VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SHARING AND REUSING SEMANTIC QUERIES FOR SEARCHING THE WEB

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE
AUTHOR

Tomáš Taraba

BRNO 2009

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SDÍLENÍ A ZNOVUPOUŽITÍ SÉMANTICKÝCH DOTAZŮ PRO VYHLEDAVANÍ NA WEBU

SHARING AND REUSING SEMANTIC QUERIES FOR SEARCHING THE WEB

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE Tomáš Taraba
AUTHOR

VEDOUCÍ PRÁCE Isabelle Mirbel
SUPERVISOR

BRNO 2009

Abstrakt

Táto práca pojednáva o vývoji grafického uživatelského prostredia pre kreslenie alebo generovanie záujmových máp využitím cieľov a stratégii.

Klíčová slova

Semantic, Web, Java, GUI, RDF, RDFS, XML, Goal

Abstract

This work is about developing a graphical user interface for drawing or generating intentional maps using goals and strategies.

Keywords

Semantic, Web, Java, GUI, RDF, RDFS, XML, Goal

Citace

Taraba Tomáš: Sharing and reusing semantic queries for searching the web. Brno, 2009, bakalářská práce, FIT VUT v Brně.

Sharing and reusing semantic queries for searching the web

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením prof. Isabelle Mirbel a doc. Černockého.
Další informace mi poskytli Pierre Crescenzo.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……………………
Jméno Příjmení
Datum

Poděkování

I would like to thank my supervisor teacher Isabelle Mirbel and doc. Jan Černocký who both allowed me to work on this research. Also my thank you is pointed at teacher Pierre Crescenzo who advised me in project implementation and all other professors who helped me that I could study at Université de Nice – Sophia Antipolis.

Contents

Introduction

Knowing, that somebody has previously solved a complex problem, but we are not able to find again the solution to that problem, record or note, can be shared with other process to help them solving a similar problem. It is not unusual that it exists a refusal to rework a solution, because some time has already been wasted to find a solution to a problem once again.

In fact, retrieving a previous solution may take much more time than redoing the past work. If we have a general look on the situation, it is obvious, that we are actively managing only a little part of all the information, which we are creating. The final effect of this is loosing of productivity and reducing the content profit.

The World Wide Web has dramatically changed the availability of electronically accessible information. According to recent survey, Web contains more than 3 billion of static documents [11] and at the same time, this huge amount of documents makes searches, accesses and management of relevant information very difficult. More or less, we can suppose that it is because document content is presented primary in human-readable language. And there is a large gap between information for automated processing and information kept in natural language for humans.

By looking for the answer of the common problem there was created several researches for enriching information by machine-readable semantics. This step set the basic advance in creating the term Semantic web and his abilities for sharing and reusing the semantic content of the web.

1 Content of the work

1.1 XML

XML allows specifying an application-independent documents and data. It has a standard syntax for meta data and a standard structure for both documents and data.

Listing 1.1. shows an example of XML document (written export of text application in which we know only binary representation, but not the meaning of it) on which it can be obviously seen the simplicity of application understanding, with usage of open standard syntax and verbose descriptions of meaning of data. XML is readable and understandable for everyone, including applications or humans, not only by the application and person who developed it. This is fundamental underpinning of the web, because it is not possible to predict the variety of systems, in fact the applications which will be managing the content on the Web. XML documents can be searched more easily than web pages, in contrast of searching in the binary data in which are documents stored by the application in which they were created and which are known and readable only by own application.

XML offers easy and standard syntax for encoding information or meaning of meta data. In the other words, it provides a masculine mechanism for encoding information and meaning of the meta data. The essential accomplishment is a standard structure convenient for expressing information for both documents and data fields. This structure is using a hierarchy or more often a tree structure. An example of this structure is shown in Figure 1.1.

```
<?xml version=”1.0” encoding=”UTF-8”?>
<!DOCTYPE office:document-content PUBLIC “-//OpenOffice.org//DTD
Office-
Document 1.0//EN” “office.dtd”><office:document-content
xmlns:office=”http://openoffice.org/2000/office”
xmlns:script=”http://openoffice.org/2000/script”
office:class=”text”
office:version=”1.0”>
<office:script/>
<office:font-decls>
<style:font-decl style:name=”Thorndale” fo:font-family=”Thorndale”
style:font-family-generic=”roman” style:font-pitch=”variable”/>
</office:font-decls>
<office:automatic-styles/>
<office:body>
<text:sequence-decls>
</text:sequence-decls>
<text:p text:style-name=”Standard”>We like Semantic Web!</text:p>
</office:body>
</office:document-content>
```

Listing 1.1 XML example of document.

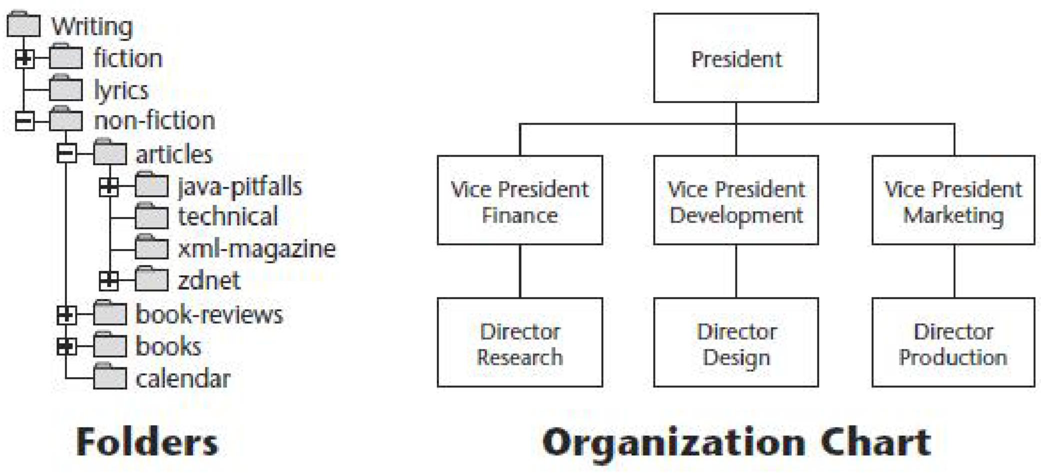


Figure 1.1 Example of XML tree.

The last, but not the less important fact is, that XML. It is a subset of the Standardized Generalized Markup language (SGML) from year 1969 by Dr. Charles Goldfarb, Ed Mosher and Ray Lorie. In small nutshell, XML is “SGML for the Web.” XML is a set of syntax rules for creating semantically rich markup languages in a particular domain. Furthermore, applying XML understand to create new languages. The key principle of XML is *markup is separate from content*. According to this principle, markup can surround or contain content. Table 1.2 shows examples of xml tags (markups).

TAG TYPE	EXAMPLE
Start tag	<description>
End tag	</description>
Empty tag	<name />

Table 1.1 Example of XML Tags.

1.2 Semantic Web

1.2.1 What is a semantic web ?

We can say that there exists two parts of vision for the future of World Wide Web. First part means to make Web more collaborative medium. Second part consists of making a web understandable, and so machine-processable. According to (Berners-Lee et al.)[9] Figure 1.2 is the original diagram of the vision. In this figure we can clearly see the relations between information items like include, describes and wrote. Fortunately, these relations between resources are not currently captured on the web. The technology for technique of capturing information like this is called Resource Description Framework (RDF), which will be described in detail in Section 1.6.

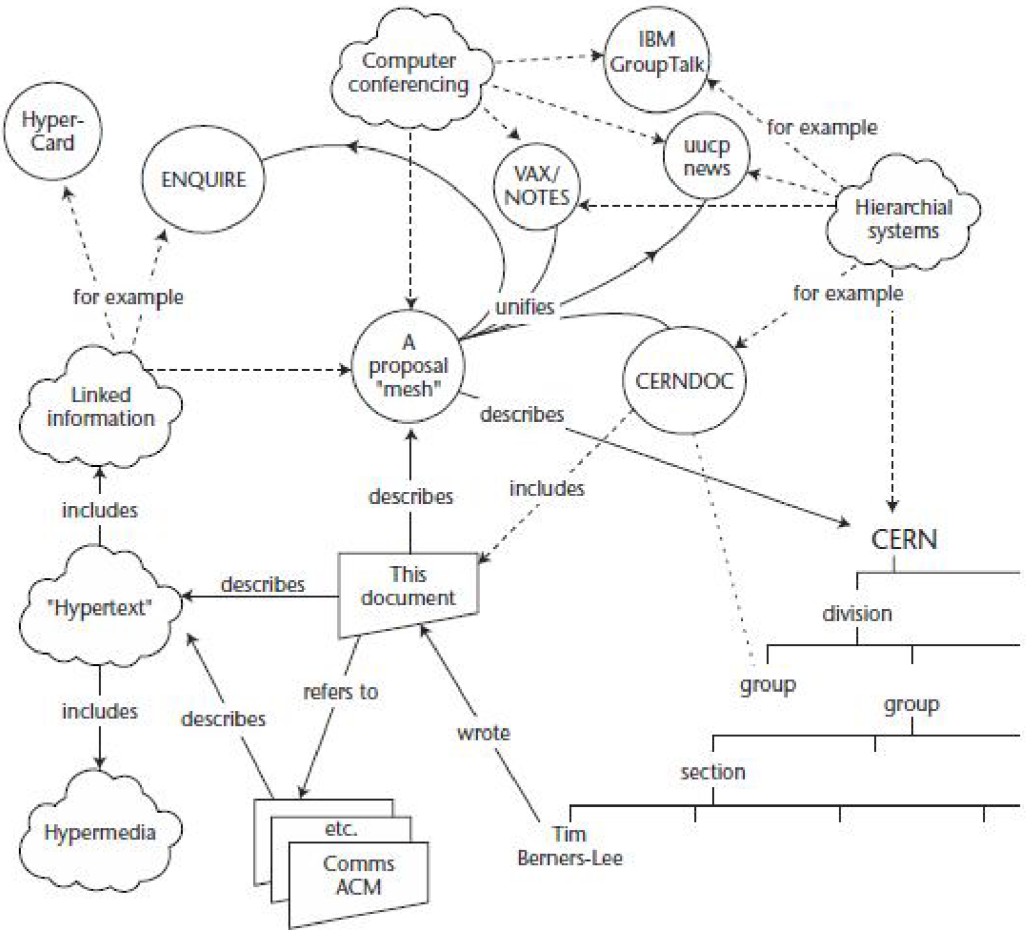


Figure 1.2 Original Web proposal to CERN by Tim Berners-Lee.

Semantic web is not only the World Wide Web. It represents a set of technologies, which will equally work on different corporate architectures. Semantic web models and techniques aim at facing the overloading of information. In [10], Paul Krill indicates, that "This condition results from having a rapid rate of growth in the amount of information available, while days remain 24 hours long and our brains remain in roughly the same state of development as they were when cavemen communicated by scrawling messages in stone".

1.2.2 Meta data

XML meta data are describing the purpose or meaning raw data values by means of text format for simple exchange, interpreting and application-independency. In fact, the meta data are increasing the fidelity of the data. Provide richer data means to provide computers more understanding for them and the more effectively they can handle complex tasks.

The following shows steps which exist beyond the basic understanding of the simple meta data.

1.2.2.1 Semantic Levels

Semantically aware applications require the evolution of data fidelity, which is shown in Figure 1.3. Instead of just meta data, we will have an information stack composed of semantic levels. It is currently at Level 1 with XML Schema, which is represented as modeling the properties of our data classes. We are capturing and processing meta data about isolated data classes like purchase orders, products, employees, and customers. On the left side of the diagram we associate a simple physical metaphor to the state of each level. Level 1 is analogous to describing singular concepts or objects. In Level 2, we will move beyond data modeling (simple meta data properties) to knowledge modeling. Knowledge modeling enables us to model statements both about the relationships between Level 1 objects and about how those objects operate. This is diagrammed as connections between our objects in Figure 1.3.

Beyond the knowledge statements of Level 2 are the superstructures or "closed world modeling" of Level 3. The technology that implements these sophisticated models of system is called ontologies.

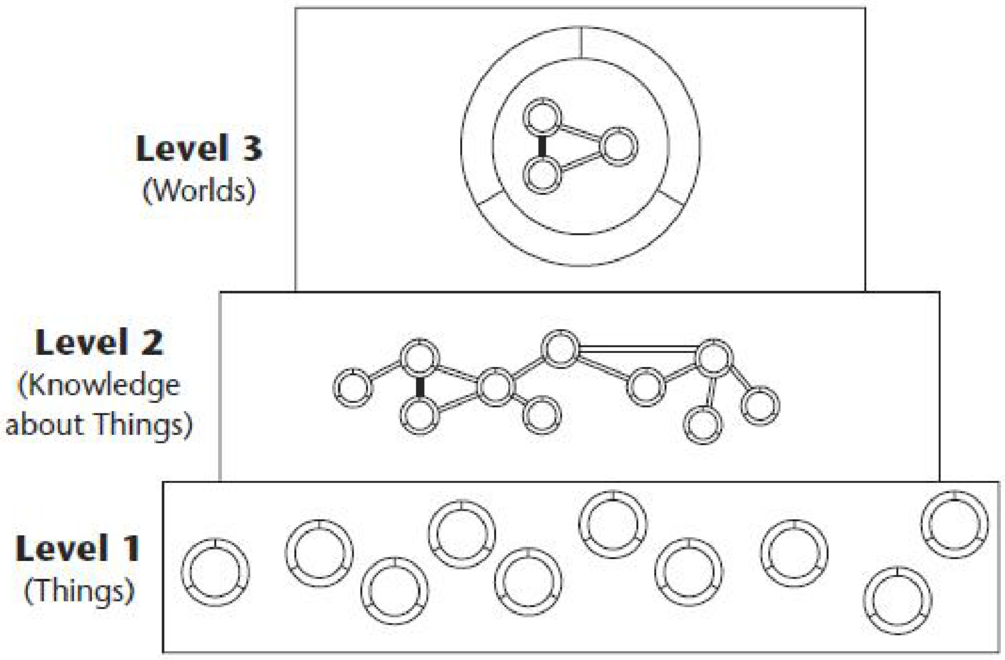


Figure 1.3 Evolution in data fidelity.

1.2.3 RDF

In this section we will focus what is it RDF, why it is not widely spread and how RDF is based on simple model, which is distinct from RDF syntax.

1.2.3.1 What is it RDF

RDF is an XML-based language for describing resources. While definition of word *resource* may be a little wide expression, it is introduced a simple formulation, that resource is a file available through the Web. This kind of resource can be accessed by Uniform Resource Locator (URL). Whereas XML documents include meta data inside their bodies, RDF aims at providing meta data to describe a document in a distinct file. Written in other words, instead of marking up the internals of a document, RDF captures meta data about "externals" of document, for example author, date of creation and type.

A particularly good use of RDF is to describe resources, which are opaque like images or audio files. Listing 1.2 displays an example of RDF file describing image resource. Besides embedding the meta data in the photo, RDF annotations are stored into an external file, as shown in Listing 1.2.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#"
xmlns:s0="http://www.w3.org/2000/PhotoRDF/dc-1-0#"
xmlns:s1="http://sophia.inria.fr/~enerbonn/rdfpiclang#"
xmlns:s2="http://www.w3.org/2000/PhotoRDF/technical-1-0#">
<rdf:Description
rdf:about="http://www.c2i2.com/~budstv/images/shop1.jpg">
<s0:relation>part-of Store Front</s0:relation>
<s0:type>image</s0:type>
<s0:format>image/jpeg</s0:format>
<s1:xmllang>en</s1:xmllang>
<s0:description>Buddy Belden's work bench for
TV/VCR repair</s0:description>
<s2:camera>Kodak EasyShare</s2:camera>
<s0:title>TV Shop repair bench</s0:title>
</rdf:Description>
</rdf:RDF>
```

Listing 1.2 Example of RDF annotations.

The first thing easy to notice is the consistent use of namespaces on all elements in the listing. In the root element **<rdf:RDF>**, four namespaces are declared. The root element specifies this document is an RDF document. An RDF document contains one or more "descriptions" of resources. A description is a set of statements about a resource. The **<rdf:Description>** element contains an rdf:about attribute that refers to the resource being described. In Listing 1.2., the rdf:about attribute points to the URL of a webpage. The rdf:about attribute is critical to understand RDF because all resources described in RDF must be denoted via a URI. The child elements of the Description element are all properties of the resource being described. In listing 1.2, two properties are bolded, one the Dublin Core namespaces and one on the technical namespace. The values of those properties are stored as the element content. In summary, Listing 1.2 has demonstrated a syntax where we describe a resource, a resource's properties, and the property values. This three-part model is separate from the RDF syntax. The RDF syntax in Listing 1.2. is considered to be one (actually of many) serializations of the RDF model.

1.2.3.2 Triple

The RDF model is often called a "triple" because it has three parts, as described previously. Though described in terms of resource properties in the preceding text, in the knowledge representation community, those three parts are described in terms of the grammatical parts of a sentence: subject, predicate and object[12]. Figure 1.4. displays the elements of the tri part model and the symbology associated with the elements when graphing them.

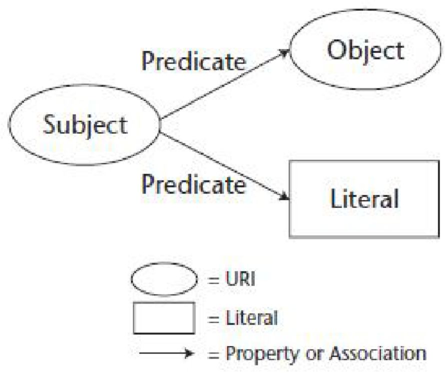


Figure 1.4 Showing the RDF triple.

The key elements of an RDF triple are following:

Subject, in grammar, this is the noun or noun phrase that is the factor of the action. In the sentence, for example, "The company sells guitars," the subject is "the company". The subject of the sentence tells us what the sentence is about. In logic, this is the term about which something is asserted. In RDF, this is the resource that is being described by the ensuing predicate and object. Therefore, in RDF we want a URI to stand for the unique concept "company" like http://www.company.org/ontology/#company to denote that it is meant a form of business ownership and not friends coming for a visit.

The general idea which is coming to the foreground is that an RDF resource stands for either electronic resources, like files or concepts, like "person". One way to think of an RDF resource is as "anything that has identity" [12].

Predicate, in grammar, this part of a sentence modifies the subject and includes the verb phrase. Returning to the sentence from previous example, the predicate is phrase "sells guitars." In other words, predicate tells some information about a subject. In logic, predicate is a function to form individuals (a particular type of subject) to truth-values with an arity based on the number of arguments it has. In RDF, a predicate is a relation between subject and object. And so, in RDF, would be defined unique URI for the concept "sells" like http://www.company.org/ontology/#sells

Object, in grammar, this is noun, which behaves upon the verb. In our example, object is noun "guitars." In logic, object behaves upon by the predicate. In RDF an object is either a resource or a literal value. In the example, a unique URI for guitars would be defined like http://www.company.org/ontology/#guitars.

Statement, in RDF is the combination of the preceding three elements, subject, predicate and object as a single unit. Figure 1.4 represents a graph of two RDF statements. Note that the object can be represented by a resource or by a literal value.

1.2.3.3 Capturing the knowledge with RDF

It exists a wide-ranged harmony that the triple-based model of RDF which we find simpler than the RDF/XML format, which is called the "serialization format." Because of this, a variety of simpler formats have been created to quickly capture knowledge expressed as a list of triples. For example, a scenario where the expression of four different ways: as natural language sentences, in a simple triple notation called N3, in RDF/XML serialization format, and, finally, as a graph of the triples.

If the model of subject, predicate and object is going to be followed, we can create a simple three statements and then capture a model.

```
Jean can play guitar.
The guitar is a archtop guitar.
Jean is a son of Orville Gibson.
```

In this example, the knowledge is managed via a bottom-up approach instead of a top-down approach. Let's examine how we capture the preceding sentences in N3 notation.

```
<#Jean> <#plays> <#guitar>
<#guitar> <#is> <#archtop>
<#Jean> <#son-of> <#Orville Gibson>
```

From every preceding sentence in the logic, it was extracted the subject, predicate and object. The sign # means that URI of the concepts would be the current document. This is a shortcut done for brevity; more accurate to replace the # sign with an absolute URI like "http://www.vutbr.cz/Jean/ontology" as a formal namespace. In N3 notation we can do that with a prefix tag like this:

```
@prefix gt: <http://www.vutbr.cz/Jean/ontology/>
```

Using the prefix, these resources will look like:

```
<gt:Jean> <plays> <guitar>
```

To help us convert a N3 notation to RDF/XML notation there exists several web semantic tools. An example of generated RDF/XML representation is shown in following Listing 1.3.

```
<rdf:RDF
      xmlns:RDFNsId1='#'
      xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
      <rdf:Description rdf:about='#Jean'>
            <RDFNsId1:plays>
                  <rdf:Description rdf:about='#guitar'>
                        <RDFNsId1:is rdf:resource='archtop' />
                  </rdf:Description>
            </RDFNsId1:plays>
            <RDFNsId1:son-of rdf:resource='#Orville Gibson/>
      </rdf:Description>
</rdf:RDF>
```

Listing 1.3 RDF/XML representation of preceding statements.

The first thing we can notice is that in RDF/XML, RDF statements are nested within each other. It is this sometimes non-intuitive translation of a list of statements onto a hierarchical XML syntax that makes the direct authoring of RDF/XML syntax difficult; however, since there are tools to generate correct syntax for us, we can focus on the knowledge engineering and not author the RDF/XML syntax. Second, note how predicates are represented by custom elements (like RDFNsId1:plays or RDFNsId:son-of). The objects are represented by either the rdf:resource attribute or a literal values.

1.2.3.4 RDF is not mainstream

The Resource Description Framework has been a W3C Recommendation (synonymous with standard) since February, 1999. Only slightly more than a year after the XML 1.0 Recommendation[9]. When giving briefings on the future of XML, it was a surprise to learn that many people have never heard about RDF. Outside of the digital library and artificial intelligence communities, RDF has not achieved mindshare with developers or corporate management. A demonstration of this mindshare gap is to compare the adoption of XML to the adoption of RDF. One simple measure is to compare the number of technical books on RDF versus XML and the number of commercial products supporting RDF versus XML as shown in Figure 1.5.

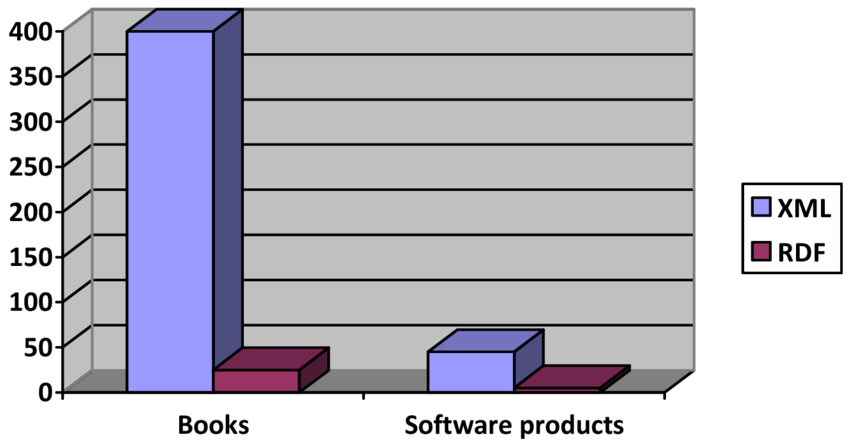


Figure 1.5 Support of XML in comparison with RDF.

The reasons why RDF adoption is so weak according to (Daconta M., Obrst L., Smith K.) [12]:

RDF and XML don't understand each other. It is not possible to validate RDF embedded in other XML or XHTML documents because of RDF's open grammar. In different meaning, RDF allows to mix in any namespace-qualified elements we want. Additionally, there is a fairly esoteric issue regarding a difference between how XML schema and RDF process namespaces. This has led many people to view RDF and XML documents as two separate paths for meta data.

Parts of RDF are complex. Several factors combined make RDF significantly more complex that XML documents. The three chief culprits in this equation are mixing metaphors, the serialization syntax and reification[3]. First, the model mixes metaphors by using terms from different data representation communities to include linguistic, object-oriented and relational data. This type of flexibility is a double-edged sword: Because it unifies modeling concepts from different domains, and that it causes confusion. This attempt to meld viewpoints is stated in the RDF Recommendation: "As a result of many communities coming together and agreeing on basic principles of meta data representation and transport, RDF has drawn influence from several different sources. The main influence has come from the Web standardization community itself in the form of HTML meta data and PICS, the library community the structured document community in the form of SGML and more importantly XML, and also the knowledge representation (KR) community." (Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation, February 22, 1999). Second, RDF syntax allows the RDF graph to be serialized via attributes of elements. In other words, we can express one RDF model in two different ways. This can be yet another problem for validation due to too much flexibility. Third, the hierarchical RDF/XML syntax (called the "striped" syntax) is difficult to author by hand and is better left to tools. In general, it is confusing to represent lists of statements as a hierarchical tree. The current method used in the RDF/XML syntax makes differentiating between object and properties very difficult. Lastly, reification has not yet proven itself and adds another level of abstraction to the RDF model. Reification matches natural language; it is a foreign concept to all of the other data communities. Most applications treat data as facts and implement data integrity procedures to ensure that axiom holds true. With reification, nothing is bedrock; everything is just an assertion, and you must follow a potentially infinite chain of assertions about assertions where one may contradict another at any time. Several RDF implementations and knowledge bases disallow the use of reification. Reification is a feature of RDF that is not for every application and can be safely avoided.

1.2.4 RDF Schema

RDF Schema is a language layered on top of RDF. This layered approach to creating the Semantic Web has been presented by the W3C and Tim Berners-Lee as the "Semantic Web Stack," as displayed in Figure 1.6. The base of the stack is the concepts of universal identification (URI) and a universal character set (Unicode). Above those concepts, we layer the XML Syntax (elements, attributes, and angle brackets) and namespaces to avoid vocabulary conflicts. On top of XML are the triple-based assertions of the RDF model and syntax we discussed in the previous section. If we use the triple to denote a class, class property and value, we can create class hierarchies for the classification and description of objects. This is the goal of RDF Schema, as discussed in Section 1.2.3. Above RDF Schema we have ontologies (taxonomy is a lightweight, ontology, and robust ontology languages like OWL). Above ontologies, we can add logic rules about the things in our ontologies. As it will be discussed in Section 1.2.4 a rule language allows us to infer new knowledge and make decisions. Additionally, the rules layer provides a standard way to query and filter RDF. The rules layer is sort of an "introductory logic" capability, while the logic framework will be "advanced logic." The logic framework allows formal logic proofs to be shared. Lastly, with such robust proofs, a trust layer can be established for levels of application-to-application trust. This "web of trust" forms the third and final web in Tim Berners-Lee's three-part vision (collaborative web, Semantic web, web of trust). Supporting this web of trust across the layers are XML Signature and XML Encryption. In this section we will try to focus on examining the RDF Schema layer in the Semantic Web Stack. RDF Schema is a simple set of standard RDF resources and properties to enable people to create their own RDF vocabularies. The data model for RDF Schema allows us to create classes of data. Class is defined as a group of things with common characteristics. In object-oriented programming (OOP), a class is defined as a template or blueprint for an object composed of characteristics (also called data members) and behavior (also called methods). An object is one instance of a class. OO languages also allows classes to inherit characteristics and behaviors from a parent class (also called a super class). The software industry has recently standardized a single notation called the Unified Modeling Language (UML) to model class hierarchies.

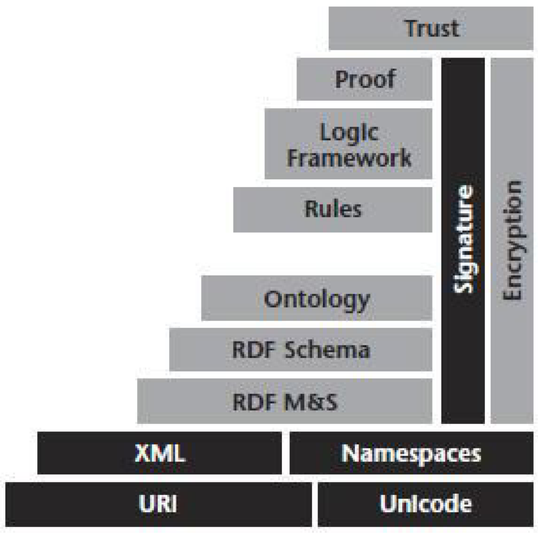


Figure 1.6 The Semantic Web Stack.

In the following practical example in Listing 1.4 we will show generated RDF schema and RDF document. Then we will describe the individual attributes in details.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!DOCTYPE rdf:RDF [
<!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
<!ENTITY example_chp5
'http://protege.stanford.edu/example-chp5#'>
<!ENTITY rdfs 'http://www.w3.org/TR/1999/PR-rdf-schema-
19990303#'>
]>
<rdf:RDF xmlns:rdf="&rdf;"
     xmlns:example_chp5="&example_chp5;"
     xmlns:rdfs="&rdfs;">
<rdfs:Class rdf:about="&example_chp5;Artifacts"
     rdfs:label="Artifacts">
<rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&example_chp5;DesignDocument"
     rdfs:label="DesignDocument">
     <rdfs:subClassOf rdf:resource="&example_chp5;Artifacts"/>
</rdfs:Class>
```

Listing 1.4 RDF Schema.

```
<rdfs:Class rdf:about="&example_chp5;Employee"
     rdfs:label="Employee">
     <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&example_chp5;Software-Engineer"
     rdfs:label="Software-Engineer">
     <rdfs:subClassOf rdf:resource="&example_chp5;Employee"/>
</rdfs:Class>
<!-- Classes SourceCode, System-Analyst, Technology, and Topic
omitted
for brevity. They are similar to the above Classes. -->
<rdf:Property rdf:about="&example_chp5;knows"
     rdfs:label="knows">
     <rdfs:domain rdf:resource="&example_chp5;Employee"/>
<    rdfs:range rdf:resource="&example_chp5;Topic"/>
</rdf:Property>
<rdf:Property rdf:about="&example_chp5;writes"
     rdfs:label="writes">
     <rdfs:range rdf:resource="&example_chp5;Artifacts"/>
     <rdfs:domain rdf:resource="&example_chp5;Employee"/>
</rdf:Property>
</rdf:RDF>
```

Listing 1.4 (continued).

rdfs:Class. An element that defines a group of related things that share a set of properties. This is synonymous with the concept of type or category. Works in conjunction with rdf:Property, rdfs:range and rdfs:domain to assign properties to the class. Requires a URI as an identifier in the rdf:about attribute.
rdfs:Label. An attribute that defines a human-readable label for the class. This is important for applications to display the class name in applications even though the official unique identifier for the class is the URI in the rdf:about attribute.
rdfs:subclassOf. An element that specifies that a class is a specialization of an existing class. This follows the same model as biological inheritance, where a child class can inherit the properties of a parent class. The idea of specialization is that a subclass adds some unique characteristics to a general concept. Therefore, going down the class hierarchy is referred to as *specialization*, while going up the class hierarchy is referred as *generalization*. The class "Software-Engineer" is defined as a subclass of "Employee." Therefore, Software-Engineer is a specialization of Employee.
rdf:Property. An element that defines a property of a class and the range of values it can represent. This is used in conjunction with rdfs:domain and rdfs:range properties. It is important to understand a key difference between modeling classes in RDFS versus modeling classes in object-oriented programming, in that RDFS takes a bottom-up approach to class modeling, whereas OOP takes a top-down approach. In OOP, we define a class and everything it contains. In RDFS, we define properties and state what class they belong to. So, in OOP we are going down from the class to the properties. In RDFS, we are going up from the properties to the class.
rdfs:domain. This property defines which class a property belongs to (formally, its sphere of activity). The value of the property must be a previously defined class. The property "knows" is the "Employee" class.
rfds:range. This property defines the legal set of values for a property. The value of this attribute must be a previously defined class. The range of the "knows" property is the "Topic" class.
Some other important RDFS definitions are as follows but they will not be described in details.

rdfs:type, rdfs:subPropertyof, rdfs:seeAlso, rdfs:isDefinedBy, rdfs:comment, rdfs:Literal.

1.2.5 Rules and logic

The semantic levels of information provide the input for software systems. The operations that a software system uses to manipulate the semantic information will be standardized into one or more rule languages. In general, a rule specifies an action if certain conditions are met.

1.3 Intentional Requirement Modeling

In this section we will describe the meaning of the goals, respectively sub-goals and what are the conceptual maps. We will describe how these notions included in search process are.

1.3.1 Semantic search process

By trying to understand what can be presented as a search process, we can say that it is a sequence of queries enabling to find comprehensive and accurate information by composing results from different information sources [1]. The novice user could have some difficulties by retrieving information which is relevant to his searched keywords and it becomes more difficult because of the deeper specialization of knowledge sources. In the web, there also exists at a domain level specific processes which are implemented by domain experts to obtain relevant information. In most cases for novice users, these processes are hard to reach, and search engines like Yahoo!, Google or Bing are not able to provide enough and satisfying details, accuracy and complexness of information about most topics.

The idea of this work is to bring possibility of reusing and sharing search queries and put them into search process. Search procedure is composed from a set of sub-goals. Sub-goals are composing one main goal which is the aim the novice user wants to reach in his intended search process. We can conclude from this statement that domain-related question types propose generalized templates useful to identify search procedure. Revert to the idea, choosing SPARQL standard query language and advantages of inference capabilities offered by ontologies specifying domain knowledge, offers us richer queries than the question types.

So again, the search process can be seen as a dependent set of smaller atomic searches carrying out domain-specific tasks. According to [1] a search process may be seen as a particular kind of business limited to search activities [1]. Bussiness process modeling formalism can be split into three categories: activity-oriented, product-oriented and decision-oriented.

Activity-oriented formalism includes:

- aiming at providing linear view of activity decomposition
- process is a set of predefined activities to be performed
- relationships are predefined among activities regarding control and data flows

Product-oriented formalism includes:

- pushing forward the result of activities
- keeping track of the product evolution

Decision-oriented formalism includes:

- based on the decision-oriented paradigm

- explanation not only how the process proceeds but why
- is more semantically powerful

If we take a think about the notions we stated, we get involved how to support knowledge transfer about search process for novice users to get sufficient results.. For example, why the search process is decomposed the way it is and how it is decomposed. In previous section we have stated that the search process is made of goals and goals are made of atomic sub-goals. It makes the process better for catching the complexity of searched knowledge, so why it is decomposed the way it is.

By providing different levels of details the aim is to handle different user's profiles and levels of knowledge. A dulcification process from novice user's side is required to be able to decompose goals ~ intentions into atomic parts of goals to help getting a satisfactory understanding of a topic about which no source contains all the relevant information. The system in this research should be able to show the same information in various levels of abstraction depending who is using the system. For novice user is acceptable higher level of abstraction changing by knowledge maturity of users in less abstraction and opening a searched object in more detailed way.

1.3.2 Map formalism

The map model is an intentional process model in which a non-deterministic ordering of intentions and strategies has been included. In our case, we focus on search intentions and search strategies [1]. In a detailed understanding, a map is a graph filled with intentions as nodes and strategies as connections between intentions. A search intention is a goal that can be achieved by the performance of a search strategy. An intention represents what we are looking for as example can be meant result that is expected to be reached and it doesn't matter how, when or where the results is provided and by who. In the map model there are 2 specific intentions to start and stop the process. A map consists of sections each which contains a triple (sources intention, target intention, strategy). The strategy is a stream which is followed from the source intention to the target intention by the way the user wants to achieve it. The map contains a finite number of paths from the start intention to the stop intention, each of the prescribing a way to achieve the goal of the search process under consideration.

Indeed, in a map, each set which is made up by a source intention, a strategy and a target intention is a *section* of the map. Let's precise what is an intentional map. An intentional map is neither a state diagram, because there is no data structure, no object, and no assigned value, nor an activity diagram, because there is always a strong context for each section of the map: its source intention and its strategy. We can attach more information to this kind of schema (in order to help the user of the map to choose the adequate strategy, for example) [2].

Let’s try a example of beginner stock trader who is looking for resources how to start trading on a financial market. To fulfill his goal, he may search for resources about brokers to decide which one is the most convenient for him (minimal opening amount, leverage, portfolio, history, trust), then resources about platforms, trading by himself giving orders to broker or just by broker, and how to open an account.. *Search for resource about brokers* and *search resources about platforms* are examples of intention. The formalized scenario with the map model is presented in Figure 1.7.

Intention: search for resources to open a trader’s account at stock exchange

- i1 – search for resources about stock exchange history
- i2 – search for resources about brokers
- i3 – search for resources about platforms
- i4 – search for resources about trading
    - personal trading
    - broker trading
- i5 – search for resources about opening an account

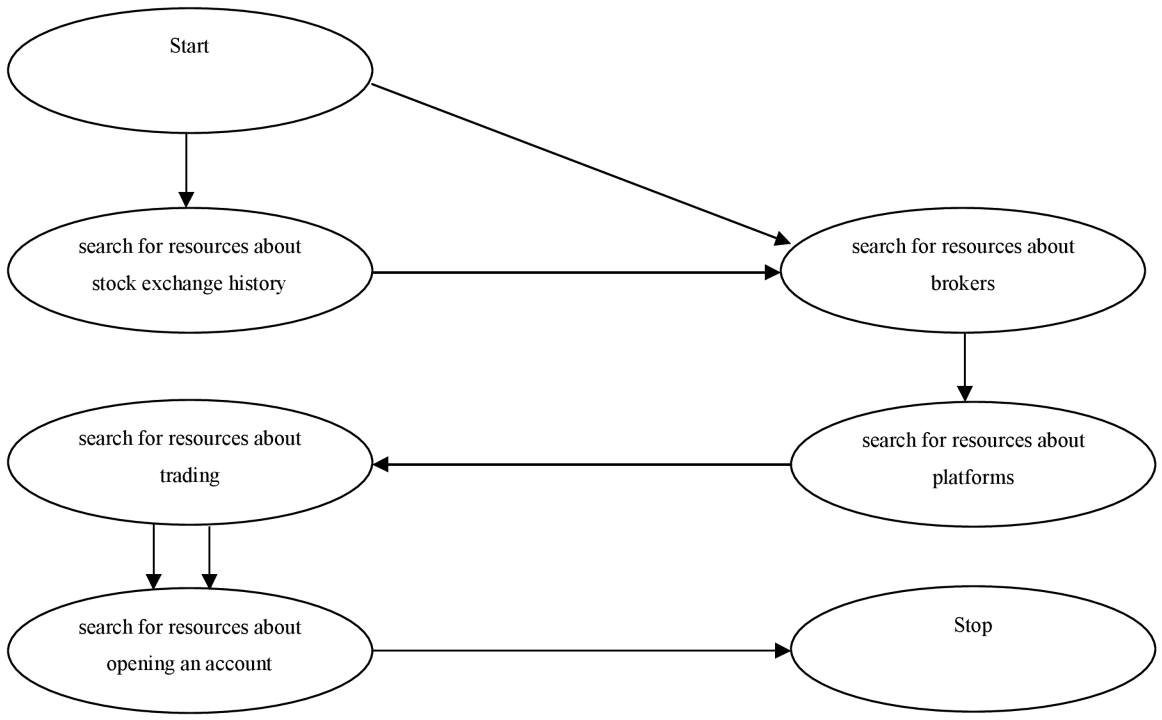

Start
search for resources about stock exchange history
search for resources about brokers
search for resources about trading
search for resources about platforms
search for resources about opening an account
Stop

Figure 1.7 Example of a intentional map.

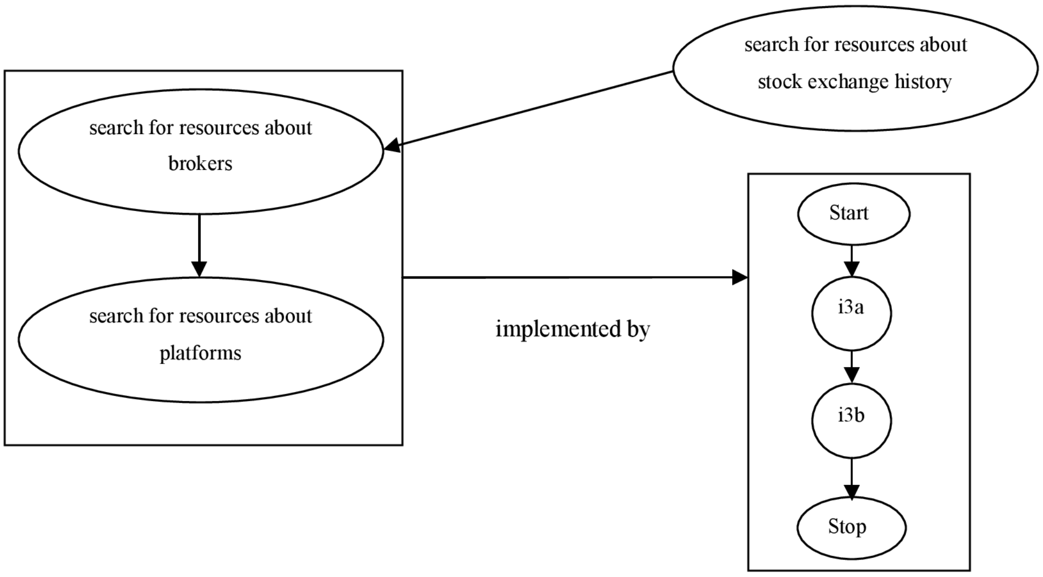
Figure 1.7 shows that there can be a more streams from a source intention to a goal; let us say to a target intention, each corresponds to a specific strategy. In the intentional map we can see a intention i5 labeled *Search for resources about opening an account* which is reachable via a strategy based on personal trading option or via a strategy based on broker trading option. Next in the picture of intentional map can be seen few other strategies to reach a same intention from different source intentions. The case which represents this situation is intention i2 labeled by *Search for resources about brokers* which is reachable from intention i1 labeled by *Search for resources about stock exchange history* or directly from a start intention i1.

The executing of each section of a map is supported by an intention achievement guideline which provides an operational or an intentional means to fulfill the target intention. In our work, we operationalize an intention achievement guideline by the execution of a query on the semantic community memory or we define it by a refined map.

Here in the example map above, the intention achievement guideline associated with the section identified by the source intention *start* and the target intention i1 labeled by *search for resources about stock exchange history* is operationalized by a query to search for relevant resources. The section identified by the source intention i2 labeled by *search for resources about brokers* and the target intention i3 labeled by *search for resources about platforms* requires. As it is shown in Figure 1.8, section i3 labeled by *search for resources platforms* is decomposed into intention i3a labeled by *search for resources about web platforms* and the intention i3b labeled by *search for resources about stand-alone platforms.*

Sub-goal: search for resources about platforms:

- i3 – search for resources about platforms
    - i3a – search for resources about web-based platforms
    - i3b – search for resources about stand-alone platforms



search for resources about stock exchange history
search for resources about brokers
search for resources about platforms
implemented by
Start
i3a
i3b
Stop

Figure 1.8 Decomposition of the intentional map of example above.

According to (Prat, 1997; Prat, 1999), an intention is represented by one verb and some parameters which play specific roles with respect to the verb. Among the parameters, there is the object on which the action described by is how the verb is processed.

1.3.3 Authoring process

1.3.3.1 Phases

We can split the intention map authoring process into 3 phases, Figure 1.9. First one is *Elicitation phase,* followed by *Formalisation phase* and finally *Fragmentation phase.* [2]

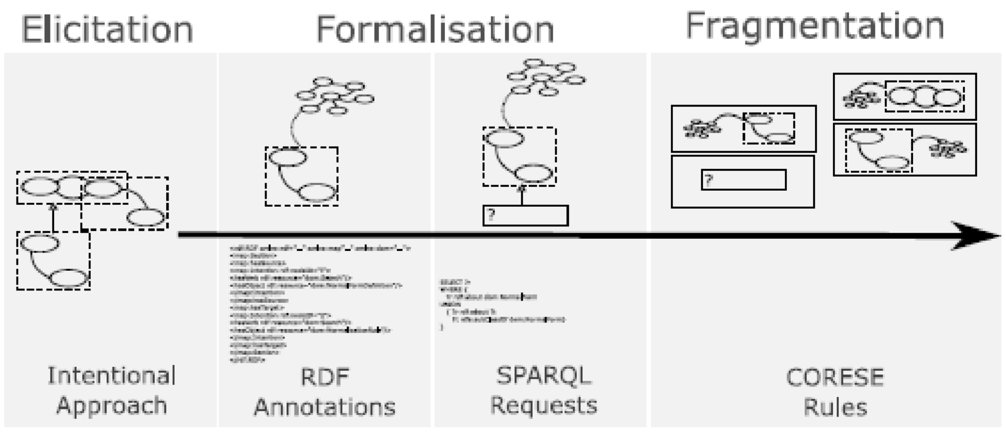


Figure 1.9 Three phases of authoring process.

1.3.3.2 Elicitation phase

To begin with, in this phase, the end users should define their way of searching by describing intentions and from this way build a pipeline consisting of goals, sub-goals and strategies (i.e. means to reach goals).

In figure 1.9 we can see an example of an intentional map. We can see two main intentions specified by the end user, let's call him a modern web user: *Download music* and *Pay for the music.* The second one is optional in the approach, because other ways can be selected and they could be easier for our user. In the area of these two intentions, we can find two strategies such as *by Ethernet 100Base-TX* or *by ITunes Store.* With these strategies we are able to find out the way how to pass from an intention to a next one. Also, there can be any other strategies which lead to the same next intentions like *by direct link* and *by YouTube steam* from *Download music* to *Stop*. Indeed, in a map, each set which is made up by a source intention, a strategy and a target intention is a *section* of the map. Let's precise what is an intentional map.

An intentional map is neither a state diagram, because there is no data structure, no object, and no assigned value, nor an activity diagram, because there is always a strong context for each section of the map: its source intention and its strategy [2]. We can attach more information to this kind of schema (in order to help the user of the map to choose the adequate strategy, for example).

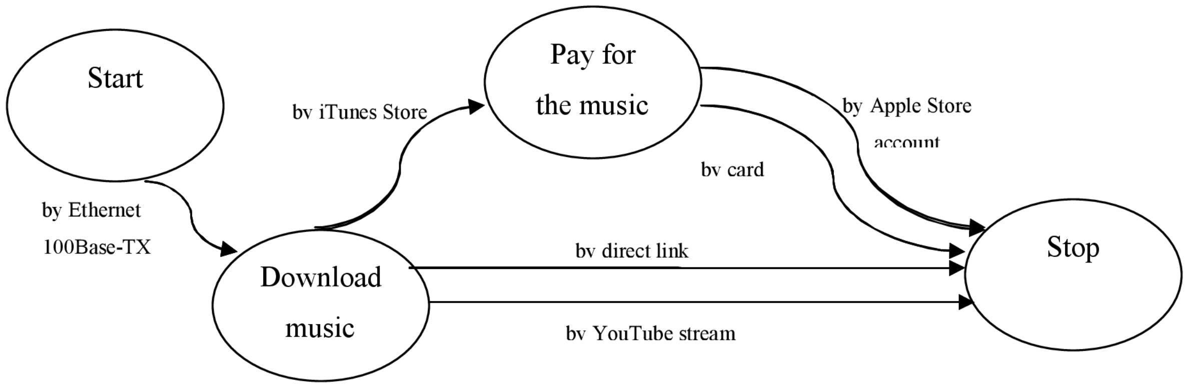


Figure 1.10 Intentional Map

This is the end of the Elicitation phase, a general and high-level definition of the image analysis pipeline under consideration, in very simple formalism.

1.3.3.3 Formalization phase

The *Formalization phase* is divided in two parts. We need to refine the intentional map which describes how to reach a target intention, by refining a section giving a new map more detailed way how to reach this intention (i.e. using more specific and low-level intentions and strategies). When this is completed, we have in use a general map and one or several level of refinements of its sections.

The second part, we understand generating and/or writing a SPARQL query in order to operationalise each section by an adequate web service specification. Indeed, the SPARQL query aims at retrieving the description of a web service or a set of web services supporting the fulfillment of the target intention of the section the query is associated to.

1.3.3.4 Fragmentation phase

And in the end the last phase, *Fragmentation phase,* transformation of all specifications created and captured in the *Formalization phase* in a set of CORESE rules. This choice has three great advantages. First, CORESE is an RDF engine bases on Conceptual Maps. It enables the processing of RDFS, OWL and RDF statements relying on Conceptual Maps formalism. It performs SPARQL Queries and run rules over RDF graphs. The second benefit is that, during this transformation into rules, the original intentions and strategies are naturally modularized and this fact far improves the reusability of the concerned discovery process fragment. Last, CORESE provides a backward chaining engine and we take advantage of it in order to offer knowledge inference. [2]

2 Application to develop

Now we will focus on the idea of need of application which will brings user a graphical interface to help him create bindings between schemas and RDF documents, creating bindings and drawing conceptual maps. Here we will discuss the phases of thought advancement of creating the whole process beginning the defining and describing the intentions and strategies.

2.1 Parsing RDF(S) files

We are going to describe in detail what the application is expected to look like and the main idea of this interface for novice user.

The idea is to have a web application allowing a user to create new search process and to run existing search process. The web application is dedicated to create new search process using existing RDF and RDF schema files. As a most comfortable environment for user we have decided to use a two directory trees for representing loaded descriptors and schemas, this we understand as loading of the domain ontology. During this process we must think about that our descriptors are in fact XML files, but they can be called "pure" XML files respectively. For our needs we will use the Java class *SAXParserFactory,* which offers us an ability to define our own way of reading of selected document. In this action, we will use our own *DocumentHandler,* which consists of following methods:

- *startDocument()*
- *startElement(String name, AttributeList atts)*
- *characters(char[] ch, int start, int length)*
- *endElement(String name)*
- *endDocument()*

Method *startDocument()* is opening our selected file with resources descriptors and is not ended while there isn't end of file and the method *endDocument()* is not called. The following method *startElement()*, which task is to distinguish the actual working element which the parser method will be working with. The method *startElement()* offers us important information from the markups from RDF file (XML file)like for example, name of the markup tag, content of opening and closing tag but not the inside value defined by these tags. We will show some attributes in example of our test descriptor file in Listing 1.5, which we are using in building the directory tree.

```
<rdf:RDF xmlns=http://www.inria.fr/2007/04/17/humans.rdfs#>
<Man rdf:ID="Harry">
  <name>Harry</name>
  <hasChild rdf:resource="#John"/>
  <hasSpouse rdf:resource="#Sophie"/>
</rdf:RDF>
</Man>
```

Listing 2.1 The content of example file with one record.

With the method *startElement(String uri, String localName, String qname, Attributes attr)*, this method is the one used the in the application, we get output shown in Listing 3.2. In this case, we can point out the following stepped output of reading the descriptor:

```
Start document:
Start element: local name: RDF qname: rdf:RDF uri:
http://www.w3.org/1999/02/22-rdf-syntax-ns#
attr:http://www.inria.fr/2007/04/17/humans.rdfs-instances
Start element: local name: Man qname: Man uri:
http://www.inria.fr/2007/04/17/humans.rdfs# attr:Harry
Start element: local name: name qname: name uri: attr:null
Characters: Harry
End element: local name: name qname: name uri:
Start element: local name: hasChild qname: hasChild uri: attr:#John
End element: local name: hasChild qname: hasChild uri:
Start element: local name: hasSpouse qname: hasSpouse uri:
http://www.inria.fr/2007/04/17/humans.rdfs# attr:#Sophie
End element: local name: hasSpouse qname: hasSpouse uri:
http://www.inria.fr/2007/04/17/humans.rdfs#
End element: local name: Man qname: Man uri:
End element: local name: RDF qname: rdf:RDF uri:
End document:
```

Listing 2.2 Output of RDF parser

Words in bold are string which are important and are parsed back to the tree directory. Parameter *attr* is actually the content of a tag, which if exists has some identificator. *Uri* (Uniform Resource Identifier) consists of a string of characters used to identify or name a ressource on the Internet. Such identification enables interaction with representations of the resource over a network (typically the World Wide Web) using specific protocols. Schemes specifying a specific syntax and associated protocols define each URI.[6]

Detailed implementation of important algorithms will be discussed in Chapter 3. Within important methods we mean mainly *startElement()*.

2.2 Building a tree directory

For this purpose we are using specialized open source Ajax, J2EE Ajax and JSF Java Framework. ICEfaces is an integrated Ajax Java application framework that enables Java EE Ajax application developers to easily create and deploy thin-client rich Internet applications (RIA) in pure Java. It is a fully featured product that enterprise developers can use to develop new or existing Java EE Ajax applications at no cost.[8]

In other words, library ICEfaces can offer functionality based on a package *java.swing.tree* and functionality of creation of a tree and moving in the newly created tree in a class *TreePath* however in graphically friendlier look for end user. For creating such a tree we have decided for combination of parser, which directly and logically communicates with method for building a tree. For movement between pointers and for moving with them, we mean movement between nodes and branches, which includes creating of node or branch, going deeper in a tree, creating of leaf or another node and following the branch back into the root node. After creating one node with deep branch, we can make another new node continuously calling this library functions:

• *DynamicNodeUserObject(rootTreeNode, this)*	creating the tree object
• *DefaultMutableTreeNode()*	for creating a node
• *DynamicNodeUserObject(branchNode, this)*	for adding this node as an object
• *add(branchNode)*	adding child node into parent node
• *add(subBranchNode)*	adding leaf into parent node

2.3 Drawing intentional map

As we stated in Section 1.3.2 that a map is a process model in which a non-deterministic ordering of intentions and strategies has been included, we must have in mind following this thought when creating a graphical application for creating such map. Basically, we must allow user to let him decide by himself which way he would like to work with this map. He has the possibility of choosing between longer or shorter way. If he has chosen the longer, it means that first he must use the tree directory part understanding loading the files, choosing the desired source and target intentions and saving them by application into xml file. When this is successfully completed he can proceed into drawing part, where first the generated file is loaded. After loading he can decide if the intentional map he created is correct or modify it by changing and/or adding more strategies or edit the desired pipeline of sub-goals. By choosing the shorter way, end-user is not required to pass all the parts of web application, but just going directly into the drawing area and by simple drag and drop commands build an intentional map.

Independently on the chosen ways, both can be saved for future reuse and can be sent to the CORESE engine to be executed. We will not use raw text format which will be created after drawing our required intentional map, but the whole map must be transformed into agreed SPARQL format.

In this part of the research, the web application is working as a demo example application. The functionality after drawing a map is restricted and is located in hypothetical layer, thus we are working with a research approach. In next Section 2.3.1 we discuss possible annotation way how to finish methodically and technically development and introduce our idea of formatting, respectively transforming intentional map into SPARQL query and executing him to the CORESE engine.

2.3.1 Search process annotation

Based on the search process ontology, search processes are then represented by RDF annotations. Our example shows the expected output from drawing application and should be revised in future in the future. If we consider our running example with *search for resources about platforms* from Section 1.3.2. is then represented by the following RDF dataset in Listing 2.3 where namespace *map* refers to the search process ontology and namespace *dom* refers to a domain ontology:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<map:Section>
 <map:hasSource>
  <map:Intention>
   <hasVerb rdf:resourc=map:Search/>
   <bind>Platforms/bind>
  </map:Intention>
 </map:hasSource>
 <map:hasTarget rdf:nodem="i3">
  <map:Intention>
   <hasVerb rdf:resourc=map:Search/>
   <with>Forex</with>
  <map:Intention>
 <map:hasTarget>
```

Listing 2.3 RDF dataset.

2.3.2 Concrete rules

Reusing search processes (or search process fragments) is intended to enable a dynamic connection of different search processes and therefore the building of whole search process by combining those (fragments of) search processes which both satisfy the global intention and retrieve available resources [1]. This goes through modeling IAGs (Intention Achievement Guideline Modeling) which connect a section of a map representing a search process either with another map representing another search process which can be viewed as a fragment of the global process which fulfills the target intention of the connected section or directly with a query. The proposed idea is to present an IAG by a rule. The rule we call *concrete* and we use *SPARQL* language which provides a unified framework to represent our *concrete* rule through the *CONSTRUCT* query form, which is supporting the fulfillment of a target intention by enabling the retrieval of relevant resources in the semantic memory. We formalize it by a *SPARQL* query that's *WHERE* clause asks for relevant resources by matching with the RDF annotations of these resources. For example, we use and present it in Listing 2.4, the instance generated from drawing application in Listing 2.3 and the intention from Figure 1.10 *Pay for the music.*

```
CONSTRUCT {
      _:s map:hasTarget _:i
      _:i map:hasObject _d:Stand-alone Platforms
      _:s map:hasResource ?r
}
WHERE {
      ?r rdf:about dom:Platforms
      UNION {
            ?r rdf:by ?t
            ?t rdfs:subClassOf dom:Platforms
```

Listing 2.3 Concrete rule in IAG annotation.

The *CONSTRUCT* clause of the query includes a statement about the resources fulfilling the section's intention and retrieved by the *WHERE* clause. This ensures that once retrieved, they are associated to the section. This query will retrieve all the resources about topics represented by sub-classes of *dom:Metatrader, dom:Marketiva, dom:DealFX, dom:SaxoTrader*, etc.

3 Implementation

3.1 Directory tree

In this section we will describe and discuss the created directory.

3.1.1 Uploading and building

These are our keywords in this part of the web application. First, the user should upload his required files with using the upload form. Files should be uploading one after another and they will be read by parser and simultaneously the tree will be build. User can decide from these loaded files which intentions he would like to reach by clicking the start intention and storing him in predefined format and clicking the target intention and storing him into predefined format. Process of storing intentions is made automatic.

In some functions of these trees, which are beside required functionalities, belong deleting nodes or leaves in trees and deleting of the intentional file.

We will have a look on how to implement such one tree, the second one is the same way as the first one without changes. Listing 3.1 shows a default constructor for setting up a pointer to our new tree. In Listing 3.2 is the basic initialization concerning the new object and binding this new object into our tree and a name of the default root node represented by *ROOT_NODE_TEXT*.

```
model = new DefaultTreeModel(rootTreeNode);
```

Listing 3.1 Setting up a new pointer for tree.

```
DynamicNodeUserObject rootObject =
                    new DynamicNodeUserObject(rootTreeNode, this);
            rootObject.setText(ROOT_NODE_TEXT);
            rootTreeNode.setUserObject(rootObject);
```

Listing 3.2 Initialization of a tree.

After successful initialization we call the parsing method *getParse(filename, rootTreeNode, model);* which is logically connected with building method of a tree. We can see that this method require to know working pointer of our model tree, the main root node and file where to go parsing.

We will show in Listing 3.3 the parsing method *startElement()* as discussed before:

```
public void startElement(String uri, String localName, String qname, Attributes attr) {
        texst.chain_start[root_node_position] = localName;
                 if (counter == 0) {
                     try {
                         TreeBean1(ROOT_N, localName, rootTreeNode, model);
                     } catch (Exception ex) {
                         Logger.getLogger(TreeBean.class.getName()).log(Level.SEVERE, null,
ex);
                     }
                 } else {
                     counter1++;
                     try {
                         TreeBean1(NODE_N, localName, rootTreeNode, model);
                     } catch (Exception ex) {
                         Logger.getLogger(TreeBean.class.getName()).log(Level.SEVERE, null,
ex);
                     }
                     if (counter1 == 1) {
                         current = tempNode;
                     }
                 }
                 if (attr.getValue(0) != null) {
                     try {
                         TreeBean1(LEAF_N, attr.getValue(0), rootTreeNode, model);
                     } catch (Exception ex) {
                         Logger.getLogger(TreeBean.class.getName()).log(Level.SEVERE, null,
ex);
                     }
                     if (tempNode != rootTreeNode) {
                     }
                 }
                 counter++;
             }
```

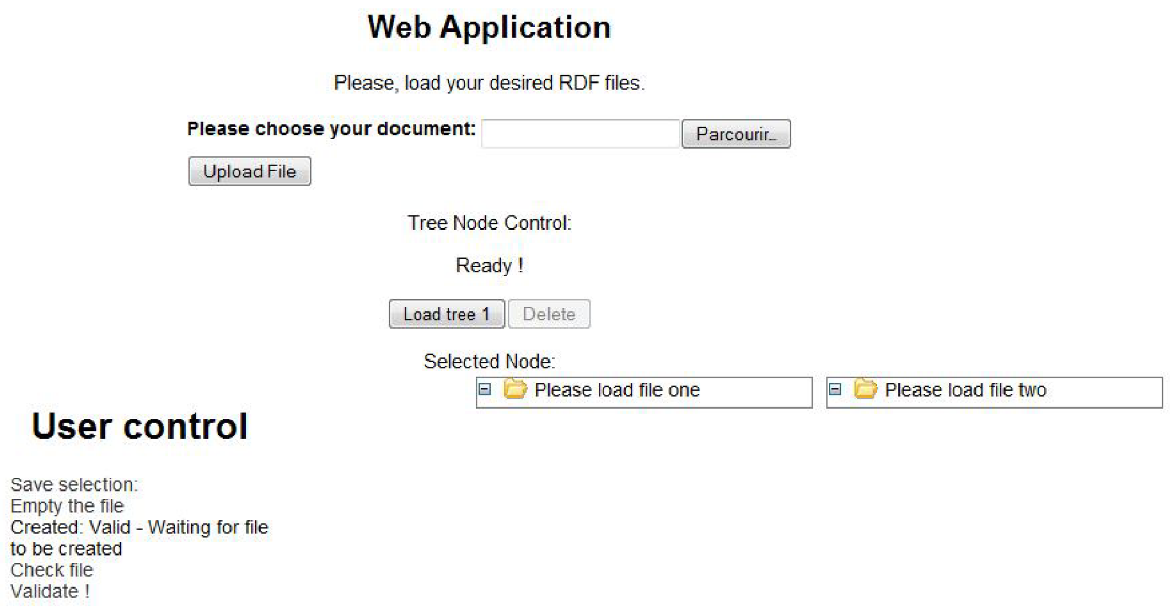
Listing 3.3 Implementation of one of parsing method *startElement()*.

The idea of adding nodes and leaves does not change heavily if we compare it with idea of creating a root node. The only thing we should be aware of is to know exactly where we are in the tree and where we would like to be and of course, what element are we building. Listing 3.4 together with Listing 3.5 show us the differences:

```
DefaultMutableTreeNode branchNode = new DefaultMutableTreeNode();
            DynamicNodeUserObject branchObject =
                    new DynamicNodeUserObject(branchNode, this);
...
tempNode.add(branchNode);
```

Listing 3.4 Creating a branch node.

```
DefaultMutableTreeNode subBranchNode = new DefaultMutableTreeNode();
            DynamicNodeUserObject subBranchObject =
                    new DynamicNodeUserObject(subBranchNode, this);
...
tempNode.add(subBranchNode);
```
Listing 3.5 Creating of sub-branch leaf.


Web Application

Please, load your desired RDF files.

Please choose your document:

Tree Node Control:

Ready !

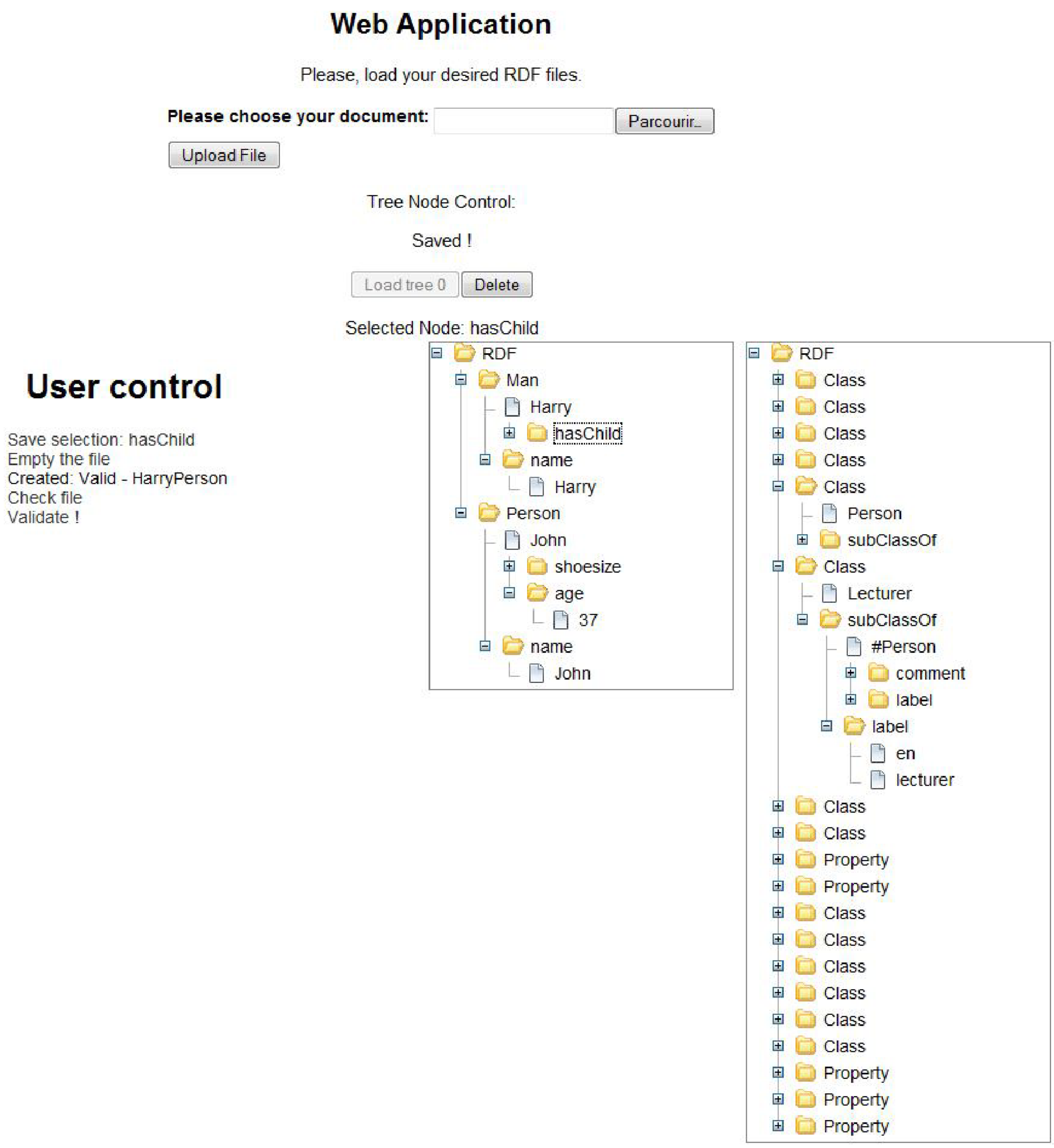
Selected Node:

User control

Save selection:
Empty the file
Created: Valid - Waiting for file to be created
Check file
Validate !

Figure 3.1 The basic view on web application, with files not loaded.

In Figure 3.1 we can see our idea implemented by Java technology. In the upper part is located the upload form waiting for user's files to be loaded and submitted. After uploading the files are loaded into the trees (this should be an semi-automatic process, user is required to confirm by clicking on a button Load 1 or 2 file) and parsing and building methods are executed. In Figure 3.2 we will see that all files are parsed without mistakes and are correctly cascaded in a tree. When information from files is processed, user is able to choose his desired source and target intention or desired strategy.


Web Application

Please, load your desired RDF files.

Please choose your document: [Parcourir...]

[Upload File]

Tree Node Control:

Saved !

[Load tree 0] [Delete]

Selected Node: hasChild

User control

Save selection: hasChild
Empty the file
Created: Valid - HarryPerson
Check file
Validate !

```
⊟ RDF
  ⊟ Man
    ├ Harry
    ⊞ hasChild
    ⊟ name
      └ Harry
  ⊟ Person
    ├ John
    ⊞ shoesize
    ⊟ age
      └ 37
    ⊟ name
      └ John
```

```
⊟ RDF
  ⊞ Class
  ⊞ Class
  ⊞ Class
  ⊞ Class
  ⊟ Class
    ├ Person
    ⊞ subClassOf
  ⊟ Class
    ├ Lecturer
    ⊟ subClassOf
      ├ #Person
      ⊞ comment
      ⊞ label
    ⊟ label
      ├ en
      └ lecturer
  ⊞ Class
  ⊞ Class
  ⊞ Property
  ⊞ Property
  ⊞ Class
  ⊞ Class
  ⊞ Class
  ⊞ Class
  ⊞ Class
  ⊞ Class
  ⊞ Property
  ⊞ Property
  ⊞ Property
```

Figure 3.2 The basic view on web application, with loaded files and trees deployed.

3.1.2 Validating

After evaluating our needs for the map modeling tool and according for the future development in the sense of standardization, our format for which we have decided, is simple and represents actual bindings created by user from Tree. What validating means, is that all pairs have their partners presented in the file and file begins and ends with introducing tags <*start*> and </*start*>. In listing 3.6 we present such validated format.

```
<start>
<bind>Pay</bind>
<with>Download</with>
</start>
```

Listing 3.6 Validated format from directory tree.

The application during its work is step-after-step building a xml file until the user clicks button *Validate*. It is obvious that until this point some errors or syntax mistakes must be made simply by user fault. We have created a validating tool which will check the file against errors before loading him into map modeling tool. Several different occasions may occur, which some of them we present in the following Listing 3.7.

```
<bind>Pay</bind>
<with>Download</with>
<bind>Cancel</bind>
```

Listing 3.7a Showing not valid file.

```
<start>
<bind>Pay</bind>
<with>Download</with>
<bind>Cancel</bind>
```

Listing 3.7b Showing not valid file.

```
<bind>Pay</bind>
<with>Download</with>
</start>
<bind>Cancel</bind>
```

Listing 3.7c Showing not valid file.

First, we must expect that user will create several bindings and one binding which is unfinished, actually is not bound and his partner is missing. The validating tool will check all bindings and erase the ones which are not bound. Usually this situation occurs at the end of file at the last record.
Second, if user clicks *Validate* the file and add something else to the bindings, application will add after introducing tags another tags (Listing 3.7c). Then the file is not valid anymore and must be checked again by clicking the button *Check file,* then the not bound partners after introducing tags

will be erased and also the introducing tags themselves. The reason why a validating tool must be integrated in our application is if the future development map modeling application will require valid file and won't check it before executing the inner content. In our map modeling application, we are expecting valid file without syntax errors. The integrated parsed can deal with not closed tags or missing tags, but the format *introducing tags - inner tags - content* has to be kept.

3.1.3 Performance

The balance tests showed us that our application is loading optimally and fluently. We did tests when application is loading, presented in Figure 3.3, when smaller RDF file is loaded and when larger RDF file is loaded, shown together in Figure 3.5. Time units used in graph are absolute units.

3.1.3.1 Application loading

- Total loading size: 281 004 bytes
- Total packets: 604
- Total loading time: 7.382591 second(s)

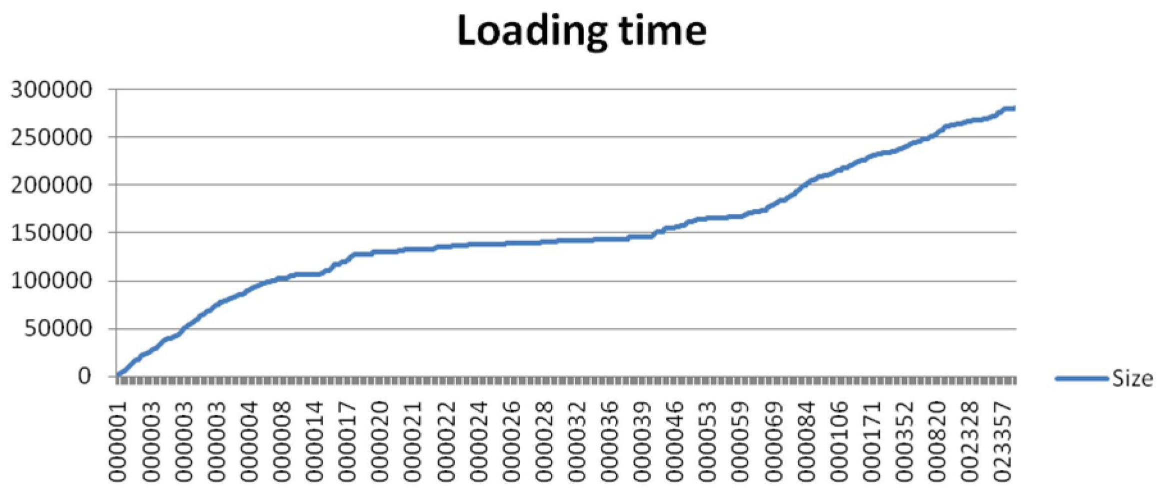


Figure 3.3 Graph showing loading size according to time.

3.1.3.2 Compiling tree with a smaller file

- Size of file: 504 bytes, 2 records
- Total loading size: 21 731 bytes
- Total packets: 53
- Total loading time: 1.514637 second(s)

3.1.3.3 Compiling tree with a larger file

- Size of file: 42 144 bytes, 70 records
- Total loading size: 198 269 bytes
- Total packets: 261
- Total loading time: 1.956699 second(s)

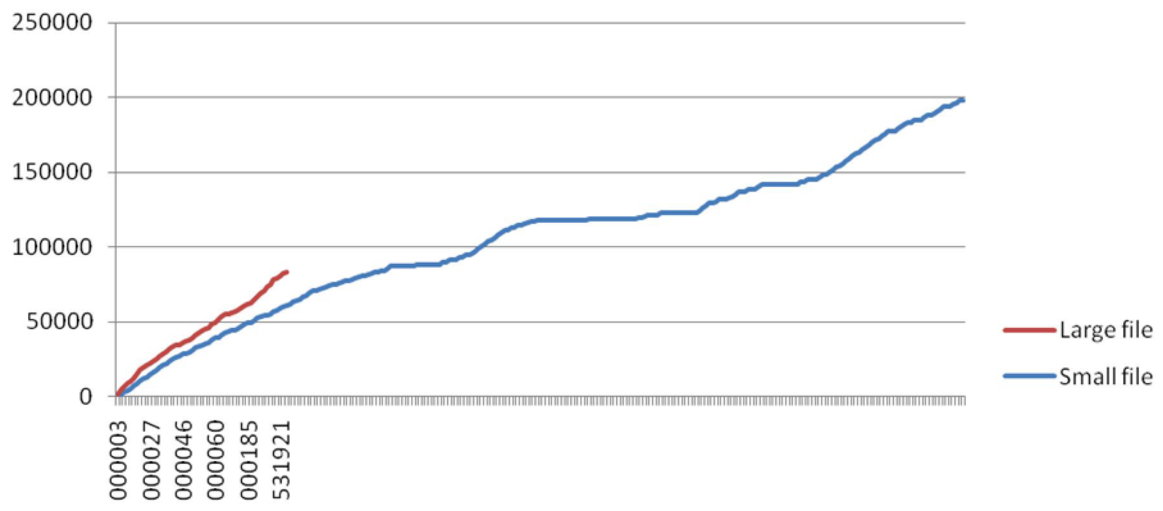


Figure 3.4 Graph showing the differences between loading a smaller and larger file.

From the results it is obvious and can be extracted that compiling a tree with different sizes of files is not important and showed a better performance even when loading a larger files.

3.2 Graphical user interface to manage intentional maps

The map management part of the application should consist of functions like loading previously generated file from directory tree and generating defined intentional map and/or drawing desired intentional map by simple drag and drop drawing commands. There is nothing more easier than using a *java.awt* and occasionally *java.swing* class. In Figure 4.6 we stated a draft out of the application to be developed.

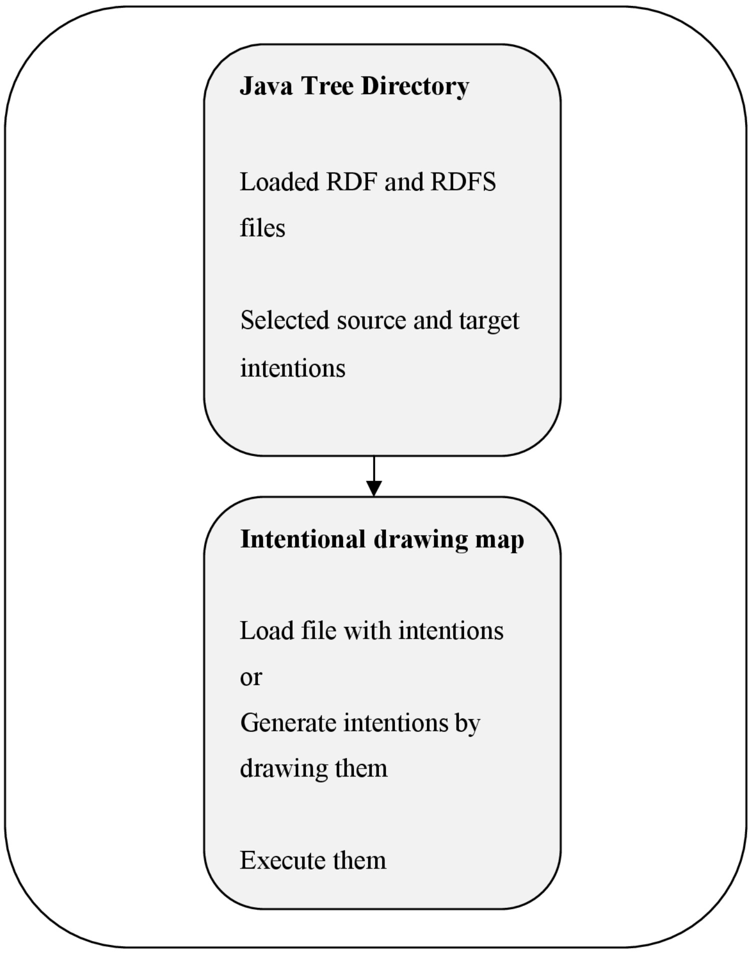
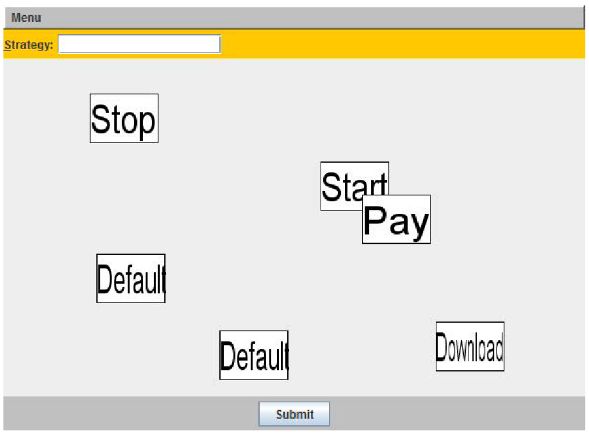

Java Tree Directory
Loaded RDF and RDFS files
Selected source and target intentions
Intentional drawing map
Load file with intentions
or
Generate intentions by drawing them
Execute them

Figure 3.6 Purpose of the web application.

Let’s suppose that the user has decided to draw his own map. First thing we should do is to easily navigate to the drawing applet and drop the rectangles, this is the shape which we decided to use to represent a goals. After drawing shapes, it is still possible to move around into preferred positions. When started to draw arrows, we have started to draw a shape which represent strategy how to reach one intention from another. Then we can decide if we need to clean the painting, save for future use or generate SPARQL query. These functionalities are restricted, because the generated format is experimental and not well optimized for fluent use. In Figure 3.7 and Figure 3.8 we present the workplace and further description will follow.



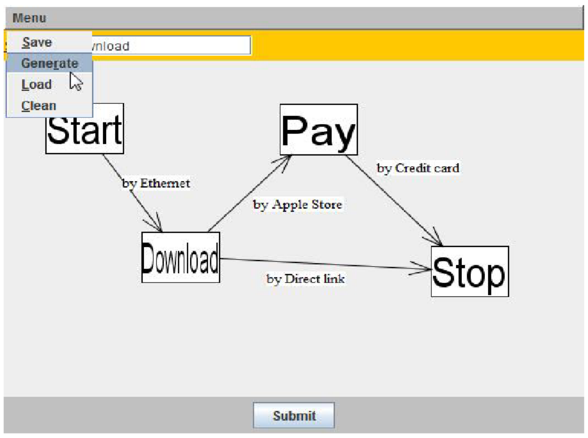


Figure 3.7 and Figure 3.8 Comparing first attempts to draw a map and correction into required appearance.

In the Figure 3.7 we can see a canvas where user tried to draw his first intentional map. The intentional map is made from example stated in Section 1.3.2. The application allows, until the strategies are not drawn, to move and delete freely the so far created goals. Before putting the square into a place, the value from text field must be submitted into application. If we find out our satisfaction with our product, there exists several options. First one is *Save* created map into a file for future use and probable sharing. Option *Generate* offers us the possibility to transform map into format convenient for executing by CORESE engine. *Load* button takes the created bindings from Directory Tree application, save bindings and draw goals into the drawing screen. User is awaited to draw lines between the goals and give them respectively names. This functionality is not present because it is a subject of study and development. We are approaching our own results in this matter and we have implemented a short demo of this feature with presented format as described in Section 3.1.2. After that, as the application takes the XML file containing bindings, we start the integrated parser to extract content of inner tags and we store them in a field prepared to be used as a buffer of strings which will be inserted into goals (eventually sub-goals). Then we ask the application to draw expected goals into screen.

We have tried not to reach out for our own way of developing the application and leave behind the current understanding of the study, but tried to follow the intentioned roadway and contribute into with the approach of our own idea. The future development should bring brighter light into the problem and decompose him in clear and agreed format, which would be used without doubts.

4 Conclusion

Participating in this project has brought me a great understanding of the Semantic Web and developing my programming skills in Java language. The beginning of the project was carried in a theoretical spirit in reading literature about descriptors and semantics, what is more in understanding the research approach by working with intentions, maps and strategies. I understood the concept and purpose of this keywords and I have moved my future thinking about World Wide Web.

In my mind I can see a future development of this project belonging in improving graphical user interface and functionalities like auto complete or suggestion in drawing a map after entering several keywords.

Moreover, I spent one semester at Université de Nice – Sophia Antipolis in south of France, which gave me a great experience and I gained a fantastic practice of French and English language.

Literature

[1] Mirbel, I., Faron-Zucker, C., Corby, O.: *Implementation of Intention-Driver Search Processes by SPARQL Queries*. 2002.

[2] Mirbel, I., Crescenzo, P.: *Improving Collaborations in Neuroscientist Community*, 8, 2004.

[3] Davies, J., Fensel, D., van Harmelen, F.: *Towards the Semantic Web*. Wiley, 2003, ISBN 0-470-84867-7.

[4] Davies, J., Studer, R., Warren, P.: *Semantic Web Technologies Trends and Research in Ontology-based Systems*. Wiley, 2006, ISBN 0-470-02596-4.

[5] Corby, O., Kefi-Khelif, L., Cherfi, H., Gandon, F., Khelif, K.: Querying the Semantic Web of Data using SPARQL RDF and XML.

[6] Wikipedia: Uniform Resource Protocol. [online], Last modification 23. June 2009.
URL http://en.wikipedia.org/wiki/URI

[7] Wikipedia: Concept Maps. [online], Last modification 23 May 2009.
URL http://en.wikipedia.org/wiki/Concept_map

[8] ICEfaces [online].
URL http://www.icesoft.com/products/icefaces.html

[9] Berners-Lee, T.: *Weaving the Web*, Harper San Francisco, 1999.

[10] Krill, P.: *Overcoming Information Overload*, InfoWorld, 7, January 2000
URL http://www.infoworld.com/articles/ca/xml/00/01/10/000110caoverload.xml

[11] Google: 3 Billion Document Index. [online]
URL http://www.google.com/3.html

[12] Daconta M., Obrst L., Smith K.: *The Semantic Web – A guide to the future of XML, Web Services, and Knowledge Management.* Wiley, 2003, ISBN 0-471-43257-1

Attachments

Attachment 1. Manual ...

Attachment 2. CD/DVD ...