

## Thesis

# **Extracting Knowledge Bases from table-structured Web Resources applied to the semantic-based Requirements Engineering Methodology Softwiki**

Rubén Navarro Piris

September 1, 2010

Universität Leipzig  
Fakultät für Mathematik und Informatik  
Institut für Informatik  
Betriebliche Informationssysteme

Academic Advisors: Prof. Dr. Klaus-Peter Fährnich  
Dipl.-Inf. Thomas Riechert

## **Authenticity Statement**

I hereby declare that I created this work independently and no other resources than the given ones were used.

Leipzig, September 1, 2010

Rubén Navarro Piris

## Thanks

This thesis would not have been possible without the help of Thomas Riechert, teacher of the Faculty of Informatics in the University of Leipzig. I want to thank him for his help and guidance in this project and with other topics outside the strict context of the thesis.

I also want to thank the rest of the *Agile Knowledge Engineering and Semantic Web* workgroup for their help solving some questions and problems with the OntoWiki framework.

Finally, I want to thank Mike Giese, student at the University of Leipzig, for helping me with some technical problems at the beginning of the project.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Technological Context</b>	<b>3</b>
2.1	The Wiki Technologies . . . . .	3
2.2	The Semantic Web . . . . .	4
2.2.1	First Attends of Semantic Web . . . . .	4
2.2.2	The RDF Language . . . . .	5
2.2.3	1st Extension Level: RDFs . . . . .	6
2.2.4	2nd Extension Level: OWL . . . . .	6
2.3	Semantic Wiki . . . . .	7
2.4	OntoWiki Project . . . . .	8
2.4.1	Main features of OntoWiki for Users and Developers . . . . .	9
2.4.2	OntoWiki Extension Framework . . . . .	9
2.5	SoftWiki . . . . .	11
<b>3</b>	<b>Objectives</b>	<b>13</b>
3.1	Specific objectives . . . . .	13
3.1.1	Section summary . . . . .	15
<b>4</b>	<b>CSVLoad</b>	<b>16</b>
4.1	Design . . . . .	16
4.1.1	Input data format . . . . .	16
4.1.2	Generated information . . . . .	18
4.1.3	Mapping information: the RDF semantic template . . . . .	19
4.1.4	Basic template for importing information to a knowledge base . . . . .	20
4.1.5	Template support of multivalued attributes . . . . .	24
4.1.6	Template support of empty values . . . . .	26
4.1.7	Template support of non-defined attributes . . . . .	27
4.1.8	Template support of data type definition . . . . .	29
4.1.9	Template support of data processing functions . . . . .	29
4.1.10	Semantic dependencies graph . . . . .	30
4.1.11	Static & Iterative Parts of the Template . . . . .	32
4.1.12	The template syntax . . . . .	33
4.1.13	Data dependencies & risks . . . . .	34
4.1.14	Section summary . . . . .	35
4.2	Application requirements . . . . .	36
4.2.1	Functional requirements / Use cases . . . . .	37
4.2.2	Non-functional requirements . . . . .	40

4.3	Implementation . . . . .	41
4.3.1	GUI Design . . . . .	41
4.3.2	Declaring a template . . . . .	41
4.3.3	Application algorithm . . . . .	42
4.3.4	File structure . . . . .	44
4.4	Google Code Issues Template . . . . .	45
4.4.1	Template .ttl file . . . . .	45
4.4.2	Template semantic dependencies reduced graph . . . . .	47
4.5	Related projects . . . . .	47
4.5.1	ConvertToRDF . . . . .	47
4.5.2	Triplify . . . . .	48
4.5.3	SCOVO to RDF data converter . . . . .	49
4.6	Legal implications . . . . .	50
<b>5</b>	<b>Gcode</b>	<b>51</b>
5.1	Design . . . . .	51
5.1.1	HTML structure . . . . .	52
5.1.2	From Google Code Issues to OntoWiki (SoftWiki) . . . . .	53
5.2	Application requirements . . . . .	53
5.2.1	Functional requirements / Use cases . . . . .	53
5.2.2	Non-functional requirements . . . . .	55
5.3	Implementation . . . . .	55
5.3.1	GUI Design . . . . .	55
5.3.2	Application algorithm . . . . .	55
5.3.3	File structure . . . . .	57
5.4	Legal implications . . . . .	58
<b>6</b>	<b>Conclusion &amp; future scope</b>	<b>60</b>
<b>7</b>	<b>Summary</b>	<b>63</b>
	<b>Installation &amp; configuration</b>	<b>64</b>
	<b>Glossar</b>	<b>65</b>
	<b>Bibliography</b>	<b>67</b>

## List of Figures

2.1	OntoWiki extension folder tree example . . . . .	10
2.2	OntoWiki framework abstract GUI . . . . .	11
2.3	SWORE Ontology . . . . .	12
4.1	From CSV data to stored RDF triples . . . . .	19
4.2	Cities table full semantic dependencies graph . . . . .	31
4.3	Cities table reduced semantic dependencies graph . . . . .	31
4.4	RDF template structure . . . . .	32
4.5	RDF template structure & syntax . . . . .	34
4.6	From CSV data to stored RDF triples using templates . . . . .	36
4.7	CSVLoad GUI Screen 1 . . . . .	41
4.8	CSVLoad GUI Screen 2 . . . . .	42
4.9	CSVLoad file tree . . . . .	44
4.10	CSVLoad generated part (marked in red) of the SWORE Ontology . . .	46
4.11	GCI Template Semantic dependencies reduced graph . . . . .	47
4.12	SCOVE to RDF mapping view . . . . .	50
5.1	Gcode generated part (marked in red) of the SWORE Ontology . . . . .	54
5.2	Gcode GUI Screen 1 . . . . .	56
5.3	Gcode GUI Screen 2 . . . . .	56
5.4	Gcode file tree . . . . .	57

## List of Tables

2.1	Example information of 2 persons . . . . .	5
2.2	Example of triples definition . . . . .	5
2.3	Example of triples definition (rd.do being a random domain of an enterprise)	5
3.1	Goal applications (conceptual) . . . . .	15
6.1	Example of contextual data (1) . . . . .	61
6.2	Example of contextual data (1) . . . . .	61

# Listings

2.1	Object-oriented class definition example . . . . .	6
2.2	RDFs class definition example . . . . .	7
2.3	OWL class definition example . . . . .	7
4.1	Basic abstract structure of a CSV file . . . . .	17
4.2	Detailed abstract structure of a CSV file . . . . .	18
4.3	Example of CSV file records (1) . . . . .	18
4.4	Example of CSV file records (2) . . . . .	19
4.5	N3 triples example (1) . . . . .	21
4.6	First idea of template . . . . .	22
4.7	Second idea of template . . . . .	22
4.8	Result of template mapping (1) . . . . .	23
4.9	Example of template (1) . . . . .	23
4.10	Result of template mapping (2) . . . . .	24
4.11	Example of CSV file records (3) . . . . .	24
4.12	Example of template (1b) . . . . .	24
4.14	Result of template mapping (3) . . . . .	25
4.13	Example of template (2) . . . . .	26
4.15	Example of CSV file records (4) . . . . .	26
4.16	Result of template mapping (4) . . . . .	26
4.17	Reduced template due to empty values . . . . .	27
4.18	Example of general template . . . . .	28
4.19	Triples reflecting the semantic dependencies graph problem . . . . .	31
4.20	RDF template syntax . . . . .	33
4.21	Google Code Issues Template . . . . .	45
5.1	Google Code Issues HTML structure (reduced) . . . . .	52



# 1 Introduction

There are currently a lot of different data sources all over the Web, each one with its own structure and semantics. Although the Semantic Web (see section 2.2), which describes methods and technologies that allow machines understanding the meaning - or *semantics* - of information on the World Wide Web by explicitly defining its semantics, gets bigger everyday, there are still a lot of systems which have not adopted Semantic Web technologies.

On the other hand, Semantic Web mechanisms need more and more semantic information in order to improve and to become an efficient and useful technology. So, tools to transform the already existing plain data into semantic data are needed in order to make the transition from Web to Semantic Web possible.

Over the last years the use of the Internet by users has evolved drastically from just consulting to publishing, sharing and modifying contents, turning the Internet into a social net in which the possibilities to collaborate and communicate grow every day bigger.

A good example are the Wiki-technologies. Their goal is to achieve a high level of collaboration among the user community of a data storage oriented system by enabling the possibility of adding or correcting content fast and easily. One of the most well known examples is Wikipedia<sup>1</sup>, a free and multilingual Encyclopedia project.

The OntoWiki project (further explained in section 2.4) merges the characteristics of the Wiki technology with the technology and benefits of the Semantic Web, giving birth to a so-called Semantic Wiki.

A good example of the use of this framework is the Catalogus Professorum Lipsiensis (CPL)<sup>2 3</sup>, a common initiative of the AKSW workgroup and the Department of Modern and Contemporary History at the History Department, also of the University of Leipzig.

The aim of the project is to build a semantic database containing all the teaching staff of the University of Leipzig in the 19th and 20th century.

SoftWiki is both a methodology and a tool (an OntoWiki extension detailed in section 2.5) for requirements engineering in a distributed stakeholders context. In order to further develop the SoftWiki tool, big amounts of semantic data are needed, but the only way to introduce non-semantic data into the system is manually, which means writing every single piece of new information. This works fine for new projects, but what about reusing already existing information? If it comes from another tool which also supports Semantic Web technologies its just a matter of importing the database,

---

<sup>1</sup> <http://en.wikipedia.org/>

<sup>2</sup> detailed info: <http://www.uni-leipzig.de/unigeschichte/professorenkatalog/info-en.html>

<sup>3</sup> example: <http://catalogus-professorum.org/website/en!>

but the problem comes when the source of the information is non-semantic data, e.g. data-tables or HTML<sup>4</sup> embedded data.

For example, the Agile Knowledge Engineering and Semantic Web (AKSW) workgroup of the University of Leipzig does not use SoftWiki, but another already solid, functional, non-semantic tool: for defining the requirements of the OntoWiki project the Google Code Issues platform. The AKSW wanted, however, to import all this information into the SoftWiki platform to be able to test and improve the tools and methodologies around it. So a new software needs to be created: a tool able to import data from the Google Code Issues platform into the SoftWiki tool.

This objective was then generalized to: import plain data from table-structured and HTML web resources into OntoWiki knowledge bases with a specific application to the semantic based requirements engineering methodology SoftWiki.

Thanks to the OntoWiki extension framework it is possible to fulfill this objective by extending an OntoWiki system, but for design reasons, two different extensions are defined: *CSVLoad* and *Gcode*.

*CSVLoad* takes care of importing data from table-structured resources into an RDF knowledge base. *CSVLoad* provides the necessary tools to define plain-to-semantic mapping using an RDF configurable template system. Although being initially designed to import CSV tables from Google Code Issues into SoftWiki, it is designed to be able to import any type of table to any type of semantic model just by defining the adequate template.

As data embedded in a HTML file follows a very specific and chaotic structure, defining a tool able to extract information out of any general HTML file to a semantic model is an extremely difficult task. It does, therefor, make no sense developing such a tool in the context of this project. As the initial goal was to import information from Google Code Issues and *CSVLoad* is not able to retrieve all the relevant or interesting information from it, the *Gcode* extension will be responsible for importing this information from the Google Code Issues HTML code to the SWORE ontology.

---

<sup>4</sup> HyperText Markup Language

## 2 Technological Context

The field in informatics in which we can find more innovative initiatives and which has the fastest increasing use rate is the Web and its technologies. Different companies and organizations compete to impose their own standards and services in a world in which new functionalities and business models are developed at a very high rate.

The Web suffered a big change at the beginning of the 21st century, changing from the know as Web 1.0, in which the users were merely information consumers of static pages, to the Web 2.0.

The Web 2.0 is the Web as we know it today: information sharing between different applications/stakeholders, dynamic interoperability, user centered design, RIA<sup>1</sup>, web applications that provide a similar look-and-feel as desktop applications by using data that can be processed both by the server and the client, exchanging this information in most cases in an asynchronous way [BK09]. Good examples of this trend are the Wiki technologies (Wikipedia), user video-sharing websites (Youtube) or social networks (Facebook, Twitter).

### 2.1 The Wiki Technologies

Wiki software is a type of collaborative web-application which allows users to collaboratively create and edit web pages directly on the web browser. The main difference between Wiki and other content management systems, such as blogs, is both that it is clearly focused on the content itself and the possibility of editing it collaboratively with other users of the community. The content (including all current and previous revisions) is usually stored in either a file system or a database on usually more than one server [Wag04].

With the help of a big community (more active users means more information precision) and moderators to ensure its correct use (avoid offensive or illegal content) this kind of applications show great results in building reliable information nets/databases. The best example for it is Wikipedia, “a free, web-based, collaborative, multilingual encyclopedia project”<sup>2</sup>. According to a research by the British journal Nature [Tec05] Wikipedia shows a similar error rate (both of serious and factual errors) to the Encyclopedia Britannica, the oldest English-language encyclopedia still in print, considered one of the most reliable information sources.

The structure of the content that users add or edit in a Wiki system is ordinarily defined by a simplified markup language which provides tools to organize the text or introduce specific elements like links to other resources. Although it makes the users

---

<sup>1</sup> Rich Internet Applications

<sup>2</sup> <http://en.wikipedia.org/wiki/Wikipedia>

lose a lot of possibilities towards editing directly the HTML or CSS documents or using JavaScript on any format, a general consistency in the look and feel of the whole Wiki-application is promoted. Some systems offer also WYSIWYG<sup>3</sup> editing tools, which are AJAX<sup>4</sup> powered, to make editing more comfortable and user friendly (although sometimes not all of the features are supported by this tools).

## 2.2 The Semantic Web

The Web community is nowadays already working on the concept of the Web 3.0. It includes not just the improvement, but also the revolution of the Web 2.0 thanks to the recent birth of new technologies that work together with already existing Web technologies.

The most important new concept is the transformation of the Web into a huge database system that can be understood and analyzed by a computer. To make that possible the use of semantic markup tools (e.g. RDF and RDF-based technologies like OWL) to define data models (creating the so called *Semantic Web*) and the use of AI<sup>5</sup> tools able to interpret this new information become essential.

The goal of the development of the Semantic Web is to make machines (computers) able to analyze the data of the Web, making functions like finding information, combining it and putting it into a specific context automatic. Such a difficult task could only be done in an appropriate way *manually* by humans [BLHL02].

It can also be seen the other way round: the Semantic Web makes machine-readable content available on the World Wide Web. Tools like Protégé<sup>6</sup> where first developed in the field of the AI, but are now adapted to the semantic web by using semantic description languages [GGP<sup>+</sup>02].

### 2.2.1 First Attends of Semantic Web

The Web is based on the HTML language. It permits the description and visualization of a document (text) and references (links) to other documents (other HTML documents, graphics or multimedia contents), but does not support the definition of semantic relationships between resources and possible attributes of them, as well of special meaning or features of the text itself.

To fulfill this objective one of the first initiatives was to extend the HTML language to enrich its semantic meaning, giving as a result the so called Semantic HTML. For example, the use of <em> denoting emphasis rather than <i>, which specifies italics. But this specification was still not enough to specify the semantics (the actual meaning and relationship) of objects referenced or described in the HTML document. Because of this the definition of a specific language for this purpose was necessary and that is when the RDF language (and derivatives like OWL) was born.

---

<sup>3</sup> What You See Is What You Get

<sup>4</sup> Asynchronous JavaScript And XML

<sup>5</sup> Artificial Intelligence

<sup>6</sup> <http://protege.stanford.edu/>

name	email	office
John	john@email.xx	E-202
Roger	roger@anotheremail.xx	F-367

**Table 2.1:** Example information of 2 persons

### 2.2.2 The RDF Language

The main idea of the Resource Description Language (RDF) is to build statements (called *triples*) in the form subject-predicate-object. Thanks to its very simple structure, labeled and directed conceptual graphs can be easily build.

An easy example. The information in table 2.1 can be defined in the triples of table 2.2.

John	<i>hasEmail</i>	john@email.xx
John	<i>isInOffice</i>	E-202
Roger	<i>hasEmail</i>	roger@anotheremail.xx
Roger	<i>isInOffice</i>	F-367

**Table 2.2:** Example of triples definition

In order to make the Web compatible with this model, the subject is defined as a web resource (an URI) and the object also as a resource or sometimes directly as a literal or label, a String containing information.

Supposing that a person has its own website (for instance an enterprise internal page) the previous example could be defined like in the triples (in N-Triples form) of table 2.3.

In this case an additional triple is added in each case because the subject is not directly the *name* of the object, but a web resource, whose associated property is a *name*.

One of the biggest advantages towards other model definition structures (as, for example, an object-oriented approach) is its high level of extendability: any time new properties can be added without having to redefine the rest of the graph.

http://rd.do/empl/178	<http://rd.do/rdf/hasName>	"John"
http://rd.do/empl/178	<http://rd.do/rdf/hasEmail>	"john@email.xx"
http://rd.do/empl/178	<http://rd.do/rdf/isInOffice>	"E-202"
http://rd.do/empls/847	<http://rd.do/rdf/hasName>	"Roger"
http://rd.do/empl/847	<http://rd.do/rdf/hasEmail>	"roger@anotheremail.xx"
http://rd.do/empl/847	<http://rd.do/rdf/isInOffice>	"F-367"

**Table 2.3:** Example of triples definition (rd.do being a random domain of an enterprise)

```
class Person {  
    ...  
}  
  
class Document {  
    Person author;  
}
```

*Listing 2.1:* Object-oriented class definition example

### 2.2.3 1st Extension Level: RDFs

RDF is good for defining attributes of resources, that means, an easy way of building traditional attribute-value pairs, which in some cases also represent relationships between resources. But this is not enough to describe the meaning of such properties or the relationships with other resources.

In this context was the RDF Schema (RDFs) defined. RDFs is an extension of RDF, build and specified also on RDF, which enriches it by defining the concepts of Class and Property. This makes it easier to build object-oriented models (similar to object-oriented languages such as Java or C++). However the definition takes place the other way round: in object-oriented languages a class is defined with the attributes that it contains (the properties) while in RDFs we can specify properties with a range or domain later. The major benefit of this architecture is the possibility of extending a class without having to redefine it (in object-oriented languages we have to either redefine the class or create a new one inheriting the characteristics of the original one). This makes reusing and extending models much easier<sup>7</sup>.

For better understanding of this here is an example. Listing 2.1 shows an object-oriented language class definition.

The attributes of the class *Document* (in this case *author* of type *Person*) is declared with the class, and therefore no other ones can be added unless we change the definition. The same conceptual definition in RDFs syntax is on listing 2.2.

As shown in the example, *Person* and *Document* are defined as classes and later the property *author* (in this case *hasAuthor*) is defined.

### 2.2.4 2nd Extension Level: OWL

Although RDFs enables the definition of classes, relationships and attributes, these features are in most of the cases insufficient to model a full ontology because of the existence of certain constraints (like cardinality or class disjointness among others). Because of that the Web Ontology Language (OWL) was defined. Its main goal is to provide support to this constraints, making it possible to define ontologies in an RDF environment<sup>8</sup>.

---

<sup>7</sup> <http://www.w3.org/TR/rdf-schema/>

<sup>8</sup> <http://www.w3.org/TR/owl-guide/>

```

@prefix ex: <http://www.example.xx/rdfsmodel#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.

ex:Person a rdfs:Class.
ex:Document a rdfs:Class.
ex:hasAuthor rdfs:domain ex:Document.
ex:hasAuthor rdfs:range ex:Person.

```

**Listing 2.2:** RDFs class definition example

The OWL Full version permits mixing OWL specifications with pure RDF specifications without restrictions, but in some cases defining ontology restrictions and constraints is necessary when using efficient reasoning systems. That is the reason why the sub-language OWL DL was developed. On the other hand, the main disadvantage is that an RDF document cannot be assumed to be a compatible OWL DL document. Another sub-language, OWL Lite, was developed with tool builders as main target. It is a subset of the OWL DL, which means it works also good with reasoning systems.

$$OWLLite \subseteq OWL DL \subseteq OWL Full$$

The representation of the example in the previous section (cf. listing 2.2) in OWL form is shown in listing 2.3.

```

@prefix ex: <http://www.example.xx/rdfsmodel#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.

ex:Person rdf:type owl:Class.
ex:Document rdf:type owl:Class.
ex:hasAuthor rdf:type owl:AnnotationProperty.
ex:Document ex:hasAuthor ex:Person.

```

**Listing 2.3:** OWL class definition example

## 2.3 Semantic Wiki

Wiki technologies are useful to build large interactive data storage systems, having Wikipedia as the best example, but they have a big problem: this data is stored/shown in the form of plain HTML. Some Wiki tools use already markup languages, although these are usually insufficient for doing complex tasks like for example information reuse, selective export or querying. As an example: finding out the names of movies, that were

first played from 1995 and on and collected more than 100 million US dollar. Finding this information in Wikipedia seems a really difficult task, as we should look every movie one by one and look for the information “manually” (that means, read all of the plain information until we find the specific information). Besides, although Wikipedia provides the users with templates for specific themes and article types, the article editor is quite free to write in the structure or format he wants, so similar objects (in this case movies) present different information structures, which makes information search even harder.

The solution to this problem is the use of semantic data models, that means, to structure the information in order to make querying and data linking possible. In this context we can find the DBPedia<sup>9</sup> project. Its purpose is to extract structured information from Wikipedia and make this information available on the Web. In this case finding the information of the example exposed before would be much easier: we just have to use a SPARQL query like in every normal (semantic) database.

Another advantage of using semantic structured data in a Wiki is that the presentation of the information to the user is uniform and makes navigating through this information more agile and, in much cases, with a wide range. An example of this can be seen comparing the Wikipedia page for the city of Leipzig<sup>10</sup> with the same information in DBPedia<sup>11</sup>. The first one is obviously more comfortable to read as a whole due to its typical text structure and an interface which has been improved for years, but if we are looking for a specific information about Leipzig, it is easier to find it in the DBPedia page. For instance, it is immediate to find out which cars are manufactured in Leipzig thanks to the *is dbpedia-owl:assembly of* property. On the other hand, in order to find this information in Wikipedia, the article has to be read only to find out that only the companies are mentioned. Then the companies pages in Wikipedia should also be read and maybe all of their manufactured cars pages. So this makes clear, that a semantic linkage and presentation shows more complete information in a very structured way.

## 2.4 OntoWiki Project

The OntoWiki project<sup>12</sup> [DAR06] is being developed since 2004 by the Agile Knowledge Engineering and Semantic Web (AKSW<sup>13</sup>) workgroup of the University of Leipzig and it enables the visualization, modification and insertion of semantic based data with a Wiki framework. Its current version is 0.9.

In order to make collaboration between the developers of OntoWiki (both the AKSW group as the developers community) easier, the source code is hosted in the Google Code platform, which permits code hosting, versioning, browser code viewing and social interaction, in the form of Issues definitions, and comments and attachment of additional documents/sources to them.

---

<sup>9</sup> <http://dbpedia.org/>

<sup>10</sup> <http://en.wikipedia.org/wiki/Leipzig>

<sup>11</sup> <http://dbpedia.org/resource/Leipzig>

<sup>12</sup> <http://ontowiki.net/Projects/OntoWiki>

<sup>13</sup> <http://aksw.org/>



### 2.4.1 Main features of OntoWiki for Users and Developers

OntoWiki provides, among others<sup>14</sup>, the following interesting features.

#### Facet-based Browsing

Thanks to it the user is able to navigate through the information along multiple paths corresponding to different orderings of the facets.

#### In-line editing

Following the WYSIWYG (What You See Is What You Get) philosophy to make editing more user friendly, OntoWiki enables in-line editing, which means that users can modify/add data directly in the graphic interface by double clicking on the desired field.

#### Integrated map and calendar tab

Data containing property values representing geographical information or associated to the data type `xsd:date`.

#### Themes modification

Just modifying the default CSS file and the *main.php* file the OntoWiki system will have a total different look.

#### Extensions

OntoWiki has a GPL License and by this can be changed by anyone to adapt to its specific needs with the help of its widgets and extension framework. Widgets are reusable components of the OntoWiki user interface that the user interacts with. Developers can activate/deactivate or add new widgets in order to adapt the functionalities to a specific schema or needs. See section 2.4.2 for more information<sup>15</sup>.

### 2.4.2 OntoWiki Extension Framework

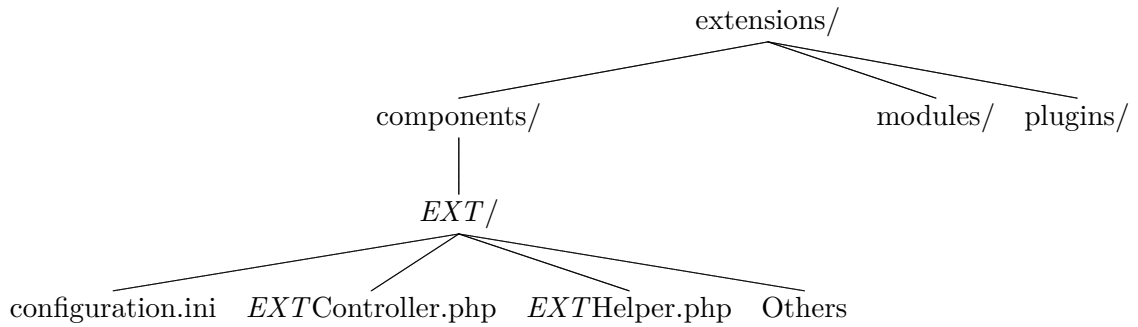
OntoWiki enables a sophisticated extension framework which permits developers adding new functionalities and personalize the deployed OntoWiki framework. Every installed extension can be enabled/disabled when wished by the system manager. There are three types of extensions:

1. Components
2. Modules
3. Plug-ins

---

<sup>14</sup> source: <http://ontowiki.net/Projects/OntoWiki/Features>

<sup>15</sup> or go to <http://code.google.com/p/ontowiki/wiki/ExtensionCookbook>



**Figure 2.1:** OntoWiki extension folder tree example

each one with its purpose and structure. All of them have their specific *.ini* file (*{component|module|plugin}.ini*) where the standard configuration (enabled, name, etc.) and specific private parameters are defined. This file, together with the extension main file (and every other file the extension needs to work) is under the extension folder to find. An example of the OntoWiki extensions folder tree with a component extension (referenced as *EXT*) can be seen in figure 2.1.

### Component

Components are controllers with a MVC<sup>16</sup> architecture. Components receive and serve requests, usually using the main window to interact with the user by showing or getting information (cf. figure 2.2), but they can also serve as asynchronous request controllers.

A component can have the so called Helper. Unlike the component controller, which serves only its specific request(s), a component helper is instantiated every time a request takes place. This results very useful for doing actions not directly related with the controller, e.g. registering a new menu option.

Every component extends `OntoWiki_Controller_Component` and every helper extends `OntoWiki_Component_Helper`. The component is configured on its *component.ini* file.

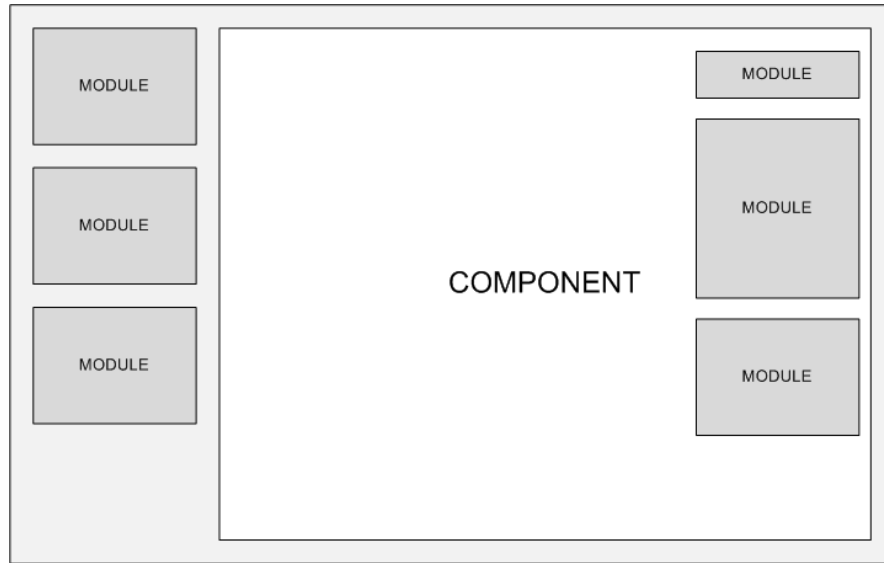
### Module

Modules shown little displays, either in the main information info or in the empty left part of the screen (cf. figure 2.2). They usually show additional information or functionalities related with the information displayed in the main window. They act more dynamically than components because they are able to modify their configuration file.

Every module extends `OntoWiki_Module` and has its own *module.ini* configuration file.

---

<sup>16</sup> Model-View-Controller



**Figure 2.2:** OntoWiki framework abstract GUI

### Plug-in

Plug-ins are the most flexible extensions. Instead on being based on a standard class with defined display and requests, a plug-in executes arbitrary code when a specific event is triggered. Plug-ins can modify the information associated to this event, so they depend on the good definition of them in order to work properly. Some events are already defined in the framework core, but they can also be triggered by extensions at any moment and with any wished information.

Every plug-in extends `OntoWiki_Plugin` (although this class does not define strong structural guidelines or properties) and has its own *plugin.ini* configuration file.

## 2.5 SoftWiki

The discussion and specification of requirements is a key aspect in the developing of both software and other innovative projects. In order to achieve a successful requirements engineering, all the project stakeholders (clients, users, managers, developers, etc.) have to take part on it. However, this is sometimes really hard when the project is big and its stakeholders groups are big or very distributed (e.g. open source or governmental projects). So a tool and/or methodology which permits the easy collaboration of different stakeholder groups is needed to solve this problem [Aue09].

Thanks to the possible use of widgets, a requirements and tags management extension called SoftWiki [LR09] was developed for the former version of OntoWiki(0.85)and is being currently adapted to Ontowiki v0.9 by M. Giese.

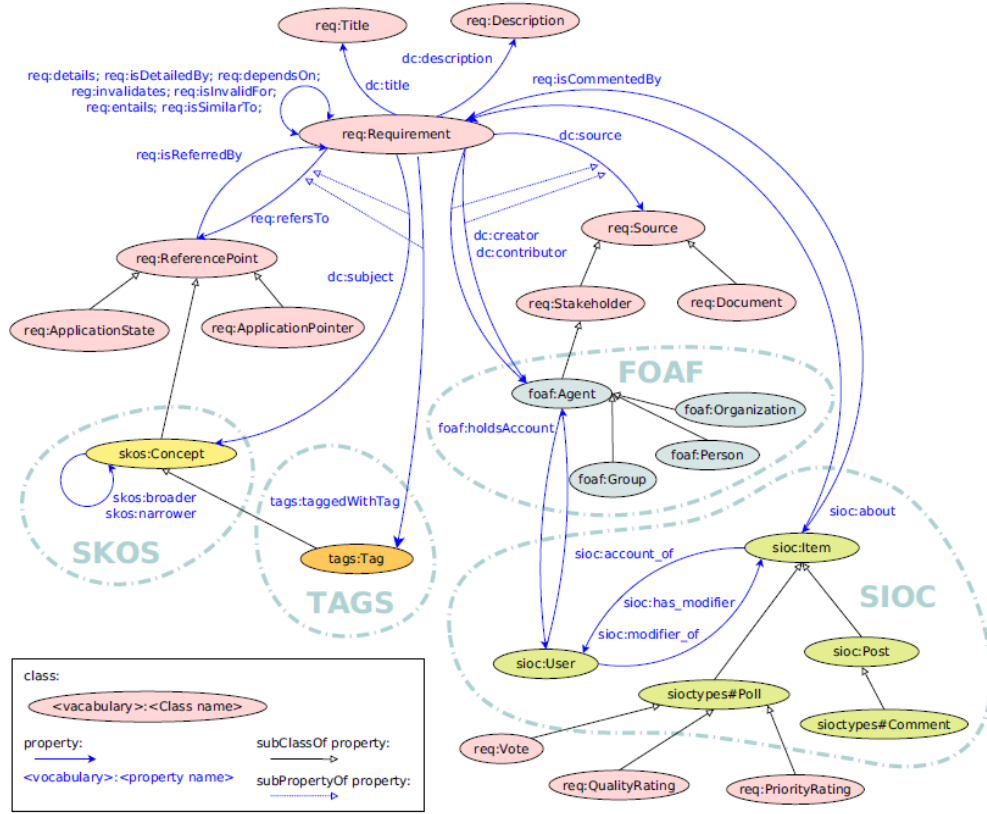


Figure 2.3: SWORE Ontology

The SoftWiki tool allows the definition of a project's requirements and the tagging of these. With the help of the OntoWiki framework it provides also visualization and search/navigation tools for these requirements.

But SoftWiki is not only a tool, it consists also in the Agile Requirements Engineering Methodology and an ontology (SWORE: Software Ontology for Requirements Engineering) that supports it.

This methodology consist on the collaboration of potentially very large and spatially distributed stakeholder groups in the process of software development, particularly in the field of software requirements engineering<sup>17</sup>.

The SWORE Ontology (see figure 2.3<sup>18</sup>) defines the semantic model of the requirements definition. It is a flexible definition which allows linking to external sources (public domain knowledge or company-specific knowledge).

Requirements can be linked to other requirements, redefined, commented, etc. and in this process it becomes more semantically rich. It is a sort of requirements *seman-tification* process [LHA<sup>+</sup>08].

<sup>17</sup> Project homepage: <http://softwiki.de/netzwerk/en/>

<sup>18</sup> source: <http://softwiki.de/netzwerk/en/methodologie/swore/>

## 3 Objectives

The main general goal of the project is to add importing functionalities to the OntoWiki (SoftWiki) platform. A large community of projects have their requirements hosted in the Google Code platform or in similar systems, included OntoWiki as explained in section 2.4. This projects contain big amounts of information, which makes the migration to another system (in this case SoftWiki) impossible if the right importing tools are not provided.

### **Importing information from other sources**

SoftWiki permits the definition and editing of requirements for a defined project hosted in the system, but a lot of software projects actually use other non-semantic tools to manage their requirements. The transition from one of this tools to the SoftWiki platform would need a lot of work, because every requirement has to be defined manually. So in this sense a main goal of the project is to make possible to automatically import great amounts of data into the OntoWiki system and more specifically importing requirements into the SoftWiki platform.

### **Transforming great amounts of imported information into semantic data**

Another think to take into consideration is that OntoWiki saves semantic data, but the information we are going to import comes in form of plain data (the import of semantic data models is already implemented in OntoWiki). Another main goal of the project is to provide tools to convert the imported plain data to the according semantic model.

### **Importing information directly from the Web**

Some services do not provide easy information access points, that means, standard table, object-like or semantic data formats to make data importing easier. Instead, all the information is shown directly in the form of HTML. That leads to the next main goal of the project: extracting information from HTML resources.

## 3.1 Specific objectives

In order to fulfill the objectives approached in the previous section (cf. section 3) software tools have to be developed to work together with the OntoWiki system. OntoWiki provides a sophisticated extension system, which makes possible the inclusion of new functionalities under the same integrated platform (cf. section 2.4.1).

The next thing to do is deciding the next topics:

- Fulfilling the proposed objectives with one extension or with several of them.
- Checking if the initial objectives are not feasible or too big and have to be reduced

In order to make importing information from very different sources possible, the use of a standard plain data format gets clear. And the simpler and more widely used this format is, the easier for getting the information. So what we need in first place is an application (OntoWiki extension) that is able to import the new information from an external source (defined by the user) in a simple data format, yet to be defined, to further work on it. This application will be referenced as *CSVLoad* (the reason for this name will become clear later on).

As discussed before, OntoWiki saves semantic data. Because of that all the information imported from other sources has to be transformed to semantic data. In this sense the user needs tools to define a semantic model and how the imported information is to be mapped to it.

Considering that the information that has to be transformed is contained in *CSVLoad*, the inclusion of this functionalities in the mentioned application is the most adequate thing to do. By this we avoid unnecessary process steps and data storage.

A user may also want to create a new semantic model from a web service which offers its information only in HTML form. But a difficult problem is that every service uses its own HTML format. This makes the importing process even harder, because specific importing tools for every web are then necessary.

It is clear that the implementation of a tool that is able to import data from any kind of HTML structure is not possible, as it would need either a sophisticated structural analysis and AI system or very complex and user-unfriendly configuration settings. Such features cannot be covered by a project of this type (a thesis project).

Considering that the whole project was build around the idea of importing requirements from the Google Code Issues platform possible and that some of its information is only available in HTML form, this project objective will be reduced to import requirements information from Google Code Issues HTML code. Including this functionality in *CSVLoad* would not make much sense due to the following reasons:

- It does not respect the philosophy of the OntoWiki extensions: one extension, as little as possible, for every functionality or process. As it is later explained (cf. section 5.1 on page 51), the importing of this specific information cannot be done at the same time as the importing and transformation of a complete model, so they take place in different processes.
- *CSVLoad* is conceptualized to be general purposed, that means, for different systems and for a standard simple format, so it will be stable and will not need major changes. On the other hand, the discussed functionality is very specific to another system that is defined by other developers. Therefore the possibilities of change are bigger, because any change on the information display in Google Code Issues can influence negatively in the correct working of the system. In this sense it is better to isolate the functionality in an independent extension, enhancing by this

way the possibility of disabling it when a change or error occurs, but keeping all other functionalities active till the problem is solved.

It is clear then that another extension (referenced as *Gcode*) is the best option.

### 3.1.1 Section summary

In order to enable the importing of large amounts of data from non semantic systems to the OntoWiki systems, new software tools are needed. This tools have to fulfill the following objectives:

- Import information from other sources
- Transform great amounts of imported information into semantic data
- Import HTML information from the Google Code Issues platform

Two OntoWiki extensions are to be developed, *CSVLoad* and *Gcode*. Their objectives are defined in the table 3.1.

CSVLoad	- Import new information in a simple data from an external source format - Map information to user-defined semantic information
Gcode	- Import requirements information from Google Code Issues HTML code

**Table 3.1:** Goal applications (conceptual)

## 4 CSVLoad

As mentioned in section 3.1 on page 13, the extension *CSVLoad* will be responsible of two main objectives:

- Import new information in a simple data format from an external source into the system
- Map this imported information to a user-defined semantic model

### 4.1 Design

#### 4.1.1 Input data format

Before designing and implementing the functionalities, the according data input format has to be defined. To fulfill the system requirements it should fit the following terms:

1. **Simplicity:** in order to make the system as flexible as possible, the best way to go is to use a data format as simple as possible, in the sense of the most plain, the better. In other words, avoid using any type of tagging, markup or type definition, so that almost any kind of data is potentially representable in this format and therefor processable by the system.
2. **Data structure:** the initial idea of the project was the importing of requirements into the OntoWiki system. This means that a lot of entities with the same structure, but with different attribute values are going to be processed. It is clear then that a collection of objects of the same class will be imported, and that means that the input will consist of an entity definition and a collection of entities which fit this definition.
3. **Multivalued attributes:** another very common property of such data entities are the multivalued attributes, that means, that a list of values of certain type are stored as one single attribute in the entity definition. A very easy example: an entity (defining an article) contains, among others, the attributes *Name*, *Theme*, *Content* and *Authors*. Although *Authors* is just one attribute in the entity definition, it can contain more than one value. In an object-like structure an attribute of a class (entity) could also be a class (entity) with different attributes, but this would conflict with the first term (*Simplicity*), so this feature will not be considered.
4. **Standard:** considering that the main goal of the application is the migration of data from one external system to the OntoWiki system, it is very important that



*DEFINITION**RECORD\_1*.  
.  
.*RECORD\_N***Listing 4.1:** Basic abstract structure of a CSV file

the used format is standard, proved and widely accepted. The objective of the application is to make the data migration as easy as possible; if a non-standard or non well-known format is used, the user is then forced to transform the data into a new, unknown format, leading to extra work and a bigger error probability.

Considering all this terms, an appropriate format is a CSV table, because it respects all the terms:

1. No tags, markups or types, only information.
2. First row defines the attributes of an entity, the rest the entities information.
3. Permits multivalued attributes by including CSV information inside a CSV attribute.
4. Most of the popular programming languages have CSV support incorporated (e.g. PHP<sup>1</sup>, Python<sup>2</sup>, Ruby<sup>3</sup>) or have free accessible libraries or tools available (e.g. Java<sup>4</sup>, C/C++<sup>5</sup>). Popular database systems (e.g. MySQL<sup>6</sup>, Oracle<sup>7</sup>) or office software suite spreadsheets (Microsoft Excel, OpenOffice Calc) also fully support the CSV format.

**CSV format**

The CSV format is a simple text format for defining information records. The structure of a CSV file is the following: every record is a line in the file, that means, that records are separated by a *new line* character. The first record defines the names of the attributes (cf. listing 4.1).

Every record consists of a series of fields, separated by commas (cf. listing 4.2).

An example of a simple CSV file can be seen in listing 4.3. As explained before, the first line defines the attributes and the following define the values of every record.

<sup>1</sup> <http://php.net/manual/en/function.fgetcsv.php>

<sup>2</sup> <http://docs.python.org/library/csv.html>

<sup>3</sup> <http://www.ruby-doc.org/stdlib/libdoc/csv/rdoc/>

<sup>4</sup> <http://sourceforge.net/projects/javacsv/>, <http://opencsv.sourceforge.net/>

<sup>5</sup> <http://www.ioplex.com/~miallen/libmba/dl/src/csv.c>

<sup>6</sup> <http://dev.mysql.com/tech-resources/articles/csv-storage-engine.html>

<sup>7</sup> [http://download.oracle.com/docs/cd/E14571\\_01/apirefs.1111/e10653/oracle/adf/model\\_adapter/dataformat/csv/CSVParser.html](http://download.oracle.com/docs/cd/E14571_01/apirefs.1111/e10653/oracle/adf/model_adapter/dataformat/csv/CSVParser.html)

```

DEF_FIELD_1 , DEF_FIELD_2 , ... , DEF_FIELD_N
FIELD_1_1 , FIELD_1_2 , ... , FIELD_1_N
.
.
.
FIELD_N_1 , FIELD_N_2 , ... , FIELD_N_N

```

**Listing 4.2:** Detailed abstract structure of a CSV file

```

City , Country , Inhabitants
Leipzig , Germany , 518.862
Barcelona , Spain , 1.621.537

```

**Listing 4.3:** Example of CSV file records (1)

Initially the field was conceived to be just a simple string of information, but this supposed a problem to include information strings that actually need to include *comma* or *new line* characters. To solve this the field information can be delimited by *double-quote* characters and every *comma* or *new line* character will be considered as part of the field information and not as a CSV file delimiter. In this case the rule *every line is a record* is only true when considering that every new line character embedded in a *double-quote* character delimited field is not being taken into account.

In case of wanting to use a *double-quote* character in the information field, the field will have to be delimited and the *double-quote* character will be encoded by a double *double-quote* character.

The listing 4.4 shows an extended version of listing 4.3, in which an attribute containing commas, new lines and double-quotes is included.

The use of double-quoting is optional (implicitly necessary when the information contains *double-quote*, *comma* or *new line* characters), but it is a good and extended practice to use them on all of the fields for uniformity and avoiding errors.

### 4.1.2 Generated information

OntoWiki saves information in form of semantic knowledge bases. This means that the information is stored in the system in the form of RDF triples. So what the CSVLoad application has to do is transform the plain CSV information into semantic data in RDF form (cf. figure 4.1).

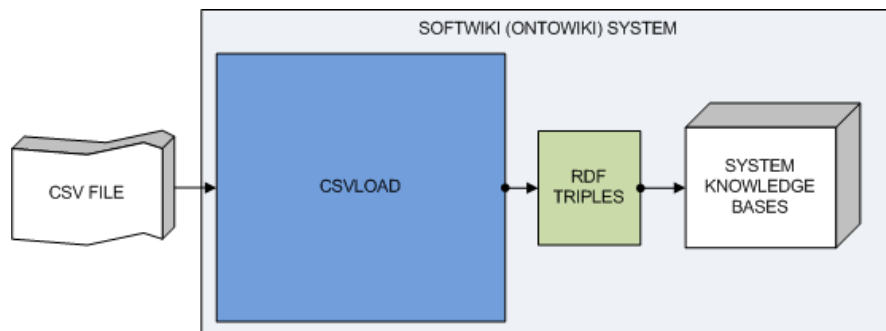
To achieve this objective the CSV file is not enough, because there is only a definition of the names of the attributes, but neither their semantic meaning nor their relationships are specified. It is clear then that some additional information to map the plain information to a semantic model is necessary.

City , Country , Inhabitants , Description

Leipzig , Germany , 518.862 , ‘ ‘... Leipzig ’s name is derived from  
the Slavic word Lipsk ,  
which means ‘ ‘ ‘ ‘ settlement  
where the lime trees (American :  
linden trees) stand ‘ ‘ ‘ ‘ ‘ .  
First documented in 1015 in the  
chronicles of Bishop Thietmar  
of Merseburg... ’ ’

Barcelona , Spain , 1.621.537 , ‘ ‘... in the 3rd century BC.  
About 15 BC, the Romans redrew  
the town as a castrum (Roman  
military camp) centered on the  
‘ ‘ ‘ ‘ Mons Taber ‘ ‘ ‘ ‘ , a little  
hill near the contemporary  
city hall... ’ ’

*Listing 4.4:* Example of CSV file records (2)



*Figure 4.1:* From CSV data to stored RDF triples

### 4.1.3 Mapping information: the RDF semantic template

Considering that the data received from the system to create the knowledge base (cf. section 4.1.1) is insufficient due to the lack of semantic information, the system has to have this information already hosted or receive it from the user.

Configuration flexibility, that means, that the user is able to add new plain-to-semantic mapping settings or modify the existing, is an important feature. The system must provide tools that permit the existence of several configuration settings, so that the user can choose the one he prefers, and provide also a way to easily add new ones.

The first step is analyzing which type of information does the system need and in which form in order to transform plain data to semantic data.

### Creating a new model

The extension needs the definition of the semantic model to adapt the imported data to it. The format will be a set of RDF triples, accordingly modified so that this definition is reusable for further knowledge base definitions and for the different record values of the CSV table, avoiding this way the redefinition of the model every time it is used.

OntoWiki offers tools for defining models and adding information directly from RDF code in either XML, JSON or Turtle format. For reasons of simplicity and readability the Turtle (N3) format is the chosen one for the mapping definition. In conclusion, the semantic model will be defined in a modified subset of the Turtle (N3) language.

In order to make the semantic model definition reusable, the use of some sort of identifying input values is the simplest yet effective option.

### Input identifying values

When creating a new knowledge base in OntoWiki two information are requested (besides the input table and the mapping information):

- knowledge base (or model) URI
- base URI

This information can be useful in order to make the semantic mapping reusable for creating new knowledge bases without having to redefine it. As a consequence some input identifying values will be accepted. The used syntax is the following:

`$<INPUT_CODE>`

where INPUT\_CODE is one of the following:

- `BASE_URI`: the introduced base URI.
- `MODEL_URI`: the introduced model URI.
- `MODEL_LABEL`: the last argument of the introduced knowledge base (model) URI, used to make the new instances easier to identify.

Those input identifying values will be included anywhere wished in the semantic mapping.

#### 4.1.4 Basic template for importing information to a knowledge base

As explained before, the goal of the extension is to import data in the form of a CSV table, that means, a lot of information following the same pattern, which is defined in the first row (cf. section 4.1.1 on page 17). So every row will be mapped the same way to the same semantic structure only changing the concrete values. To implement this functionality, constructing the semantic model definition (the semantic mapping) as a mapping template is the best way to go.

```

Germany a Country;
label ‘‘Germany’’ .
Leipzig a City;
label ‘‘Leipzig’’;
isCityOf Germany;
hasInhabitants ‘‘518.862’’;
hasDescription ‘‘...Leipzig’s
                name is derived from the Slavic
                word Lipsk, which means
                ’’’settlement where the lime
                trees (American: linden trees)
                stand’’’.
                First documented in 1015 in the
                chronicles of Bishop Thietmar
                of Merseburg...’’ .

```

```

Spain a Country;
label ‘‘Spain’’ .
Barcelona a City;
label ‘‘Barcelona’’;
isCityOf Spain;
hasInhabitants ‘‘1.621.537’’;
hasDescription ‘‘...in the 3rd century BC.
                About 15 BC, the Romans redrew the town as
                a castrum (Roman military camp) centered
                on the ’’’Mons Taber’’’, a little hill
                near the contemporary city hall...’’

```

**Listing 4.5:** N3 triples example (1)

Basically the template will be a definition of all the semantic triples that describe the semantics of a record according to the user wished semantic model, but using the attribute names instead of their values.

The following example clarifies this proposal. If the information shown in the listing 4.4 on page 19 is to be imported to OntoWiki, a possible way of defining its semantics is the set of Turtle (N3) triples shown in listing 4.5.

Now transforming this into a template in the formerly proposed format the result is something like the shown in listing 4.6.

It is easy to see that the direct substitution of the values by the attribute name can lead to confusion when processing it, because the system is, for example, unable to tell the difference between the word *Country* that represents the attribute and the one that is just a part of the semantic definition in the first line. The same happens with the *City* attribute.

```
Country a Country;
label  ‘‘Country’’ .
City a City;
label  ‘‘City’’;
isCityOf Country;
hasInhabitants  ‘‘Inhabitants’’;
hasDescription  ‘‘Description’’ .
```

**Listing 4.6:** First idea of template

```
!Country a Country;
label  ‘‘!Country’’ .
!City a City;
label  ‘‘!City’’;
isCityOf !Country;
hasInhabitants  ‘‘!Inhabitants’’;
hasDescription  ‘‘!Description’’ .
```

**Listing 4.7:** Second idea of template

Solving this situation is that simple as adding a special coding character to the attributes names to make them easy to recognize. The chosen symbol is the exclamation (!) mark as shown in listing 4.7:

Now the system is able to recognize the attributes, but further problems become clear. The attribute *Country* is used indistinctly in the first and second lines although it is a URI<sup>8</sup> in one case and a literal<sup>9</sup> in the other respectively. In the previous examples this did not suppose a problem, but if the next record is included :

Los Angeles , United States of America , 3.833.995 , “...”.

using direct substitution on the template the result would be the shown in listing 4.8.

Direct translation works when the attribute is a literal, but not when it is a URI. In this example, both the first and the third lines do not respect the triple structure. So two important facts are deduced:

1. Attributes must be analyzed (and modified if necessary) to avoid violating the correct RDF syntax.
2. The set of supported characters by a URI differs from the one supported by a literal.

---

<sup>8</sup> Unified Resource Identifier

<sup>9</sup> literal information

```

United States of America a Country;
label ‘‘United States of America’’ .
Los Angeles a City;
label ‘‘Los Angeles’’ ;
isCityOf United States of America;
hasInhabitants ‘‘3.833.995’’ ;
hasDescription ‘‘...’’ .

```

**Listing 4.8:** Result of template mapping (1)

```

!Country a Country;
label ?Country .
!City a City;
label ?City;
isCityOf !Country;
hasInhabitants ?Inhabitants;
hasDescription ?Description .

```

**Listing 4.9:** Example of template (1)

It is then necessary to define those supported character sets, make somehow URI and literal attributes differentiable and apply the right set in every case. In the case of a literal the only restriction is that the content is a standard string, while in the case of a URI the only accepted characters will be the alphanumeric<sup>10</sup> and the underscore (.). In this sense, every character that differs from the ones mentioned will be substituted by an underscore. This same restriction is applied to the mapping attributes, in order to reduce complexity and error provability. Also, in order to avoid excessively long values which could lead to malfunctions in the resource system, URIs will be cut to a maximum length of 40 characters.

To make URI and literal different between each other we can use different characters to encode them, for example the exclamation mark (!) for the URI and the question mark (?) for the literal. As literals are always double-quoted the marks are not necessary anymore and will therefor not be used. The resulting template would be the shown in listing 4.9 and the resulting output for the previous example is the shown in listing 4.10.

With those resulting template syntax and features there are enough tools to import a CSV table into an OntoWiki knowledge base. However only the direct substitution of simple attribute values is supported. Other features like multivalued attributes or values pre-processing are not supported by it at this point.

---

<sup>10</sup> case-sensitive Latin letters and Arabic digits

```

United_States_of_America a Country;
label ‘‘United States of America’’ .
Los_Angeles a City;
label ‘‘Los Angeles’’;
isCityOf United_States_of_America;
hasInhabitants ‘‘3.833.995’’;
hasDescription ‘‘...’’ .

```

**Listing 4.10:** Result of template mapping (2)

```

City , Country , Inhabitants , Universities
‘‘Leipzig’’ , ‘‘Germany’’ , ‘‘518.862’’ , ‘‘Uni Leipzig ,HIWK’’
‘‘Barcelona’’ , ‘‘Spain’’ , ‘‘1.621.537’’ , ‘‘UPC,UB,UPF’’

```

**Listing 4.11:** Example of CSV file records (3)

#### 4.1.5 Template support of multivalued attributes

Normally every attribute contains one single value. But there are some cases where one single attribute contains complex values, for example, in the form of a list of values of the attribute type. It could also contain a simpler representation of an object that fits to a certain class. However, as explained before, the extension accepts only simple CSV tables as input, so this type of representation is not accepted. In conclusion, the only complex attribute type accepted is a multivalued attribute, that means, a list of values (separated by comma or any other selected character) of the same type.

To clarify this let us keep on with the example data of listing 4.3 on page 18. Extending the table with an attribute called *Universities* containing a list of the universities of the specified *City* the result would be the shown in listing 4.11.

A possible template that would fit this data table is the shown in listing 4.12.

But this way the resulting semantic model would be very poor and would not represent every piece of information (in this case every University) as a meaningful and unique

```

!Country a Country;
label ?Country .
!City a City;
label ?City;
isCityOf !Country;
hasInhabitants ?Inhabitants;
hasUniversities ?Universities .

```

**Listing 4.12:** Example of template (1b)



entity. The most appropriate way to do it is defining every value in the list *Universities* as a *University* and refer it to the city as *University isLocatedIn City*.

In order to achieve this, the template has to support some kind of additional syntax that accepts multivalued attributes. The features of this extension should at least contemplate the following:

- **Abstract semantic definition** : following the style of the whole template, this syntax will define the abstract model for every general value in the multivalued attribute and the system will be the one to do the necessary substitutions and processing.
- **Range selection**: in some cases the user will not want to use all of the values of the multivalued attribute. He could wish only one of the values or a subset of them. The syntax must also accept some sort of index range definition to allow the user a bigger personalization level.
- **Delimiter identifying**: in the most common case the character used to separate values in a same field is the comma. In a CSV file, as example, a multivalued attribute usually contains itself a CSV record, with the particularity that all fields correspond to the same attribute definition. But the delimiter can also be some other depending on the source or type of data. In some countries the comma is used as the separating character between the integer and the decimal part of a numeric value and, in order to avoid problems with it, CSV files use the semicolon (;) character. Because of this the new syntax has to allow defining the delimiter character.

The selected way to do this is defining the attribute mapping as usual but concatenated with rectangular brackets ([ / ]) with the necessary configuration parameters inside of them, which are the range definition and the delimiter. The supported range definition at the time is the following:

1. \*: it means that all of the values should be taken.
2. *N*: where *N* is a positive integer from 0 to 9 which determines the only specific element of the values list to be mapped.

The delimiter comes next quoted ('<DEL>') or double-quoted ("<DEL>") and the parameters are comma-separated.

Considering all the previous the resulting template that would fit the data shown in listing 4.11 is the one shown in listing 4.13.

Using the improved substitution algorithm, the result of applying the template to the second record of listing 4.11 on the previous page (the information about the city of Leipzig) are the triples shown in listing 4.14.

```
Germany a Country;
label  "Germany".
```

```

!Country a Country;
label ?Country.
!City a City;
label ?City;
isCityOf !Country;
hasInhabitants ?Inhabitants.
!Universities[*,' ',''] a University.
!Universities[*,' ',''] label ?Universities[*,' ',''].
!Universities[*,' ',''] isLocatedIn !City.

```

**Listing 4.13:** Example of template (2)

```

Leipzig a City;
label 'Leipzig';
isCityOf Germany;
hasInhabitants '518.862'.
Uni_Leipzig a University.
Uni_Leipzig label 'Uni Leipzig'.
Uni_Leipzig isLocatedIn !City.
HIWK a University.
HIWK label 'HIWK'.
HIWK isLocatedIn Leipzig.

```

**Listing 4.14:** Result of template mapping (3)

#### 4.1.6 Template support of empty values

The nature of CSV tables and of information itself permits the existence of empty values. Let us consider the previous example shown in listing 4.11 with an additional record for Markkleeberg, a little city near Leipzig which does not have universities (cf. listing 4.15).

```

City , Country , Inhabitants , Universities
'Leipzig','Germany','518.862','Uni Leipzig , HIWK'
'Barcelona','Spain','1.621.537','UPC , UB , UPF'
'Markkleeberg','Germany','24.254',''

```

**Listing 4.15:** Example of CSV file records (4)

Applying the actual algorithm to this record with the template of listing 4.13, the result would be the shown in listing 4.16.

```

Germany a Country;
label 'Germany'.

```

```

Markkleeberg a City;
label 'Markkleeberg';
isCityOf Germany;
hasInhabitants '24.254'.
a University.
label ''.
isLocatedIn Markkleeberg.

```

**Listing 4.16:** Result of template mapping (4)

The last three lines are obviously wrong because they do not respect the RDF triple syntax. To avoid this problem the system will recognize if an attribute is empty and delete any RDF expression that contains it from the record semantic mapping. That means that the system would apply the reduced template shown in listing 4.17 to the mentioned record.

```

!Country a Country;
label ?Country.
!City a City;
label ?City;
isCityOf !Country;
hasInhabitants ?Inhabitants.

```

**Listing 4.17:** Reduced template due to empty values

#### 4.1.7 Template support of non-defined attributes

Along the definition of this template 2 different data tables have been used:

1. *City* , *Country* , *Inhabitants* , *Description*
2. *City* , *Country* , *Inhabitants* , *Universities*

It is clear that the templates that fit to those tables (listings 4.9 and 4.13) are the same for the first 3 attributes (*City* , *Country* , *Inhabitants*) and differ only in the last attribute (*Description* and *Universities* respectively). But even so they need their own specific template. For example, if the template of the 1. table was applied to an input table of the second type, two irregularities would occur:

1. The system would not be able to apply the semantic mapping to the input attribute *Universities*.
2. The system would not find the information to substitute the *Description* mapping definition and the resulting output triples would be wrong.

The first case just means a loss of information, but the system would still work correctly and the import process would not stop, so it is just responsibility of the user.

This loss could also be intentional if the user simply does not want or does not need to import this information.

In the second case the output would be wrong and the importing process would be interrupted. Therefore the system will also detect when an attribute, that is defined in the template, does not exist in the input table and will delete the RDF expressions just like with empty attributes, as explained in the previous section (cf. section 4.1.6).

This implies two big advantages: first, the problem of the second irregularity is solved, which means that tables with similar attributes can use the same template for importing at least all their similar information. Second, along the same line, the user could define a template with a group of attributes, but use a CSV input table containing only a subset of this attributes. This means that a general template (cf. listing 4.18) can be defined and after that different types of tables with just some attributes in common can be imported using the same template.

```
!Country a Country;
label ?Country.
!City a City;
label ?City;
isCityOf !Country;
hasInhabitants ?Inhabitants;
hasDescription ?Description.
!Universities[*,"","'"] a University.
!Universities[*,"","'"] label ?Universities[*,"","'"].
!Universities[*,"","'"] isLocatedIn !City.
```

**Listing 4.18:** Example of general template

According to the previous idea, a template that would map the following table:

- *City , Country , Inhabitants , Description, Universities*

would also map the tables mentioned before:

- *City , Country , Inhabitants , Description*
- *City , Country , Inhabitants , Universities*

As well as those:

- *City , Country , Description*
- *City , Country*
- *City , Country, Universities*

However, not every table is a valid input of the template (see section 4.1.10 on page 30).

The templates of the previous examples are simplified (no prefixes or ontology definition are used) in order to make them easier to understand. A complete, usable version of the template in listing 4.18 is available in the CSVLoad source code folder.

### 4.1.8 Template support of data type definition

RDF supports the explicit definition of the data type of a literal in this form:

```
'LITERAL_VALUE'^^<TYPE_DEFINITION_URI>
```

Because of its simplicity and utility the template supports this feature following this syntax:

```
?ATTRIBUTE_NAME^^<TYPE_DEFINITION_URI>
```

### 4.1.9 Template support of data processing functions

Sometimes the same data can be represented in different ways. A very good example is the data type *Date/Time*. This can be shown in different ways or with a different amount of information (minutes or seconds are not always necessary or used). Examples of different representations (also with different precision) of the same *Date/Time* value:

- 2010-07-16T19:20+01:00
- 2010-07-16T19:20:18+01:00
- July 16, 2010, 19:20
- Fri Jul 16 19:20:18 2010

Although they all represent the same (approximate) moment in time, they have different formats. And that is not the only problem; the first two alternatives give the information of the time zone, but the other two do not. This can lead to misunderstanding of the data. To avoid this, some systems use some kind of unique, time zone independent representation to store this data. A widely spread format is the UNIX timestamp. Originally defined for UNIX systems, but used currently in several other OS and file systems, it represents the number of seconds passed from 1970-01-01T00:00+00:00 (Second 0 of minute 0 at 00:00 on the first of January 1970) till the current moment <sup>11</sup>. In this sense the UNIX timestamp representation of the *Date/Time* shown in the upper example is 1279304418.

So this kind of data formats are very useful for sending information in a standard, error free form, but not user friendly. A user needs this information represented as year-month-day-time (or similar) format, but the template processing algorithm only substitutes the input data into RDF triples as specified. Considering the mentioned problematic, an interesting feature is offering some transforming functions that the template defining user can request when needed. This makes more sense for literals, because they are the ones that respond to such problematic in a clear way due to their data representing function.

Again the best way to include this feature is using a special encoding character (in this case the \$ character) together with an identifier code of the specific function. The resulting representation is the following (taking into consideration the syntax of the previous section):

<sup>11</sup> more info: <http://www.unixtimestamp.com/>

?ATTRIBUTE\_NAME\$FUNCTION\_CODE^^<TYPE\_DEFINITION\_URI>

being both processing function and type definition optional.

At this point only the *DATE* function code, which transforms the attribute from a UNIX timestamp to the ISO 8601 YYYY-MM-DDThh:mmTZD date-time format, is defined.

#### 4.1.10 Semantic dependencies graph

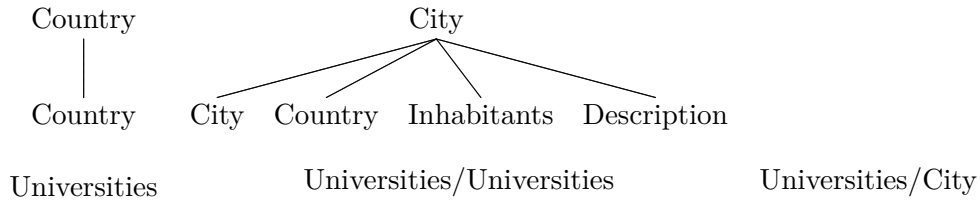
As explained in section 4.1.7 on page 27, attributes defined in the template that do not exist in the input table will be deleted from the template before executing the substitution algorithm. This feature is designed to permit the user importing new data using a general template without having to import all the defined data and without having to redefine the template for every specific table. Even so the user does not have total freedom to decide the input data because of the semantic dependencies graph.

Having a correct template as input, the semantic dependencies graph is defined like that:

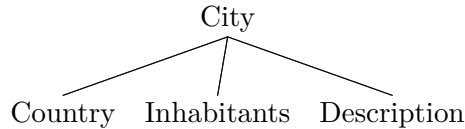
1. An expression of type “ $N \ a \ <Type>.$ ”, being  $N$  a URI, creates a node with label  $N$ .
2. An expression of type “ $N_1 \ <property> \ N_2.$ ”, being  $N_1$  and  $N_2$  URIs (or also a literal in the case of  $N_2$ ), creates a  $N_1N_2$  node.
3. An expression of type “ $\langle expression \rangle; \ <property> \ N_1; \ <property> \ N_2; \dots; \ <property> \ N_N.$ ”, being  $\langle expression \rangle$  an expression as defined in 1. or 2. and  $N_1, N_2, \dots, N_N$  URIs, creates a dependency tree, graphically shown by  $N_1, N_2, \dots, N_N$  being the child nodes (the dependent nodes) and the node defined by  $\langle expression \rangle$  being the parent node (the node provoking a dependency).
4. If a node  $N_1$  is dependent to an equivalent one  $N_2$  (a node with the same label), the nodes joint, that means, from now on there is only one node  $N_{12}$ , whose children are both the former ones from  $N_1$  and  $N_2$
5. When all nodes have been defined according to the previous rules, independent nodes (nodes with no dependency) are deleted from the graph.

The first 3 rules create the full graph and, by applying the last 2, a reduced graph (a tree) is build, clean from redundant or useless information.

When deleting an attribute from the template because it is not in the input information, the system deletes every expression in which this attribute appears (cf. sections 4.1.6 on page 26 and 4.1.7 on page 27), deleting it implicitly from the semantic dependencies graph. According to the third rule of the graph definition, the direction of the dependency is bottom-up, which means that a node that has no children can be deleted with no problem, but a parent node can only be deleted if all of their child nodes are also deleted. Otherwise a conflict is created.



**Figure 4.2:** Cities table full semantic dependencies graph



**Figure 4.3:** Cities table reduced semantic dependencies graph

The following example clarifies this. Applying the first 3 rules to the template shown in listing 4.18 on page 28 the generated full graph is the shown in figure 4.2.

The reduced version (applying the full set of rules) is the one shown in figure 4.3.

So, according to the previous statement, if the input data does not contain an attribute *City* (either it is empty or it is non-defined) the system will output wrong triples. Let us think that the 2nd record of listing 4.15 on page 26 (the information about Leipzig) has the field *City* empty. Applying the mapping algorithm with the template of listing 4.18 on page 28 the result would be the shown in listing 4.19.

An argument is missing in the third row (and consequently also in the fourth and fifth), giving a non valid RDF output as a result. To avoid such a problem there are 2 different ways to go:

```

Germany a Country;
label "Germany".
isCityOf Germany;
hasInhabitants "518.862";
hasDescription "...".
Uni_Leipzig a University.
Uni_Leipzig label "Uni Leipzig".
HIWK a University.
HIWK label "HIWK".
HIWK isLocatedIn Leipzig.
  
```

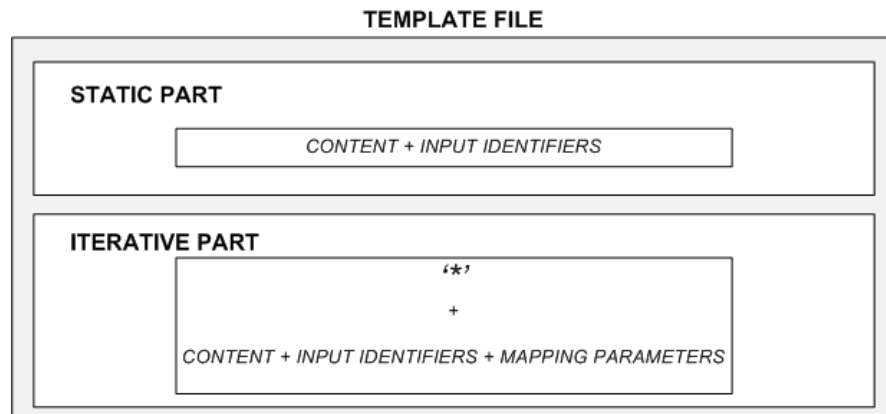
**Listing 4.19:** Triples reflecting the semantic dependencies graph problem

- **Use of identifying attributes:** make sure that an attribute that is parenting other nodes in the semantic dependencies graph is the identifier<sup>12</sup>, part of the identifying group of attributes or at least an attribute that will never be empty or non-defined.
- **Avoiding parent nodes:** if all the expressions are defined as shown in points 1. and 2. of the semantic dependencies graph, the graph will be empty at the end of the defining process, which means that there are no dependencies and therefore no conflict. In other words, avoid interconnecting expressions with the semicolon character if the first condition cannot be guaranteed.

#### 4.1.11 Static & Iterative Parts of the Template

Taking all the previous into consideration, the designed syntax for a template is a mix of a subset of the Turtle (N3) syntax and new self-defined elements. The user must define a new abstract knowledge base model, which will be personalized when instantiating, directly on the template. The template defines also the abstract structure of every record in the input CSV table.

So there is a part of the template which is used only at the beginning for defining the new model and any needed related data. This is the so-called *static part* of the template. On the other hand, there is a part that is iteratively used for mapping every record to its according semantic representation. This is the so-called *iterative part*.



**Figure 4.4:** RDF template structure

The differences between the *static part* and the *iterative part* are:

- The *Static part* defines a set of triples which are loaded in the first place. Afterwards all the triples generated by the *iterative part* are introduced.
- Attribute mapping (attr. substitution, empty values and non-defined attr. support) and data processing functions are only supported by the *iterative part*.

<sup>12</sup> an attribute with unique value and not empty



- Input identifying values are supported by both parts.
- The template is build like this:  $\langle static\ part \rangle * \langle iterative\ part \rangle$ , with an asterisk character (\*) at the beginning of the *iterative part* to mark the separation (cf. figure 4.4).

#### 4.1.12 The template syntax

The static part does not need any special treatment (except for the processing of the input identifying values) since its syntax is the usual of a Turtle (N3) document:

- Definition of a base (@base <BASE\_URI>).
- Definition of prefixes (@prefix PREFIX\_LABEL <PREFIX\_URI>).
- RDF triples.

On the other hand, the iterative part is defined with a modified subset of this language. It contains a sequence of 0 or more RDF\_EXPRs following the syntax of listing 4.20.

```

RDF_EXPR => (EXPR | MULT_EXPR)
EXPR => [ '*' '.' ';' ] ( CLASSDEF | URITOLIT | TOURI | TOLIT )
CLASSDEF => URI ' a ' PROP
URITOLIT => URI ' ' PROP ' ' LIT
TOURI => PROP ' ' URI
TOLIT => PROP ' ' LIT

MULT_EXPR => [ '*' '.' ';' ] ( M_CLASSDEF | M_URITOLIT | M_TOLIT | M_TOMURI )
M_CLASSDEF => URI [ ' ' RANGE ' ' DEL ' ] a ' PROP
M_URITOLIT => URI [ ' ' RANGE ' ' DEL ' ] ' PROP ' ' LIT [ ' ' RANGE ' ' DEL ' ] . '
M_TOLIT => [ ' ' RANGE ' ' DEL ' ] ' PROP ' ' LIT ' . '
M_TOMURI => [ ' ' RANGE ' ' DEL ' ] ' PROP ' ' URI [ ' ' RANGE ' ' DEL ' ] . '

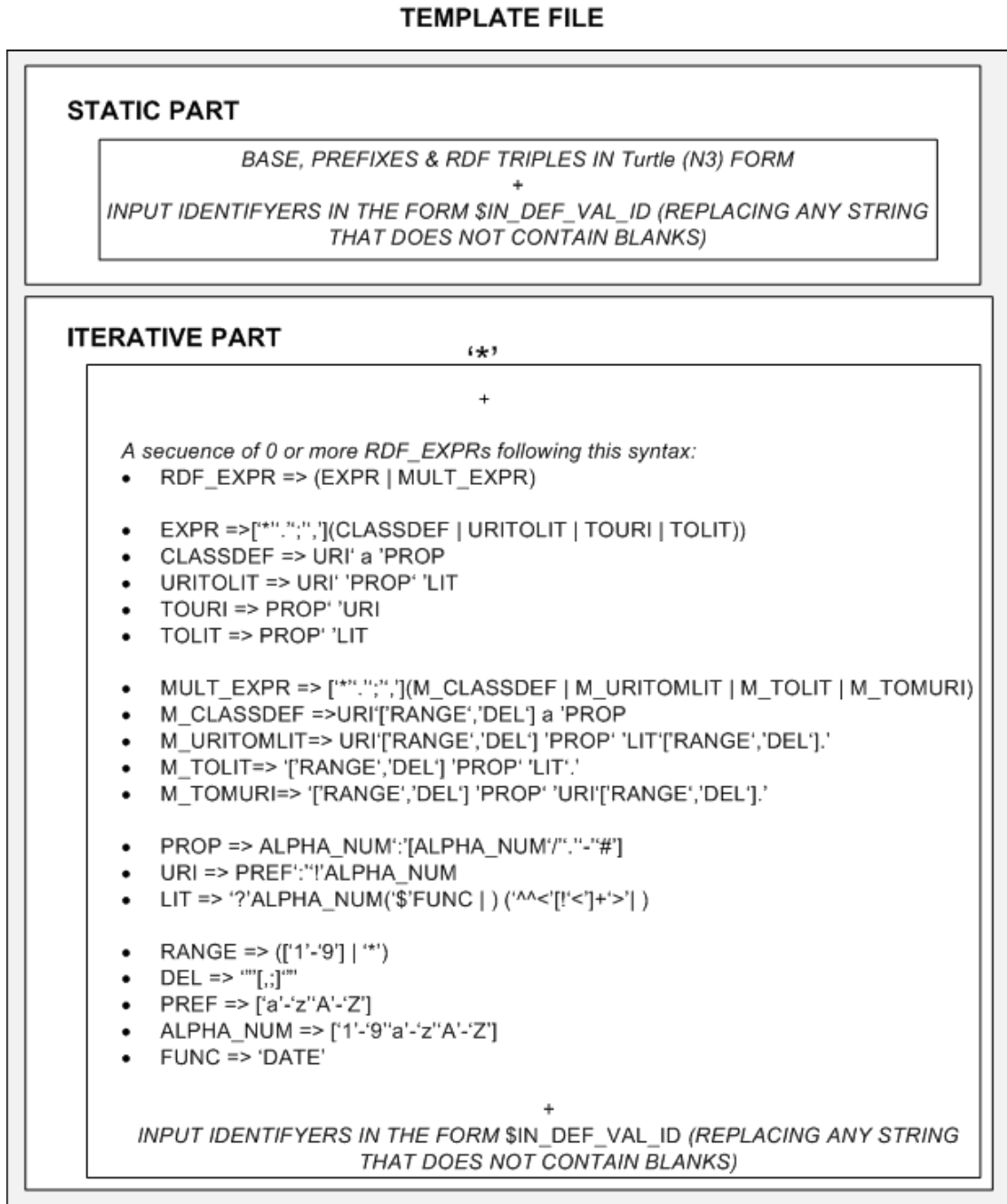
PROP => ALPHA_NUM ':' [ ALPHA_NUM / ' ' ' ' - ' ' # ' ]
URI => PREF ' : ' ! ALPHA_NUM
LIT => ' ? ' ALPHA_NUM ( ' $ ' FUNC | ) ( ' ^ ^ < ' [ ! ' < ' ] + ' > ' | )

RANGE => ( [ ' 1 ' - ' 9 ' ] | ' * ' )
DEL => ' " ' [ ; ; ] ' " '
PREF => [ ' a ' - ' z ' ' A ' - ' Z ' ]
ALPHA_NUM => [ ' 1 ' - ' 9 ' ' a ' - ' z ' ' A ' - ' Z ' ]
FUNC => ' DATE '

```

**Listing 4.20:** RDF template syntax

So the final structure of the template is the shown in figure 4.5.



*Figure 4.5:* RDF template structure & syntax

#### 4.1.13 Data dependencies & risks

The application needs the input of several data in a specific format in order to work properly. That means that the system depends on this data and, depending on the

degree of dependence on this data, the system suffers the risk of not working properly or not working at all.

The *CSVLoad* extension has 3 information sources, each with its own risks.

### **CSV table data**

The application requires an specific type of file as an input. This could mean a risk if the file type became obsolete and lesser used, or was redefined or substituted. In this case the application would stop working properly.

However, this risk should not be high due to the extended use of CSV. In addition to this, it is a very simple format conceived with the idea of being simple, so redefinition or substitution are not likely to happen.

### **Semantic template**

In the case of the semantic template the input dependency is greater than with the CSV table. The syntax of the template is specific, complex and recently developed, which makes it a source of risk. Although it is based in a subset of the Turtle (N3) language, it has new elements to which the users are not used to. Plus it is a very young technology and still needs more testing and bug fixing in order to be fully reliable.

The template system provides the user with a lot of flexibility which, in addition to the mentioned complexity and its lack of proved reliability, produces a high error possibility (both from user mistakes and from possible undetected bugs). The possibility of modifying and adding templates is only available for user with access to the file system. This type of users are in general semantic web experts or have at least a good RDF knowledge, so the learning an adapting curve should not be excessively pronounced.

### **User input data**

The system has already tools for filtering the demanded values when creating a model (knowledge base URI, base URI, file location). Even so, the system needs for them to be correct (well-defined URIs, file exist) and not used in previously existing knowledge bases, otherwise the process will stop when trying to create a knowledge base (described model already exists) or trying to load the template and CSV files (template parameters not respected or bad-defined; input CSV file does not exist, cannot be loaded or does not have the right format).

If the templates were previously well-defined by the file system administrator, no problem should occur when being selected by the application user. Otherwise the system will not be able to open the template file or the mapping will not be correctly done, and therefor the process will be stopped.

#### **4.1.14 Section summary**

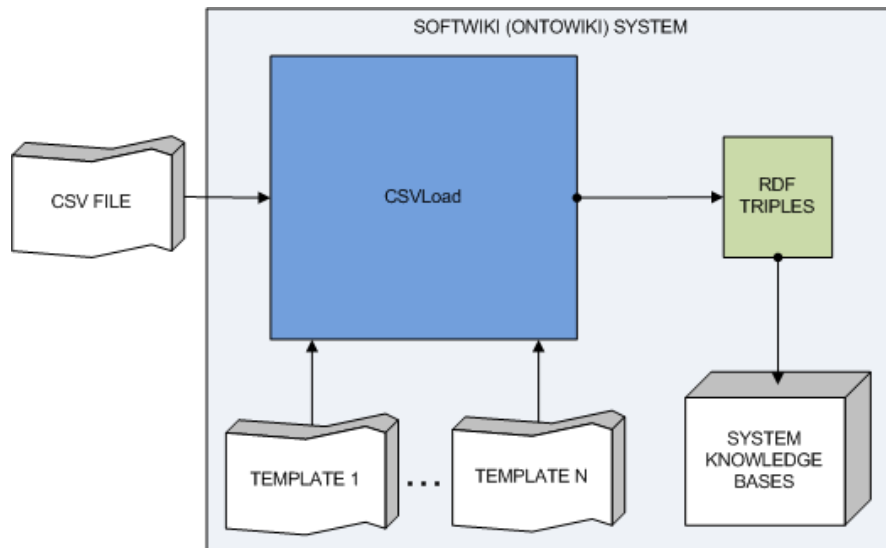
As determined in section 4.1.1 on page 16 the input information will be in the form of a simple CSV data table. In order to make the importing possible, the declaration of a new knowledge base and a semantic mapping are necessary. This mapping is implemented

with a semantic template which defines the general semantic structure that each record should follow and a specific engine that applies this semantic structure to every data record in the input CSV table. Other features of this template are:

- support of multivalued attributes
- support of empty values
- support of non-defined attributes
- support of data type definition
- support of data processing functions

The structure and syntax of this template consists of two parts (cf. figure 4.5 on page 34): the *static part*, which contains the triples that define the ontology and general properties of the model, and the *iterative part*, which contains the mapping definition for every inputted record.

The extension takes the input information (CSV + identifying values) and, with the help of one of the defined templates (which can be changed/defined by someone with access to the file system), transforms plain information into a semantic model and stores it in the defined system knowledge base, as shown in figure 4.6.



**Figure 4.6:** From CSV data to stored RDF triples using templates

## 4.2 Application requirements

In order to make sure that the main objectives are satisfactory fulfilled, concrete, measurable requirements are defined. Those requirements can be divided into two categories:

- **Functional requirements:** they define processes together with their according input data, behavior and output data. So they define *what the system does*, in other words, its use cases. The main objectives are here reflected.
- **Non-functional requirements:** they define different criteria in terms of quality, such as usability, extensibility or performance (among others). So they define *how a system is supposed to be and work*, in other words, they represent the quality and general behavior and look that the system/application should have.

#### 4.2.1 Functional requirements / Use cases

##### Import new knowledge base from CSV

###### Actors:

- OntoWiki (SoftWiki) user with administrator privileges

###### Preconditions:

- None

###### Case scenario:

1. User enters the required input data:
  - a) **String:** `knowledgeBaseUri`
  - b) **String:** `baseUri`
  - c) Depending on the CSV file source chosen option (Web/local):
    - **URL:** `webFileURL` (*optional*)
    - **URL:** `localFileURL`
  - d) **Select:** `templateId` (selection from HTML list)and clicks on 'Create Knowledge Base'.
2. The system:
  - a) creates the knowledge base
  - b) reads the CSV file
  - c) loads the chosen template
  - d) executes the mapping algorithm
  - e) introduces the new RDF triples into the knowledge base system
3. The system shows the model configuration screen with the information of the new loaded model

###### Error scenario:

- 2a → the described new knowledge base does already exist in the system
- 2b → the built path to the file is wrong or the file does not exist
- 2c → the template path was not good defined in the *component.ini* file or the file does not exist
- 2e → the inputed model is not valid (probably because of a violation in the semantic dependencies graph terms)

**Postconditions:**

1. A knowledge base with the specified knowledge base uri and base uri exists in the knowledge base system and contains the data of the CSV input table accordingly mapped to the semantic model specified in the selected template

**Add a new template****Actors:**

- User with access to the *CSVLoad* extension files and modification privileges

**Preconditions:**

- None

**Case scenario:**

1. The user:
  - a) defines the parameters of the template in the [private] section of the configuration file (component.ini) following this pattern:
    - i. properties.ttltemplates[] = "<TEMPL\_ID>"
    - ii. properties.<TEMPL\_ID>[label] = "<LABEL>"
    - iii. properties.<TEMPL\_ID>[descr] = "<DESCRIPTION>"
    - iv. properties.<TEMPL\_ID>[template] = "<TEMPL\_FILE\_NAME>.ttl"
    - v. properties.<TEMPL\_ID>[url] = "<URL>"
    - vi. properties.<TEMPL\_ID>[params] = "<PARAMETERS>"
  - b) saves the <TEMPL\_FILE\_NAME>.ttl file containing the template definition in the *ttl* folder under the extension folder (.../csvload/ttl/<TEMPL\_FILE\_NAME>.ttl).

**Postconditions:**

1. The template is accessible from now on when executing the use case *Import new knowledge base from CSV* (cf. section v4.2.1)

**Activate/deactivate template****Actors:**

- User with access to the *CSVLoad* extension files and modification privileges

**Preconditions:**

- The system contains a well-defined template with the identifier <TEMPL\_ID>

**Case scenario:**

1. The user uncomments (if wanting to activate) or comments (if wanting to deactivate) the following line of the template:
  - a) `properties.ttltemplates[] = "<TEMPL_ID>"`
2. *Optional:* the user uncomments (if wanting to activate) or comments (if wanting to deactivate) the following lines of the template:
  - a) `properties.<TEMPL_ID>[label] = "<LABEL>"`
  - b) `properties.<TEMPL_ID>[descr] = "<DESCRIPTION>"`
  - c) `properties.<TEMPL_ID>[template] = "<TEMPL_FILE_NAME>.ttl"`
  - d) `properties.<TEMPL_ID>[url] = "<URL>"`
  - e) `properties.<TEMPL_ID>[params] = "<PARAMETERS>"`

**Postconditions:**

1. The template is from now on accessible (if activated) or no longer accessible (if deactivated) when executing the use case *Import new knowledge base from CSV* (cf. section 4.2.1 on page 37).

**Delete template****Actors:**

- User with access to the *CSVLoad* extension files and modification privileges

**Preconditions:**

- The system contains a well-defined template with the identifier <TEMPL\_ID> and a file under `.../csvload/ttl` named `<TEMPL_FILE_NAME>.ttl`

**Case scenario:**

1. The user:
  - a) deletes the following lines (in case of existing) of the template in the [private] section of the configuration file (component.ini):

- i. `properties.<TEMPL_ID>[label] = "<LABEL>"`
  - ii. `properties.<TEMPL_ID>[descr] = "<DESCRIPTION>"`
  - iii. `properties.<TEMPL_ID>[template] = "<TEMPL_FILE_NAME>.ttl"`
  - iv. `properties.<TEMPL_ID>[url] = "<URL>"`
  - v. `properties.<TEMPL_ID>[params] = "<PARAMETERS>"`
- b) deletes the `<TEMPL_FILE_NAME>.ttl` file containing the template definition in the `ttl` folder under the extension folder (`.../csvload/ttl/<TEMPL_FILE_NAME>.ttl`).

**Postconditions:**

1. The template with ID `<TEMPL_ID>` and file `<TEMPL_FILE_NAME>.ttl` is completely deleted from the system.

**4.2.2 Non-functional requirements****System architecture consistency**

OntoWiki provides a sophisticated extension architecture. It is important to choose the correct extension in order to keep the system consistent, according to the information given in section 2.4.2 on page 9.

**Requirement terms:**

- Extensions, that need a high level of user interaction (more than two fields/options) or that show information of different type in the same context should be defined using a *component* extension architecture.
- Extensions, that need a high level of processing but not much user interaction and/or not much display options should be defined using a *component* extension architecture.
- Extensions, that need to show some additional/new information directly related to some existing and in an already established GUI, or that provide any event-based functionality should be defined using a *plug-in* extension architecture.

**Application:**

Considering this, it gets clear that the *CSVLoad* extension has to follow a *component* extension architecture.

**GUI consistency**

In order to provide the user with a satisfying navigation experience, new extensions and functionalities have to share a common structure with the already existing.

**Requirement terms:**



1. Every new extension has to follow the same element distribution of previous extensions/system elements accordingly.

#### Application:

As the CSVLoad extension provides a very similar functionality to the *createAction* function of the *ModelController*, the structure will be as similar as possible, in other words, CSVLoad will also be a component and will have similar (or equal) data arranged in the same way in its display.

## 4.3 Implementation

### 4.3.1 GUI Design

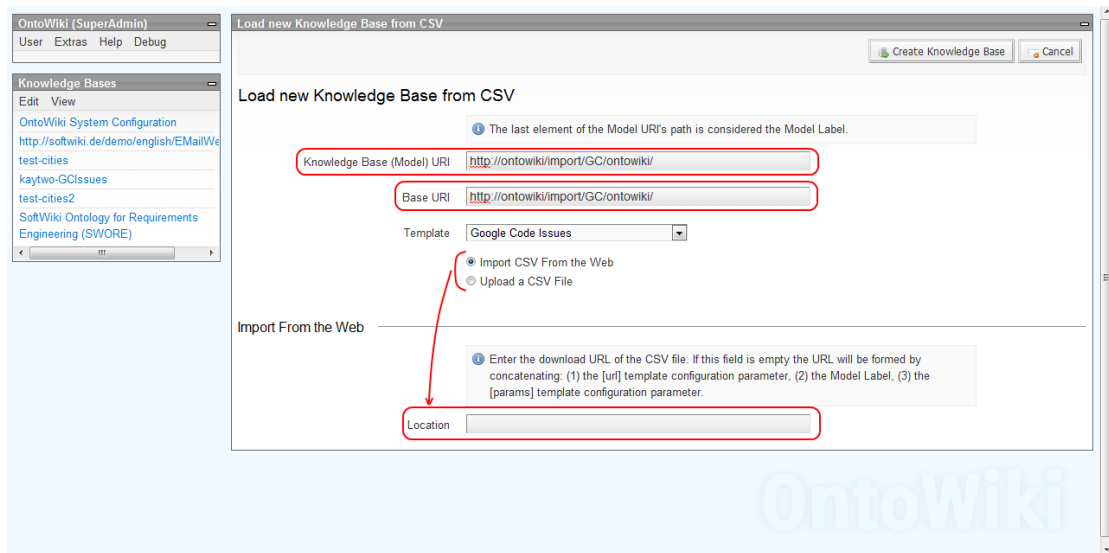


Figure 4.7: CSVLoad GUI Screen 1

The initial screen of the use case *Import new knowledge base from CSV* of CSVLoad is shown in figure 4.7. In case of successful importing, the screen of figure 4.8 is shown.

### 4.3.2 Declaring a template

In order to make the template accessible, it has to be declared in the `component.ini` file. The required parameters are:

- **templ.id:** template internal identifier
- **label:** name of the template
- **template:** name of the template file in the `ttl` folder

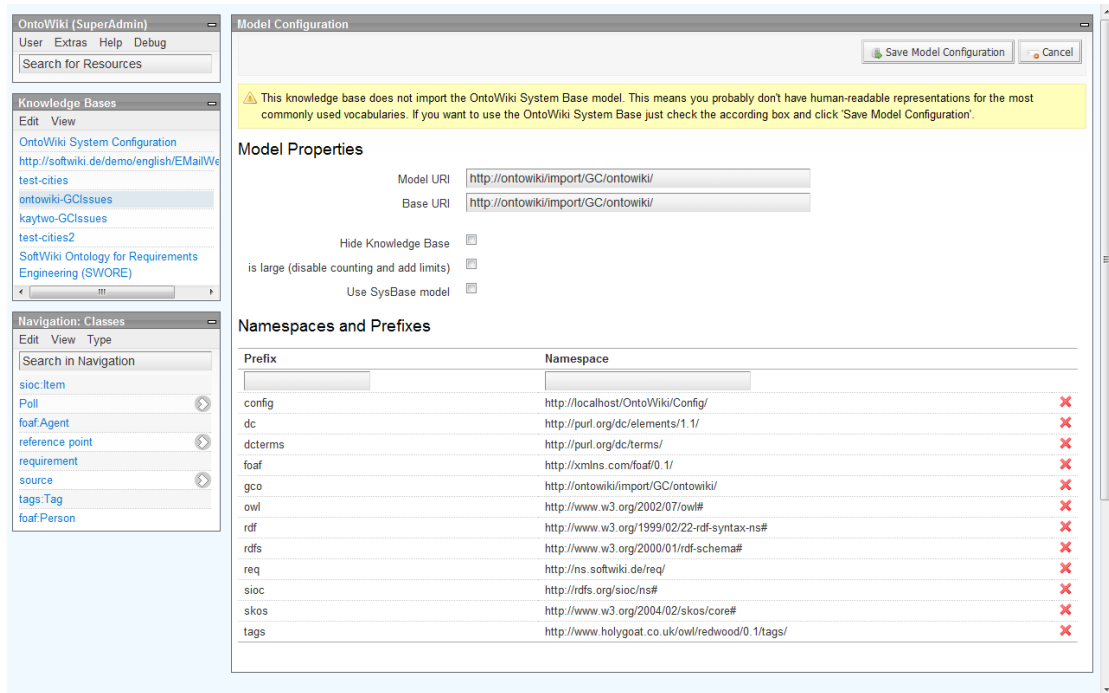


Figure 4.8: CSVLoad GUI Screen 2

- **descr** (optional): template description
- **url** (optional): url to host system of the CSV file
- **params** (optional): additional path and parameters to the CSV file

The exact syntax to define this parameters is described in the `component.ini` file of the CSVLoad extension.

### 4.3.3 Application algorithm

#### Input data & first steps

The required input data are the shown in figure 4.7 on the preceding page. If no file URL is explicitly defined, the source of the file is build by concatenating the `<url>` and `<params>` values of the selected template definition in the configuration file (`component.ini`) with the specified `MODEL_URI` in this form: `<url>MODEL_URI<params>`. This is very helpful when a template is clearly focused on a specific system which has a complex or very configurable URL. The *Google Code Issues Template* (cf. section 4.4 on page 45) is a good example; in this case *url* contains the url to the Google Code projects site while *params* contains the definition of the wished information (issue types, columns, sorting).

When the user clicks on “Create Knowledge Base” the system loads the CSV file and the template file. If the input data and the template definition are correct, the system creates the knowledge base and calls the mapping algorithm.

### Mapping algorithm

The system takes the following steps:

1. Declares the template syntax (cf. listing 4.20 on page 33).
2. Substitutes the input identifying values of the template with the according values (cf. section 4.1.3 on page 20).
3. Splits the modified template into the *Static* and the *Iterative* parts (cf. section 4.1.11 on page 32).
4. Splits the *Iterative* part into *RDF\_EXPR* expressions according to the template syntax.
5. Establishes the mapping index between the template variables array and the input columns. Taking the template on listing 4.18 on page 28 as an example:
  - *template variables array*: vars{[0]→Country, [1]→City, [2]→Inhabitants, [3]→Description, [4]→Universities}
  - *example input columns(0-4)*: City , Country , Description , Extension , Universities
  - *resulting mapping index*: matchArray{[Country]→, [City]→0, [Description]→2, [Universities]→4}
6. For every row of the CSV table (except for the name definition row) the system:
  - a) Makes a copy of the modified template to work on.
  - b) Looks for the variables defined in every *RDF\_EXPR* of the template.
  - c) If they do not exist in the mapping array, the *RDF\_EXPR* is deleted from the template copy.
  - d) For every variable:
    - i. Gets the appropriate value for the variable using the matching array.
    - ii. Applies the according content cleaning method depending on the variable type (cf. section 4.1.4 on page 20).
    - iii. Applies the according data processing function if there is any valid one specified (cf. section 4.1.9 on page 29).
    - iv. Substitutes the processed value in the template with the variable reference.
  - e) Once every *RDF\_EXPR* has been processed, the resulting N3 triples are temporally stored in an array.

7. Once every row of the CSV input table has been mapped to its according N3 triples, the processed *Static part* is concatenated with all the elements in the array (the resulting N3 triples for every record) giving as a result a String which contains the wished semantic model.

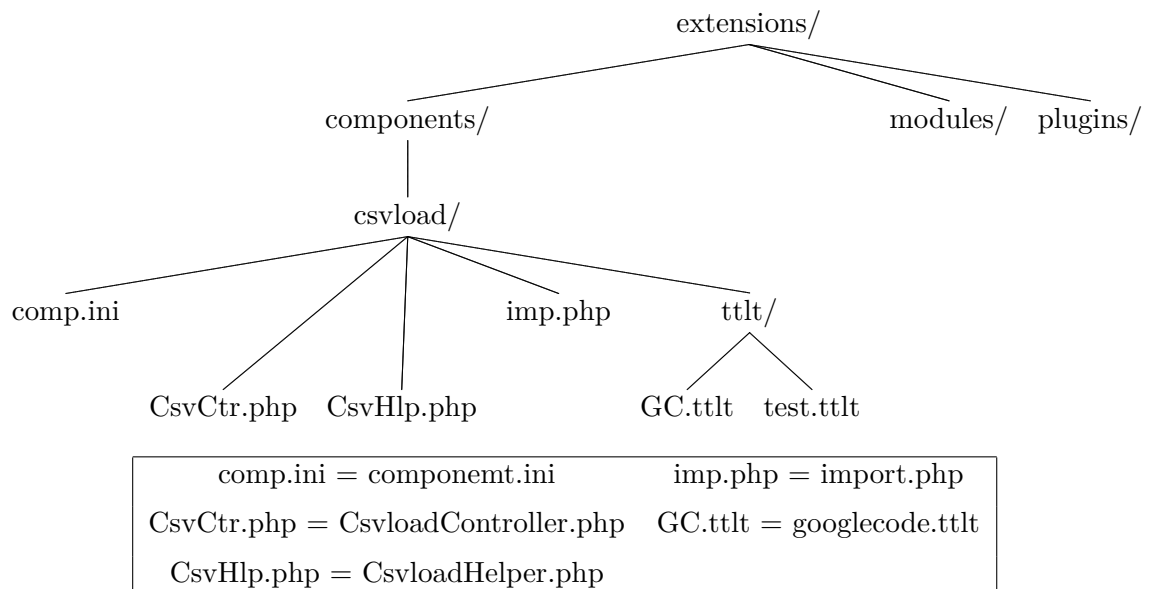
### Storage of the created model

The last step is saving the semantic model in the system knowledge bases. To do it, OntoWiki already provides a tool for this matter, the `importRdf` function<sup>13</sup>.

If the process was successful, the model configuration screen (cf. figure 4.8 on page 42) is shown. Otherwise the system reports of the problem in the CSV model import screen (cf. figure 4.7 on page 41).

#### 4.3.4 File structure

The file structure of the extension follows the abstract model of section 2.4.2 on page 9 in form shown in figure 4.9.



**Figure 4.9:** CSVLoad file tree

The functionalities of the files are the following:

- *component.ini*: contains the configuration of the extension. In the private part the templates are declared as explained in section 4.3.2 on page 41
- *CsvloadController.php*: creates the GUI, gets the basic input data (URIs) and calls the mapping algorithm (*import.php* file). Afterwards, it saves the defined model in the OntoWiki system knowledge bases.

<sup>13</sup> <http://code.google.com/p/ontowiki/source/browse/Erfurt/Store.php?repo=erfurt#982>

- *CsvloadHelper.php*: adds the “Import Knowledge Base from CSV” menu option to the “Extras” menu.
- *import.php*: loads the input CSV table and the template file, and executes the mapping algorithm. This file is defined so that does not depend on the OntoWiki framework. In other words, so that it can be easily adapted to other systems or be independent to major changes in the basic OntoWiki structure.
- *googlecode.ttl*: contains the *Google Code Issues Template* (cf. section 4.4).
- *test.ttl*: the *Cities* template, used as an example in the template definition (cf. section 4.1.4 on page 20).

## 4.4 Google Code Issues Template

The *Google Code Issues Template* defines a mapping from a CSV file to the SWORE ontology of the SoftWiki project. It gets this name because the names of the mapping attributes correspond to those of the CSV files provided by Google Code Issues, but any table following a similar structure to this files can also be imported into a SWORE knowledge base using this template.

However, there is a problem importing data from Google Code Issues, which is that the requirement description (and also the comments about this requirement) are not provided by the CSV table and have to be acquired in another way (cf. section 5 on page 51).

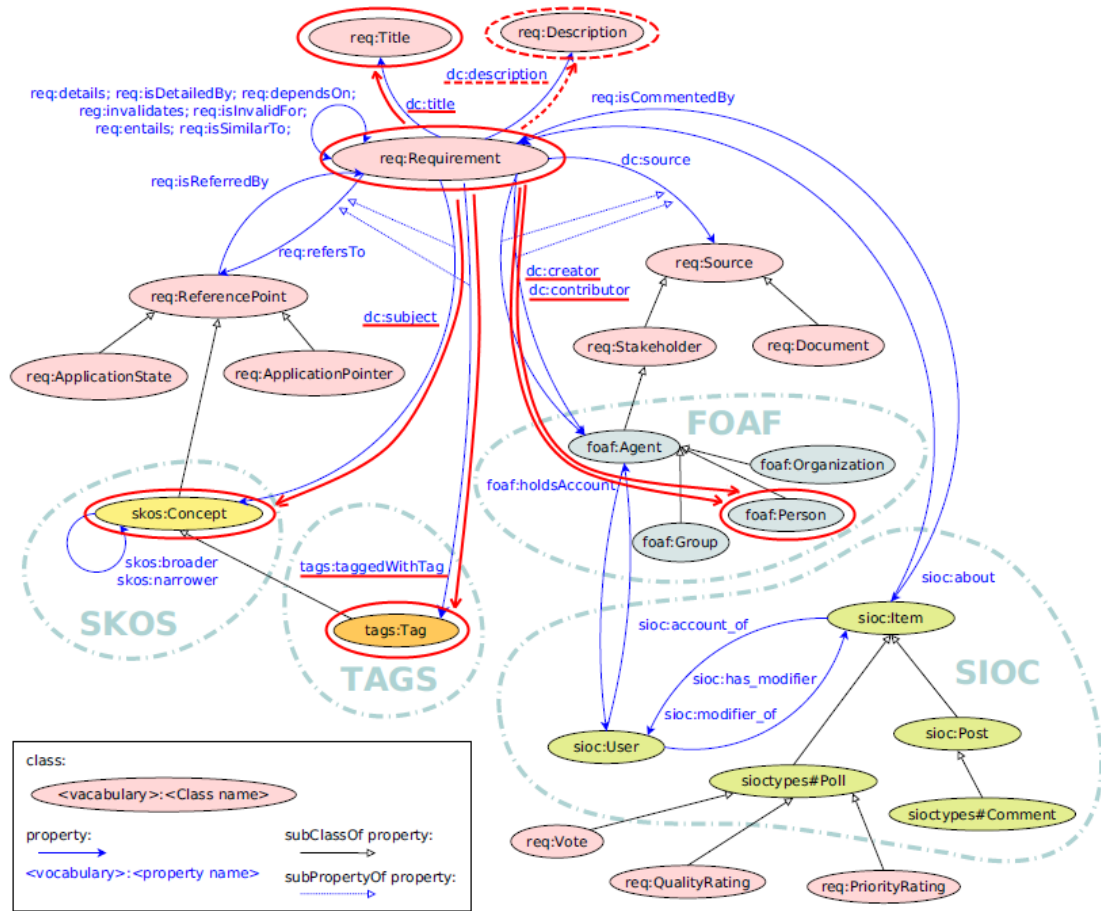
The generated semantic data corresponds to a subset of the SWORE ontology as shown in figure 4.10. As previously said, the description of the requirement is not imported from Google Code Issues, but even so the template supports this input value.

### 4.4.1 Template .ttl file

The designed template, which maps information from a CSV file provided by Google Code Issues (or a similar table) to the SWORE ontology, is shown in listing 4.21.

```
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix tags: <http://www.holygoat.co.uk/owl/redwood/0.1/tags/> .
@prefix req: <http://ns.softwiki.de/req/> .
@prefix sioc: <http://rdfs.org/sioc/ns#> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix config: <http://localhost/OntoWiki/Config/> .

<$BASE_URI> a owl:Ontology ;
```



**Figure 4.10:** CSVLoad generated part (marked in red) of the SWORE Ontology

```

rdfs:comment "Requirements for the development of the
  GoogleCode project: $MODELLabel";
rdfs:label "$MODELLabel-GCIssues" ;
owl:imports <http://ns.softwiki.de/req/>,
<http://rdfs.org/sioc/ns#>,
<http://www.holygoat.co.uk/owl/redwood/0.1/tags/>,
<http://www.w3.org/2004/02/skos/core>.

```

```

*
gco:$MODELLabelReq!ID a req:Requirement ;
dc:title ?Summary ;
dc:description ?Description ;
dc:creator config:!Reporter ;
dc:contributor config:!Owner;
dc:subject gco:!Type ;

```

```

req:modified ?ModifiedTimestamp$DATE
      ^^<http://www.w3.org/2001/XMLSchema#dateTime> ;
dcterms:created ?OpenedTimestamp$DATE
      ^^<http://www.w3.org/2001/XMLSchema#dateTime>;
skos:primarySubject ?Type ;
tags:taggedWithTag gco:!AllLabels[*,""] .

gco:!AllLabels[*,""] a tags:Tag .

gco:!AllLabels[*,""] tags:name ?AllLabels[*,""] .

config:!Reporter a foaf:Person;
rdfs:label ?Reporter .

config:!Owner a foaf:Person;
rdfs:label ?Owner .

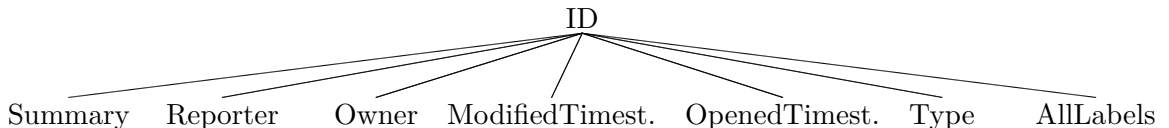
gco:!Type a skos:Concept;
rdfs:label ?Type .

```

**Listing 4.21:** Google Code Issues Template

#### 4.4.2 Template semantic dependencies reduced graph

According to the conditions described in section 4.1.10 on page 30, the semantic dependencies of the Google Code Issues Template is the shown in figure 4.11.

**Figure 4.11:** GCI Template Semantic dependencies reduced graph

The attribute *ID* is essential to build a valid output graph, but since it is an identifier value of the record, the problem defined in section 4.1.10 is solved.

## 4.5 Related projects

### 4.5.1 ConvertToRDF

*“Convert To RDF is a tool for automatically converting delimited text data into RDF via a simple mapping mechanism”* [GGP<sup>+</sup>02].

Interesting features of this tool<sup>14</sup> are:

<sup>14</sup> ConvertToRDF is available under: <http://www.mindswap.org/~mhgrove/convert/>

- Direct mapping of attributes (columns) to object with properties.
- Possibility of choosing the column that will identify the RDF object and the columns that will be mapped as object attributes, together with its RDF syntax.
- Use of a GUI<sup>15</sup> for defining the mapping. There is also a former version<sup>16</sup> that uses predefined mapping files (with either a simple own defined syntax or an RDF syntax based on a predefined ontology).

There are however some features that are not supported by ConvertToRDF but in CSVLoad:

- **Multiple objects from a single row:** A row does not necessarily contain information of only one object. The ontology may include, for example, the City and Country objects. In this case, using the information of listing 4.3 on page 18, both a City object and a Country object are to be defined.
- **Multivalued attributes:** Using as example the information of listing 4.11 on page 24, the ontology may also contain the University object. In this case the system has to be able of splitting the multivalued attribute and process every attribute separately.
- **Missing or empty attributes:** CSVLoad works with predefined templates, but does not require that the defined attributes appear the exact way in the input CSV table. If an attribute does not exist in the table (or it is empty) it is automatically omitted of the resulting N3 triples. Also if there are attributes in the table, which are not defined in the template, they are omitted too.
- **Data types:** The data type of an attribute can also be defined (e.g. for a Date/-Time attribute).
- **Data processing functionalities:** CSVLoad offers some data converting/processing functions. In this case just by adding the specific tag to an attribute the system will execute an specific routine/conversion to the tagged attribute.

### 4.5.2 Triplify

The largest part of information on the Web is already stored in structured form, often as data contained in relational databases, but usually published by Web applications only as HTML mixing structure, layout and content. The Triplify application<sup>17</sup>, born with the idea of overcoming the chicken-and-egg dilemma (simultaneously lacking of semantic representations and semantics-conscious Web search facilities) that delays the expansion of the Semantic Web, permits the conversion of web information (extracted from a relational DB) into RDF, JSON and Linked Data [ADL<sup>+</sup>09].

Interesting features of this tool are:

---

<sup>15</sup> Graphical User Interface

<sup>16</sup> available under: <http://www.mindswap.org/~mhgrove/ConvertToRDF/>

<sup>17</sup> <http://triplify.org/About>



- Easy to install and configure (with few SQL knowledge).
- Already pre-configured mappings to several popular Web applications.
- Focused on deploying the information of a Web into the Semantic Web, fact that provides several advantages:
  1. Search engines can better evaluate the content and find content more easily.
  2. Possibility of creating customized data queries, for example, easy searching for a product with certain characteristics.

Triplify, like CSVLoad, is focused on the conversion of table-based data to RDF. However, CSVLoad and Triplify differ in some aspects:

- Triplify works directly on SQL while CSVLoad does it with CSV and template-based mapping, in which case the flexibility is higher because of the possibility of defining different templates, each supporting different types of tables.
- CSVLoad is able to ignore non defined or empty attributes, so it is easier to define more general purpose input configurations than with Triplify, which demands very specific configuration.
- Triplify does not support data processing or converting.

### 4.5.3 SCOVO to RDF data converter

At the point of finishing this documentation, there is another project also developed by the AKSW, which also imports data from CSV files, but with a very different approach. This project is focused on converting biomedical statistical data of the *Global Health Observatory* (GHO) to RDF models. According to the *Statistical Core Vocabulary* (SCOVO) [HHR<sup>+</sup>09], this data consists on three concepts:

- **Dataset:** A dataset represents the container of the data, such as a table holding data in its cells.
- **Data Item:** A data item represents a single piece of data, such as a cell in a table.
- **Dimension:** A dimension represents the unit of a single piece of data, such as a time period, location or a disease.

As transforming SCOVO data into RDF in a fully automated way is not possible, since publication formats often contain implicit assumptions that have to be discovered by humans, a semi-automatic approach implements this mapping. thanks to it, he user is able to define the dimensions of the CSV table (which is shown in the form of a HTML table as in figure 4.12) and select the elements that belong to this dimension. The user can also select the range of statistical items that fit to the dimensions and store this configuration for further tables with the same structure (e.g data from different years).

**Import CSV Data**  
Configurations

Data Range: (5,3) to (16,16)

**Countries Dimension**

**Country Codes Dimension**

**Diseases Dimension**

**Data Range**

WHO Country code	3010	4005	1010	4008	1020	2010	2020	4007	5020	4010	4012	2030	3020	3025
GBD cause (b)														
Tuberculosis	127	0	7	0	40	0	6	2	0			0	0	672
STDs excluding HIV	52	1	75	0	44	0	29	2	3	1	7	0	0	258
a. Syphilis			6	0	12	0	2	0	0	0	0	0	0	24
b. Chlamydia			38	0	16	0	17	2	3	1	5	0	0	146
c. Gonorrhoea	15	0	30	0	15	0	9	0	0	0	1	0	0	85
HIV/AIDS	0	0	5	0	163	0	16	0	0	1	0	2	0	1
Diarrhoeal diseases	1,206	3	97	0	866	0	23	3	3	1	33	0	0	1,117
Childhood-cluster diseases	170	0	19	0	108	0	1	0	0	0	0	0	0	365
a. Pertussis	110	0	16	0	77	0	1	0	0	0	0	0	0	62
b. Poliomyelitis	0	-	0	-	0	-	0	-	0	0	0	-	-	0
c. Diphtheria	2	0	0	-	2	0	0	0	-	-	-	-	-	2
d. Measles	6	0	2	0	16	0	0	0	0	0	0	-	0	236

Figure 4.12: SCOPE to RDF mapping view

With this information the system is able to import the data items to RDF, with the according dimensions as properties of it.

So, in comparison to CSVLoad, this tool provides an in-line simple mapping configuration system, which can be more easy to use and more useful for importing data structures just one time in comparison to a predefined template. However, CSVLoad is more powerful in defining wide-ranged mappings that can be reused multiple times by slightly different structures (structures with very similar attribute sets, but not necessarily exactly the same).

CSVLoad focuses on typical table structures (first row defines attributes, the following the values) which define entities, while this tool focuses on data items that belong to several dimensions and that are represented in a multidimensional table, which is clearly focused on visualization, not on computing.

## 4.6 Legal implications

CSVLoad uses self-generated content (the templates, input URIs) and input data from the user (the CSV file, additional templates). The first case does not present any legal problems, but in the second case there could be. If the imported data are bound to restricted use (because of copyright or privacy terms), importing and using this data could mean a violation of some type. However CSVLoad provides only a tool for hosting data according to the wishes of the user. So in case of rights violation it is the user's responsibility.

## 5 Gcode

As explained in section 3.1 on page 14 the Gcode extension for OntoWiki (SoftWiki) is designed to import additional information from Google Code Issues and display it in the resource view screen of OntoWiki.

In oder words, the extension *Gcode* is responsible of fulfilling one main objective:

- Import information from the Google Code Issues platform and display it on the OntoWiki (SoftWiki) framework.

### 5.1 Design

Every requirement defined in Google Code Issues has its own HTML page with all the detailed information about it. The information provided by the CSV table imported with the CSVLoad extension (cf. section 4 on page 16) and the Google Code Template (cf. section 4.4 on page 45) provides some important information (titel, owner, date, etc.) but not all. Detailed description, community comments and attached files are not included in this data table, but only in the specific requirement HTML page. Therefor, in order to get this information displayed in OntoWiki (SoftWiki), the HTML page is taken as input information and analyzed by the *Gcode* extension.

*CSVLoad* imports at once the basic information of all the requirements included in the input CSV table. *Gcode* on the other hand receives an HTML page of a single requirement to analyze. The possibility of iteratively analyze all of the requirements to import the hole information was already discussed, but discarded. The reason why lays on the DoS<sup>1</sup> attack preventing mechanism. As an imported model can be really extense, the number of needed calls would also be very big. This could lead to a temporal banning from the source (Google), which could not only imply not getting all the information, but also being banned from other Google's services (in this case access to other CSV tables or others like Gmail, Google Docs) until Google's security systems could determine that it was not an attack.

Because of that, the information will be imported on demand. That means, that when a requirement which was previously imported from Google Code Issues is requested, Gcode imports the additional information from the requirement's (issue) Google Code page and stores then the information. If a requirement is requested, but its information was already imported, Gcode does not import it again.

---

<sup>1</sup> Denial of Service: saturating the target (victim) machine with external communications requests, such that it cannot respond to legitimate traffic

### 5.1.1 HTML structure

In order to get the wished information, the specific structure of the HTML website has to be defined and followed. At the moment of finishing the application's development and the writing of this documentation, the structure of the relevant information of this HTML document is the shown in listing 5.1 (capital letter names refer to the wished information).

```
<html>

  <td class="vt issuedescription">
    <a>(AUTHOR)</a>
    <span class="date" title="(DATE)"></span>
    <pre>REQ_DESCRIPTION</pre>
    <div class="attachments">
      <a href="FILE_URL">Download</a>
    </td>

  <td class="vt issuecomment">
    <a name="c1"></a>
    <a>AUTHOR</a>
    <span class="date" title="DATE"></span>
    <pre>COMMENT</pre>
    <div class="attachments">
      <a href="FILE_URL">Download</a>
    </td>

    .
    .
    .

  <td class="vt issuecomment">
    <a name="cN"></a>
    <a>AUTHOR</a>
    <span class="date" title="DATE"></span>
    <pre>COMMENT</pre>
    <div class="attachments">
      <a href="FILE_URL">Download</a>
    </td>

</html>
```

**Listing 5.1:** Google Code Issues HTML structure (reduced)

The HTML code presents an issue description, from which the REQ\_DESCRIPTION value represents the description of the requirement, followed by a list of comments with similar structure. Every comment has (among other) relevant information corresponding to this list of attributes:

- **AUTHOR:** comment author

- **DATE:** moment in time in which the comment was posted
- **COMMENT:** the comment text content
- **FILE\_URL [0..N]:** download link to the comment attachments (if any)

in the form expressed in the previous listing.

### 5.1.2 From Google Code Issues to OntoWiki (SoftWiki)

As explained in the previous section, Google Code Issues stores the comments to a requirement with *Author*, *Date*, *Comment* and attachments (*File\_urls*), as well as the requirement's *Description*. OntoWiki follows the same comments structure except for the attachments, which are directly related to the requirement. So *Gcode* stores the comments with *Author*, *Date* and *Description* as independent entities and puts all the attachments as properties of the requirement. The extension also provides the user with a direct link to the Google Code Issue website.

To avoid the DoS problem (cf. section 5.1 on page 51) the extension acts only when a requirement that was previously imported using the *CSVLoad* extension and the *Google Code Template* is requested. The initial goal of importing the data from Google Code was the possibility of changing the requirements engineering environment, keeping on with the work right from where it was, but using a different tool: the OntoWiki (SoftWiki) framework. Therefore the extension only imports the data in case that the requested requirement does not have a description. The figure 5.1.2 shows the relationships and properties of the SWORE ontology imported by Gcode.

## 5.2 Application requirements

The requirements of an application are divided in two groups: (1) Functional requirements and (2) Non-functional requirements, as explained on section 4.2 on page 36.

### 5.2.1 Functional requirements / Use cases

#### Import comments & attachment from Google Code Issues

**Actors:**

- None

**Preconditions:**

- A resource is requested to the resource view controller

**Case scenario:**

1. The system:



- 54

- d) if the requirement has already a defined *Description*:
  - i. shows a message telling that the information was already imported

**Postconditions:**

1. If the requirement was imported from Google Code Issues with the *CSVLoad* and *Google Code Template* tools, it has now the *Description*, *Comments* and *Attachments* information from the requirements Google Code Issues HTML website as accessible properties in the OntoWiki (SoftWiki) knowledge base system.

## 5.2.2 Non-functional requirements

### System architecture consistency

See description and terms in section 4.2.2 on page 40.

**Application:**

Considering this, it gets clear that the *Gcode* extension has to follow a *module* extension architecture.

### GUI consistency

See description and terms in section 4.2.2 on page 40.

**Application:**

As *Gcode* is a module extension, the rendering format inside the extension display is relatively free and can be fully adapted to the nature of the data.

## 5.3 Implementation

### 5.3.1 GUI Design

The screen of the use case *Import comments & attachment from Google Code Issues* of *Gcode* when the information was not previously imported is shown in figure 5.2. If, on the other hand, the information was already imported, the screen of figure 5.3 is shown.

### 5.3.2 Application algorithm

When the user requires some information to be shown in the resource view screen, and in order to check if the extension has to activate its display, the system:

1. Checks that the required resource is a requirement that was imported from Google Code Issues with the *CSVLoad* and *Google Code Issues Template* tools. For this, it analyzes the last part of the URI, which, in case of fitting to the mentioned conditions should have this structure: <MODEL\_LABEL>Req<REQUIREMENT\_ID>.

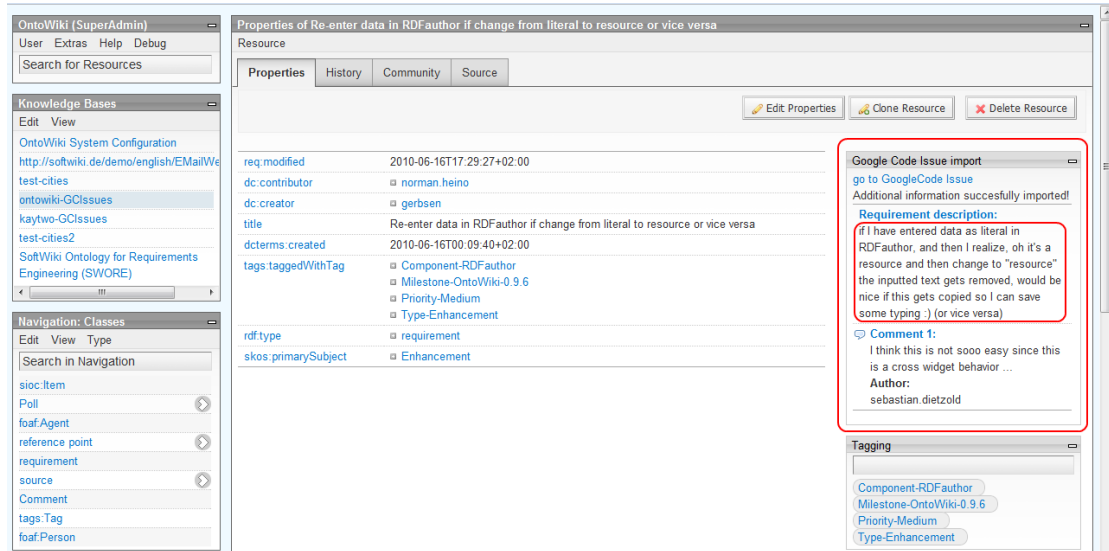


Figure 5.2: Gcode GUI Screen 1

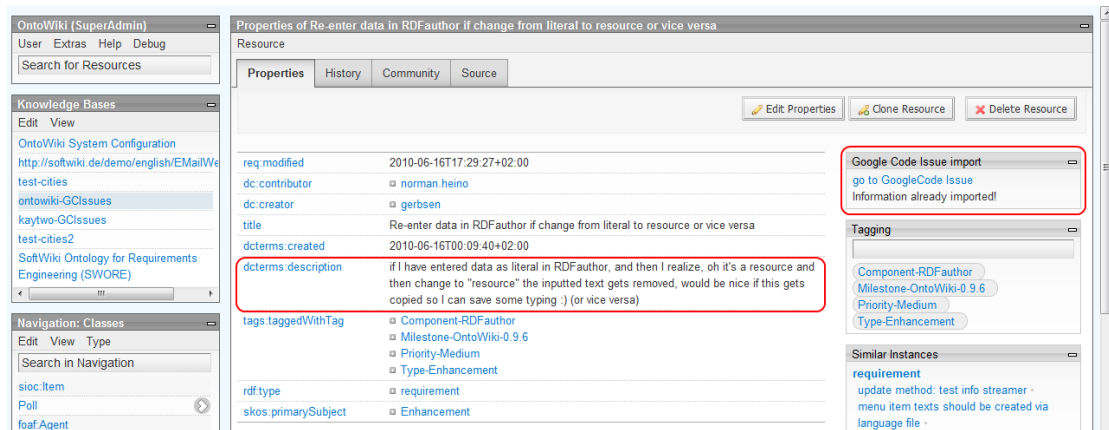


Figure 5.3: Gcode GUI Screen 2



2. Splits this structure to get the information.
3. Downloads the HTML page of the requirement from Google Code Issues.

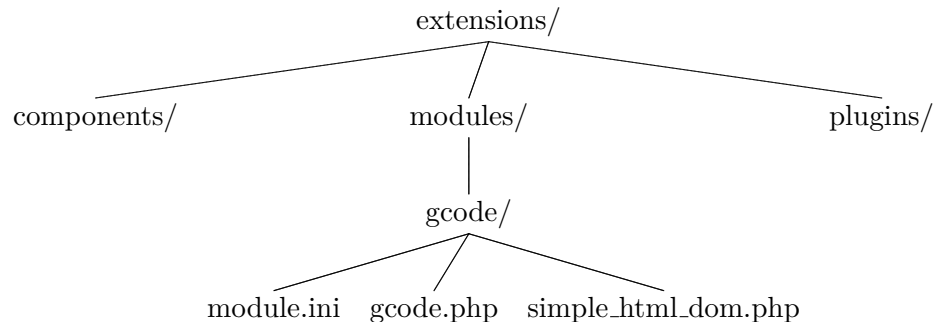
If this process was successful, it means that the resource is in fact a requirement previously imported from Google Code Issues. So what the system does next is:

1. Looks for the description property of the requirement.
  - a) If the result is not empty, the display shows a message informing that the additional information was already imported (cf. figure 5.3).
  - b) If the result is empty, the system:
    - i. Analyzes the downloaded HTML page structure and extracts its relevant information as explained in section 5.1.1 on page 52.
    - ii. Shows the extracted information (comments and attachments) in the extension display (cf. figure 5.2).
    - iii. Stores the information in the according semantic form into the system knowledge bases.

In order to get the relevant information of the HTML structure, the *PHP Simple HTML DOM Parser v1.11* tool<sup>2</sup> is used. This tool permits the easy information extracting and modifying of HTML files.

### 5.3.3 File structure

The file structure of the extension follows the abstract model of section 2.4.2 on page 9 in the form shown in figure 5.4.



**Figure 5.4:** Gcode file tree

The functionalities of the files are the following:

- *module.ini*: contains the extension configuration.
- *gcode.php*: executes the application algorithm (cf. 5.3.2).

<sup>2</sup> <http://simplehtmldom.sourceforge.net/>

- *simple\_html\_dom.php*: provides the necessary functions for navigating the HTML document.

## 5.4 Legal implications

### Simple HTML DOM Parser

This tool is licensed under the MIT License:

*“Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:*

*The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.*

*THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.”*

So the use of software of this license establishes few boundaries, less actually than the *GNU General Public License*<sup>3</sup> (or short GPL). The OntoWiki project is under the GPL license and both GPL and MIT license are compatible, so the use of the *PHP Simple HTML DOM Parser v1.11* Software is permitted and does not suppose any legal problem.

### Google

The extension is specifically designed for importing data from Google Code. Google is a private enterprise and therefore it is important to keep their Proprietary Rights and Terms of Use policy<sup>4</sup> in mind.

*“(...)Google claims no ownership or control over any Content submitted, posted or displayed by you on or through Google services. You or a third party licensor, as appropriate, retain all patent, trademark and copyright to any Content you submit, post or display on or through Google services and you are responsible for protecting those rights, as appropriate. By submitting, posting or displaying Content on or through Google services which are intended to be available to the members of the public, you grant Google*

<sup>3</sup> <http://www.gnu.org/licenses/old-licenses/gpl-2.0.html>

<sup>4</sup> available under <http://code.google.com/intl/en/projecthosting/terms.html>

*a worldwide, non-exclusive, royalty-free license to reproduce, adapt, modify, publish and distribute such Content on Google services for the purpose of displaying, distributing and promoting Google services.”*

According to this, Google does neither have any copyright of this data nor the obligation of defending any rights the owner could claim for them.

However, the author of content published in Google Services can retain rights over this content, but it is his/her obligation to defend them. This implies that the use of the Gcode extension (both commercial and non-commercial) could provoke a copyright conflict, specially taking into account that the data are automatically imported without any specific request from the user.

On the other hand, the information Gcode imports is additional information of a project hosted in Google Code. Such information has, in general, the same legal terms as the additional imported information, so the legal violation first takes place by misusing the CSVLoad extension.

Even so, in case of considering the public use of this extension, a deeper legal study would be recommendable.

## 6 Conclusion & future scope

### Conclusion

One of the most important topics of the project is the analysis and processing of data structured with different formats, which can be summarized in three levels:

#### Data embedded in HTML documents

Although this data always follows a certain pattern, it does not necessarily follow a rational, well-defined, easy to process structure, but a very specific one, with different formats and in different elements (e.g. tag attributes, tag content, different tags). The HTML structure of the Google Code Issues platform (cf. section 5.1.1 on page 52) is a good example of that. In addition to this, HTML is strongly focused on visual presentation, not on data storage. In this sense, its structure will easily change anytime that the visual presentation suffers modifications. The work on the Gcode extension shows that tools for working with this type of data are highly structure-dependent and too specific, having to change them every time the structure of the HTML site is changed, which could have only visual reasons.

#### Data in table-based form

On the other hand, if the information is stored in table-structured databases (CSV as the simplest example, a relational database as an advanced table-based architecture), it is presentation independent. This provokes the definition of an extra layer mapping the information to its visual presentation necessary, but it is yet an advantage, since the same information can be shown in different ways (depending on every necessity). If the use of standard formats is added to this, the independence of the data gets even clearer, since the share of information becomes much easier. The use of the CSV file format with the CSVLoad tool (cf. section 4.1.1 on page 17) proves it.

#### Semantic Data

However, data in table-form shows very little or none contextual information (the relationship between the different data units). Having a table like this one (*City,Country,Inhabitants*) a person could easily identify that a City is in a Country and, depending on the shown data and previous knowledge, determine to which one of the previous belongs the attribute Inhabitants. Having the data on table 6.1, with few previous knowledge almost anyone could easily identify that *Inhabitants* refers to *City*.

On the example on table 6.2 the attribute *Inhabitants* belongs clearly to the *Country* (even being very similar to the value in the previous example). But a computer is not able to recognize it for itself, so it needs to have such information specifically defined.

City	Country	Inhabitants
New York	USA	10.573.000

**Table 6.1:** Example of contextual data (1)

City	Country	Inhabitants
Brussels	Belgium	10.827.000

**Table 6.2:** Example of contextual data (1)

The first way of doing it is with a classical object-oriented structure, in which it can be defined that a Country has City and Inhabitants as attributes, or a Country has City as attribute and this City has the Inhabitants attribute. However, an object definition is final, which means that if its attributes and semantics are to be expanded, either redefining or inheriting are necessary. The semantic data, thanks to the use of triples that define the properties and attributes, can be gradually defined and expanded (cf. section 2.2.3 on page 6). So the semantic data provides the advantages of object-oriented data structures, but with a much bigger level of flexibility and expansion. This, added to the use of standard languages (RDF, RDFs, OWL) makes semantic data perfect for distributed, incremental environments like the whole Web itself.

## Future scope

CSVLoad was recently developed in an investigation environment and has therefore lots of room for improvement. The importance of implementing the following or any other features depends mainly on the use of this tool (by developers? by general users?) and the users feedback and own desires. Some interesting possible improvements or new features are:

- Allow users without access to the application file system to load new own templates. Depending on the needs and permissiveness of the system towards the users, it could be loaded just for this execution or added to the application server's template collection.
- Extend the template syntax to cover the hole Turtle (N3) syntax. At this point the template's supported syntax does not cover the hole Turtle (N3) syntax, which could represent a problem when defining a new template. Supporting the whole syntax could imply a bigger definition flexibility.
- Change the expression management in order to avoid the problems defined by the semantic dependencies graph (cf. section 4.1.10 on page 30), in other words, if a parent node is deleted, automatically delete its child nodes also. Although the

resulting models could be almost empty or incongruent since big sets of expressions could be deleted due to the lack of one single attribute, it would give the users more flexibility in defining their templates.

- Permit the configuration of CSV attribute names to template attribute names mapping. At this point, the names of the attributes in the input table have to be equal as the ones on the template in order to be matched. Allowing in-line mapping definition could be useful to allow importing from other languages (e.g. *Stadt* [City in German]  $\rightarrow$  *City*) or attributes with similar or equal meaning, but written differently (e.g. *Detail*  $\rightarrow$  *Description*) without having to change the input table.
- Possible fusion or exchange of functionalities with the SCOVO to RDF data converter tool. Combining the flexibility and potential of the template syntax with the in-line properties defining and its application to a HTML version of the input CSV table could help creating a more complete, user-friendly and less error prone CSV importing framework.
- Add more data processing functions (cf. 4.1.9 on page 29). Additional functions like type-specific checking or additional transforming ones could increase the system's data type support.
- Show template information. At this point the extension is focused in investigation and development and the users of it will normally also have access to the files and will be able to see all the information directly in the template's definition and declaration. If not, a user will only be able to see the title of the template. Adding tools to see its description and possibly its structure would improve usability.

On the other hand, Gcode is a very specific tool, designed to import specific information to a specific ontology. Therefor the most important work on it would be updating the code to possible changes in the HTML structure of the Google Code Issues platform or in the basic structure of the SWORE ontology. An improvement on the extensions behaviour could also be an option, e.g. implementing a hidden behaviour, which means importing and adding the information without showing a specific display, in order to avoid confusion or excessive information in the resource view.

It could also serve as an example for other specific tools for importing data embedded in HTML documents from other sources.

## 7 Summary

Over the last years the use of the Internet by users has evolved drastically from just consulting to publishing, sharing and modifying contents, turning the Internet into a social net in which the possibilities to collaborate and communicate grow every day bigger. A good example are the Wiki systems, which are collaborative, content-focused platforms in which the work of a community is the key to its good performance. Another of the biggest web technology developments of the Internet nowadays is the so-called Semantic Web, a Web in which every piece of data has its context clearly specified and machines are able to understand it.

The OntoWiki project merges both Semantic Web and Wiki technology, enabling the definition, modification and visualization of agile, distributed knowledge engineering scenarios. Profiting from the complex extension system of OntoWiki, the SoftWiki platform was born. Thanks to this tool and the associated Agile Requirements Engineering methodology, potentially very large and spatially separate stakeholder groups are able to gather, semantically enrich, classify and aggregate software requirements in an easy manner.

Originally created from the desire to import non-semantic requirement data from the Google Code Issues platform to SoftWiki, the *CSVLoad* extension for OntoWiki enables importing plain data out of CSV table files into OntoWiki with the help of an administrator-defined *RDF semantic template*, defined with a modified subset of the Turtle (N3) language with support of input and mapping values. The use of *CSVLoad* and the already defined *Google Code Issues Template* makes importing the requirements of a project hosted in Google Code into SoftWiki (in other words, into a SWORE ontology) very easy.

Some platforms permit exporting only a part (or in some cases none) of their information in standard formats like CSV or RDF. Instead they just show their data in HTML documents, which makes creating general, effective plain-to-semantic importing tools an extremely difficult (and in some cases impossible) task, forcing developers to build custom-made tools. The *Gcode* extension is a tool specifically built to extract additional requirements information from the Google Code Issues platform HTML code and, together with the *CSVLoad* tool, it turns importing all the requirements information from Google Code Issues into SoftWiki into an easy, automatic process.

By comparing both extensions, their input data and features, the advantages of using structured, view-independent data compared to view-representation-embedded data (e.g. data in a HTML document) become clear. But this data needs a next step, the semantic mark-up, so that computers are able to know the context of the information in an expandable, flexible environment.

# Installation & configuration

In order to install the CSVLoad and Gcode extensions, the first step is to have a working installation of OntoWiki. It can be installed from the repository with this command<sup>1</sup>:

```
hg clone https://ontowiki.googlecode.com/hg/ ontowiki
```

downloaded as a compressed file from <http://code.google.com/p/ontowiki/downloads/list> or downloaded from Mercurial following the instructions at <http://code.google.com/p/ontowiki/wiki/InstallFromMercurial>.

Once OntoWiki is installed and working, CSVLoad and Gcode can be installed. The source packages are available at the CSVLoad project page<sup>2</sup> at the SoftWiki project Web site<sup>3</sup>. After downloading, uncompress them in the OntoWiki main folder.

Developing versions can be downloaded from the Bitbucket repository at <http://bitbucket.org/tausendeins/ontowikiextensions>.

Once the files are copied, go to each extension folder (*.../ontowiki/extensions/components/csvload* and *.../ontowiki/extensions/modules/gcode* respectively) and copy the *(...).ini-dist* files to *(...).ini* in order to activate them.

The installation is now complete!

---

<sup>1</sup> go to <http://code.google.com/p/ontowiki/source/checkout> for more information

<sup>2</sup> <http://softwiki.de/netzwerk/plattform/csv-load-projekt/>

<sup>3</sup> <http://softwiki.de/>



# Glossar

**AJAX** *Asynchronous JavaScript and XML*, a group of interrelated web development techniques used on the client-side to create interactive web applications.

**AKSW** *Agile Knowledge Engineering and Semantic Web*, a workgroup of the Faculty of Informatics at the University of Leipzig.

**CSS** *Cascading Style Sheets*, a style sheet language used to describe the the look and formatting of a document written in a markup language.

**CSV** *Comma Separated Values*, a simple text format for a database table where each record in the table is one line of the file and whose record field value are separated with a comma.

**GHO** *Global Health Observatory*, an access point to the data and analyses of the World Health Organization for monitoring the global health situation.

**GNU GPL** *GNU General Public License*, the most widely used free software license.

**GPL** Short form of **GNU GPL**.

**GUI** *Graphical User Interface*, a type of user interface that allows users to interact with programs with graphical elements rather than text commands.

**HTML** *HyperText Markup Language*, the predominant markup language for web pages, which provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items.

**MVC** *Model-View-Controller*, a software architecture or pattern used in software engineering that isolates domain logic from input and presentation, permitting independent development, testing and maintenance of each.

**OWL** *Web Ontology Language*, a knowledge representation language for authoring ontologies.

**PHP** *PHP: Hypertext Preprocessor*, a widely used, general-purpose scripting language that was originally designed for web development to produce dynamic web pages.

**RDF** *Resource Description Framework*, a general method for conceptual description or modeling of information that is implemented in web resources.

**RDFs** *RDF Schema*, extensible knowledge representation language, providing basic elements for the description of ontologies.

**SCOVO** *Statistical Core Vocabulary*, an RDF vocabulary used to represent statistical data in the form of a semantic model.

**SPARQL** *SPARQL Protocol and RDF Query Language*, an RDF query language considered a key semantic web technology.

**SWORE** *Software Ontology for Requirements Engineering*, ontology for Agile Requirements Engineering.

**URI** *Uniform Resource Identifier*, a string of characters used to name and locate a resource on the Internet.

**WYSIWYG** *What You See Is What You Get*, used to describe a results-oriented graphical user interface.

# Bibliography

- [ADL<sup>+</sup>09] Sören Auer, Sebastian Dietzold, Jens Lehmann, Sebastian Hellmann, and David Aumüller. Triplify - lightweight linked data publication from relational databases. In *Proceedings of the 17th International Conference on World Wide Web, WWW 2009, Madrid, Spain, April 20-24, 2009*, pages 621–630, 2009. 48
- [Aue09] Sören Auer. Das projekt softwiki: Methoden- und softwareunterstützung für agiles, endanwender-getriebenes requirements engineering. In Sören Auer, Kim Lauenroth, Steffen Lohmann, and Thomas Riechert, editors, *Agiles Requirements Engineering für Softwareprojekte mit einer großen Anzahl verteilter Stakeholder*, volume XVIII of *Leipziger Beiträge zur Informatik*, pages 3–7. Leipziger Informatik-Verbund (LIV), 2009. 11
- [BK09] Marianne Busch and Nora Koch. Rich internet applications. state-of-the-art. technical report, ludwig-maximilians-universität münchen. 2009. 3
- [BLHL02] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web - a new form of web content that is meaningful to computers will unleash a revolution of new possibilities. *ScientificAmerican.com*, 2002. 4
- [DAR06] Sebastian Dietzold, Sören Auer, and Thomas Riechert. Kollaborative wissensarbeit mit OntoWiki. In *Proceedings of the INFORMATIK 2006 Workshop: Bildung von Sozialen Netzwerken in Anwendungen der Social Software*, 2006. 8
- [GGP<sup>+</sup>02] Jennifer Golbeck, Michael Grove, Bijan Parsia, Aditya Kalyanpur, and James Hendler. New tools for the semantic web. In *Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web*, pages 23–38. 2002. 4, 47
- [HHR<sup>+</sup>09] Michael Hausenblas, Wolfgang Halb, Yves Raimond, Lee Feigenbaum, and Danny Ayers. Scovo: Using statistics on the web of data. In *ESWC*, pages 708–722, 2009. 49
- [LHA<sup>+</sup>08] Steffen Lohmann, Philipp Heim, Sören Auer, Sebastian Dietzold, and Thomas Riechert. Semantifying requirements engineering the softwiki approach. 2008. 12
- [LR09] Kim Lauenroth and Thomas Riechert. Der SoftWiki-Ansatz für verteiltes requirements engineering mit großen stakeholdergruppen. In Sören Auer, Kim Lauenroth, Steffen Lohmann, and Thomas Riechert, editors, *Agiles*

*Requirements Engineering für Softwareprojekte mit einer großen Anzahl verteilter Stakeholder*, volume XVIII of *Leipziger Beiträge zur Informatik*, pages 39–48. Leipziger Informatik-Verbund (LIV), 2009. 11

[Tec05] BBC News Technology. Wikipedia survives research test. *BBC News*, 2005. 3

[Wag04] Christian Wagner. Wiki: A technology for conversational knowledge management and group collaboration. In *Communications of the Association for Information Systems (Volume13)*, pages 265–289, 2004. 3