Web to lower the barriers to linking data currently linked using other methods.

1.3.1 The problem of data silos

Currently the Web is almost completely human-oriented and much less machine-oriented so that building services for accessing and managing information from different and heterogeneous data sources may be a quite difficult task. The reason is not in the tiered architecture itself, but in what it produces. HTML pages generate a Web of documents in which machines are not really welcome as data are completely hidden in the documents. Furthermore, the hyperlinks just represent existing relations among documents but not what kind and who is the subject and the object of the relations so that they are meaningless for machines.

The sources of data used for feeding the "traditional" Web are like "silos" with no links to each other. Each contains different information, may be based on different systems and potentially may belong to different companies, but they may also contain homogeneous information and belong to the same company. The issue is that, in both cases, all the data is really hidden by the HTML layer and no machine-to-machine communication is made possible.
What has been just described is a strong limitation of the existing Web as, for instance, it does not allow to any program, or better, to any agent to retrieve data from silos for querying and reasoning on data in order to reply to user requests. The final goal is to transform a Web of documents into a Web of data, in which silos may be also physically heterogeneous and far, but virtually they are homogeneous

and interconnected by machine-comprehensible relations.

1.3.2 Exposing, sharing, and connecting data in the Web

Traditionally, data published on the Web has been made available as raw dumps in formats such as CSV [Lam86] or XML [SMYM$^+$08], or marked up as HTML tables, sacrificing much of its structure and semantics. In the conventional hypertext Web, the nature of the relationship between two linked documents is implicit, as the data format (e.g. HTML) is not sufficiently expressive to enable individual entities described in a particular document to be connected by typed links to related entities. [BHBL09]
However, in recent years, the Web has evolved from a global information space of linked documents to one where both documents and data are linked. Underpinning this evolution is a set of best practices for publishing and connecting structured data on the Web known as Linked Data.
Berners-Lee [BL06] outlined a set of "rules" for publishing data on the Web in a way that all published data becomes part of a single global data space:

- Use URIs [BLFM05] as names for things
- Use HTTP URIs so that people can look up those names
- When someone looks up a URI, provide useful information, using the standards (RDF [Bec04], SPARQL [PS08])
- Include links to other URIs, so that they can discover more things

These have become known as the "Linked Data principles", and provide a basic recipe for publishing and connecting data using the infrastructure of the Web while adhering to its architecture and standards. There is a W3C project named *Linking Open Data* [W3C] that is the

most visible example of adoption and application of the Linked Data principles. The original and ongoing aim of the project is to bootstrap the Web of Data by identifying existing data sets that are available under open licenses, converting these to RDF according to the Linked Data principles, and publishing them on the Web.
An indication of the range and scale of the Web of Data originating from the Linking Open Data project is provided in figure 1.5 Each

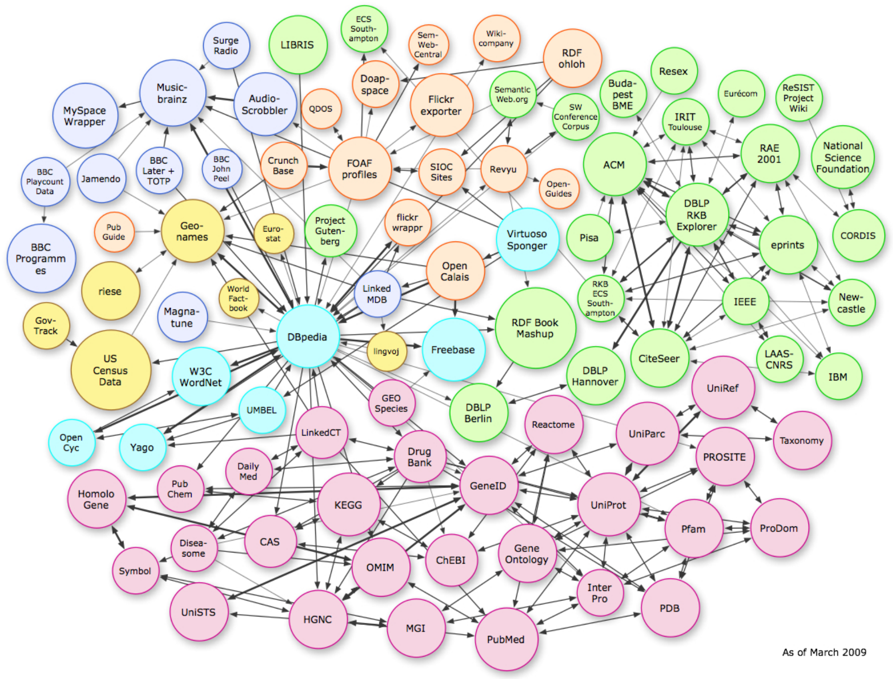


Figure 1.5: Linking Open Data cloud diagram giving an overview of published data sets and their interlinkage relationships. [BHBL09]

node in this cloud diagram represents a distinct data set published as Linked Data, as of March 2009. The arcs in figure 1.5 indicate that links exist between items in the two connected data sets. Heavier arcs roughly correspond to a greater number of links between two data sets, while bidirectional arcs indicate the outward links to the other exist in each data set.
The technology that is critical to the Web of Data is RDF (Resource Description Framework). While HTML provides a means to structure and link documents on the Web, RDF provides a generic, graph-based

data model with which to structure and link data that describes things in the world.

1.4 Motivation and objectives

The aim of this thesis is to develop a tool for transforming relational databases into Linked Data that can be extended at supporting other types of transformations, based on a theoretical framework for data sources reengineering called “Semion”.
At the state of the art, as it will be exposed in the next chapter, other existing tools allow the “semantic lifting” (the reengineering from a non-RDF source to an RDF one). But, if on one hand, they could be good to obtain RDF datasets from legacy systems, on the other, their reengineering is driven by the RDF metamodel, namely RDF Schema. This fact introduces some collateral problems related to the semantics of RDF Schema non well adapted to that one of the data source schema that has to be lifted.
For this reason the tool, that will be projected, implements a methodology that is based on two main steps:

1. the reengineering to RDF triples following the metamodel derived by the non-RDF data source (e.g. an RDF vocabulary of a relational database schema).
2. the refactoring of the dataset obtained in 1. to any possible vocabulary or design pattern that better adds the desired semantics to data.

The semantic lifting of the tool that will be studied differs from that one of existing tools because the latter add semantics to data automatically and in a single step ignoring domain specific issues and the quality of the input data source (e.g. a non well projected relational database), while it first extract pure RDF triples and then, with a customized and user driven approach, add semantics to data.
The metamodel that is want to be used as the target of the refactoring

is the Linguistic Meta-Model (LMM) [PGG08], that is an OWL-DL implementation of the Peirce semiotic triangle [Pei31], and that allows a semiotic-cognitive representation of knowledge. The reason of this choice has to be found in the fact that LMM is able to support the representation of different knowledge sources developed according to different underlying semiotic theories. This is possible because most knowledge representation schemata, either formal or informal, can be put into the context of the semiotic triangle.

Though LMM is the target metamodel for refactoring, the tool that will be analyzed is projected to perform refactoring according to other metamodel, vocabulary or ontologies in a way that allows the user to customize and to direct the semantic lifting process.

Chapter 2

Opening the data silos

Linked Data is about using the Web to connect related data that was not previously linked, or using the Web to lower the barriers to linking data currently linked using methods of the Semantic Web. In summary, Linked Data is simply about using the Web to create typed links between data from different sources.

2.1 Publishing heterogeneous data sources in RDF Linked Data

The basic tenets of Linked Data are:

- to use the RDF data model to publish structured data on the Web
- to use RDF links to interlink data from different data sources

Applying both principles leads to the creation of a data commons on the Web, a space where people and organizations can post and consume data about anything. This data commons is often called the Web of Data or Semantic Web.

To publish data on the Web, it is first needed to identify the items of interest in the domain as resources identified using Uniform Resource Identifiers (URIs) in a way they could be dereferenced and fetched through the HTTP protocol. Then it is possible to organize these resources in triples using RDF statements.
The main benefits of using the RDF data model in a Linked Data context are that:

- Clients can look up every URI in an RDF graph over the Web to retrieve additional information.
- Information from different sources merges naturally.
- The data model enables you to set RDF links between data from different sources.
- The data model allows you to represent information that is expressed using different schemata in a single model.
- Combined with schema languages such as RDF Schema or OWL, the data model allows you to use as much or as little structure as you need, meaning that you can represent tightly structured data as well as semi-structured data.

RDF links enable Linked Data browsers and crawlers to navigate between data sources and to discover additional data. These links can be added manually or generated with pattern-based algorithms such as it happens in the RDF Book Mash-up [BCG07], which for instance uses the URI *http://www4.wiwiss.fu-berlin.de/bookmashup/books/0747581088* to identify the book "Harry Potter and the Half-blood Prince". Having the ISBN number in these URIs made it easy for DBpedia to generate *owl:sameAs* links between books within DBpedia and the Book Mash-up.

2.2 The state of art

There are some tools that make possible to expose and publish non-RDF legacy data sources into Linked Data. During the translation process they assume as meta-model and so as vocabulary RDF Schema.

2.2.1 Meta-model driven approaches

Generally, taking the relational databases domain as an example, these tools translate:

- database tables into RDF resources having *rdfs:class* as type
- table columns into RDF resources having rdf:property as type

Among the others we can cite:

- **D2R Server** [BC07], that is a tool for publishing relational databases on the Semantic Web. It enables RDF and HTML browsers to navigate the content of the database, and allows applications to query the database using the SPARQL query language. D2R Server uses the D2RQ Mapping Language to map the content of a relational database to RDF. A D2RQ mapping specifies how resources are identified and which properties are used to describe the resources.
- **Talis Platform**, that provides Linked Data-compliant hosting for content and RDF data. Data held in the platform is organized into “stores” which can be individually secured if need be. Any kind of content can be added to a store along with arbitrary RDF metadata. The content and metadata becomes immediately accessible over the Web and discoverable using both SPARQL and a free text search system with built in ranking of results according to relevance to the search terms.

- **Triplify**, that is a small plug-in written in PHP for Web applications, which reveals the semantic structures encoded in relational databases by making database content available as RDF, JSON or Linked Data.

- **Virtuoso**, that is a middleware and database engine hybrid that combines the functionality of a traditional RDBMS, ORDBMS, virtual database, RDF, XML, free-text, web application server and file server functionality in a single system. Rather than have dedicated servers for each of the aforementioned functionality realms, Virtuoso is a "universal server"; it enables a single multithreaded server process that implements multiple protocols.

2.2.2 Limitations of meta-model driven approaches

In the previous section it was said that the meta model driven approaches add typing informations to the generated RDF resources generally using, as vocabulary, RDF Schema.
For some aspects this assumption could be seen as a pattern choice, but for others, it introduces some problems in terms of ontology engineering. Consider a database table *Purchase* that realizes the relation between the tables *Product* and *Customer*. In such meta model driven approach the table *Purchase* would be an *rdfs:Class*, but this conclusion may be wrong because *Purchase* is a table that expresses a relation and it is better translated as an *rdf:Property*.
Moreover, it would be better to have no type assertions in the data after the reengineering process towards Linked Data, but just triples and then modelling these triples in order to align them to some ontology design patterns or vocabulary, also reusing knowledge organization schemata. After the triples are organized using patterns it is possible to assert that a certain resource is an *owl:Class* instead of an *rdfs:class*, or also an *owl:ObjectProperty* instead of owl:DataProperty or *owl:Class* itself.
In few words, the desirable goal is to add semantics not in a once, but after having modelling and structured the domain of interest and data using the good engineering practices of the ontology design patterns.

Chapter 3

Semion: design

In this chapter it will be described the design of the Semion tool. First they will be analyzed the problems, that drove to design Semion, related to their solutions. Then it will be introduced to the Linguistic Meta-Model (LMM) [PGG08], that is a metamodel that enables a semiotic-cognitive representation of knowledge that is important because most knowledge representation schemata, either formal or informal, can be put into the context of so-called semiotic triangle of which LMM is an OWL-DL implementation. Finally, Semion will be described and designed.

3.1 Motivations

Motivations can be represented as a couple composed by problems, that currently exist in the Web, and solutions that are proposed in order to solve problems.

3.1.1 Problems

In section 1.1.2 it was introduced the problem of *data silos* consisting in a world of data sources hidden behind the Web of documents, that is, as it was said, a Web from humans to humans and in which machines are not really welcome. The scenario is even more complex because these data sources are not homogeneous and adding interoperability between them is a very difficult task.

The scheme in figure 3.1 is useful to analyze this scenario. In

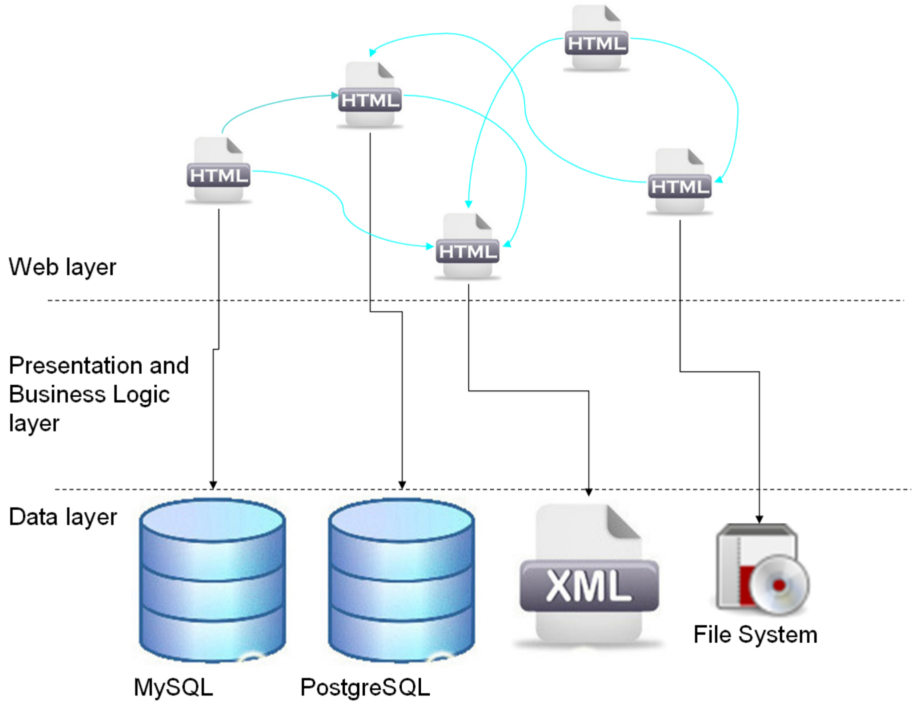


Figure 3.1: Web of documents: a summary scenario

fact, at the top of the figure there is the Web layer in which it lays HTML [RLHJ99] documents connected among them through hyperlinks. These documents may contain data retrieved from the data layer through the presentation and business logic layer that transforms data into HTML contents. It is easy to realize that no connection and communication at the data layer is possible because data are heterogeneous in their type (e.g. relational databases, XML [SMYM+08]

data sources, file systems, etc.) and in their inner architecture (e.g. MySQL databases, PostgreSQL databases, different DTDs or XML Schemata [WF04], NTFS file systems, ext3 file systems, etc.).
Only connection at the Web layer are allowed through HTML hyperlinks, but, as said in chapter 1, hyperlinks are just anchor relations between HTML documents and they do not care about data.
In chapter 2 we introduced Linked Data, a set of best practices for publishing and connecting structured data on the Web. With Linked Data the problem of connecting and sharing heterogeneous data sources is quite solved, because data are translated into RDF/OWL models that have a formal semantics to describe the data they represent and the relations among these data.
With the Linked Data best practices a new problem arise and it concerns the quality and the design of these RDF/OWL models obtained by the translation of non-RDF data sources. In fact, although these practices describe how to represent data using the Semantic Web technologies, they do not care how these data sources have to be translated into RDF triples. To be more concrete they do not care the quality of the translation.

3.1.2 Goals

The objective of the thesis is to design and implement a method to transform heterogeneous data sources into Semantic Web models following the principles of Linked Data.
The method should provide techniques in order to give the possibility to reuse knowledge organization schemata, formally ontology design patterns, that ensure high quality RDF data sets, that are the output of the transformation. Concerning the last feature, it is important to point out that a good method should be able to transform heterogeneous data sources allowing interoperability among them and interoperability with already existing Semantic Web applications. Practically, it is not required simply a transformation, but a transformation that introduces multilinguality among created and existing data sets.

3.1.3 Solutions

In order to satisfy the goals, it was designed Semion, a software that implements the method introduced above, or more formally, a customized method for transforming relational databases into Linked Data. Due to complexity and time issues the domain of the transformation was fixed only to relational databases, but the methodology has to be intended usable for any structured data source.
The transformation can be seen as a pipeline composed by two main modules:

- a reengineering module, that translates relational databases into RDF triples
- a refactoring module, that performs alignments to specific pattern or vocabulary

The second component is very important to the aims of the thesis, because it provides a method to align datasets to specific ontologies thought as patterns or vocabulary, among them the Linguistic Meta-Model (LMM) [PGG08] has the ability to support the representation of different knowledge sources, developed according to different schemata, as they were a single one. This is possible because most knowledge representation schemata, either formal or informal, can be put into the context of so-called semiotic triangle [Pei31], illustrated in chapter 3, of which LMM is an OWL-DL implementation.

3.2 The Linguistic MetaModel: LMM

Following the intuition of the semiotics, in [PGG08] was developed the Linguistic Meta-Model (LMM), an OWL-DL ontology, that in its core component called LMM1 (figure 3.2) formalizes the distinctions of the semiotic triangle introduced by Peirce [Pei31]. Figure 3.2 shows the semiotic triangle implemented by LMM, and it is composed by:

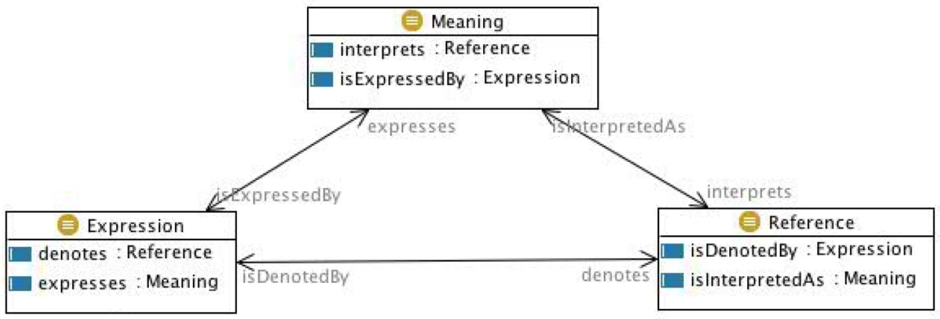


Figure 3.2: The semiotic triangle in LMM1
[PGG08]

- **Reference** could be populated by any possible individual in the logical world, being it either a concrete object or any other social object whose existence is stipulated by a community. Individuals are related by the fact that they co-occur into events. As shown in figure 3.3, LMM specializes the class of Reference distinguishing among:
  - *Physical objects* that allows to talk of artifacts in a very general sense. The concept is derived from DOLCE [GGM$^+$02]
  - *Individual references* can have members that are individual references and they are typically Named Entities
  - *Multiple references* can have instances that are collective individuals, whose members have a superclass in MultiReference.
  - *Situations* is the circumstantial context where entities and events occur. This is a very important class, because it can belong either to the class *Reference* or the class *Meaning*.
- **Meaning** is the class of instances of concepts that could be related to each other by subsumption relations, that organize them into hierarchies of subclasses (e.g. the dog is an animal), or by descriptions expressing the possibility for events to occur. The class *Meaning*, as shown in figure 3.4, is specialized by:
  - *Description* that, as just said, can be thought can as a "descriptive context" that uses or defines concepts in order

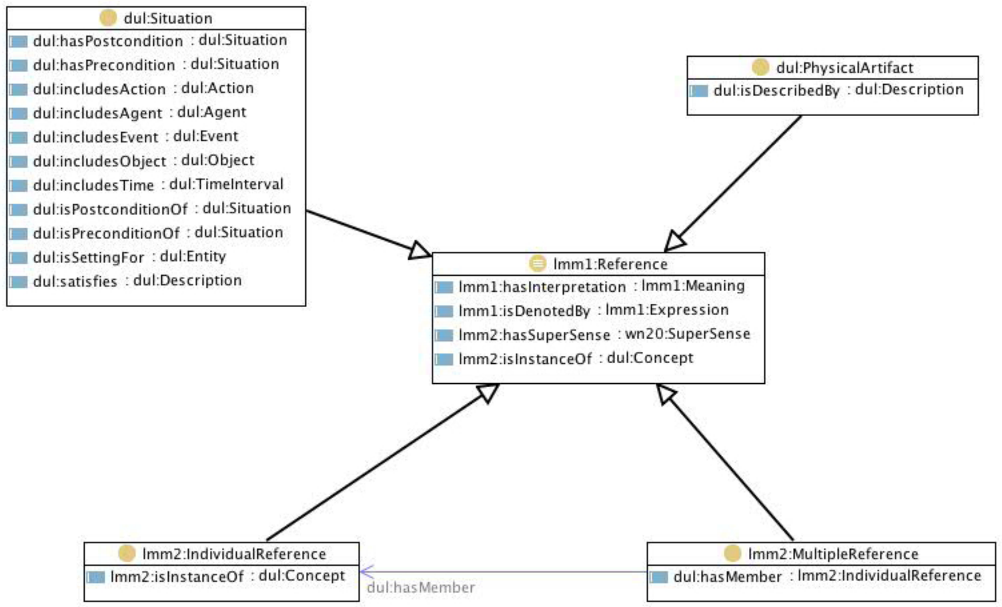

dul:Situation
dul:hasPostcondition : dul:Situation
dul:hasPrecondition : dul:Situation
dul:includesAction : dul:Action
dul:includesAgent : dul:Agent
dul:includesEvent : dul:Event
dul:includesObject : dul:Object
dul:includesTime : dul:TimeInterval
dul:isPostconditionOf : dul:Situation
dul:isPreconditionOf : dul:Situation
dul:isSettingFor : dul:Entity
dul:satisfies : dul:Description
dul:PhysicalArtifact
dul:isDescribedBy : dul:Description
lmm1:Reference
lmm1:hasInterpretation : lmm1:Meaning
lmm1:isDenotedBy : lmm1:Expression
lmm2:hasSuperSense : wn20:SuperSense
lmm2:isInstanceOf : dul:Concept
lmm2:IndividualReference
lmm2:isInstanceOf : dul:Concept
dul:hasMember
lmm2:MultipleReference
dul:hasMember : lmm2:IndividualReference

Figure 3.3: Hiearachy of the class Reference
[PGG08]

to create a view on a "relational context" out of a set of data or observations.

- *Collection* has as main task to give a unique coherent term to the class Description by means of the main relation isUnifiedBy. It can be thought as any container for entities that share one or more common properties.
- *Situation* is the realization of a certain description. It is very important to point out that the this class serves as a bridge between the class *Meaning* and the class *Reference* passing through the class *Description*
- *Concept* can be used in other descriptions by means of the main relation isConceptUsedIn.

- **Expression** is the class including any social object produced by an agent in the context of communicative acts, so they are natural language terms, symbols in formal languages, icons, and whatever can be used as a vehicle for communication.

As said for the *Reference* and *Meaning* classes also the Expression one is specialized by (figure 3.5):

– *Concept Expression* denotes multireferences (cf. lmm2:MultipleReference). A concept expression is a a term that expresses a Meaning, and denotes a MultipleReference, and examples are "Dog", "Black box". Concept expression can be composed by single or multiple words.

– *Name* denotes either named entities (cf. lmm2:NamedEntity) or collective references (cf. lmm2:ExtensionalReference). A name is a proper noun that denotes an IndividualReference, be it singular or plural and examples are "John Zorn", "Daimler Benz", "FaceBook" (as a community).

– *Contextual Expression* denotes either contextual references (cf. lmm2:ContextualReference, or collective references (cf.

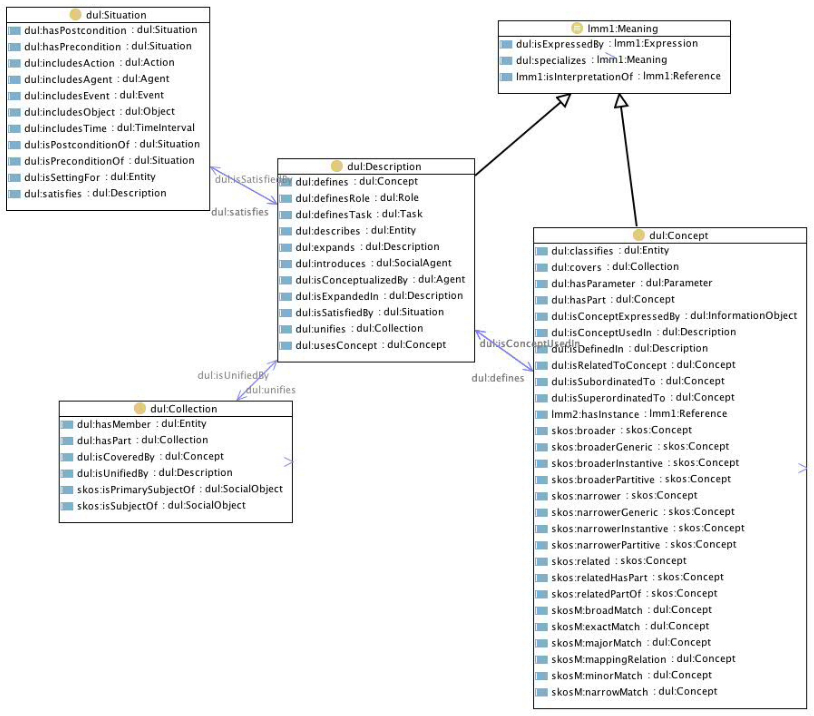

dul:Situation
dul:hasPostcondition : dul:Situation
dul:hasPrecondition : dul:Situation
dul:includesAction : dul:Action
dul:includesAgent : dul:Agent
dul:includesEvent : dul:Event
dul:includesObject : dul:Object
dul:includesTime : dul:TimeInterval
dul:isPostconditionOf : dul:Situation
dul:isPreconditionOf : dul:Situation
dul:isSettingFor : dul:Entity
dul:satisfies : dul:Description
lmm1:Meaning
dul:isExpressedBy : lmm1:Expression
dul:specializes : lmm1:Meaning
lmm1:isInterpretationOf : lmm1:Reference
dul:isSatisfiedBy
dul:satisfies
dul:Description
dul:defines : dul:Concept
dul:definesRole : dul:Role
dul:definesTask : dul:Task
dul:describes : dul:Entity
dul:expands : dul:Description
dul:introduces : dul:SocialAgent
dul:isConceptualizedBy : dul:Agent
dul:isExpandedIn : dul:Description
dul:isSatisfiedBy : dul:Situation
dul:unifies : dul:Collection
dul:usesConcept : dul:Concept
dul:Concept
dul:classifies : dul:Entity
dul:covers : dul:Collection
dul:hasParameter : dul:Parameter
dul:hasPart : dul:Concept
dul:isConceptExpressedBy : dul:InformationObject
dul:isConceptUsedIn : dul:Description
dul:isDefinedIn : dul:Description
dul:isRelatedToConcept : dul:Concept
dul:isSubordinatedTo : dul:Concept
dul:isSuperordinatedTo : dul:Concept
lmm2:hasInstance : lmm1:Reference
skos:broader : skos:Concept
skos:broaderGeneric : skos:Concept
skos:broaderInstantive : skos:Concept
skos:broaderPartitive : skos:Concept
skos:narrower : skos:Concept
skos:narrowerGeneric : skos:Concept
skos:narrowerInstantive : skos:Concept
skos:narrowerPartitive : skos:Concept
skos:related : skos:Concept
skos:relatedHasPart : skos:Concept
skos:relatedPartOf : skos:Concept
skosM:broadMatch : dul:Concept
skosM:exactMatch : dul:Concept
skosM:majorMatch : dul:Concept
skosM:mappingRelation : dul:Concept
skosM:minorMatch : dul:Concept
skosM:narrowMatch : dul:Concept
dul:isConceptUsedIn
dul:defines
dul:isUnifiedBy
dul:unifies
dul:Collection
dul:hasMember : dul:Entity
dul:hasPart : dul:Collection
dul:isCoveredBy : dul:Concept
dul:isUnifiedBy : dul:Description
skos:isPrimarySubjectOf : dul:SocialObject
skos:isSubjectOf : dul:SocialObject

Figure 3.4: Hiearachy of the class Meaning [PGG08]

lmm2:Ex- tensionalReference). A contextual expression is a a term that denotes a reference via anaphora or deixis and examples are “the dog over there”, “all my family”, “the current ACME employees”, “the lion described above”.

3.3 Software description

Semion is projected to be a stand alone modular tool and it is extensible to manage different data sources by providing plug-in packages. Its core is composed by three main and independent components:

- the **Reengineer**, that is responsible for the reengineering of the data from non-RDF and legacy data sources into RDF triples. It is an independent module, able to run alone. It is important to say that the reengineering process and the reengineering choises are kept divided, because the first one is an hardcoded part of

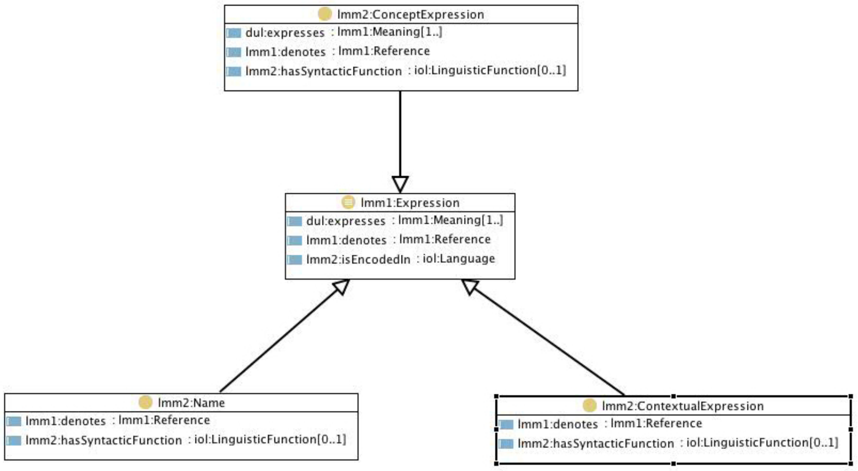

lmm2:ConceptExpression
dul:expresses : lmm1:Meaning[1..]
lmm1:denotes : lmm1:Reference
lmm2:hasSyntacticFunction : iol:LinguisticFunction[0..1]
lmm1:Expression
dul:expresses : lmm1:Meaning[1..]
lmm1:denotes : lmm1:Reference
lmm2:isEncodedIn : iol:Language
lmm2:Name
lmm1:denotes : lmm1:Reference
lmm2:hasSyntacticFunction : iol:LinguisticFunction[0..1]
lmm2:ContextualExpression
lmm1:denotes : lmm1:Reference
lmm2:hasSyntacticFunction : iol:LinguisticFunction[0..1]

Figure 3.5: Hiearachy of the class Expression
[PGG08]

the module, while the seconds are the result of user choices and are represented using RDF/OWL. In the next sections we are going to explore this subject.

- the **Refactor**, that is responsible for the alignments from reengineered data sets to the meta models are wanted to use.
  Alignments consist in transforming an RDF dataset in a new RDF dataset, formally they are refactoring. The module is needed to comply to selected vocabularies, that are meant to be customizable ontology content patterns or standard ontologies. These alignments are performed by reasoning on sets of rules expressed in SWRL [HPSB$^+$04] syntax, that is a Semantic Web Rule Language based on a combination of the OWL-DL and OWL Lite sublanguages of the OWL Web Ontology Language with the Unary/Binary Datalog RuleML [Rul] sublanguages of the Rule Markup Language.

- the **Graphical User Interface (GUI)** has the role of supporting interactions between the software and the users. The intent is to design a full stand-alone software, a program that runs as a separate computer process, not an add-on of an existing process (e.g. not a plug-in). This is the reason why the GUI is built up using *Widgets* ("Widget" is short for "window gadget") that are basic visual building blocks which, combined in an application, hold all the data processed by the application and the available interactions on this data.

3.4 Software architecture

The following subsections will explore the requirements of the rengineer and the refactor modules.

3.4.1 Reengineer

The basic idea about the features that the reengineer must implement can be given by figure 3.6. In fact, it should simply be able to read data from any possible data source and then translate them into RDF data sets.

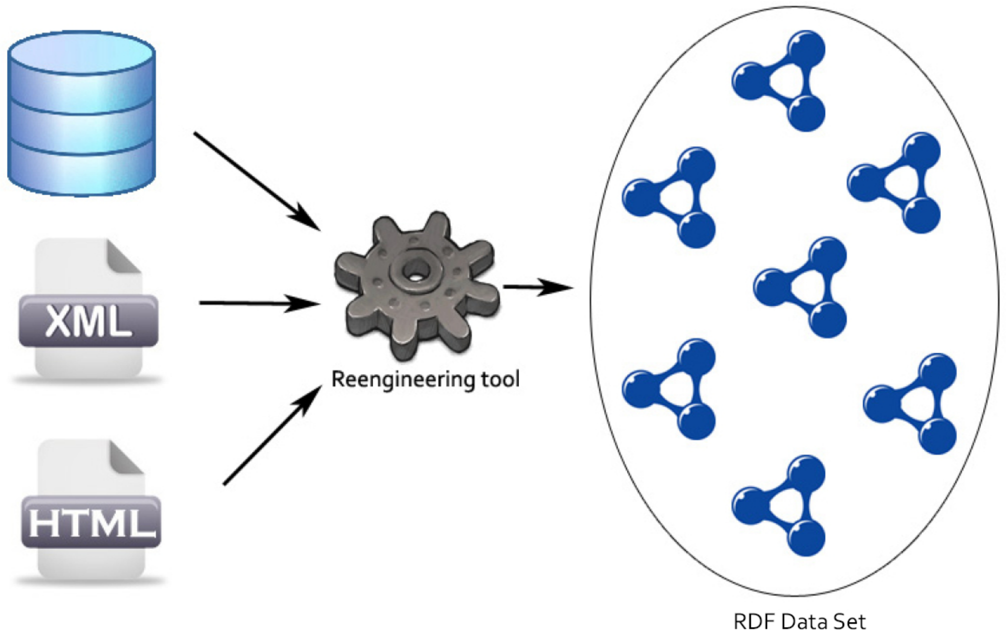


Figure 3.6: Scheme of a possible reengineering tool

What is not immediately clear analyzing the figure 3.6 is that the reengineering is not completely hardcoded, because, even if the translation process from data source contents to RDF data sets is transparent to users, its definition is not, as it is the user to suggest to the software what to translate and how to translate it. This is possible using RDF or OWL models to describe the data source schema, the data schema, the mapping between the former two and the RDFs that are wanted as output and, last but not less important, the mapping vocabulary.
Users write their own RDF or OWL model to tell Semion how to reengineer data sources and they can do that in three different ways:

- using the Semion GUI to write reengineering models. It is fully graphical oriented and integrated in the software;
- using the default models that Semion provide them. It is allowed to users to start from default models and eventually modify them within the GUI;
- using external tools for the Semantic Web (e.g. Protégé [KFNM04], TopBraid Composer [Top], etc.) or common editors to write reengineering models that will be imported in Semion.

This feature allows a customized reengineering process of high level, that is crucial if it is wanted to translate non-trivial data sources in which automated or hardcoded methods could return undesirable outputs, or, worst, they could fail. About what it has been just said, it is possible to consider a database with a non well-projected schema in which, for example, no foreign keys are identified. As result it is expected an RDF data set that does not contain useful information about those entries for which foreign keys would be needed to express relations with other entries.

As the customization is ensured by RDF or OWL models that describe the data source they want to reengineer, then in figure 3.7 is shown the hierarchy and the function of these models.

Starting from the top of the figure, there is:

- **L5** that contains the database mapping meta model and is formally the vocabulary of our mapping between non-RDF data sources to RDF ones.
- **L4/L3b** that contains the mapping model itself and uses L5 to describe it (at this layer there are the instances of the vocabulary we chose). It could possibly be put together with L3 so it is explained why it has been indicated also as L3b.
- **L3/L3a** that contains the data model.
- **L2** that is the RDF or OWL representation of the data source schema (e.g. a database schema).

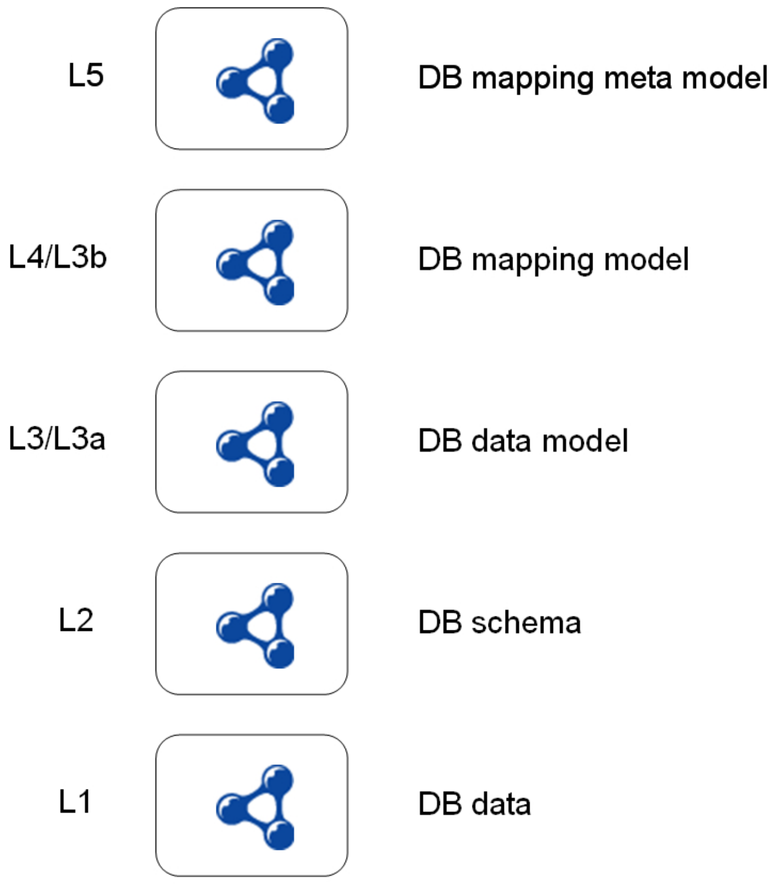


Figure 3.7: Hierarchy of models for the database mapping and translation

- **L1** that contains the data of the input data source translated into and RDF or OWL format.

Considering these layers, the informal architecture in figure 3.6 should be modified in the one in figure 3.8. It is possible to observe how the former reengineer tool is now put in a context in which the Graphical User Interface is the broker of each interaction between users and the tool. Moreover the core of all the reengineering process are the five layers just explained.

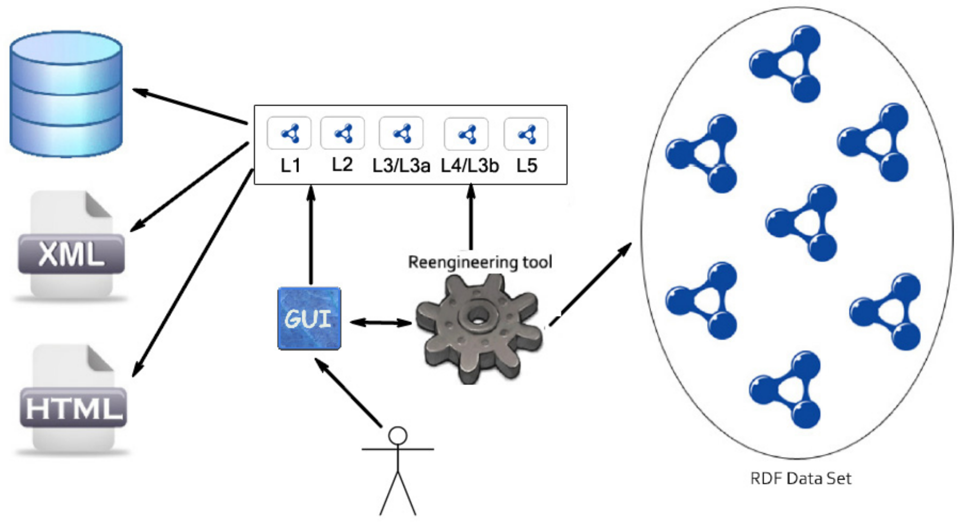
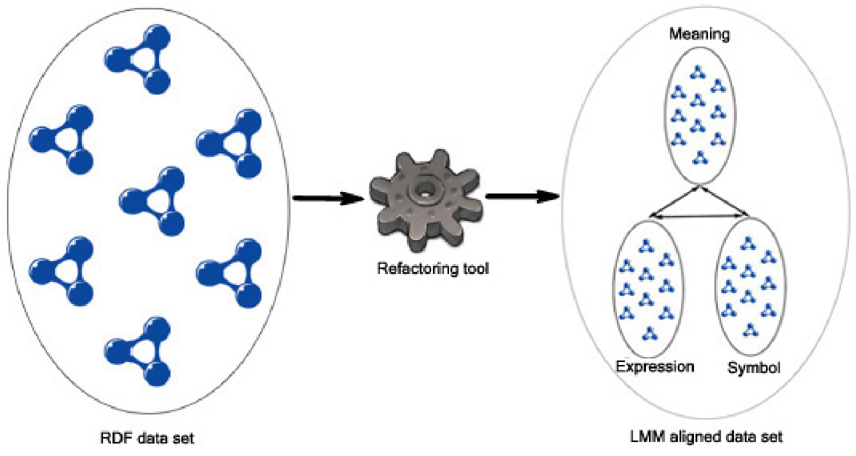


Figure 3.8: Scheme of a possible reengineering tool

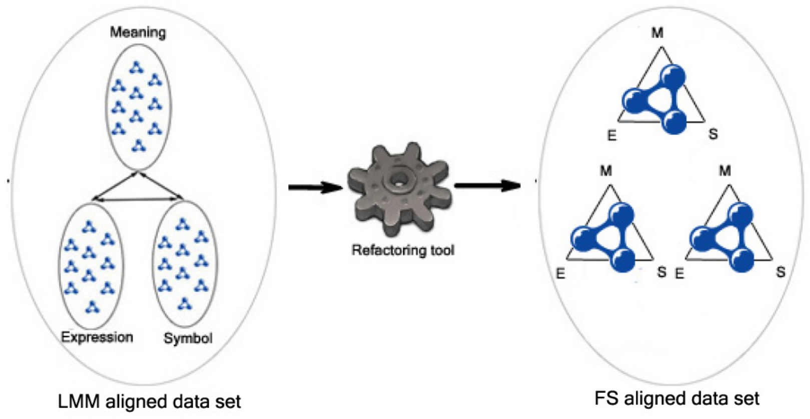
3.4.2 Refactor

Once the starting data source is reengineered into RDF data sets, the next step is to align these data sets to content patterns or standard vocabulary. This module should be able not only to perform alignments from datasets to content patterns or standard vocabulary but also from the lattern to any other ontology that is useful to add semantics to data. To this purpose, it is useful to say tha an user may want to project his refactoring in order to align the dataset to the LMM meta model, then, the result of the alignment, to the FormalSemantics meta model, that contains the vocabulary to express the formal semantics, and finally grounding the latter in a logic language that can be expressed by the OWL vocabulary itself.
In figure 4.11 is given the scheme of a possible refactoring tool. As it is possible to note, the figure is divided into two diagrams:

- in figure 3.9(a) is synthetically shown the refactoring process about the alignment of an RDF data set, obtained by the reengineering process, to the LMM meta model
- in figure 3.9(b), instead, is shown the process of the alignment



(a)



(b)

Figure 3.9: Scheme of the refactoring tool.

of the data set, that is the output of the previous point, to the FormalSemantics meta model

The aims involved in the analysis and in the design of the refactoring module are centred around the will to project a software module with the following features:

- an high cohesion level of its inner sub-components that ensures that the latter to work together towards the same goal: refactoring RDF data sets
- a low coupling level with other software components that ensures a full independent module and the desirable goal of a reusable software component
- the possibility to customize the refactoring process, in order to satisfy user requirements. Furthermore, the refactoring choices must be represented in a formalism that ensures the integration with existing Semantic Web applications and that is not hard-coded

The first two point require a thorough knowledge of the code and of the engineering choices, so they will be clearer after reading the next chapter. Instead, the third point concerns quite exclusively the formalism used to represent users choices in the refactoring process. So the questions are:

1. What should users represent as their own customization choices regarding the refactoring process?
2. What is the formalism to represent these choices?

The answer to the first question is simple because users should write **rules** to represent their refactoring choices. Rules allow both backward-chaining and forward-chaining reasoning [RN03]. The backward-chaining is an inference method used in automated theorem provers, proof assistants and other artificial intelligence applications. It is one of the two most commonly used methods of reasoning with inference rules and logical implications. Backward chaining is implemented in logic programming by SLD resolution. [RN03]

Backward chaining starts with a list of goals (or a hypothesis) and works backwards from the consequent to the antecedent to see if there is data available that will support any of these consequents. An inference engine using backward chaining would search the inference rules until it finds one which has a consequent (*Then* clause) that matches a desired goal. If the antecedent (*If* clause) of that rule is not known to be true, then it is added to the list of goals (in order for one's goal to be confirmed one must also provide data that confirms this new rule). For example, suppose that the goal is to conclude the color of my pet Fritz, given that he croaks and eats flies, and that the rule base contains the following four rules:

1. *If* **X** croaks and eats flies *Then* **X** is a frog
2. *If* **X** chirps and sings *Then* **X** is a canary
3. *If* **X** is a frog *Then* **X** is green
4. *If* **X** is a canary *Then* **X** is yellow

This rule base would be searched and the third and fourth rules would be selected, because their consequents (*Then* Fritz is green, *Then* Fritz is yellow) match the goal (to determine Fritz's color). It is not yet known that Fritz is a frog, so both the antecedents (*If* Fritz is a frog, *If* Fritz is a canary) are added to the goal list. The rule base is again searched and this time the first two rules are selected, because their consequents (*Then* **X** is a frog, *Then* **X** is a canary) match the new goals that were just added to the list. The antecedent (*If* Fritz croaks and eats flies) is known to be true and therefore it can be concluded that Fritz is a frog, and not a canary. The goal of determining Fritz's color is now achieved (Fritz is green if he is a frog, and yellow if he is a canary, but he is a frog since he croaks and eats flies; therefore, Fritz is green).

Contrariwise forward chaining starts with the available data and uses inference rules to extract more data (from an end user for example) until a goal is reached. An inference engine using forward chaining searches the inference rules until it finds one where the antecedent (*If* clause) is known to be true. When found it can conclude, or infer, the consequent (*Then* clause), resulting in the addition of new information to its data.
Considering the previous rule base, to satisfy the goal the first rule would be selected, because its antecedent (*If* Fritz croaks and eats flies) matches our data. Now the consequents (*Then* **X** is a frog) is added to the data. The rule base is again searched and this time the third rule is selected, because its antecedent (*If* Fritz is a frog) matches our data that was just confirmed. Now the new consequent (*Then* Fritz is green) is added to our data. Nothing more can be inferred from this information, but we have now accomplished our goal of determining the color of Fritz.

It seems obvious that different rule bases could be written in different syntaxes for different reasoners, but it is needed to be far from concrete reasoners, and, at the same time, it is needed to keep the expressivity of their rule languages. This introduces the answer to the second question. So what it is needed is a rule language that could work accordin with the Semantic Web principles and it is SWRL [Top]
The Semantic Web Rule Language (SWRL) is a proposal for a Semantic Web rules-language, combining sublanguages of the OWL Web Ontology Language (OWL DL and Lite) with those of the Rule Markup Language (RuleML) [Rul], that is a markup language developed to express both forward (bottom-up) and backward (top-down) rules in XML for deduction, rewriting, and further inferential-transformational tasks.
A possible example is the following rule:

$$hasParent(?x1, ?x2) \wedge hasBrother(?x2, ?x3) \Rightarrow hasUncle(?x1, ?x3)$$

This rule say that if an individual $x1$ has an individual $x2$ as parent and $x2$ has another individual $x3$ as brother then $x1$ has $x3$ as uncle. The rule in SWRL expressed with an RDF concrete syntax become:

```
<swrl:Variable rdf:ID="x1"/>
```

```
<swrl:Variable rdf:ID="x2"/>
<swrl:Variable rdf:ID="x3"/>
<ruleml:Imp>
  <ruleml:body rdf:parseType="Collection">
    <swrl:IndividualPropertyAtom>
      <swrl:propertyPredicate rdf:resource="&eg;hasParent"/>
      <swrl:argument1 rdf:resource="#x1" />
      <swrl:argument2 rdf:resource="#x2" />
    </swrl:IndividualPropertyAtom>
    <swrl:IndividualPropertyAtom>
      <swrl:propertyPredicate rdf:resource="&eg;hasBrother"/>
      <swrl:argument1 rdf:resource="#x2" />
      <swrl:argument2 rdf:resource="#x3" />
    </swrl:IndividualPropertyAtom>
  </ruleml:body>
  <ruleml:head rdf:parseType="Collection">
    <swrl:IndividualPropertyAtom>
      <swrl:propertyPredicate rdf:resource="&eg;hasUncle"/>
      <swrl:argument1 rdf:resource="#x1" />
      <swrl:argument2 rdf:resource="#x3" />
    </swrl:IndividualPropertyAtom>
  </ruleml:head>
</ruleml:Imp>
```

Following the just given guide lines it is possible to figure out a refactoring tool like that one in figure 3.10.

In the figure are clear the interactions between the user and the graphical interface that is the bridge for the user to the application. Through the GUI the user write his rules that are parsed and executed by the reasoner that is part of the refactoring tool.

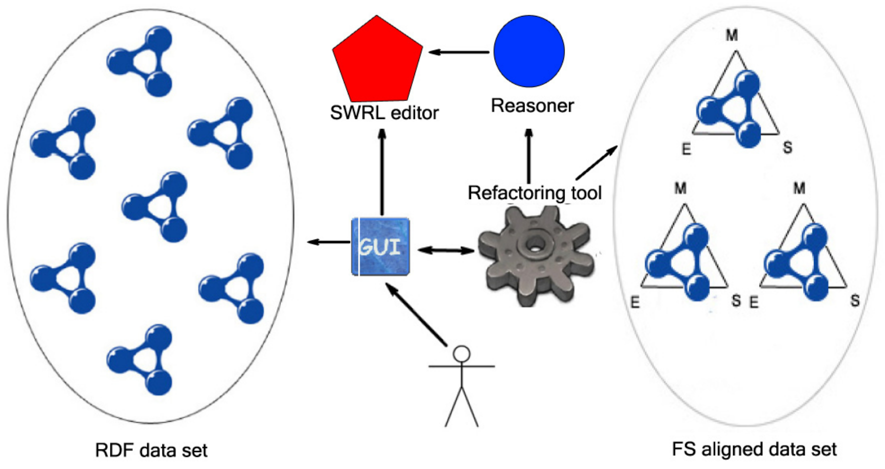


Figure 3.10: Scheme of a possible refactoring tool

Chapter 4

Semion: implementation

As said, Semion is a tool for transforming relational databases into Linked Data based on a customized method. The next sections will give an implementation of Semion.

4.1 Environment and development tools

Semion is a tool implemented completely in Java [GJSB05]. Its Graphical User Interface is developed using the Standard Widget Toolkit (SWT) [Eclb] and JFace [Ecla]. SWT is an open source widget toolkit for Java designed to provide efficient, portable access to the user-interface facilities of the operating systems on which it is implemented. JFace is a UI toolkit with classes for handling many common UI programming tasks. JFace is window-system-independent in both its API and implementation, and is designed to work with SWT without hiding it. JFace includes the usual UI toolkit components of image and font registries, text, dialog, preference and wizard frameworks, and progress reporting for long running operations. Two interesting features are actions and viewers. The action mechanism allows user commands to be defined independently from their exact whereabouts in the UI. Viewers are model based adapters for certain SWT widgets, simplifying the presentation of application data structured as lists, tables or trees.

All components that manage RDF [Bec04] or OWL [HM04] documents are implemented with Jena [Sou], that is a Java framework for building Semantic Web applications. It provides a programmatic environment for RDF, RDFS [BG04b] and OWL, SPARQL [PS08] 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 [BL98] and N-Triples [BG04a]
- An OWL API
- In-memory and persistent storage
- SPARQL query engine

Alignments are made possible thanks to the Jena reasoner and to Pellet, that is an open-source Java based OWL DL reasoner based on the tableaux algorithms [BS01] developed for expressive Description Logics. It supports the full expressivity OWL DL including reasoning about nominals (enumerated classes).
Rules for alignments can be written in the following syntaxes:

- SemionRule
- SWRL [HPSB+04]

The software development environment we used is Eclipse 3.5.1 (Galileo), that is is written primarily in Java and can be used to develop applications in Java and, by means of the various plug-ins, in other languages as well.

4.2 Treating Non-RDF databases as RDF graphs

The reengineering process and so the transformation from relational databases into RDF can be divided into the OWL model, that semantically describes the database and the transformation, and the Java code, that performs the reengineering. In the next two subsections these features will be explained.

4.2.1 RDF customized mapping of non-RDF sources

As discussed in section 2.2, existing tools for reengineering data from relational databases (RDB) into RDF follow approaches based on the direct mapping between RDB and RDFS, for instance a possible translation schema could be:

- A RDB table maps to a RDFS Class.
- A column name of a RDB table maps to a RDFS Property.
- A RDB table record maps to a RDF node.
- A cell of a RDB table maps to a value.

Though this translation pattern can generally work for mapping RDB to RDF, it is strongly related to the quality of the RDB.
In fact, assuming a low-quality RDB with few domain semantic relationships at the level of schema, like $Andrea \rightarrow lives\_in \rightarrow Bologna$, the result of the translation will be an RDF model without or with a few relations among its resources, but, contrariwise, only or a lot of literals that do not express so much but the strings or the integers that they represent.
Also the assumption to map directly RDB to RDFS is a strong limitation as it is wanted to add the semantics to data step by step reusing existing meta-model and vocabularies that allows an ontology design

patten approach and, where possible, to relate entities extracted into RDF from the database with that ones from other ontologies making sense to Linked Data.
To fulfill these problems in this chapter it is wanted to give give a concrete implementation to the method discussed in chapter 3.

First of all it was projected *semionDB.owl*, an ontology that is thought to be the vocabulary to represent relational databases into an RDF format. It was obtained by specializing two well known content ontology design patterns:

- *collectionentity.owl* whose intent is to represent domain (not set theory) membership
- *objectrole.owl* whose intent is to represent objects and the role they play

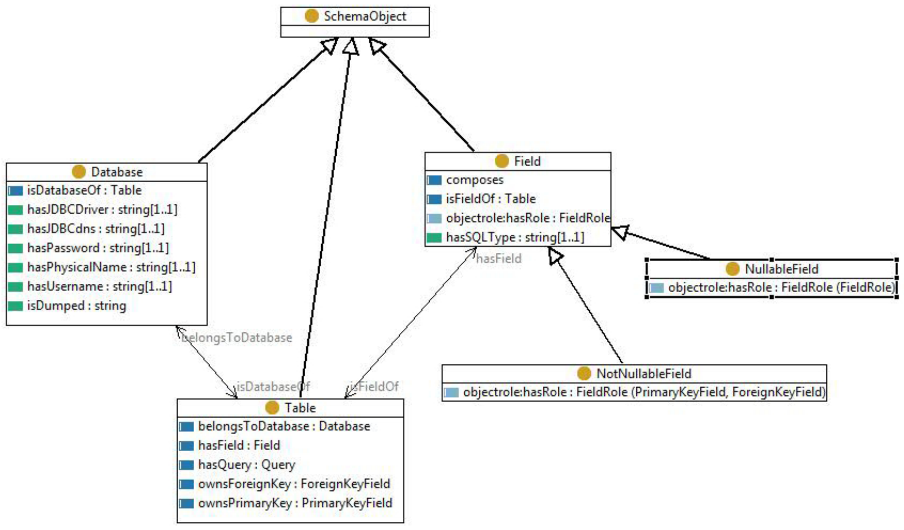

SchemaObject
Database
isDatabaseOf : Table
hasJDBCDriver : string[1..1]
hasJDBCdns : string[1..1]
hasPassword : string[1..1]
hasPhysicalName : string[1..1]
hasUsername : string[1..1]
isDumped : string
Field
composes
isFieldOf : Table
objectrole:hasRole : FieldRole
hasSQLType : string[1..1]
NullableField
objectrole:hasRole : FieldRole (FieldRole)
NotNullableField
objectrole:hasRole : FieldRole (PrimaryKeyField, ForeignKeyField)
hasField
belongsToDatabase
isDatabaseOf
isFieldOf
Table
belongsToDatabase : Database
hasField : Field
hasQuery : Query
ownsForeignKey : ForeignKeyField
ownsPrimaryKey : PrimaryKeyField

Figure 4.1: semionDB ontology: SchemaObject classes.

In figure 4.1 there is the diagram that represents for objects that compose the database schema, and they are three disjoint classes:

- *Database* that contains all the information about the connection and the database type. Instances of this class are any possible database.

- *Table* is the class used to represent database table objects. Instance of this class are related to that ones of *Database* by the property *belongsToTable*, whose inverse is *hasTable*.

- *Field* is the class used to represent table columns. Instances of this class are related to that one of *Table* by the property *isFieldOf*. The inverse relation is realized by the property *hasField*. In relational databases it is common to distinguish between two disjoint type of Columns that in the ontology are realized by:

  - *NotNullableField* whose instances are those columns that do not admit null values.
  - *NullableField* whose instances are those columns that may admit null values.

These three classes are, de facto, the core concepts of database schemata, but they are not enough to express all the information needed to represent relationships among them and, above all, with data.
In fact, in relational databases a column can not be distinguished from others only because it may admit null values, but also because it may play the role of being a primary key and can be used to uniquely identify a database record. It may als play the role of being a foreign key and can be used to realize relationships joining on other table columns.

Figure 4.2 explains what it has been just introduced.

Roles and fields are two disjoint concepts, so that a *PrimaryKeyField* is not a type of field, but a possible role played by an instance of the class *Field*. In our ontology it is clear how only instances of *NotNullableFild* can play the role of being PrimaryKeyField or *ForeignKeyField*, because not null values are required in relational database to identify primary and foreign keys.

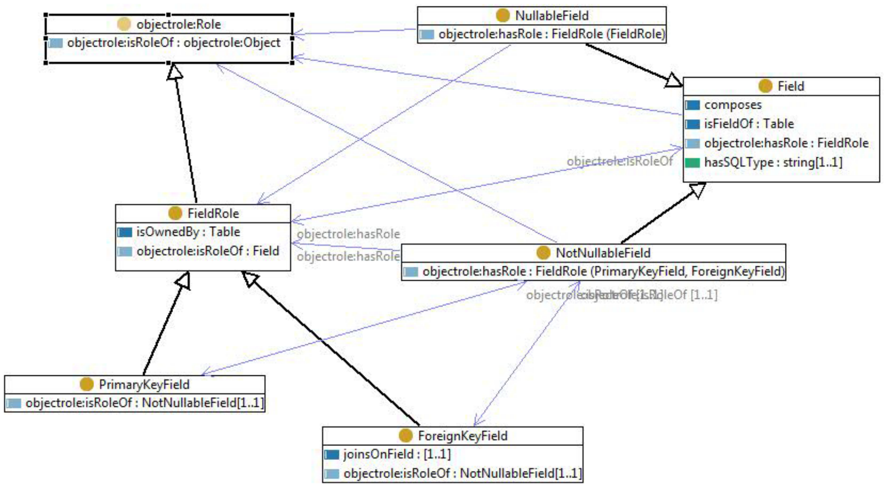

objectrole:Role
objectrole:isRoleOf : objectrole:Object
NullableField
objectrole:hasRole : FieldRole (FieldRole)
Field
composes
isFieldOf : Table
objectrole:hasRole : FieldRole
hasSQLType : string[1..1]
objectrole:isRoleOf
FieldRole
isOwnedBy : Table
objectrole:isRoleOf : Field
objectrole:hasRole
objectrole:hasRole
NotNullableField
objectrole:hasRole : FieldRole (PrimaryKeyField, ForeignKeyField)
objectrole:isRoleOf [1..1]
PrimaryKeyField
objectrole:isRoleOf : NotNullableField[1..1]
ForeignKeyField
joinsOnField : [1..1]
objectrole:isRoleOf : NotNullableField[1..1]

Figure 4.2: semionDB ontology: Roles of Fields.

It is possible to associate queries to tables, in order to fetch data in the database, thanks to the property *hasQuery* that allows to relate instances of the class *Table* to instances of the class *Query*.
Any individual of the class *Query* should be associated to:

- a literal that is the concrete SQL [IBM] query used to fetch data by property *hasSQLQuery*
- a *Table* by the property *isQueryOf*, that is the inverse of *hasQuery*
- an individual of the class *ResultSet*, that is a collection of individuals of the class *Record*, that maps a physical database record

An individual of the class *Record* is both a member of a *ResultSet* and a collection of values, so it has instances of the class *Value* as members. This relations are shown in figure 4.3.
A value in this vocabulary for relational databases is a single cell in a database table and is associated to a specific column of the table. As individuals of the class *Field*, that represents physical table columns, can play roles such as being primary or foreign keys in a table schema, also values can play the same roles but in the data schema of the

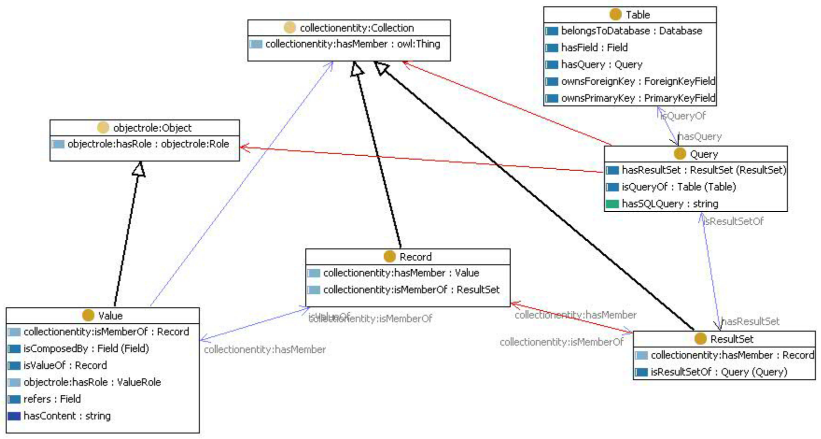

collectionentity:Collection
collectionentity:hasMember : owl:Thing
Table
belongsToDatabase : Database
hasField : Field
hasQuery : Query
ownsForeignKey : ForeignKeyField
ownsPrimaryKey : PrimaryKeyField
isQueryOf
hasQuery
objectrole:Object
objectrole:hasRole : objectrole:Role
Query
hasResultSet : ResultSet (ResultSet)
isQueryOf : Table (Table)
hasSQLQuery : string
isResultSetOf
Record
collectionentity:hasMember : Value
collectionentity:isMemberOf : ResultSet
Value
collectionentity:isMemberOf : Record
isComposedBy : Field (Field)
isValueOf : Record
objectrole:hasRole : ValueRole
refers : Field
hasContent : string
collectionentity:isMemberOf
collectionentity:hasMember
collectionentity:hasMember
collectionentity:isMemberOf
hasResultSet
ResultSet
collectionentity:hasMember : Record
isResultSetOf : Query (Query)

Figure 4.3: semionDB ontology: Tables, Queries, ResultSets, Records and Values schema.

tables. So if we have a column in a table that is a primary key, the value of a specific table record associated to that column must necessary be a primary key, also remembering that only not nullable cells are admitted to be primary or foreign keys. Figure 4.4 shows what it has been just said.

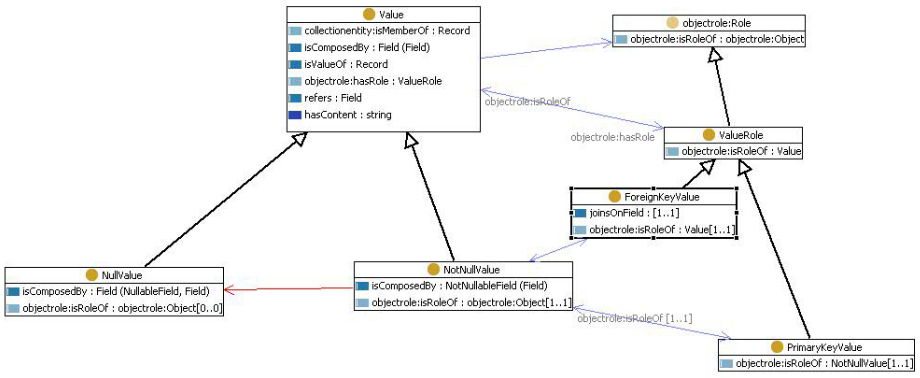

Value
collectionentity:isMemberOf : Record
isComposedBy : Field (Field)
isValueOf : Record
objectrole:hasRole : ValueRole
refers : Field
hasContent : string
objectrole:Role
objectrole:isRoleOf : objectrole:Object
objectrole:isRoleOf
objectrole:hasRole
ValueRole
objectrole:isRoleOf : Value
ForeignKeyValue
joinsOnField : [1..1]
objectrole:isRoleOf : Value[1..1]
NullValue
isComposedBy : Field (NullableField, Field)
objectrole:isRoleOf : objectrole:Object[0..0]
NotNullValue
isComposedBy : NotNullableField (Field)
objectrole:isRoleOf : objectrole:Object[1..1]
objectrole:isRoleOf [1..1]
PrimaryKeyValue
objectrole:isRoleOf : NotNullValue[1..1]

Figure 4.4: semionDB ontology: Roles played by Values.

4.2.2 Reengineering: from databases to RDF

In section 4.3.1 were introduced the five layers stack for mapping and translating non-RDF and legacy data sources into RDF. Now it is given an implementation of that method that uses that stack to drive the Semantic Lifting of relational databases.

Spending other few words about the five layers stack, the users suggest to the reengineer their customization choices passing as parameter RDF/OWL files that lay in:

- *L5* that is the vocabulary of the mapping model
- *L4* that is the concrete mapping model for the data source selected
- *L3* that contains the vocabulary both for schema and for data
- *L2* that is the concrete schema of the data source
- *L1* is the translation of the non-RDF data into RDF following the vocabulary of data given in *l3*

In figure 4.5 there is the UML [Pre04] class diagram of the reengineering module that is the core of the Semantic Lifting.

The class *SemionSchemaGenerator* is the director of the reengineering. Its inputs are a connection to a data source, that is represented in figure 4.5 by the interface *ConnectionSettings*, that is implemented by the class *ConnectionSettingsImpl* in order to manage connections to relational database via JDBC [FEB03], and an RDF/OWL description of the mapping between the physical data source (i.e. a relational database) and the vocabulary for representing the schema of the database described in section 4.2.1.
With connection information and the mapping description it is possible to translate the physical schema of a specific database in RDF/OWL and then extract its content.

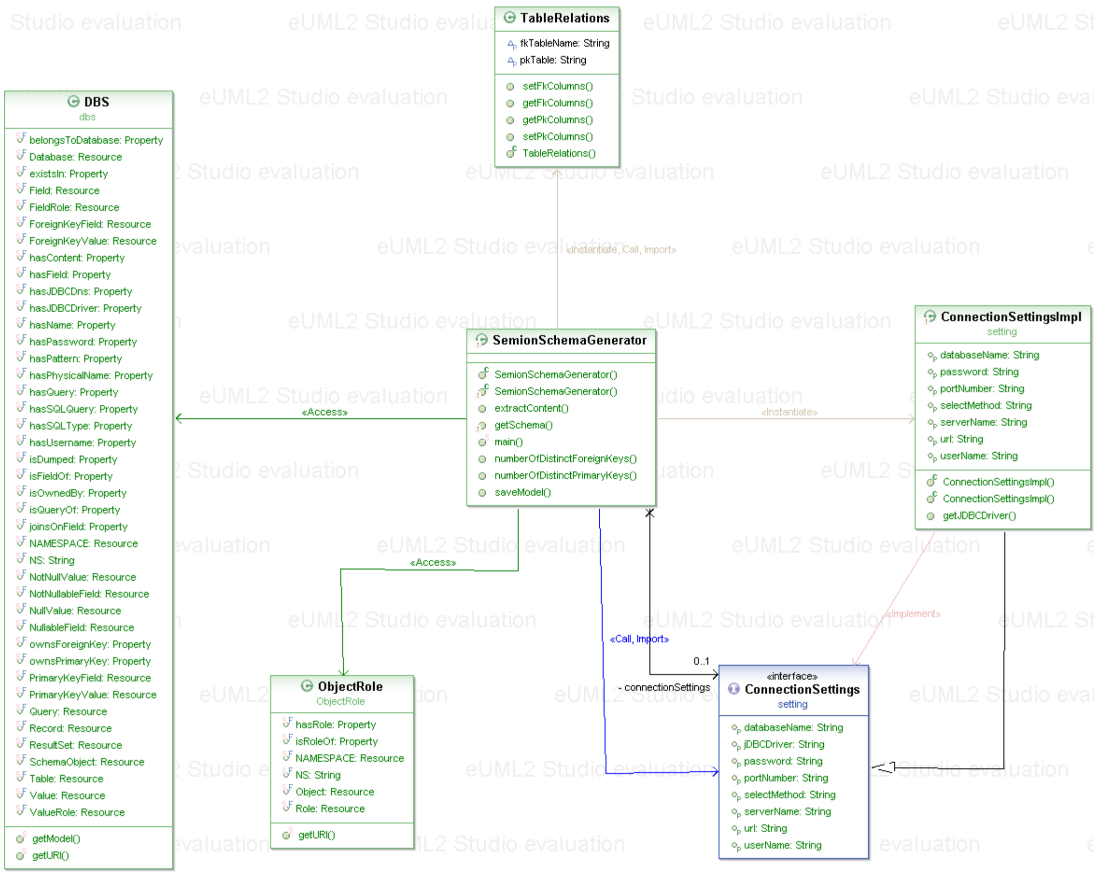


Figure 4.5: Class Diagram of the reengineer.

To avoid the standard interpretation and extraction of the database physical schema, it is also possible to pass the latter as an input and allowing the reengineer to extract only data.
In order to perform the semantic lifting of relational databases, it was said that a mapping is needed. This is an RDF document that contains informations about the mapping between Java JDBC objects and classes and properties defined in the ontology *semionDB*. When the *SemionSchemaGenerator* starts the computation the are two possibilities:

1. extract both schema and data from the relational database

2. extract just data passing the schema as input

In both cases, as the object of the reengineering are relational databases, the *SemionSchemaGenerator* needs to know how to represent a JDBC object in the *semionDB* ontology, so now it is clear the role of the mapping in the architecture.
The two methods of the class *SemionSchemaGenerator*, that are involved in the Semantic Lifting are:

- *extractSchema* for the translation in RDF of the database schema
- *extractContent* for the translation in RDF of the database content

The class *TableRelations* is used by *SemionSchemaGenerator* in order to collect relations among the tables of the database expressed as foreign keys. It is used only for utility reasons during the reengineering process.

At the end of the computation the result is an object that is an instance of the class *SemionModel* that implements the Jena *Model* interface, that represents RDF models as Java objects. Methods are provided for creating resources, properties and literals and the Statements which link them, for adding statements to and removing them from a model, for querying a model and set operations for combining models.

4.3 Adding multilinguality to data

Once a database is translated into RDF it is possible to proceed to the refactoring of its resources in order to align reengineered datasets to defined content patterns or standard ontologies.
The approach followed for aligning datasets is based on reasoning on sets of rules that the user submits as input to the refactoring module of Semion. Though the refactoring method is thought to align datasets to LMM, Semion is implemented to manage alignments to any other ontology and from any ontology. That allows Semion to use as input already aligned datasets, and, more generally, to combine ontologies during the refactoring.

For instance, the approach that is followed in the first evaluation, regarding relational databases in chapter 5, is based on three alignments:

1. the alignment of the reengineered dataset to LMM
2. the alignment of the refactored LMM dataset to the ontology FormalSemantics.owl
3. the alignment of the refactored FormalSemantics.owl dataset to the logic of OWL

The module is opened to accept any ontology for the alignment. The reason is to be found in the fact that an user can perform the rafactoring by choosing what he considers the best content pattern or ontology for adding semantics to dataset that he has.

4.3.1 SemionRule: Alignment as inference on set of rules

In Semion the refactoring is thought to be performed as reasoning on sets of alignment rules expressed in SWRL syntax. If, on one hand, SWRL rules ensure portability, being both expressed with an OWL vocabulary and not bind to any specific reasoner, on the other they may result complex to write for an user. For this reason it was defined in Semion a human-friendly language to represent rules, that before performing the alignments, is interpreted and translated to SWRL and then used as an input for the Pellet reasoning.
Below it is described the syntax for the rule language that was defined:

Now it is possible to write a rule that is able to infer that any individual of the class *Table* of the semionDB ontology is an LMM *Meaning*:

```
myRules:CMRule [ semionDB:Table(?x) -> LMM_L1(?x)]
```

where:

```
      axiom ::= r-name [ rule ]
       rule ::= antecedent → consequent
 antecedent ::= atom
              | atom . antecedent
 consequent ::= atom
              | atom . consequent
       atom ::= description(i-object)
              | dataRange(d-object)
              | individualvaluedPropertyID(i-object, i-object)
              | datavaluedPropertyID(i-object, i-object)
              | same(i-object, i-object)
              | different(i-object, i-object)
   i-object ::= i-variable | individualID
   d-object ::= d-variable | dataLiteral
 i-variable ::= ? URIreference
 d-variable ::= ? URIreference)
     r-name ::= URIreference
```

Figure 4.6: The SemionRule language.

- *myRules* is the namespace in which rules are defined
- *CMRule* is the name chosen for the rule
- *?x* is a variable

The *CMRule* is translated by the refactoring module of Semion in the following SWRL rule

```
<swrl:Variable rdf:ID="x"/>
<swrl:Imp rdf:about="http://www.myRules.org#CMRule">
  <swrl:body rdf:parseType="Collection">
    <swrl:ClassAtom>
      <swrl:classPredicate
      rdf:resource="http://andriry.altervista.org/dbs.owl#Table"/>
      <swrl:argument1 rdf:resource="http://www.myRules.org#x"/>
      </swrl:ClassAtom>
    </swrl:body>
  <swrl:head rdf:parseType="Collection">
    <swrl:ClassAtom>
  <swrl:classPredicate
  rdf:resource="http://www.ontologydesignpatterns.org/ont/
  lmm/LMM_L1.owl#Meaning"/>
  <swrl:argument1 rdf:resource="http://www.myRules.org#x"/>
  </swrl:ClassAtom>
  </swrl:head>
</swrl:Imp>
```

4.3.2 Refactoring module: concrete implementation

As explained in the previous section, the refactoring of RDF datasets is realized by reasoning on sets of rules written lithe in SWRL syntax or in in SemionRule syntax.
We are now interested in how Semion performs the translation from SWRL to SemionRule and vice versa and the reasoning.

Figure 4.7 shows the UML class diagram of the refactoring module.

The class *SemionRefactorer* is responsible of the coordination of the whole computation of the module.
Translations from SemionRule to SWRL are performed by a *SemionRefactorer* object, allocating an instance of the class *SemionRuleParser*, that implements the interface *ISWRLRule* and whose translation task is executed by the method *parse*, that is declared by *ISWRLRule* and implemented by *SemionRuleParser*. On the other hand, translations from SWRL to SemionRule are performed by a *SemionRefactorer* object, allocating an instance of the class *SWRLToSemionRule*, that is also an implementation of the interface *ISWRLRule* and also in this case, the method involved in the translation is *parse*.
It is important to point out that both the input and the output of the method *parse* is a Java *Object*. The reason of this choice is to be found in the two different ways of translation:

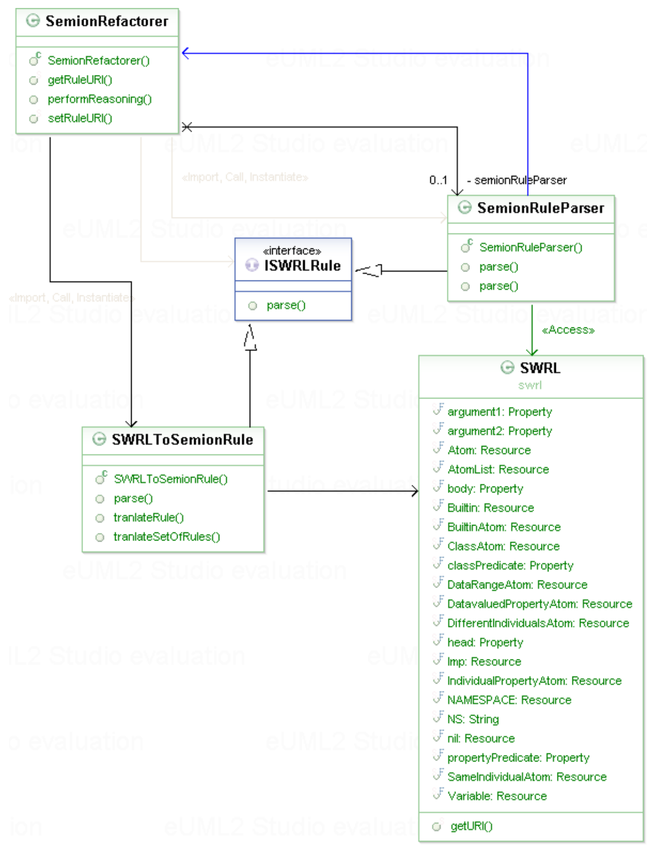


Figure 4.7: Class Diagram of the refactoring module.

- from a Jena *Model* to a Java *String* when the system is running a translation from SWRL to SemionRule
- from a Java *String* to a Jena *Model* when the system is running a translation from SemionRule to SWRL

It is not a problem, because an instance of the class *SemionRefactorer* does not know so much about the concrete implementations of the *ISWRLRule* interface, but it knows that it should aspect a *String* or *Model* depending on the direction of the translation as just mentioned.

The class *SWRL* of the module provides static methods in order to directly access the OWL classes and properties of the SWRL ontology described in *http://www.w3.org/2003/11/swrl*.

The translation from SWRL to SemionRule is important only for reasons concerning human readability and it has not any affect on the reasoning.
In fact, the reasoning is provided by the refactoring module of Semion passing to Pellet SWRL rules through the Jena interface. Pellet has an implementation of a direct tableau algorithm [BS01] for a DL-safe rules extension to OWL-DL. This implementation allows one to load and reason with DL-safe rules encoded in SWRL and includes support for some SWRL built-ins.

4.4 The graphical tool

The graphical tool interacts with the other modules as the software was designed following the ModelViewController (MVC) [Wikb] pattern. This pattern allows to represent the reengineering and the refactoring components as the models (see figure 4.8), that elaborate the data upon which the application operates. The view renders the model into a form suitable for interaction, a user interface element. The controller receives input and initiates a response by making calls on model objects.

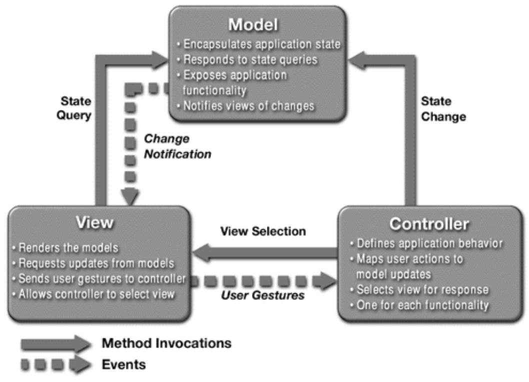

Model
• Encapsulates application state
• Responds to state queries
• Exposes application functionality
• Notifies views of changes
State Query
State Change
Change Notification
View
• Renders the models
• Requests updates from models
• Sends user gestures to controller
• Allows controller to select view
View Selection
User Gestures
Controller
• Defines application behavior
• Maps user actions to model updates
• Selects view for response
• One for each functionality
Method Invocations
Events

Figure 4.8: The Model-View-Controller pattern.

It is implemented using the Java Standard Widget Toolkit (SWT) [Eclb], that is is an open source widget toolkit for Java designed to provide efficient, portable access to the user-interface facilities of the operating systems on which it is implemented, and JFace [Ecla], that is a window-system-independent in both its API and implementation, and is designed to work with SWT without hiding it. JFace includes the usual UI toolkit components of image and font registries, text, dialog, preference and wizard frameworks, and progress reporting for long running operations. Two of its more interesting features are actions and viewers. The action mechanism allows user commands to be defined independently from their exact whereabouts in the UI. Viewers are model based adapters for certain SWT widgets, simplifying the presentation of application data structured as lists, tables or trees.

As it is possible to see in figure 4.9, the core of the graphical tool is the class *Semion*, that contains the *main* method and allows the built and the visualization of the user interface.

The user interface is composed by two principal area, as shown in figure 4.10:

- the Project Explorer on the left, that allows to browse through the various Semion Projects and also to add new and to manage existing ones.
- the Tab Folder on the right, that allows to the users to customize or simple to view the RDF representation of a relational database and to align the latter to LMM or to any ontology design pattern or vocabulary.

The Project Explorer is realized implementing a JFace *TreeViewer*, that is designed to be instantiated with a pre-existing SWT *Tree* control and configured with a domain-specific content provider (i.e. the class *MappingTreeContentProvider* in figure 4.10), that allows to associate to tree items Java objects, and with a domain-specific label provider (i.e. the class *MappingTreeLabelProvider* in figure 4.10), that provide labels and images to be displayed as tree items.
The Semion projects are physically stored into an user defined workspace and accessed through an RDF document that describes the workspace itself.
The class *Workspace* is used to manage the RDF that semantically

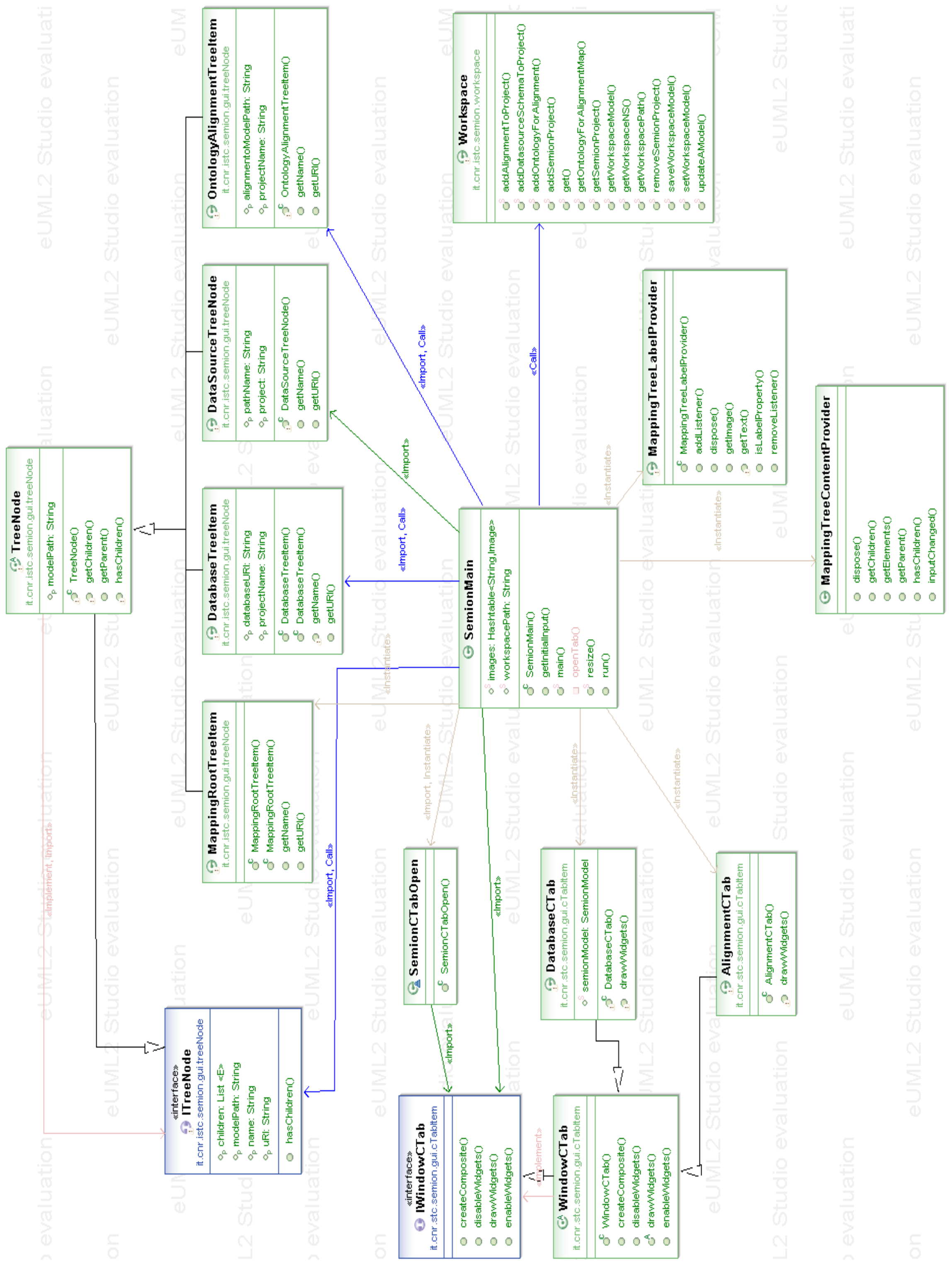
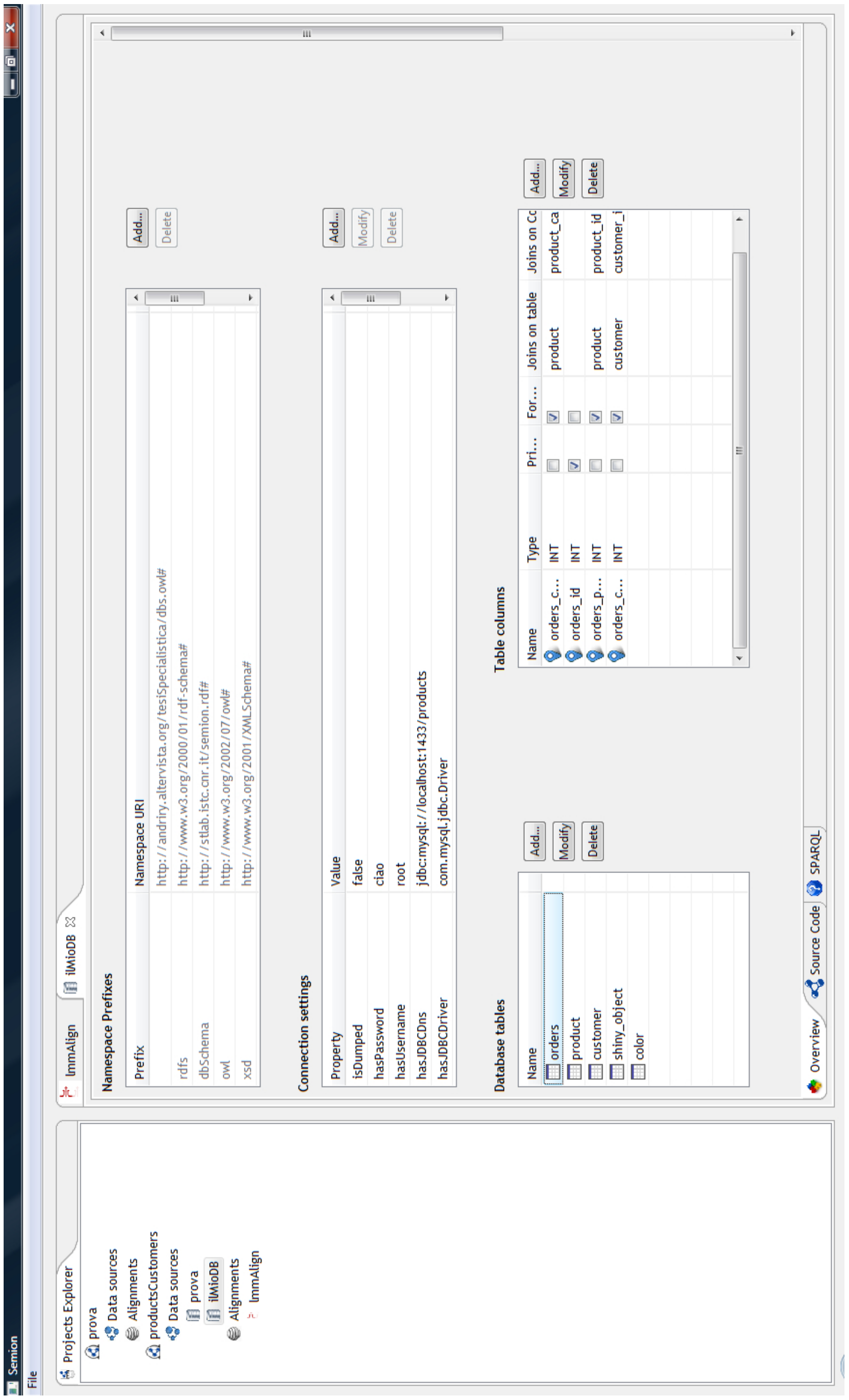


Figure 4.9: Class Diagram of the graphical tool.



Namespace Prefixes

Prefix	Namespace URI
	http://andriry.altervista.org/tesiSpecialistica/dbs.owl#
rdfs	http://www.w3.org/2000/01/rdf-schema#
dbSchema	http://stlab.istc.cnr.it/semion.rdf#
owl	http://www.w3.org/2002/07/owl#
xsd	http://www.w3.org/2001/XMLSchema#

Connection settings

Property	Value
isDumped	false
hasPassword	ciao
hasUsername	root
has.JDBCDns	jdbc:mysql://localhost:1433/products
has.JDBCDriver	com.mysql.jdbc.Driver

Database tables

Name
orders
product
customer
shiny_object
color

Table columns

Name	Type	Pri...	For...	Joins on table	Joins on Co
orders_c...	INT	☐	✓	product	product_ca
orders_id	INT	✓	☐		
orders_p...	INT	☐	✓	product	product_id
orders_c...	INT	☐	✓	customer	customer_i

Figure 4.10: Semion Graphical User Interface.

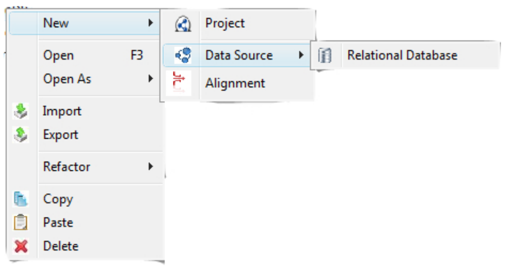
describes the Semion workspace.
To perform actions on projects it is available a pop-up menu for the Project Explorer, as shown in figure 4.11(a). In the specific case it is possible to see how to add an new database in a Semion project:

- in the first (figure 4.11(b)) step it is required to choose between an empty database or to extract it with the reengineering wizard. The first choice means that it will be the user to add tables, columns and relations manually, while the second one means that the database will be automatically lifted into an RDF form, thanks to the reengineering module described in section 4.2
- if the user chooses the first option, the database with the name selected will be added to the project (notice: untill the user does not add tables and columns the database is empty), while if the user choses the second option a new page is shown in order to fill the form with the information that the reengineer requires to perform a semantic lifting (figure 4.11(c))

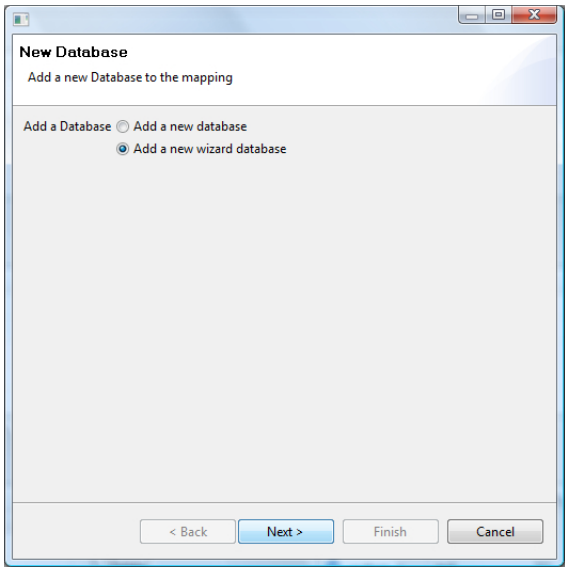
All the tree items of the Project Explorer extend the abstract class *TreeNode* (see the class diagram in figure 4.9, that implements the interface *ITreeNode*, and they are:

- *MappingRootTreeItem*: is the root node for each project
- *DataSourceTreeNode*: is the father node for the tree items that represents data sources
- *DatabaseTreeItem*: is the item to display relational databases in the tree viewer
- *OntologyAlignmentTreeItem*: is the item to display aligned ontologies

Both *DatabaseTreeItem* and *OntologyAlignmentTreeItem* if opened clicking on the "Open" entry in the pop-up menu or with a double click of the mouse, allows to analyze the structure of the RDF translation of the database and of the alignment in the tab folder in the right (see



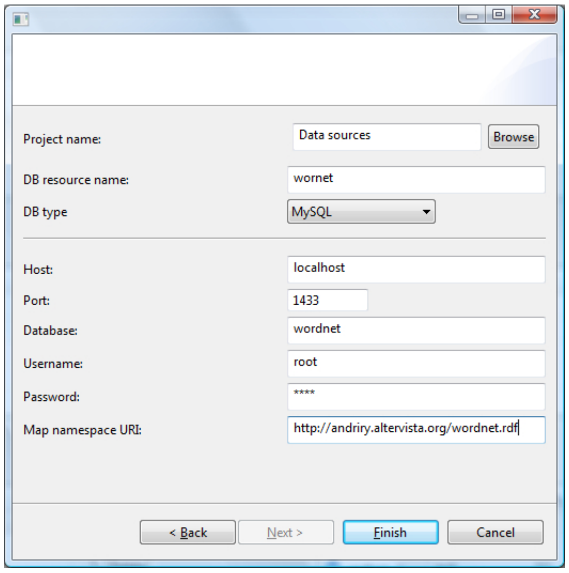
(a)


New Database

Add a new Database to the mapping

Add a Database: Add a new database [✓ Add a new wizard database]

(b)



Project name:	Data sources
DB resource name:	wornet
DB type	MySQL
Host:	localhost
Port:	1433
Database:	wordnet
Username:	root
Password:	****
Map namespace URI:	http://andriry.altervista.org/wordnet.rdf

(c)

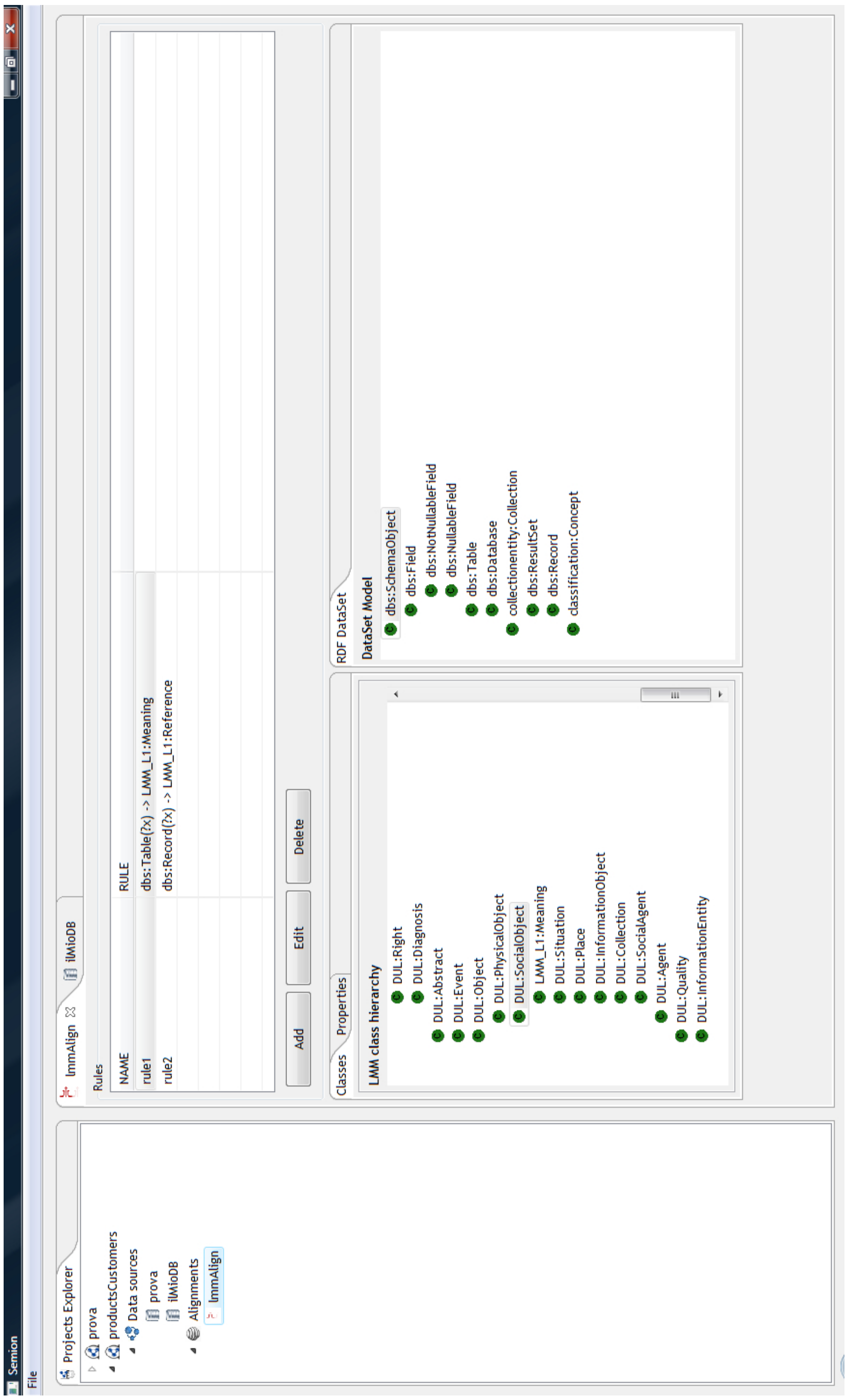
Figure 4.11: Creation of a new database in a Semion project.

figure 4.10).
In particular when the tree item *DatabaseTreeItem* is opened a tab item with the same name and the same icon is displayed in the tab folder and it allows to add, modify or delete resources from an RDF model of a database. It is composed by three sub-tab items:

- **Overview**, that organizes all the resources of the RDF model in a tabular way, so that in the top are listed in a table all the namespaces and their relative prefixes, then there is a table that displays all the connection settings to the physical database and in the bottom of the window there are two tables that lists all the database table and the columns associated to the selected table respectively. In the columns table is provided all the information relative to the role of the column in the database, so it is possible to know if it is a primary or a foreing key and on wich columns of another table it joins.
- **Source Code**, that displays the source code of the RDF model in three possible syntaxes, that are RDF/XML, RDF/XML-ABBREV and N3.
- **SPARQL**, that provides a SPARQL endpoint in order to query the RDF model of the database.

When the tree item *OntologyAlignmentTreeItem* is opened, another tab item, always with the same name and the same icon, is displayed in the tab folder and it allows to perform alignments. Alignments, as already said, are obtained infering on sets of SWRL rules. In the alignment tab item there is a table (see figure 4.12) that lists all the inference rules defined for that model. This set is customized both through a wizard editor, that accepts rules in SemionRule syntax, and through the drag and drop of a tree item from "RDF DataSet" tree in the right into the target ontology in the left (the two trees in the bottom of the figure 4.12)

When the user completes the refactoring he can proceed with another alignment to another ontology (i.e. FormalSemantics.owl) or exports the inferred ontology so he can publish it as Linked Data.



- prova
- productsCustomers
  - Data sources
    - prova
    - ilMioDB
  - Alignments
    - ImmAlign

NAME	RULE
rule1	dbs:Table(?x) -> LMM_L1:Meaning
rule2	dbs:Record(?x) -> LMM_L1:Reference

LMM class hierarchy

- DUL:Right
  - DUL:Diagnosis
- DUL:Abstract
- DUL:Event
- DUL:Object
  - DUL:PhysicalObject
  - DUL:SocialObject
    - LMM_L1:Meaning
    - DUL:Situation
    - DUL:Place
    - DUL:InformationObject
    - DUL:Collection
    - DUL:SocialAgent
  - DUL:Agent
- DUL:Quality
- DUL:InformationEntity

DataSet Model

- dbs:SchemaObject
  - dbs:Field
    - dbs:NotNullableField
    - dbs:NullableField
  - dbs:Table
  - dbs:Database
- collectionentity:Collection
  - dbs:ResultSet
  - dbs:Record
- classification:Concept

Figure 4.12: Alignments in the Semion Graphical User Interface.

Chapter 5

Semion: evaluation

Semion was tested both on reengineering relational databases and PGN and LaTex file containing information about chess games. The first test is described in section 1 and the second in section 2. While, relational databases were the declared case study, PGN and LaTex were chosen to test the capability of Semion to be easily extended in order to manage the reengineering from other kind of data sources.

5.1 Testing Semion on relational databases

In order to make an exhaustive and non trivial test of the Semion tool and its components we focused our attention on a large database in which the alignment to LMM and then to a vocabulary that expresses the formal semantics (i.e. FormalSemantics) makes much sense.

5.1.1 Evaluation scenario: WordNet

The chosen database is WordNet [Mil95] in its MySQL [WA02] version. WordNet is a lexical database for the English language. It groups

English words into sets of synonyms called synsets, provides short, general definitions, and records the various semantic relations between these synonym sets. The purpose is twofold:

- to produce a combination of dictionary and thesaurus that is more intuitively usable
- to support automatic text analysis and artificial intelligence applications.

WordNet was created and is being maintained at the Cognitive Science Laboratory of Princeton University under the direction of psychology professor George A. Miller.
WordNet distinguishes between nouns, verbs, adjectives and adverbs because they follow different grammatical rules. Every synset contains a group of synonymous words or collocations (a collocation is a sequence of words that go together to form a specific meaning, such as "car pool"); different senses of a word are in different synsets. The meaning of the synsets is further clarified with short defining glosses (Definitions and/or example sentences).

Most synsets are connected to other synsets via a number of semantic relations. These relations vary based on the type of word, and include:

- Nouns
  - hypernyms: Y is a hypernym of X if every X is a (kind of) Y (canine is a hypernym of dog)
  - hyponyms: Y is a hyponym of X if every Y is a (kind of) X (dog is a hyponym of canine)
  - coordinate terms: Y is a coordinate term of X if X and Y share a hypernym (wolf is a coordinate term of dog, and dog is a coordinate term of wolf)
  - holonym: Y is a holonym of X if X is a part of Y (building is a holonym of window)
  - meronym: Y is a meronym of X if Y is a part of X (window is a meronym of building)

- Verbs
  - hypernym: the verb Y is a hypernym of the verb X if the activity X is a (kind of) Y (to perceive is an hypernym of to listen)
  - troponym: the verb Y is a troponym of the verb X if the activity Y is doing X in some manner (to lisp is a troponym of to talk)
  - entailment: the verb Y is entailed by X if by doing X you must be doing Y (to sleep is entailed by to snore)
- Adjectives
  - related nouns
  - similar to
  - participle of verb
- Adverbs
  - root adjectives

While semantic relations apply to all members of a synset because they share a meaning but are all mutually synonyms, words can also be connected to other words through lexical relations, including antonyms (opposites of each other) which are derivationally related, as well.

A W3C working draft for the RDF/OWL representation of WordNet already exists [vAGS06]. It was developed performing the conversion by using a Prolog program written in SWI-Prolog. Our aim is to use a different methodology that expresses the conversion as an RDF/OWL itself that contains SWRL rules and does not hide the details into a specific programming language such as Prolog.

5.1.2 Testing the reengineering module

The reengineering of the WordNet MySQL database was performed with the Semion graphical tool and the database wizard extractor

provided by the user interface of the tool.
The input database was composed by 19 tables, 647,843 total records and 62MB of memory size. The exact record and memory size distribution can be seen in figure 5.1
Each table was translated into an RDF resource and it was reen-

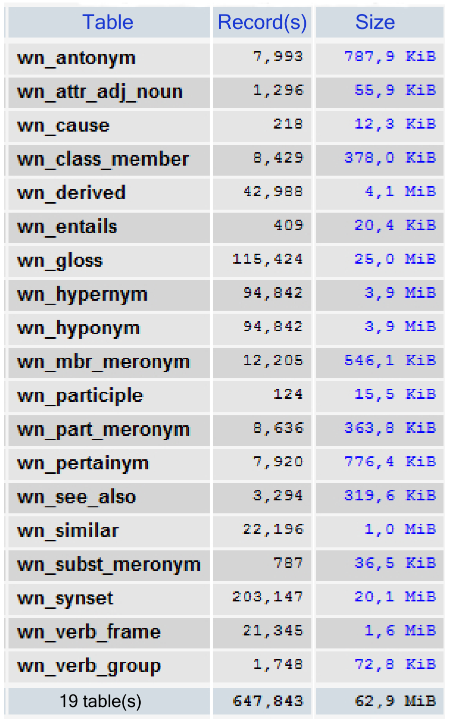


Table	Record(s)	Size
wn_antonym	7,993	787,9 KiB
wn_attr_adj_noun	1,296	55,9 KiB
wn_cause	218	12,3 KiB
wn_class_member	8,429	378,0 KiB
wn_derived	42,988	4,1 MiB
wn_entails	409	20,4 KiB
wn_gloss	115,424	25,0 MiB
wn_hypernym	94,842	3,9 MiB
wn_hyponym	94,842	3,9 MiB
wn_mbr_meronym	12,205	546,1 KiB
wn_participle	124	15,5 KiB
wn_part_meronym	8,636	363,8 KiB
wn_pertainym	7,920	776,4 KiB
wn_see_also	3,294	319,6 KiB
wn_similar	22,196	1,0 MiB
wn_subst_meronym	787	36,5 KiB
wn_synset	203,147	20,1 MiB
wn_verb_frame	21,345	1,6 MiB
wn_verb_group	1,748	72,8 KiB
19 table(s)	647,843	62,9 MiB

Figure 5.1: WordNet MySQL tables.

gineered into an instance of the *semionDB:Table* class. The URIs identifying new created *semionDB:Table* instances, followed the pattern:

$$baseURI\# < table\_name >$$

where:

- $baseURI\#$ identifies the namespace chosen for the dataset (i.e. in our case it was *http://andriry.altervista.org/tesiSpecialistica/wn.rdf#*)
- $< table\_name >$ has to be substituted by the name of the physical database table (e.g. wn_synset, wn_gloss, etc...)

Each column was translated into an RDF resource and, according to its nullable SQL option, it was reengineerd into:

- a *semionDB:NotNullableField*, if the physical column table did not admit null value in the database
- a *semionDB:NullableField*, if the physical column table admitted null value in the database

The pattern used for column URI generation was the following:

$$baseURI\# < table\_name > \_ < column\_name >$$

where:

- $baseURI\#$ identifies the namespace chosen for the dataset as before
- $< table\_name > \_ < column\_name >$ identifies the *semionDB:Field* (both *semionDB:NotNullableField* and *semionDB:NullableField* are subclasses of its) and where
  - $< table\_name >$ has to be substituted by the name of the physical database table in which the column exists
  - $< column\_name >$ has to be substituted by the name of the physical table column

During the generation of the column RDF resources, *semionDB:hasField* and *semionDB:isFieldOf* object properties were automatically added to table resources and column resources respectively. The existence in the database tables of primary and foreign keys was reflected in the RDF dataset by the creation of *semionDB:PrimaryKeyField* and *semionDB:ForeignKeyField* class instances related to *semionDB:Not NullableField* by the object property *objectrole:isRoleOf*.

Data were extracted defining the SQL queries as literals in *semionDB:Query* resources related to *semionDB:Table* resources. In this evaluation test queries were projected in order to obtain a single database dump, so that to each table was associated a query of the form "SELECT * FROM <table_name>". In other situations could be better to project queries oriented to paginate or to filter result sets only to needed data. The execution of the queries produced:

- 19 instances of the *semionDB:ResultSet* class related to *semionDB: Query* instances by the object property *semionDB:hasResultSet*
- 647,843 instances of the *semionDB:Record* class related to *semionDB:ResultSet* instances by the object property *collectio-nentity:isMemberOf*
- 2,264,506 instances of both the *semionDB:NullValue* and the *semionDB:*
  NotNullValue classes related to *semionDB:ResultSet* instances by the object property *collectionentity:isMemberOf*
- 752,623 relations between synset value resources and value resources identified in other table instances. This relations are realized by foreign key resources related to *semionDB:NotNullValue* instances that maps physical table cells of the WordNet database that contained a foreign key. In table 5.1 to each table resource is associated the number of foreign key resource that joins on synset resources.

All the results in table 5.1 and all the other given results in this section are obtained as output of SPARQL queries on the RDF dataset generated from the reengineering process.

Dumping the whole WordNet MySQL relational database into an RDF dataset required approximately 480 seconds.

The reengineerd dataset was published at
http:/wwww.ontologydesignpatterns.org/ont/WNet/wnet.rdf.

Table 5.1: Foreign keys joining on synsets

Table URI	**# of foreign keys individuals**
wn:wn_antonym	15986
wn:wn_attr_adj_noun	2592
wn:wn_cause	436
wn:wn_class_member	16858
wn:wn_derived	85976
wn:wn_entails	818
wn:wn_gloss	115424
wn:wn_hypernym	189684
wn:wn_hyponym	189684
wn:wn_mbr_meronym	24410
wn:wn_participle	248
wn:wn_part_meronym	17272
wn:wn_pertainym	15840
wn:wn_see_also	6588
wn:wn_similar	44392
wn:wn_subst_meronym	1574
wn:wn_verb_frame	21345
wn:wn_verb_group	3496

5.1.3 Testing the refactoring module

After having reengineered the WordNet database into an RDF dataset a series of alignment tests were profiled. The tests were planned in way to obtain a pipeline in which the output of the previous test session was the input of the next, namely:

1. from the WordNet RDF dataset to LMM
2. from LMM to FormalSemantics
3. from FormalSemantics to OWL vocabulary

As the Semion refactoring module works on alignments based on the Pellet reasoning on set of rules in SWRL syntax, for each one of the three previous points were defined rules using the Semion user interface and the SemionRule language in order to prove the efficiency of the translation from SemionRule to SWRL performed by the software.

Below are listed some topic rules defined for alignment in point 1:

1. *same(wn:wn_synset, ?x)*
   $\rightarrow$
   dul:Collection(?x)

2. *semionDB:hasQuery(wn:wn_synset, ?x) .*
   semionDB:hasResultSet(?x, ?y) .
   collectionentity:hasMember(?y, ?z) .
   semionDB:Record(?z) .
   $\rightarrow$
   dul:Concept(?z) .
   dul:covers(?z, ?x)

3. *semionDB:Record(?z) .*
   collectionentity:hasMember(?z, ?w) .
   collectionentity:hasMember(?z, ?t) .
   semionDB:NotNullValue(?w) .
   semionDB:NotNullValue(?t) .
   semionDB:refers(?w, wn:wn_part_meronym_synset_id1) .
   semionDB:refers(?t, wn:wn_part_meronym_synset_id2) .
   objectrole:hasRole(?w, ?k) .
   objectrole:hasRole(?t, ?j) .
   semionDB:joinsOnField(?k, ?p) .
   semionDB:joinsOnField(?j, ?q) .
   collectionentity:isMemberOf(?p, ?x) .
   collectionentity:isMemberOf(?q, ?y) .
   $\rightarrow$
   dul:isPartOf(?x, ?y) .
   dul:covers(?z, wn:wn_part_meronym)

4. *semionDB:Record(?z) .*
   collectionentity:hasMember(?z, ?w) .
   collectionentity:hasMember(?z, ?t) .
   semionDB:NotNullValue(?w) .

semionDB:NotNullValue(?t) .
semionDB:refers(?w, wn:wn_gloss_synset_id) .
semionDB:refers(?t, wn:wn_part_gloss) .
objectrole:hasRole(?w, ?k) .
semionDB:joinsOnField(?k, ?p) .
collectionentity:isMemberOf(?p, ?x) .
$\rightarrow$
LMM_L1:Expression(?t) .
LMM_L1:expresses(?t, ?x) .
dul:Relation(?z)

The rule (1) allowed to infer that the instance *wn:wn_synset* that identified a *semionDB:Table* was a also an instance of *dul:Collection*. This pattern was adopted for most of the instances of the *semionDB:Table* in the WordNet RDF dataset because a *dul:Collection* can be thought as any container for entities that share one or more common properties.
The rule (2) let the system to infer that any *semionDB:Record* is a *dul:Concept* and covers the *dul:Collection* identified by the specific *semionDB:Table*, which the record is related to through an instance of the *semionDB:Query*, that, in the dataset, represents the SQL query executed to generate the considered *semionDB:Record*.
The rule (3) was used to infer meronym relationships between synsets as relationships *dul:isPartOf* in LMM. To find instances satisfying this rule, it was needed to look for all *semionDB:Value* instances members of some *semionDB:Record* (interpreted as a *dul:Relation*, that had a reference with *wn:wn_part_meronym_synset_id2* and *wn:wn_part_meronym_synset_id1* fields instances in table *wn:wn_part_meronym*.
In rule (4) values of the column *wn:wn_part_gloss* (?t) was inferred as *LMM_L1:*
Expression individuals, that express a particular meaning defined by the synset record (?x) indetified by *wn:wn_gloss_synset_id*(?w). Records of the gloss table (?z) are interpreted as *dul:Relation* individuals.

Then, the output of this refactoring was aligned to the ontology FormalSemantics. Below are listed some main rules defined:

1. *LMM_L1:Meaning(?x)* $\rightarrow$ *dul:Set(?x)*

2. *LMM_L1:Reference(?x) → FormalSemantics:SetElement(?x)*
3. *dul:Relation(?x) → FormalSemantics.owl:Relation(?x)*
4. *dul:Concept(?x) → FormalSemantics.owl:Class(?x)*

All these rules are immediately clear and they do not need any further explanation but they infer, for any individual declared on the left side of the arrow, the type suggested on the right side.

Finally there was the alignment of the resulting ontology to a logic language expressed by the OWL vocabulary and some main rules were:

- *FormalSemantics:Relation → owl:ObjectProperty*
- *FormalSemantics:Class → owl:Class*
- *FormalSemantics:Set → owl:Class*

The last aligned ontology was the final output of the test. All the inferences performed by the Pellet reasoner needed approximately 900 seconds.

The refactored ontology was published at
http:/wwww.ontologydesignpatterns.org/ont/WNet/wnet-ref.owl.

5.2 Testing Semion on other data sources

It was said that Semion is a tool for transforming relational databases into Linked Data and this capability was tested on the WornNet database in its MySQL version.
However, Semion and the transforming methodology it implements is thought, in potentia, to manage any kind of data source.

5.2.1 Evaluation scenario: PGN and LaTex chess

Semion was also tested for transforming into Linked Data non-relational databases consisting in PGN (Portable Game Notation) [Wikc] repositories and in LaTex [Lam86] documents, both containing information about chess games.
PGN files and chess LaTex documents store information about games using two different syntaxes, but they organize this information in a similar way. For example the code in figure 5.2 is expressed in PGN:

```
[Event "F/S Return Match"]
[Site "Belgrade, Serbia Yugoslavia|JUG"]
[Date "1992.11.04"]
[Round "29"]
[White "Fischer, Robert J."]
[Black "Spassky, Boris V."]
[Result "1/2-1/2"]

1. e4 e5 2. Nf3 Nc6 3. Bb5 {This opening is called the Ruy
Lopez.} 3... a64. Ba4 Nf6 5. O-O Be7 6. Re1 b5 7. Bb3 d6
8. c3 O-O 9. h3 Nb8 10. d4 Nbd7 11. c4 c6 12. cxb5 axb5
13. Nc3 Bb7 14. Bg5 b4 15. Nb1 h6 16. Bh4 c5 17. dxe5 Nxe4
18. Bxe7 Qxe7 19. exd6 Qf6 20. Nbd2 Nxd6 21. Nc4 Nxc4
22. Bxc4 Nb6 23. Ne5 Rae8 24. Bxf7+ Rxf7 25. Nxf7 Rxe1+
26. Qxe1 Kxf7 27. Qe3 Qg5 28. Qxg5 hxg5 29. b3 Ke6
30. a3 Kd6 31. axb4 cxb4 32. Ra5 Nd5 33. f3 Bc8 34. Kf2 Bf5
35. Ra7 g6 36. Ra6+ Kc5 37. Ke1 Nf4 38. g3 Nxh3 39. Kd2 Kb5
40. Rd6 Kc5 41. Ra6 Nf2 42. g4 Bd3 43. Re6 1/2-1/2
```

Figure 5.2: A PGN file.

While the LaTex syntax for chess games is in figure 5.3:

In both formats, the chess moves are given in algebraic chess notation, in which each square of the chessboard is identified with a unique pair of a letter and a number. The vertical files are labeled *a* through *h*, from White's left (i.e. the queenside) to his right. Similarly, the horizontal ranks are numbered from 1 to 8, starting from White's home rank. Each square of the board, then, is uniquely identified by its file letter and rank number. The white king, for example, starts the game

```
\title{{\bf Kasparov Karpov 5}\\
Chess World Championship\\
NewYork-Lion 1990}

\section{Game 1:\\ New York, Oct 8, 1990}

{\sc Karpov-Kasparov}

{\sc King's Indian Defense (E81/14)}

\newgame

\move d2d4 g8f6
\move c2c4 g7g6
\move b1c3 f8g7
\move e2e4 d7d6
\ply f2f3


|5 Nf3| - games 3,5,7.
Some months ago Kasparov has lost a game as Black
in this variant against Gulko.

\ply e8g8

\move c1e3 c7c6
```

Figure 5.3: A LaTex document with skak macros.

on square e1. The black knight on b8 can move to a6 and c6. Chess notations are a way to determine any unique point on the board. Again, in the two code are stored informtaion regarding the event (i.e. [Event "F/S Return Match"] in PGN and in LaTex), the site, the date, the white player, the black player and so on.

5.2.2 PGN and LaTex chess to RDF

PGN and LaTex chess represent chess games and moves in a similar way, even if they are expressed in two different syntaxes and are used for two different purposes (the first one to record chess games and the

second one also to present them in electronic document like PDF).
The reason of the similarity between the two notations is to be found in their aim to map chess games, with event, site, players, moves and comments. So the structure of a chess game is, in a certain way, the schema of the two formats thought as data sources and on this schema it was constructed the OWL-DL vocabulary that is used by Semion to represent chess games extracted from PGN or LaTex sources. The figure 5.4 shows the diagram of the vocabulary.

The class *xc:Game* represents the concept of a chess game and is related to the Round attribute in PGN files and to the first part in the sections of chess LaTex documents (e.g. \section{Game 2:...}). Both PGN and LaTex associate a game to an event, so individuals of *xc:Game* are member of *xc:Championship* individuals, that map any kind of chess event.
Again, both formats associate a player to white or black chess pieces and this is reflected in the OWL vocabulary specializing the agent role pattern [GP]. In fact, individuals of the class *Person* (subclass of *agentorole:Agent*) can be related by *xs:playAs* (sub property of *objectrole:hasRole*) or to individuals of *WhitePlayer* or to individuals of *BlackPlayer* (sublcasses of *objectrole:Role*).
The information about the date in which a match occurs, is represented as an instances of the class *timeinterval:Timeinterval* from the time interval content pattern [Preb].
In a game, the moves are represented as instances of the class *xc:ChessMoveInGame* that is formally a particular move (individual of the class *xc:ChessMove*) occuring in a specic game. So a *xc:ChessMove* individual is any possible move of a chess piece in the chessboard. The knowledge representation schema for moves is obtained by specializing the ontology design pattern called object role [Prea].

What it is needed now is the description of the mapping between Java objects, obtained by parsing PGN or LaTex document, and the OWL vocabulary. The mapping vocabulary will be not analyzed because it is very trivial as it is just a collection of triples asserting <*Java object*> *map:mapsTo* <*xc vocabulary class*>.
The Semion reengineering module was tested both on a PGN file and on a LaTex documents and the resutls were two RDF datasets expressed by using the vocabulary that has just been described.

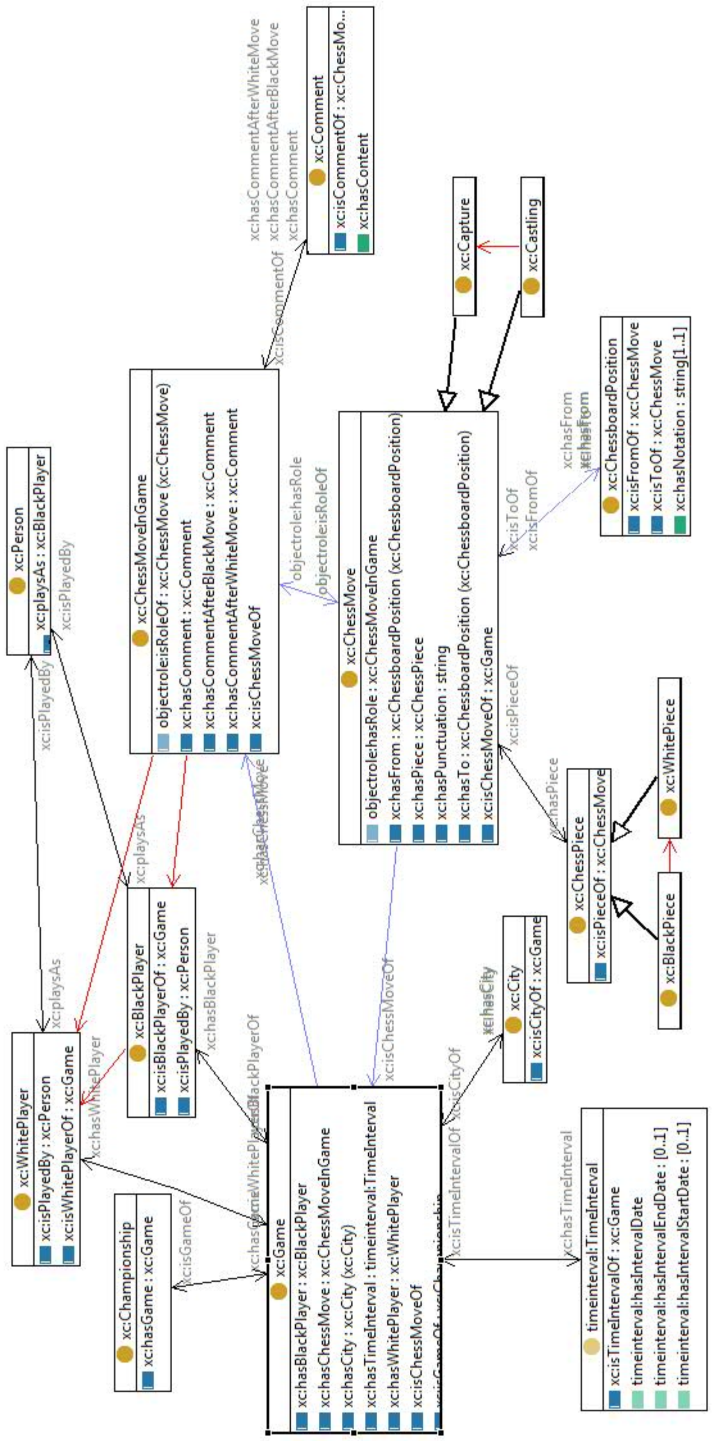

xc:Comment
xc:isCommentOf : xc:ChessMo...
xc:hasContent
xc:hasCommentAfterWhiteMove
xc:hasCommentAfterBlackMove
xc:hasComment
xc:isCommentOf
xc:Capture
xc:Castling
xc:Person
xc:playsAs : xc:BlackPlayer
xc:isPlayedBy
xc:ChessMoveInGame
objectrole:isRoleOf : xc:ChessMove (xc:ChessMove)
xc:hasComment : xc:Comment
xc:hasCommentAfterBlackMove : xc:Comment
xc:hasCommentAfterWhiteMove : xc:Comment
xc:isChessMoveOf
objectrole:hasRole
objectrole:isRoleOf
xc:ChessMove
objectrole:hasRole : xc:ChessMoveInGame
xc:hasFrom : xc:ChessboardPosition (xc:ChessboardPosition)
xc:hasPiece : xc:ChessPiece
xc:hasPunctuation : string
xc:hasTo : xc:ChessboardPosition (xc:ChessboardPosition)
xc:isChessMoveOf : xc:Game
xc:ChessboardPosition
xc:isFromOf : xc:ChessMove
xc:isToOf : xc:ChessMove
xc:hasNotation : string[1..1]
xc:hasFrom
xc:isToOf
xc:isFromOf
xc:isPieceOf
xc:hasPiece
xc:ChessPiece
xc:isPieceOf : xc:ChessMove
xc:WhitePiece
xc:BlackPiece
xc:playsAs
xc:BlackPlayer
xc:isBlackPlayerOf : xc:Game
xc:isPlayedBy : xc:Person
xc:hasBlackPlayer
xc:isBlackPlayerOf
xc:hasChessMove
xc:isChessMoveOf
xc:City
xc:isCityOf : xc:Game
xc:hasCity
xc:isCityOf
xc:WhitePlayer
xc:isPlayedBy : xc:Person
xc:isWhitePlayerOf : xc:Game
xc:hasWhitePlayer
xc:isWhitePlayerOf
xc:Championship
xc:hasGame : xc:Game
xc:isGameOf
xc:hasGame
xc:Game
xc:hasBlackPlayer : xc:BlackPlayer
xc:hasChessMove : xc:ChessMoveInGame
xc:hasCity : xc:City (xc:City)
xc:hasTimeInterval : timeinterval:TimeInterval
xc:hasWhitePlayer : xc:WhitePlayer
xc:isChessMoveOf
xc:isTimeIntervalOf
xc:hasTimeInterval
timeinterval:TimeInterval
xc:isTimeIntervalOf : xc:Game
timeinterval:hasIntervalDate
timeinterval:hasIntervalEndDate : [0..1]
timeinterval:hasIntervalStartDate : [0..1]

Figure 5.4: Diagram of the OWL vocabulary for chess games.

The chess dataset was published at
http:/wwww.ontologydesignpatterns.org/ont/chess/chess.rdf.

5.2.3 Refactoring the chess dataset

After having tranlated the PGN and the chess LaTex original sources, it was tested the refactoring module on the resulting datasets.
To this purpose they were planned two different refactoring approaches in order to align the dataset to:

- LMM, FormalSemantics and OWL as it was explained in section 5.1.3
- DBpedia and GeoNames that give to Semion a real Linked Data case study.

The alignment to LMM organizes the dataset in a semiotic-cognitive way, as it was explained that the LMM core is an OWL implementation of the semiotic triangle of Peirce [Pei31]. As Semion performs alignments reasoning on sets of rules expressed in SemionRule, that is the syntax to succinctly represent SWRL [HPSB$^+$04] rules defined in section 4.6, they were defined rules like the following:

1. *xc:ChessMove(?x)* $\rightarrow$ *LMM_L1:Meaning(?x)*
   This rule infers that any individuals of the class *xc:ChessMove*, that represents any possible move in the chessboard of any possible chess piece, is also an individual of the class *LMM_L1:Meaning*, that represents concepts in the semiotic triangle (see figure 3.2).
2. *xc:ChessMoveInGame(?x)* $\rightarrow$ *LMM_L1:Reference(?x)*
   This rule infers that any individuals of the class *xc:ChessMoveInGame*, that represents the specific move in a game, is a *LMM_L1:Reference*, that represents concrete objects in the semiotic triangle.
3. *xc:Comment(?x)* $\rightarrow$ *LMM_L1:Expression(?x)*
   This rule infers that any individuals of the class *xc:Comment*,

that represents the comment related to a specific move in a game, is a *LMM_L1:Expression*, that represents natural language terms, symbols in formal languages, icons, and whatever can be used as a vehicle for communication.

4. *xc:ChessMove(?x) .*
   xc:ChessMoveInGame(?y) .
   objectrole:hasRole(?x, ?y)
   → *LMM_L1:interprets(?x, ?y)*
   This rule infers that the property *objectRole:hasRole* occurring between two individuals of the classes *xs:ChessMove* and *xs:ChessMoveInGame* as to be intended as *LMM_L1:interprets*, that is the property that relates concepts to concrete ojbects.

5. *xc:ChessMoveInGame(?x) .*
   xc:Comment(?y) .
   xc:hasComment(?x, ?y)
   → *LMM_L1:denotes(?y, ?x)*
   This rule infers that the property *xc:hasComment*, that associates a comment to a specific move in a game, has to be intended as a denotation between an expression and a concrete object.

The refactoring rules defined to align the dataset to FormalSemantics, in order to express the dataset itself in the language that describes the formal semantics, and to OWL, that formally describes a logic language, were the same chosen for the previous test with relational databases.

The second refactoring test involved existent Linked Data source that can be seen in the bouble diagram in figure 1.5 and they are DBpedia and GeoNames.
For this reason rules like the following were written:

1. *xc:City(?x) .*
   same(?x, games:NewYork)
   → *same(?x, geonames:5128581)*
   that infers that the individual *games:NewYork* of the class *xc:City* in our dataset is the same individual in *geonames:5128581* that is the resource associated to the city of New York.

2. *xc:Person(?x) .*
   same(?x, games:Kasarov)
   → same(?x, dbpedia:Kasparov)
   that infers that the person Kasparov in our dataset is the same Kasparov in DBpedia.

3. *xc:Game(?x) → rdfs:seeAlso(?x, dbpedia:Chess_match)*
   that allows a refrence to the DBpedia resource that contains information about chess matchs from individuals of *xc:Chess*.

4. *same(xc:WhiteBishop, ?x) → rdfs:seeAlso(?x, dbpedia:Bishop_(chess))*
   that, as before, allows a reference to the DBpedia resource that contains information about the bishop chess piece from the individul *xc:WhiteBishop* in the dataset.

When published in the Linked Data, the dataset is available to be queried via SPARQL and an user or an agent can request information, not only limited about specific games and moves, but also about the place in which the event took place (e.g. the city of New York in GeoNames) or detailed information about the two players of the game (e.g. Kasparov and Karpov in DBpedia) or, again, information regarding chess pieces (e.g. the bishop in DBpedia).
Considering a particular scenario in which other datasets containing information about any domain regarding chess, such as chess World Championships, famous chess players or chess tactics, were published in the Linked Data, it could be possible and simple with Semion to connect all this datasets as they were a whole knowledge base.

The refactored ontology was published at
http:/wwww.ontologydesignpatterns.org/ont/chess/chess-ref.owl.

Chapter 6

Conclusion and future work

In this thesis work it was studied, designed and implemented a customized method for transforming relational databases into Linked Data [BHBL09].
Though the subject chosen as case study was relational databases, this method has to be intended applicable to any kind of structured data source. In fact, it bases the reengineering process on the description of the data source schema as an RDF/OWL model made following a well defined vocabulary.
This feature is not completely new, but it was suggested from some tools like D2R [BC07] that uses a mapping driven approach in translating relational databases into RDF, in which the whole reengineering is driven by the definition, in formal terms of a declarative language (D2RQ for the D2R tool), of the mapping between the physical data source and RDF. But this approach limits the description only on the mapping, while we are interested also about the schema of the data.
Furthermore these existing mapping driven approach tools makes a lot of assumption on the RDF dataset that can be good for general and theoretical situations, but may fail in many real cases in which databases are not well projected. In fact they general work mapping:

- a database table to a RDFS Class
- a column name of a database table to a RDF property

- a database table record to a RDF node
- a cell of a RDB table to a value

As introduced this approach may be not completely useful as for most of the existing databases. Many entities, that can be potentially RDFS Class, are not in the form of table records but some implicit forms, e.g. as for a table describing information about persons, it includes a column called “birth place” which describes the city where the person was born and each value in this column is a name of some city and it should refer to a concerned record within the table Location, but it is only a data with type “varchar”. Now if we apply the general mapping, we obtain that a person, represented by an RDF resource, has some relation that says where he was born in some city, but this city is only a literal and not another resource that semantically identifies the city.
In terms of Linked Data this is a crucial point, as if we have a lot of datasets in the Web of Data but they are not connected each other, in some way we fail our objective.
For this reason in Semion the reengineering is driven by a mapping that says to the software what is a physical object of the database in the domain specific vocabulary we defined during this work, namely semionDB.owl. As result a database table will be an instance of the class *Table* of this vocabulary, a column will be a *Field* and a record a *Record* and no further assumption is made in terms of RDF Schema or OWL.
Publishing a dataset that uses as vocabulary the representation of the database schema is really the aim we want to reach, because behind tables, columns and records are hidden a lot of possible concepts that they do not necessary mean just “table”, “column” or “record”. To solve this problem it was studied a methodology in order to align reengineered datasets to any possible Linked Data vocabulary (e.g. SKOS, FOAF) or to any possible ontology design pattern. In this field was analyzed the alignment to LMM [PGG08], a Linguistic Meta-Model that provides a semiotic-cognitive representation of linguistic knowledge and grounds it in a formal semantics. The most important feature of LMM is its ability to support the representation of different knowledge sources developed according to different underlying semiotic theories. This is possible because most knowledge representation

schemata, either formal or informal, can be put into the context of so-called semiotic triangle [Pei31], that is used to discuss the differences between objects, concepts and symbols.
Of course LMM is only a possible alignment choice and one may want to align an already LMM aligned dataset to another ontology in order to reorganize LMM itself within another vocabulary.
The most important thing in the method we defined is that as much as possible anything is customized.
The final result of the thesis work is Semion a graphical tool composed by two main independent modules:

- the reengineering module for relational databases translation into RDF datasets
- the refactoring module for ontology alignment

The project has not to be considered accomplished, but ongoing work because Semion wants to be a tool able to manage heterogeneous data sources and not only relational databases. The aim is to project and implement a tool that is helpful to translate into Linked Data any kind of legacy content source and following this way it is part of the Interactive Knowledge Stack (IKS) project [IP], that is an Integrating Project part-funded by the European Commission. It started in January 2009 and will provide an open source technology platform for semantically enhanced content management systems.
Another issue that will be analyzed is the possibility to use the Rule Interchange Format (RIF) [dB07] instead of SWRL [HPSB$^+$04], that is currently used, to represent alignment rules in the refactoring module. At the state of the art RIF is a W3C working draft and is a format for interchanging rules over the Web. Rules that are exchanged using RIF may refer to external data sources and may be based on data models that are represented using a language different from RIF.
The tool will be extended with a package manager that enables users to add plug-ins developed separately from Semion but designed to work in cooperation with it. This feature will be useful, for instance, in order to manage different data soures as Semion plug-ins.

Appendix A

Engineering details

The three main components of the Semion tool, namely the graphical user interface, the reengineering module and the refactoring module, were built as three different java projects. Specifically, the reengineering and the refactoring modules were implemented as two completely independent components in order to possibly use them separately, while the graphical user interface, as it plays the role of being a bridge to user for the two former modules, imports both projects.
Figure A.1 shows the diagram about the project depedencies existing among the three java projects that are the core of the Semion tool.

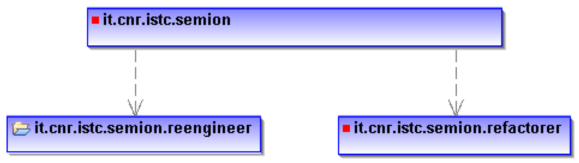


Figure A.1: Projects dependencies in the Semion tool.

Expanding each project in the diagram in figure A.1, it is possible to see that, as figure A.2 shows, the *it.cnr.istc.semion*, that contains classes and methods related to the graphical user interface, is compoed by the following packages:

- *it.cnr.istc.semion*, that contains tha basic classes to build the user interface
- *it.cnr.istc.semion.workspace*, that contains classes in order to manage the user workspace during the working sessions
- *it.cnr.istc.semion.treeNode*, that contains interfaces, abstract classes and classes that represent Java objects for the tree viewer of the Project Explorer (see figure 4.10)

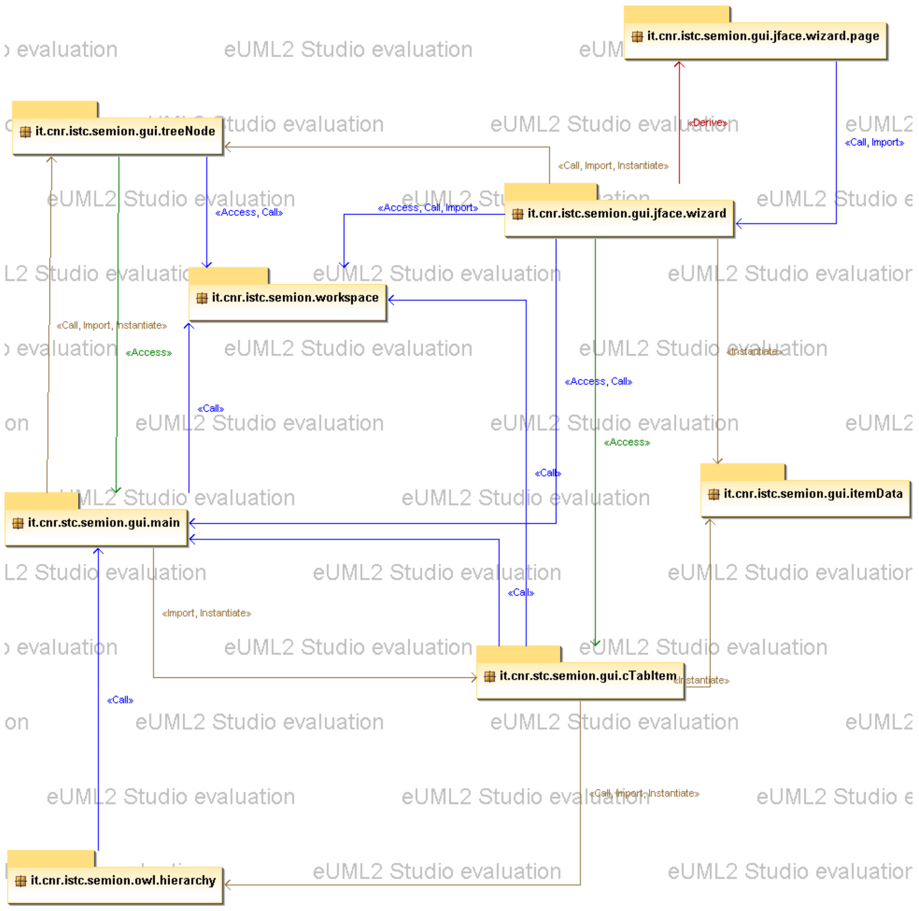

it.cnr.istc.semion.gui.jface.wizard.page
it.cnr.istc.semion.gui.treeNode
«Derive»
«Call, Import»
«Call, Import, Instantiate»
«Access, Call»
«Access, Call, Import»
it.cnr.istc.semion.gui.jface.wizard
it.cnr.istc.semion.workspace
«Call, Import, Instantiate»
«Access»
«Instantiate»
«Access, Call»
«Call»
«Access»
«Call»
it.cnr.istc.semion.gui.itemData
it.cnr.stc.semion.gui.main
«Call»
«Import, Instantiate»
it.cnr.stc.semion.gui.cTabItem
«Instantiate»
«Call»
«Call, Import, Instantiate»
it.cnr.istc.semion.owl.hierarchy
eUML2 Studio evaluation

Figure A.2: Projects dependencies in the Semion tool.

- *it.cnr.istc.semion.cTabItem*, that contains interfaces, abstract classes and classes that represent Java object for the tab elementes of the tab viewer(see figure 4.10)

- *it.cnr.istc.semion.ItemData*, that contains classes for representing the inner data associated to tab items
- *it.cnr.istc.semion.hierarchy*, that contains some utility classes that ensure to infer RDF/OWL classes and properties hierarchies
- *it.cnr.istc.semion.wizard* and *it.cnr.istc.semion.wizardPage*, that allows to Semion to display wizard menus in order to perform action like adding a new transformation from a database or creating a new project

The project *it.cnr.istc.semion.reengineer* has the following packages, as figure A.3 shows:

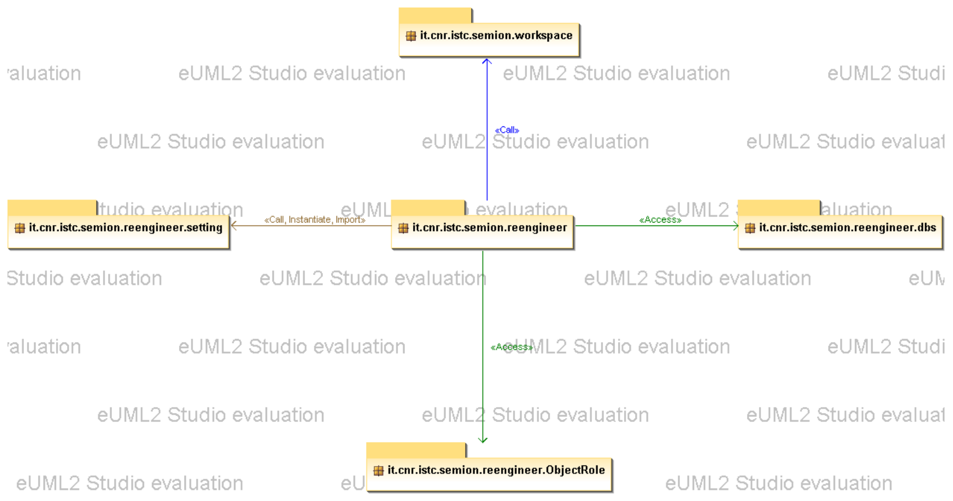

it.cnr.istc.semion.workspace
«Call»
it.cnr.istc.semion.reengineer.setting
«Call, Instantiate, Import»
it.cnr.istc.semion.reengineer
«Access»
it.cnr.istc.semion.reengineer.dbs
«Access»
it.cnr.istc.semion.reengineer.ObjectRole
eUML2 Studio evaluation

Figure A.3: Package dependencies in the *it.cnr.istc.semion.reengineer* project.

- *it.cnr.istc.semion.reengineer*, that is the core package of the project and accesses and calls all the other packages
- *it.cnr.istc.semion.reengineer.dbs*, that contains a class that allows to manage the *semionDB* ontology via the Jena framework

- *it.cnr.istc.semion.reengineer.objectrole*, that contains a class that allows to manage the *object role* content pattern via the Jena framework
- *it.cnr.istc.semion.reengineer.setting*, that contains classes and methods in order to connect and query a relational database via JDBC [FEB03]
- *it.cnr.istc.semion.reengineer.workspace*, that contains classes in order to manage the user workspace during the working sessions

Finally the project *it.cnr.istc.semion.refactorer* is composed by the packages shown in figure A.4, and thery are:

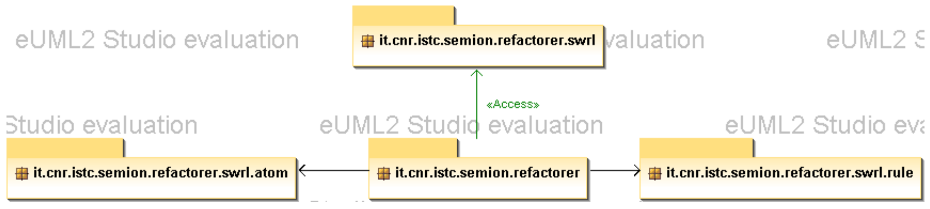

it.cnr.istc.semion.refactorer.swrl
«Access»
it.cnr.istc.semion.refactorer.swrl.atom
it.cnr.istc.semion.refactorer
it.cnr.istc.semion.refactorer.swrl.rule
eUML2 Studio evaluation

Figure A.4: Package dependencies in the *it.cnr.istc.semion.refactor* project.

- *it.cnr.istc.semion.refactorer*, that is the core package of the project and accesses and calls all the other packages. In this package is performed the reasoning on sets of rules for the alignments
- *it.cnr.istc.semion.refactorer.swrl*, that contains classes and methods that allow to tranlate from SWRL [HPSB+04] to SemionRule and vice versa
- *it.cnr.istc.semion.refactorer.swrl.atom*, that contains classes and methods that implement the SWRL atoms in SemionRule
- *it.cnr.istc.semion.refactorer.swrl.tule*, that contains classes and methods that implement the SWRL rules in SemionRule

The three projects are composed by 11728 lines of code.

Bibliography

[AH08] Grigoris Antoniou and Frank van Harmelen. *A Semantic Web Primer*. The MIT Press, second edition, 2008.

[BC07] Christian Bizer and Richard Cyganiak. D2RQ - Lessons Learned. *W3C Workshop on RDF Access to Relational Databases*, October 2007.

[BCG07] Christian Bizer, Richard Cyganiak, and Tobias Gauss. The RDF Book Mashup: From Web APIs to a Web of Data. In Sren Auer, Christian Bizer, Tom Heath, and Gunnar Aastrand Grimnes, editors, *SFSW*, volume 248 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[Bec04] Dave Beckett. RDF/XML Syntax Specification (Revised). W3C recommendation, W3C, February 2004. http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/.

[BG04a] Dave Beckett and Jan Grant. RDF Test Cases. W3C recommendation, W3C, February 2004. http://www.w3.org/TR/2004/REC-rdf-testcases-20040210/.

[BG04b] Dan Brickley and Ramanathan V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. W3C recommendation, W3C, February 2004. http://www.w3.org/TR/2004/REC-rdf-schema-20040210/.

[BHBL09] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked Data - The Story So Far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009.

[BLFM05] Tim Berners-Lee, Roy Fielding, and Larry Masinter. RFC 3986: Uniform Resource Identifier (URI): Generic Syntax. Technical report, W3C/MIT, http://www.ietf.org/rfc/rfc3986.txt, January 2005.

[BLHL01] Tim Berners-Lee, James A. Hendler, and Ora Lassila. The semantic web. *Scientific American*, 284(5):34–43, 2001.

[BS01] Franz Baader and Ulrike Sattler. An Overview of Tableau Algorithms for Description Logics. *Studia Logica*, 69:5–40, 2001.

[dB07] Jos de Bruijn. RIF RDF and OWL Compatibility. W3C working draft, W3C, October 2007. http://www.w3.org/TR/2007/WD-rif-rdf-owl-20071030.

[FEB03] Maydene Fisher, Jon Ellis, and Jonathan Bruce. *JDBC API Tutorial and Reference.* Addison-Wesley, Boston, MA, 2003.

[GGM$^+$02] Aldo Gangemi, Nicola Guarino, Claudio Masolo, Alessandro Oltramari, and Luc Schneider. Sweetening Ontologies with DOLCE. In *EKAW '02: Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web*, volume 2473 of Lecture Notes in Computer Science, pages 166–181, London, UK, October 2002. Springer-Verlag.

[GJSB05] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *Java(TM) Language Specification, The (3rd Edition) (Java (Addison-Wesley)).* Addison-Wesley Professional, 2005.

[HM04] Frank van Harmelen and Deborah L. McGuinness. OWL Web Ontology Language Overview.

W3C recommendation, W3C, February 2004. http://www.w3.org/TR/2004/REC-owl-features-20040210/.

[HPSB$^+$04] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C member submission, W3C, May 2004. http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/.

[KFNM04] Holger Knublauch, Ray W. Fergerson, Natalya Fridman Noy, and Mark A. Musen. The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications. In *International Semantic Web Conference*, pages 229–243, 2004.

[Lam86] L. Lamport. *LaTeX: A Document Preparation System.* Addison-Wesley, 1986.

[Mil95] George A. Miller. WordNet: A lexical database for English. *Communications of the ACM*, 38(1):39–41, 1995.

[Pei31] Charles S. Peirce. *Collected Papers of Charles Sanders Peirce.* Harvard University Press, 1931.

[PGG08] Davide Picca, Alfio M. Gliozzo, and Aldo Gangemi. LMM: an OWL-DL MetaModel to Represent Heterogeneous Lexical Knowledge. In *Proceedings of the Sixth International Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco, May 2008. European Language Resources Association (ELRA). http://www.lrec-conf.org/proceedings/lrec2008/.

[Pre04] Roger S. Pressman. *Software Engineering: A Practitioner's Approach.* McGraw-Hill Science/Engineering/Math, sixth edition, April 2004.

[PS08] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. W3C recommendation, W3C, January 2008. http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/.

[RLHJ99] Dave Raggett, Arnaud Le Hors, and Ian Jacobs. HTML 4.01 specification. W3C recommendation, W3C, December 1999. http://www.w3.org/TR/1999/REC-html401-19991224.

[RN03] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003.

[SMYM$^+$08] C. Michael Sperberg-McQueen, François Yergeau, Eve Maler, Jean Paoli, and Tim Bray. Extensible Markup Language (XML) 1.0 (Fifth Edition). W3C proposed edited recommendation, W3C, February 2008. http://www.w3.org/TR/2008/PER-xml-20080205.

[vAGS06] Mark van Assem, Aldo Gangemi, and Guus Schreiber. RDF/OWL Representation of WordNet. W3C working draft, W3C, April 2006. http://www.w3.org/2001/sw/BestPractices/WNET/wn-conversion-20062304/.

[WA02] Michael Widenius and Davis Axmark. *Mysql Reference Manual*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2002.

[WF04] Priscilla Walmsley and David C. Fallside. XML Schema Part 0: Primer Second Edition. W3C recommendation, W3C, October 2004. http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/.

Webography

[BL89] Tim Berners-Lee. Information Management: A Proposal. http://www.w3c.org/History/1989/proposal.html, 1989. Visited on February 2010.

[BL98] Tim Berners-Lee. Notation 3. http://www.w3.org/DesignIssues/Notation3, 1998. Visited on February 2010.

[BL06] Tim Berners-Lee. Linked Data. Worldwidewebdesignissues, July 2006. Visited on January 2010.

[BM] Dan Brickley and Libby Miller. The Friend Of A Friend (FOAF) Vocabulary Specification. http://xmlns.com/foaf/spec/. Visited on January 2010.

[Ecla] Eclipse. JFace. http://wiki.eclipse.org/index.php/JFace, Visited on January 2010.

[Eclb] Eclipse. SWT: The Standard Widget Toolkit. http://www.eclipse.org/swt/, Visited on January 2010.

[GP] Aldo Gangemi and Valentina Presutti. Agent Role Content Pattern. http://ontologydesignpatterns.org/cp/owl/agentrole.owl. Visited on February 2010.

[IBM] IBM. Structured Query Language (SQL). http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.admin.doc/doc/c0004100.htm. Visited on February 2010.

[IP] IKS-Project. Interactive knowledge stack for small to medium cms/kms providers. http://www.iks-project.eu/. Visited on February 2010.

[Prea] Valentina Presutti. Object Role Content Pattern. http://ontologydesignpatterns.org/cp/owl/objectrole.owl. Visited on February 2010.

[Preb] Valentina Presutti. Time Interval Content Pattern. http://ontologydesignpatterns.org/cp/owl/timeinterval.owl. Visited on February 2010.

[Rul] RuleML. The Rule Markup Initiative. http://ruleml.org/. Visited on January 2010.

[Sou] Sourceforge. Jena A Semantic Web Framework for Java. http://jena.sourceforge.net/. Visited on January 2010.

[Top] TopQuadrant. TopBraid Composer. http://www.topbraidcomposer.com/index.html. Visited on January 2010.

[W3C] W3C. Linking Open Data. http://esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpenData. Visited on February 2010.

[Wika] Wikipedia. Comma-separated values. http://en.wikipedia.org/wiki/Comma-separated_values. Visited on February 2010.

[Wikb] Wikipedia. Modelviewcontroller. http://en.wikipedia.org/wiki/Model-View-Controller. Visited on February 2010.

[Wikc] Wikipedia. Portable game notation. http://en.wikipedia.org/wiki/Portable_Game_Notation. Visited on February 2010.

[Wikd] Wikipedia. Semantic Web Stack. http://en.wikipedia.org/wiki/Semantic_Web_Stack. Visited on February 2010.