

5

DogmaBank

In the last section of the previous chapter we have described a meta-model of emergent semantic systems. We applied the model to social bookmarking tools and discovered that the technical system of most tools stays on the linguistic level. In this chapter we will describe our DogmaBank system that explores a possible implementation of the technical system of a social bookmarking tool at the conceptual level.

5.1 Introduction

We start with a birds' eye view of DogmaBank and gradually zoom in down to the code level. We first clarify the high-level design decisions in section 5.2 and characteristics of DogmaBank in section 5.3. Then we describe the user interface and provide screenshots in section 5.4. We provide more detail about the architecture in section 5.5 and illustrate the inner workings of the system with UML diagrams in section 5.6. Finally we want to offer to future DogmaBank developers a head start by listing and discussing the most important pieces of code in section 5.7. Finally, in section 5.8 we explain how we have tested the system.

We checked out the latest stable version of Piggy Bank from the MIT subversion tree and adapted the code to connect with STAR.Lab’s DOGMA Concept Definition Server. The name “DogmaBank” reflects the integration of the two code bases of “DOGMA” and “Piggy Bank”.

5.2 Design decisions

In this section we describe the different possible approaches to tagging and explain why we choose tagging with concepts from an ontology. We also describe why we choose to use Piggy Bank, and why we choose to use DOGMA.

5.2.1 Different approaches to tagging

DogmaBank is a social bookmarking tool. It can be used to tag web pages. Throughout this thesis text, we have described several approaches to tagging. We will summarize them here before we select one particular approach. We can split them into two groups: tagging with freely chosen keywords or tagging with concepts.

Tagging with freely chosen keywords

Below, we describe three approaches to tagging web pages using natural language words (or sequences of words when the social bookmarking tool allows spaces in a tag).

- **A tag is its own identifier.** This means a tag equals another tag if they consist of the same string. This is the approach used in del.icio.us.
- **A tag equals another tag if their strings are the same, unless specifically indicated otherwise.** This is the approach used in Piggy Bank, as described in section 4.5.2 on page 73. For this to work, a URI is generated for each tag. This approach works similarly as the previous, except that the OWL constructs OWL:DifferentFrom and OWL:SameAs are explicitly specified between tags¹.

¹Note that the URI generation is implemented in the current Piggy Bank version, but the user interface does not yet support specifying these OWL:DifferentFrom and OWL:SameAs relations

- **A tag is different from any other tag, except if they are mapped to each other.** This approach is exactly the inverse of the previous approach and is described in section 4.4.4 on page 67.

Tagging with concepts

The approaches described here use concepts defined in an ontology to tag a web page.

- **Rigid** One way would be to allow people to tag a web page only with one of the concepts present in an ontology. This is similar to the approach of a controlled vocabulary that has been a priori exhaustively defined, the problems of which are touched upon in section 4.3.3 on page 62.
- **Anarchistic** The opposite approach is to allow everybody to specify new concepts in the ontology. This approach is called a folkontology (see section 4.4.1 on page 64) and the expectation is that the “best” concepts are used most and stand out thanks to their frequency of usage.
- **Moderated** An in-between approach is to use a moderated system. Users can propose new concepts to be added to the ontology. A group of moderators or ontology engineers review the proposed concepts. This is similar to the approach used in the Open Directory Project DMOZ (first mentioned in section 3.4.1 on page 39). Following this approach, users can propose new subcategories, which are moderated by more experienced users. The higher up the hierarchy, the more experienced a user needs to be to approve a new category.

We believe the rigid approach is not flexible enough. The need for new concepts will invariably arise, so one fixed ontology once and for all without any facilities for change is not an option. The intensive research on ontology evolution [Lee04] shows the need to cater for change.

At the other side of the spectrum, we have the extremely flexible anarchistic approach. The drawback of this approach is that the ontology gets polluted with everybody’s concept definitions. After a while, when a user performs a simple search for a concept like “dog” or “person”, he will be presented with a list of tens of definitions that probably differ only in a slight way. This is not very user friendly and rather confusing. One way to cope with this problem is to put the most used definition at the top in order to promote agreement on concepts by having the most

popular ones bubble up. The problem is that popularity is not a good measurement of quality and correctness of a concept. It could be that a definition is not used very often, but is correct and relevant nevertheless. For example, a user searches for the concept “latex” and has to scroll ten or twenty definitions of \LaTeX the typesetting system, before he finds the milky fluid derived from the rubber tree he was looking for.

To avoid this kind of ontology pollution, we choose the moderated approach. When a user cannot find the concept he is looking for, he can suggest a new concept. The moderator decides which concepts are added to the ontology. When similar, only slightly different concepts are suggested, he can choose to merge them into one concept.

5.2.2 Why Piggy Bank

Advantages of Piggy Bank

Piggy Bank is a relatively well known, open source, academic system for collecting and sharing RDF data and tagging web pages (see section 3.3.3 on page 36).

We could have chosen to build YABT (Yet Another Bookmarking Tool) from scratch, using one of the Web2.0 rapid development frameworks (like Ruby on Rails, see section 3.4.4) that are increasingly becoming available and maturing rapidly. We feel this would have been an isolated attempt. Instead, we choose to adapt a well known system, so maybe our ideas will find their way to the main branch of the Piggy Bank source code tree and our code will not be lost. The developers of Piggy Bank are currently putting systems into place to accept contributions from the community. Even when our code does not make it into the main branch, we can still take advantage of the further enhancements of the main Piggy Bank branch by regularly merging our code with the main branch, or by making it available as a patch (or extension) to Piggy Bank.

We choose to preserve all of the existing Piggy Bank functionality in our Dogma-Bank implementation. This makes the transition for an existing Piggy Bank user easier. A user can still choose to tag a web page with a freely chosen keyword instead of with a concept. Why would a user want to do this? It is possible he is waiting for a concept to be added to the ontology by the moderator, and uses a freely chosen keyword for now. It is also possible he wants to mark a web page as

“toread”, which is a popular tag on current bookmarking tools, but has no relationship with the content or topic of the page.

Disadvantages of Piggy Bank

A disadvantage of the Piggy Bank architecture is that it is a Firefox extension. This limits the number of potential users of the system not to those using a certain operating system platform, as Firefox and the Piggy Bank extension are available for Linux, Mac OS X and Windows, but to the users of the Firefox web browser. To early adopters and technology savvy people this sounds like an advantage, but in reality it is a serious limitation. We have already been asked by a business partner of STAR.Lab [Ver06] what the possibilities are for our extension in the context of i-City², a city wide wireless internet project in Hasselt, Belgium. One constraint is that we would have to use Internet Explorer, as the project is Microsoft sponsored. Re-implementing Piggy Bank as an Internet Explorer plug-in will not be a priority to the Piggy Bank developers any time soon.

An other disadvantage of Piggy Bank being a Firefox extension is that the data is stored locally on the computer. An initial goal of on-line bookmarking tools is to make the bookmarks available to a user no matter which computer he is using (at work, at home, at school, ...). Piggy Bank requires a user to publish his bookmarks to one or more Semantic Banks as described in section 3.3.3, e.g. the Semantic Bank of his research group, company, organization, etc., before he can access them on an other computer and manually re-save them in an other local Piggy Bank. At the moment, there are no facilities to automatically synchronize local Piggy Banks stored on different computers.

Note that the main goal of our DogmaBank implementation is to be a proof of concept, so people can experience how the “Tag with concept” approach works in practice, and have them consider to implement it in their own system.

²<http://www.i-city.be>

5.2.3 Why DOGMA

We first explained about the DOGMA Concept Definition Server in section 2.5 on page 22. The server contains a set of concepts. Each concept has a numerical, system generated, unique ConceptID, a definition, and a gloss. In practice, the definition is a synset and the gloss a definition (often containing a gloss). A synset is a set of one or more synonyms [Fel98]. This is one example of a concept:

```
ConceptID: 15649
Definition: car, elevator_car
Gloss:     where passengers ride up and down;
           "the car was on the top floor"
```

In DogmaBank, the user can search the concepts. We make use of two look-up methods offered by the API of the Concept Definition Server. The two sets of concepts that are retrieved with these two methods are merged to avoid duplicate entries.

The first is `retrieveConceptsByDefintion(String termLabel)`. This method searches the start of the definition field (which is a synset) of every concept. It would be better to do a full text search of the definition, but as the most commonly used word for that concept is usually the first word in the definition (synset), in practice this seems to yield enough results to have a workable system.

The second method in the API that we call to get a second set of lexons is `retrieveAssociatedConcepts(String termLabel)`. This needs some more explanation, for which it is helpful to have read section 2.5 on page 22 about DOGMA. The lexons in the Lexon Base are grouped into contexts. A term in a lexon can be linked to a concept in the Concept Definition Server. A term and a context have a unique mapping to a concept. The `retrieveAssociatedConcepts` method retrieves all concepts that are linked to a term. A term can occur in different contexts and so it can be linked to different concepts. This is how homonymy is implemented in DOGMA.

The advantage of the second method is that concepts can be retrieved with a term that is not even present in the definition (synset) of those concepts. It can be compared with the Google search algorithm that also uses the words that are used to link to a page to retrieve that page. The most widely known abuse of this well known algorithm is the biography of George W. Bush Jr. on the web site of the White House that comes up as the first search result for "miserable failure", even

though those words do not occur on that page. What happened is that a group of bloggers agreed to use those words to link to the White House web page to influence the Google search results. An other, more practical example is that a del.icio.us search for “folksonomies” also brings up pages only talking about the word “tagging”.

5.3 Characteristics of DogmaBank

We will first talk about the high-level characteristics of our system. The next section will talk about the exact interactional behavior of the system.

In [SdM06] a list of characteristics for emergent semantics systems is discussed. We will apply these characteristics to DogmaBank.

5.3.1 Natural language words vs. language independent concept labels

Most tagging systems use natural language words as tags and assume the language used is US English. With a few quick examples we will show that it is wrong to assume that the native language of all the users is English or to assume that all the tagged content is in English. In del.icio.us, there is no way to indicate what language a tag is in. The problem is similar to homonymy and synonymy.

A tag can have different meanings in different languages.

The tag “beer” has a different meaning in English or in Dutch. A quick del.icio.us search for the latest twenty web pages tagged with “beer”³ include results in English, Russian, Italian, and German. All results are about the English meaning of the concept that is defined in the DOGMA Concept Definition Server (CDS) as “a general name for alcoholic beverages made by fermenting a cereal (or mixture of cereals) flavored with hops”. None talk about the Dutch meaning of the tag, which is translated to English as “bear” and described in the CDS as “massive plantigrade carnivorous or omnivorous mammals with long shaggy coats and strong claws”.

³<http://del.icio.us/tag/beer>

A concept can have different translations.

When we do a search for “bier” (which is both Dutch and German for “beer”), we get some results in German, but still most of the results are in English. All the pages are about the concept “beer” (“a general name for alcoholic beverages made by fermenting a cereal (or mixture of cereals) flavored with hops”), even though in English “bier” means “a stand to support a corpse or a coffin prior to burial”.

A search for “appel” returns quite some results about “appel d’offres” (French for “invitation to tender”), but also about a famous artist called Karel “Appel”, recipes with “appels” (Dutch for “apple”), a drama group called “de Appel”, and Apple computers.

We believe the above examples show the current approach to multilinguality sometimes results in a big mess. Some users use their native language to tag items in English and other users use English to tag items in their native language.

Our system does not use natural language words to tag items, but concepts. These concepts are not identified by a label in some language, but by unique concept identifiers. The concepts are stored in the DOGMA CDS with a number as unique identifier. For the moment, the CDS is populated with concepts that are described in US English only. However, recently, the CDS has been adapted to cater for multilinguality along the lines set out in [BSM03].

The current implementation of the DogmaBank system does not take multilinguality into account and requires the use of US English. Since the CDS is ready for multilinguality, we will outline how a future version of DogmaBank could work to deal with different languages in section 6.3.1 about Future Work.

5.3.2 Informal vs. formal semantics

Ontologies are formal models containing concepts and relationships between these concepts. These relationships are expressed in some first order language to enable reasoning. A concept in an ontology is formally defined by its relationships, e.g. a car has wheels, a car is a kind of vehicle, etc. The reasoning is done using concept identifiers, in our case unique numbers. This is fine for machine-machine interaction. For humans to understand the concepts we need informal meaning descriptions and human readable intuitively understandable concept labels. These

descriptions and labels are stored in the Concept Definition Server.

The Concept Definition Server contains a definition (synset) and a gloss (definition) for each concept, e.g. concept number 15647 has as definition “car, auto, automobile, machine, motorcar” and as gloss “4-wheeled motor vehicle; usually propelled by an internal combustion engine; “he needs a car to get to work””.

5.3.3 A bag of loose words vs. a well structured semantic network

DogmaBank chooses an approach that combines both. For a user, it looks like DogmaBank uses a bag of loose concepts, but in the back-end we can make use of the relationships between the concepts as stored in the commitment layer. In section 6.3.3 about Future Work we describe one way to exploit these semantic relations at retrieval time.

5.3.4 Effort at creation time vs. effort at usage time

Most tagging systems make it very easy for a user to tag a web page: just type in a list of words and click the “Tag” button. On the other hand, the effort to retrieve the tagged web pages is bigger. A user has to try different notational variants, e.g. “semweb”, “semantic_web”, “semantic-web”, etc. A user also has to think of different synonyms and maybe even different languages to be sure to get most of the pages on the topic.

Tagging web pages with concepts requires a user to perform an extra step. He types in a word and gets a list of concepts related to this word. He then selects one of the concepts and hits the Tag with concept” button. This process requires extra effort at the creation time of the bookmarks. The payback comes in the form of more accurate results and less noise when querying bookmarks.

5.3.5 Individual creation decision vs. group-wise creation agreement

In most tagging systems, creating a new tag is an individual decision: a user just types in the tag name when bookmarking a web page. Which concepts are included in an ontology is generally the result of a group-wise decision process, supported by a methodology as DOGMA-MESS [dM05a]

In our system the creation of new concepts is decided by a single ontology engineer, so there is no group-wise creation agreement (yet). Our current system does not support a real meaning negotiation process between the ontology engineer and the users.

5.3.6 The (unbearable?) lightness of being a tag creator vs. the authoritative weight of a relevant stakeholder

In most tagging systems, creating a new tag is very easy. Usually, for the creation of an ontology, a group of domain experts and relevant stakeholders is consulted.

In our system, users can only make suggestions for new concepts to be added. We assume there is an ontology engineer approving the concepts suggested by users. This decision is made, because we do not want the Concept Definition Server to be polluted with overlapping concepts. An ontology engineer can bundle suggestions and refine definitions before adding new concepts to avoid such pollution.

5.4 User interface

In this section we describe the user interface of DogmaBank. We go over the steps to bookmark a web page, to browse the bookmarks, and to suggest a concept.

5.4.1 The tagging process

In this section we describe the process of a user tagging a web page. We will first describe the tagging process of a user tagging a web page with a freely chosen keyword. Then we will describe the concept tagging process of a user tagging a web page with a concept.

Tag with a freely chosen keyword

We maintained the Piggy Bank functionality of tagging with a keyword by putting it in a different tab as can be seen in the screenshot in figure 5.1. The reasons have

been described in the design decisions section (5.2.2). One of the reasons is that we do not want to deprive the user from the ability to tag a page as “toread”.

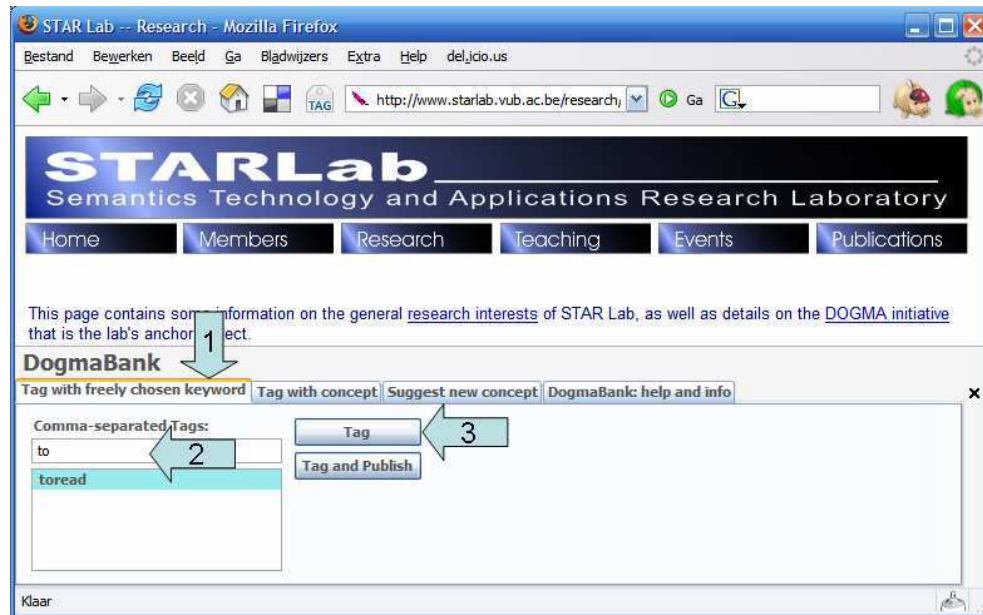


Figure 5.1: DogmaBank screenshot of the Piggy Bank tagging functionality

When a user activates the Tag Bar by selecting the appropriate menu option (“Tools” > “Piggy Bank” > Tag this page”) or by hitting the backslash key, the Tag Bar appears at the lower half of the Firefox browser window. The Tag Bar contains four tabs: one for tagging with a freely chosen keyword, a second for tagging with a concept, a third to suggest a concept, and a fourth for help and info about DogmaBank. The second is selected by default, so the user selects the first (1). He then sees an input field (2) and a “Tag” button (3). The user inputs a list of comma separated keywords. Spaces are allowed in a tag. While the user is typing, a list of possible tag completions is suggested in a listbox just below. Selecting a tag completion is done with the up and down arrow keys. Activating the completion is done with the “enter” key. The suggestions are tags that have been used by the user for other pages. When the user hits the “Tag” button, the tags are stored in his local Piggy Bank.

Tag with a concept

The new “Tag with concept” functionality is put into a separate tab (1) that is selected by default, as can be seen in the screenshot of the Volkswagen home page being tagged with the “car, auto, automobile, ...” concept in figure 5.2.

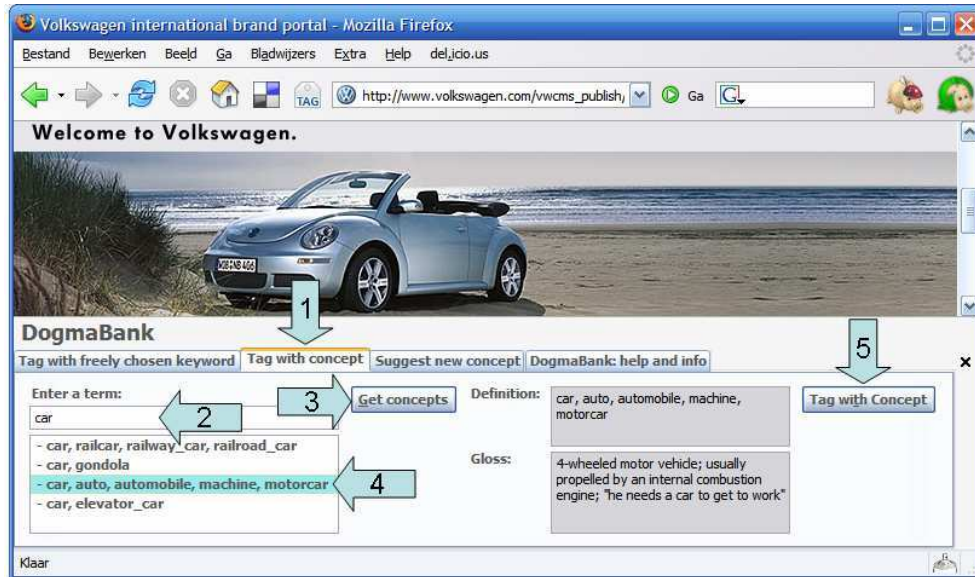


Figure 5.2: DogmaBank screenshot of the “tag with concept” process

The user enters a search term (2) to search for concepts and hits the “enter” key or clicks the “Get concepts” button (3). The results appear in a listbox (4). The user selects a concept. The definition and gloss appear on the right. When happy with that concept, the user clicks the “Tag with concept” button.

Note that we had also implemented a search-as-you-type solution in which each key stroke triggers a remote connection to the DOGMA server. The connections happen asynchronously, which causes the search results to arrive in random order. This leads to errors like the search results for “tes” being displayed while the user has typed “test”, because the search results of “tes” arrived after those of “test” and the window is updated with the search results that arrive last.

5.4.2 Browsing bookmarks

To browse the bookmarks, the user selects the menu option “Tools < Piggy Bank < Browse my Piggy Bank”, clicks the little piggy button in the top right corner of the Firefox browser, or hits the key combination “Alt - Shift - P”. The user is presented with the user interface as shown in figure 5.3. At the left of the page the user sees a “Browse Data by Tag” and a “Browse Data by Concept” area. The “Browse Data by Tag” area is part of the original Piggy Bank. The “Browse Data by Concept” area is what we added for DogmaBank. We will only focus on the area we added.

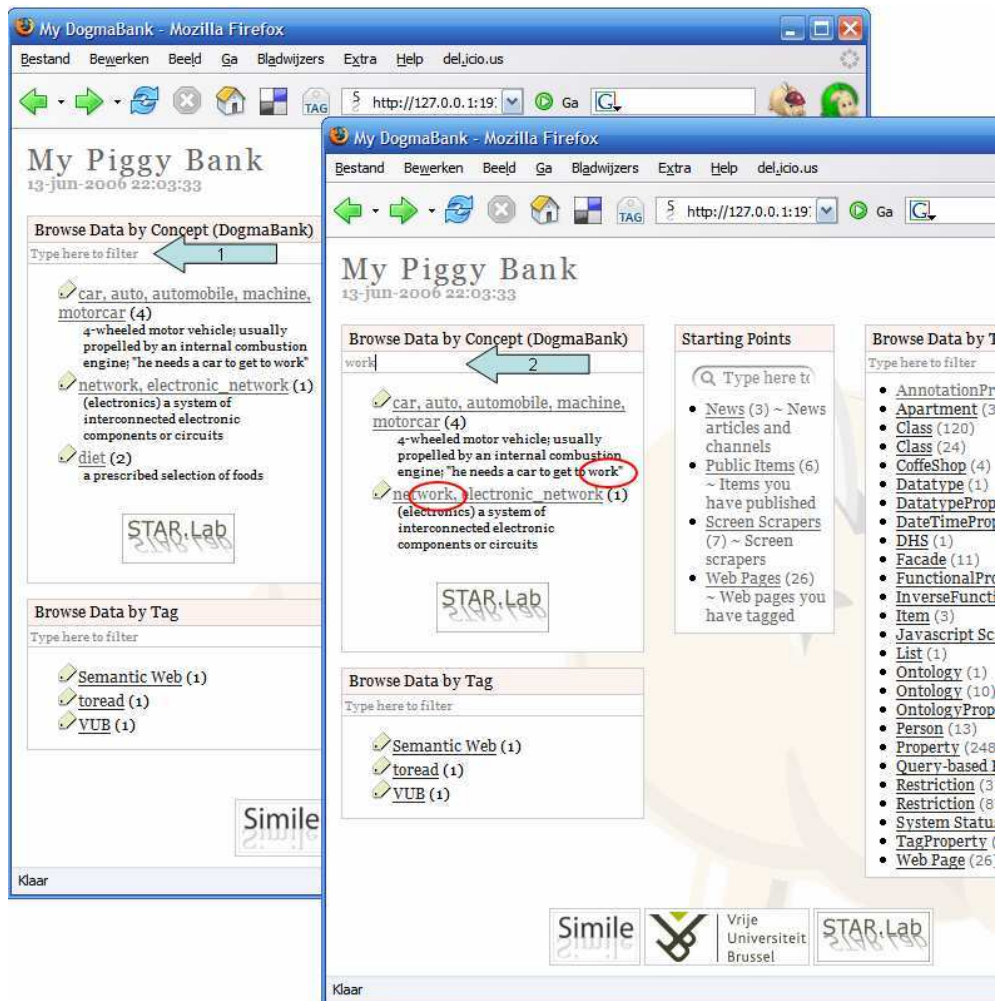


Figure 5.3: DogmaBank screenshot of the browsing functionality

The user is presented with a full list of all the concepts he ever used to tag a web page. He can filter this list by typing in the “Type here to filter” input box (1). For example, when he types “work” (2) only the concept “car” en “network” are shown, because the word “work” occurs in the gloss of “car” and in the definition of “network”. The filtering happens at each key stroke.

When a user clicks on a concept, he gets the list of web pages that are tagged with that concept (see figure 5.4).

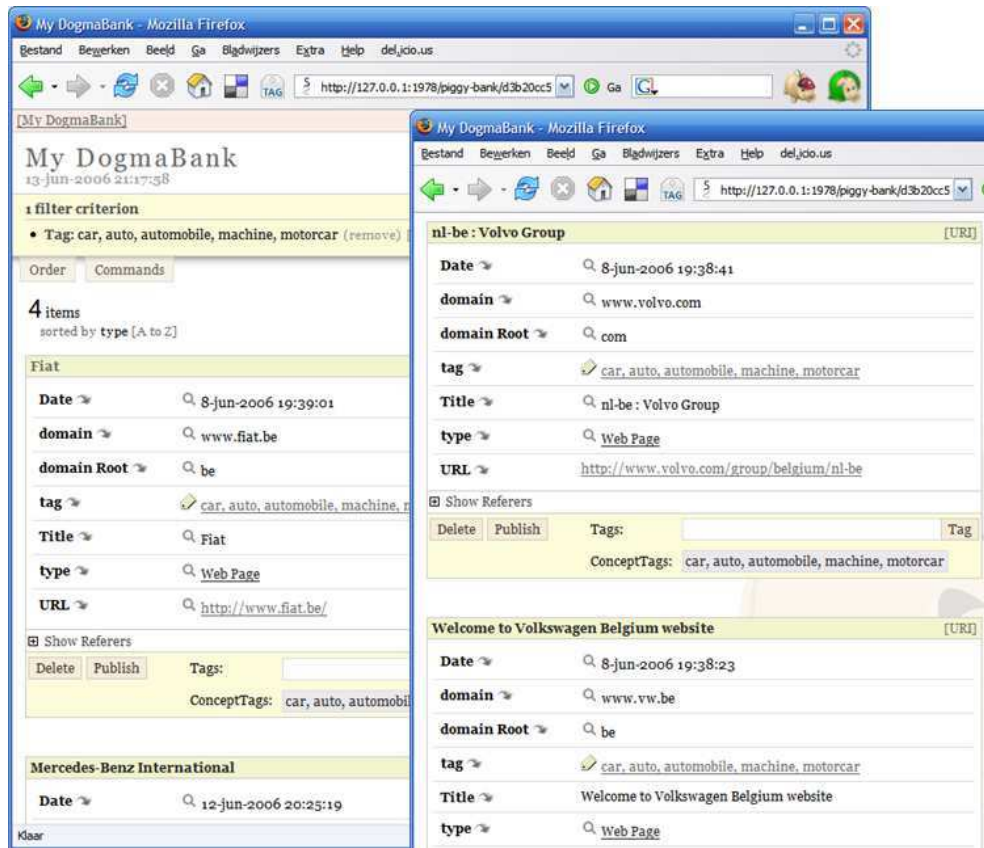


Figure 5.4: DogmaBank screenshot: browse web pages tagged with the “car, auto-mobile” concept

5.4.3 Suggest a concept

When a user wants to tag a page with a concept that is not yet present in the Concept Definition Server, then he can suggest the concept by clicking on the “Suggest Concept” tab and filling in the form (see figure 5.5).

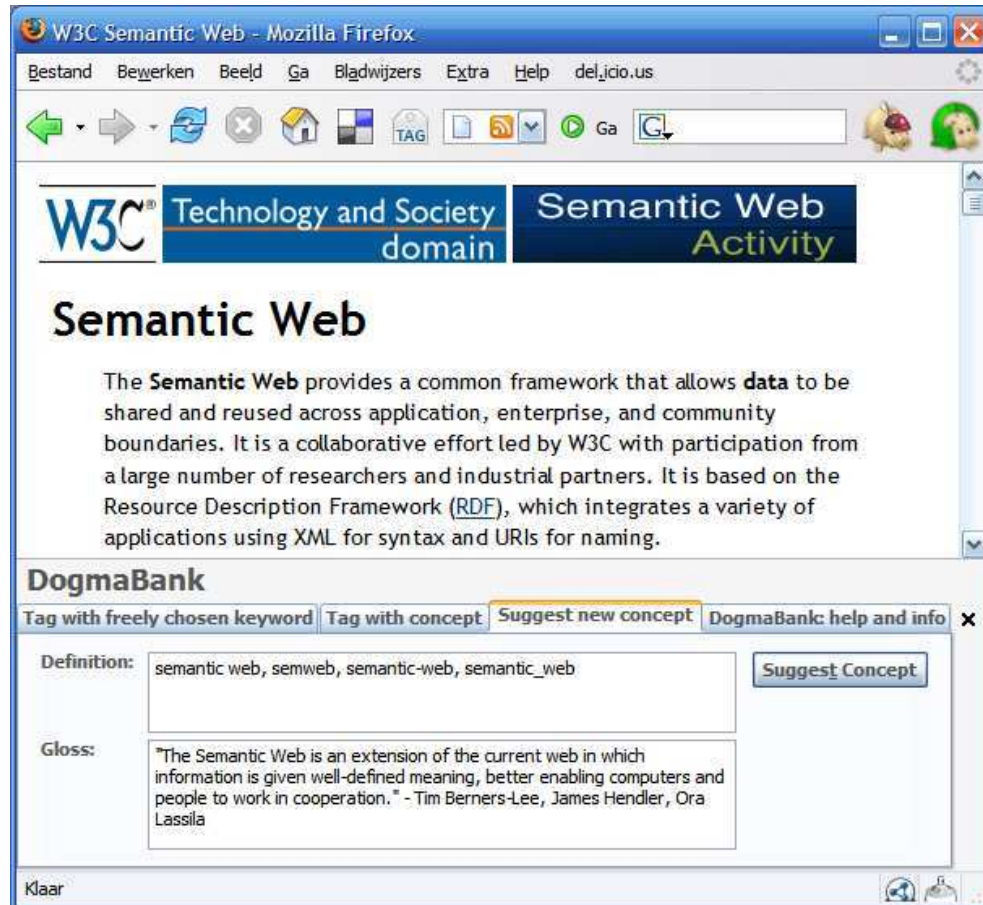


Figure 5.5: DogmaBank screenshot: Suggest Concept

The suggested concepts are logged on a remote web server at STAR.Lab. These can be approved or declined by an ontology engineer. When they are approved, then they are added to the Concept Definition Server.

Note that our DogmaBank tool does not impose any methodology about when to suggest and approve a new concept, but assumes there is one. Note that we

describe a possible collaborative ontology editing extension to DogmaBank in our Future Work section 6.3.4.

5.5 Architecture

5.5.1 Used technologies

Piggy Bank combines a lot of different technologies. Instead of explaining each and every one of them as we first encounter them, we group the explanations in this section.

XUL⁴ The XML User Interface Language (XUL) is a markup language for describing user interfaces. A feature in XUL, called an overlay, allows us to create extensions to the Firefox browser itself, for example to add a custom toolbar, change menus, or add other features.

Chrome⁵ consists of the user interface parts of the application window that are outside of the Firefox browser window's content area. Toolbars, menu bars, progress bars, and window title bars are all examples of elements that are typically part of the chrome. The Piggy Bank chrome consists of three parts:

- content: where Piggy Bank stores the XUL and Javascript files.
- locale: where Piggy Bank stores the DTD (Document Type Description) localization files. Only English localization strings are provided.
- skin: where Piggy Bank stores the CSS (Cascading Style Sheets) and image files. They describe the visual appearance of the chrome.

XPCOM⁶ stands for Cross Platform Component Object Model. It is a framework for writing cross-platform, modular software. It can be compared to Microsoft COM. In Piggy Bank we use an XPCOM component to set up the LiveConnect Java to Javascript bridge.

⁴<http://xulplanet.com/>

⁵<http://www.mozilla.org/xpfe/ConfigChromeSpec.html>

⁶<http://www-128.ibm.com/developerworks/webservices/library/co-xpcom.html>

Javascript⁷ is a purely interpreted object scripting language originally developed by Netscape. It has no direct relation to Java, so a special XPCOM component is needed to make the Java and Javascript code communicate.

Jetty⁸ is a Java HTTP Server and Servlet Container. Jetty is a fully featured web server for static and dynamic content. It is simply included in Piggy Bank as a Java component, avoiding the need to run a separate web server (like Apache).

Ajax⁹ is shorthand for Asynchronous JavaScript and XML. It is a Web development technique for creating interactive web applications. It is intended to make web pages feel more responsive by exchanging small amounts of data with the server behind the scenes, so that the entire web page does not have to be reloaded each time the user makes a change. Piggy Bank uses it for the tag completion suggestion functionality.

5.5.2 DogmaBank architecture

Piggy Bank is open source. Not a lot of developer documentation is available about the inner workings of the system. Only one high-level diagram has been published¹⁰. We adapted this diagram by including the parts we added to Piggy Bank and removing the parts DogmaBank does not use: see figure 5.6. We will only discuss the parts that concern our tagging system. The technologies mentioned in this section are explained in the previous section (5.5.1).

The system consists of three main parts:

- As any Firefox extension, it contains chrome additions in XUL and Javascript, including the TagBar mentioned in the user interface section (5.4) above.
- The back-end is entirely written in Java. It stores the tags in a database and supplies a HTTP interface to browse the tags.
- The Javascript code in the chrome and the Java code in the back-end communicate through an XPCOM component written in Javascript.

⁷<http://www.mozilla.org/js/>

⁸<http://jetty.mortbay.org/jetty/>

⁹<http://en.wikipedia.org/wiki/AJAX>

¹⁰<http://simile.mit.edu/piggy-bank/architecture.html>

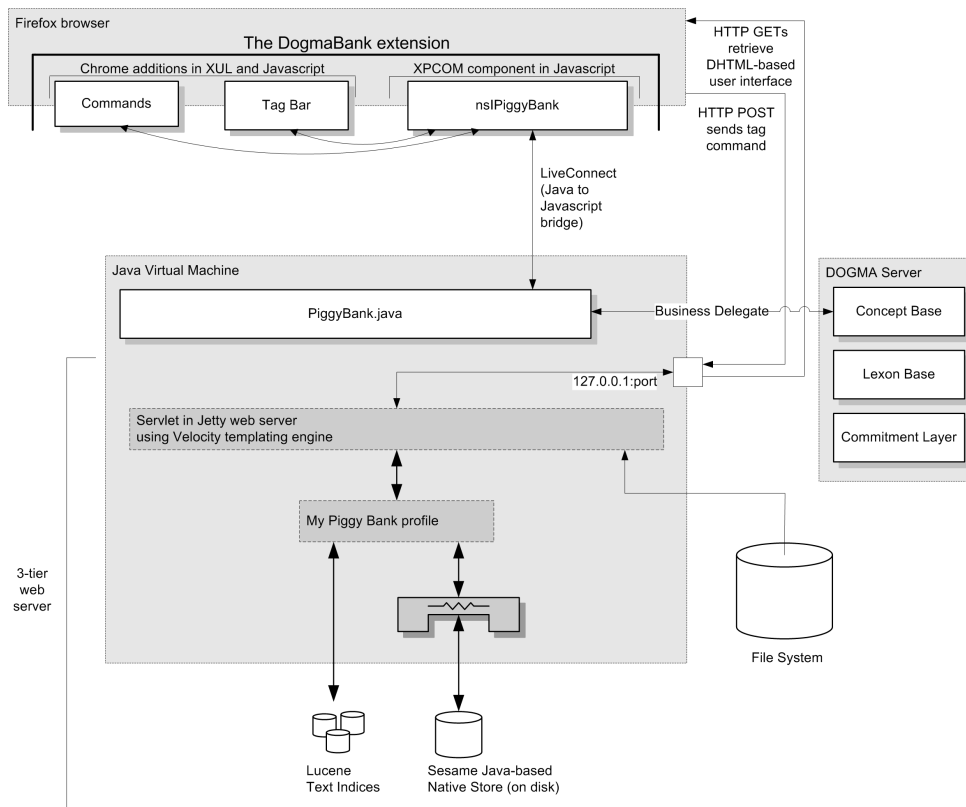


Figure 5.6: DogmaBank screenshot of the browsing functionality

The back-end can be considered a 3-tier web application, as it contains a Jetty web server that uses the Velocity templating engine to serve information in databases through a DHTML+Javascript-based user interface. The web server is completely stateless, which means all the data is encoded in the URL (GET) or in POST variables.

The “Tag with concept” command is sent as an HTTP POST to the embedded web server. The Ajax technique is used to retrieve the concepts while tagging. This means the TagBar is not completely reloaded when searching for concepts. Only the listbox is updated. The Javascript code is executed by the local web browser and calls some Java code in the DogmaBank back-end. This Java code contains a Business Delegate object that communicates with the DOGMA server. The results of each query are propagated back to the Javascript code that updates the listbox.

5.6 UML diagrams

Up until now in this chapter we have explained the high level design decisions and characteristics of DogmaBank, described the user interface and discussed the general technical architecture. We keep drilling down by illustrating the general flow of information with some UML diagrams in this section 5.6 before we comment on the raw code listings in the next section 5.7.

5.6.1 Tag with freely chosen keyword

The Piggy Bank code contains very little useful comments and almost no documentation. To be able to make sense of the code, we had to draw a few UML diagrams ourselves (as none are provided).

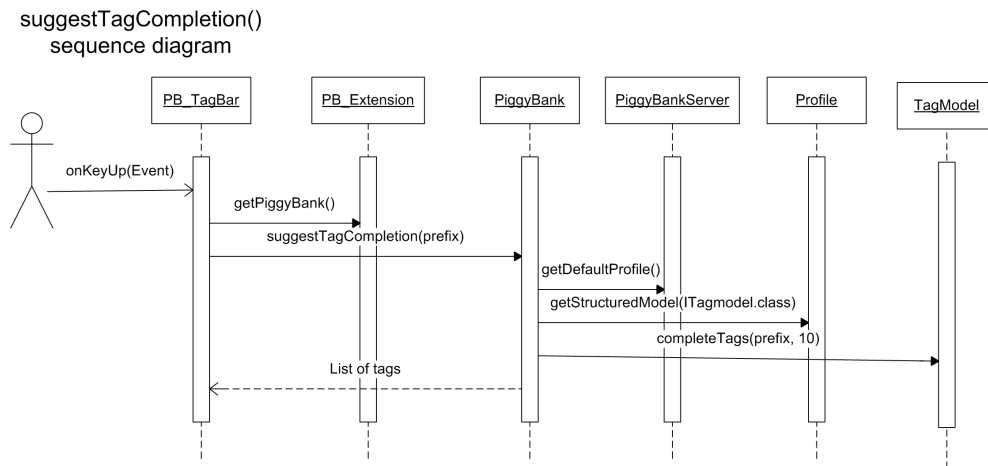
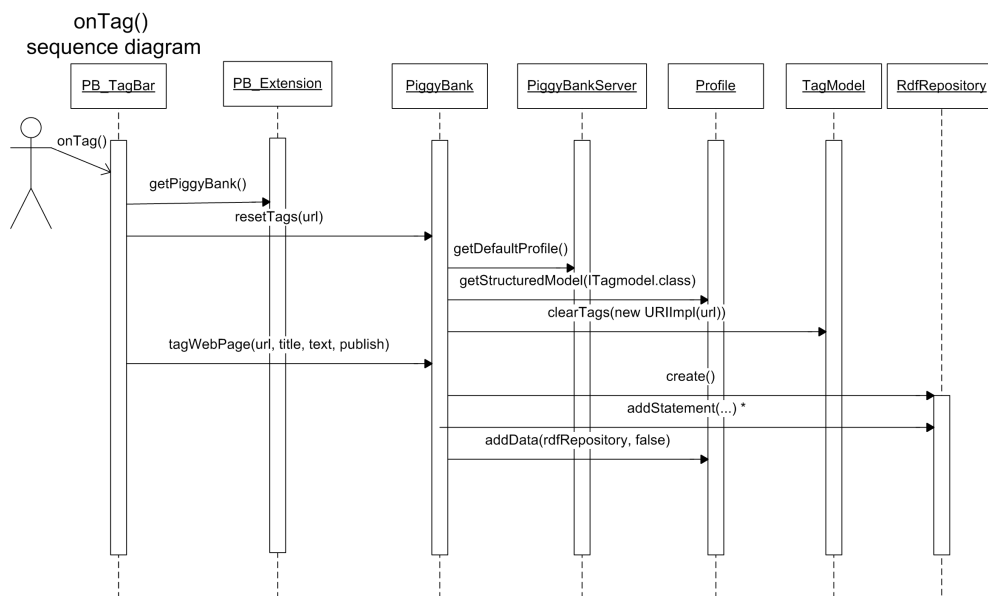
suggestTagCompletion()

As described in the requirements, Piggy Bank provides a user with suggestions of previously used tags as-he-types. The previously used tags are looked up in the database on each `onKeyUp()` event (see figure 5.7 on page 97). The capturing of this event is defined in the `master-overlay.xul` file.

```
<textbox id="piggyBank-tagToolbar-textbox"
         onkeyup="PB_TagBar.onKeyUp(event);" />
```

This event causes the `TagBar` to ask the extension object for the singleton instance of the Piggy Bank web server. The `TagBar` asks the Piggy Bank web server for a list of suggested tags. The web server gets the list of already used tags from the `TagModel` and checks if any of them match the first few characters the user has typed in.

Note that the sequence diagrams make no mention of what is a Java object and what is a Javascript object, as objects are passed between the two languages through the XPCOM component, as can be seen in the architecture overview figure on page 95.

Figure 5.7: Piggy Bank UML sequence diagram: `suggestTagCompletion()`Figure 5.8: Piggy Bank UML sequence diagram: `onTag()`

onTag()

We discuss the sequence diagram in figure 5.8 on page 97.

When a user clicks on the “Tag” button (as described in the requirements section), again the TagBar object asks the extension object for a reference to the Piggy Bank web server singleton. Then, the TagBar asks the web server to reset (clear) the tags for the current web page. Subsequently, the TagBar object asks the Piggy Bank web server to tag the current web page with the given keywords. The web server creates an RDF repository object, adds the list of tags to that object and adds the RDF repository object to the database.

Class diagram

We looked up the classes that are used in the above sequence diagrams and composed a class diagram (see figure 5.9). The Piggy Bank web server is a specialization of the web server used in the MIT Longwell¹¹ project. A web server contains exactly one profile (the default profile). That profile contains a set of structured models, among which the TagModel.

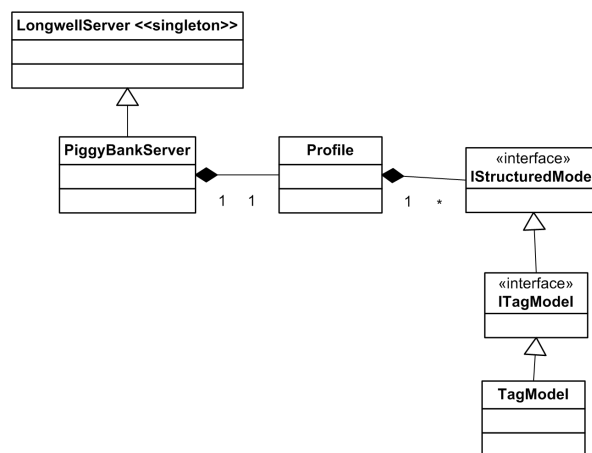


Figure 5.9: Piggy Bank UML class diagram

¹¹<http://simile.mit.edu/longwell/>

5.6.2 Tag with Concept

onGetConcepts()

The DogmaBank tagging system uses concepts to tag web pages. These concepts are stored in the Concept Definition Server. As described in section 5.4, a user types in a term and hits the “Get Concepts” button. What happens next can be seen in figure 5.10. A connection has to be made from within the DogmaBank Firefox extension to the DOGMA server using a Java Business Delegate.

Two methods are invoked to retrieve the relevant concepts from the Concept Definition Server: `retrieveAssociatedConcepts()` and `retrieveConceptsByDefinition()`. The latter searches the beginning of the definition of every concept. The former looks for concepts that are linked to the given natural language word. These links between the linguistic level and the conceptual level can be defined by ontology engineers using DOGMA Studio.

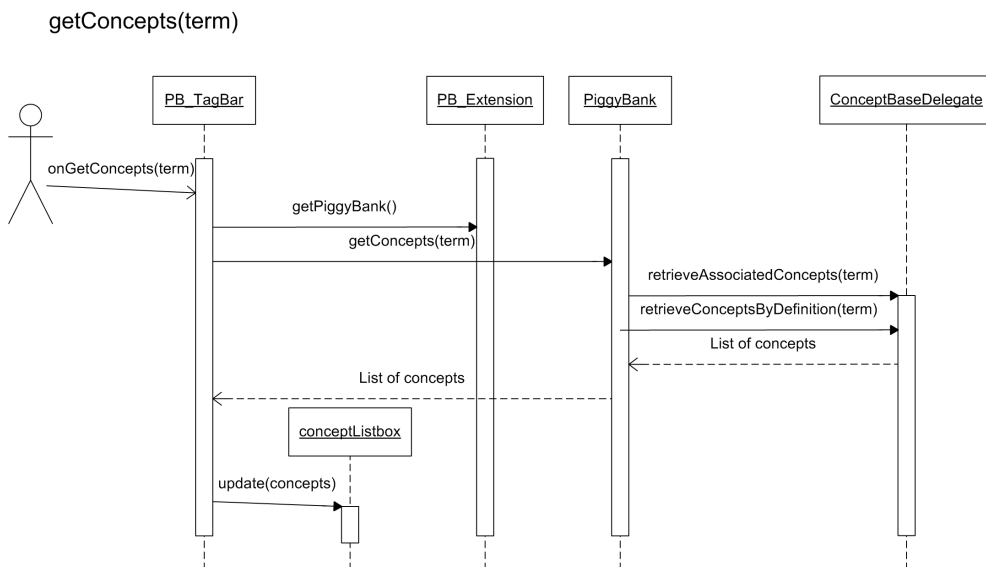


Figure 5.10: UML Sequence diagram of the `onGetConcepts()` method

onConceptTag()

When a user has selected an appropriate concept out of the list of concepts he clicks on the “Tag with Concept” button. The UML sequence diagram in figure 5.11 looks similar to the sequence diagram for tagging a web page with a freely chosen keyword, except that now the definition and the gloss of the concept is stored by the TagModel class in a file on the local hard drive. The definitions and the glosses of the concepts that have been used to tag a web page are cached locally in order to avoid unnecessary queries to the DOGMA server.

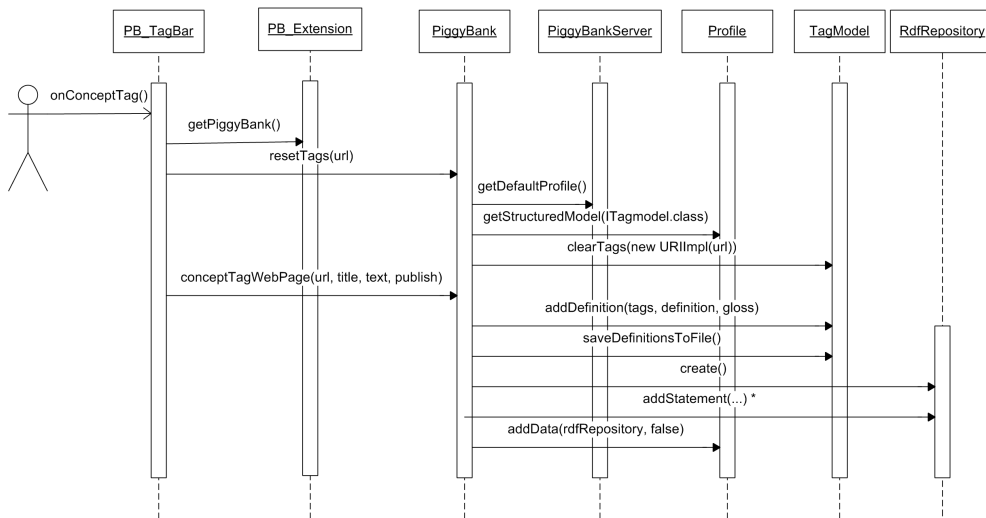


Figure 5.11: UML Sequence diagram of the onConceptTag() method

5.7 Code

This section is meant for developers that want to build upon the DogmaBank source code. Because there is almost no developer documentation and no code comments, we had to read almost all of the Piggy Bank code in order to understand which were the architecturally most clean places to make the necessary adjustments and additions for our DogmaBank system. This section should give future developers a head start.

5.7.1 The tag toolbar

As any Firefox extension all user interface elements and scripts are stored in the chrome. In fact, a Firefox extension is little more than a zipped chrome directory with a .xpi file name extension. The chrome contains the user interface additions to the browser interface: menu items, buttons, windows, ...

The chrome directory contains three subdirectories:

```
chrome
|__ content
|__ locale
|__ skin
```

content

The content directory contains the XUL files and the Javascript files. We will explain the Javascript files in section 5.7.2.

Below we list only a small part of the XUL code that is used for the user interface. The listed code corresponds with what can be seen in figure 5.2. The XUL code is formatted in XML and stored in the master-overlay.xul file. The tabpanels, labels and listboxes are simply nested XML tags, which makes the code easy to read. XUL uses a boxed layout scheme (as may be known from the Java BorderLayout) combining horizontal and vertical boxes which may or may not be flexible.

The Javascript methods that are called on certain events are specified in special attributes of the user interface element.

```
chrome/content/master-overlay.xul
```

```
<!-- TAG WITH CONCEPT -->
<tabpanel flex="1">
  <hbox flex="1">
    <vbox flex="1">
      <label value="&piggyBank.tagToolbar.conceptLabel;" />
      <textbox id="piggyBank-tagToolbar-conceptTextbox"
        tabindex="1"
        onkeyup="PB_TagBar.onConceptKeyUp(event);"
      />
      <listbox id="piggyBank-tagToolbar-conceptListbox"
```

```

        tabindex="2"
        onselect="PB_TagBar.onSelectConceptListbox(event);"
        rows="5"
        style="background-color: #eeeeee"
        flex="1"
    />
</vbox>
<vbox>
    <button label="&piggyBank.tagToolbar.GetConcepts;"
        tabindex="4"
        accesskey="g"
        oncommand="PB_TagBar.onGetConcepts();"
    />
</vbox>
<grid flex="1">
    <columns>
        <column />
        <column flex="1" />
    </columns>
    <rows flex="1">
        <row>
            <label value="&piggyBank.tagToolbar.definition;" />
            <textbox id="piggyBank-tagToolbar-definitionTextbox"
                tabindex="5"
                multiline="true"
                readonly="true" flex="1" />
        </row>
        <row flex="1">
            <label value="&piggyBank.tagToolbar.gloss;" />
            <textbox id="piggyBank-tagToolbar-glossTextbox"
                tabindex="6"
                multiline="true"
                readonly="true"
                flex="1" />
        </row>
    </rows>
</grid>
<vbox>
    <button label="&piggyBank.tagToolbar.ConceptTag;"
        tabindex="3"
        oncommand="PB_TagBar.onConceptTag();"
        onblur="PB_TagBar.onBlur();"
        onfocus="PB_TagBar.onFocus();"
        accesskey="t"

```

```

        />
    </vbox>
</hbox>
</tabpanel>

```

locale

The translations for the strings that are used for the labels and buttons in the user interface (see XUL code above) are stored in DTD files. Below we list part of the en-US browser.dtd file which corresponds with the above XUL code.

```
chrome/locale/en-US/browser.dtd
```

```

<!ENTITY piggyBank.tagToolbar.conceptGroupboxLabel "Tag with concept">
<!ENTITY piggyBank.tagToolbar.tabLabelConcept      "Tag with concept">
<!ENTITY piggyBank.tagToolbar.GetConcepts          "Get concepts">
<!ENTITY piggyBank.tagToolbar.ConceptTag           "Tag with Concept">
<!ENTITY piggyBank.tagToolbar.TagAndPublish        "Tag and Publish">
<!ENTITY piggyBank.tagToolbar.definition           "Definition:">
<!ENTITY piggyBank.tagToolbar.gloss                "Gloss:">

```

skin

The look of the DogmaBank Firefox extension is separated into CSS files. Below we list part of the classic browser.css file which corresponds with the XUL and DTD code above.

```
chrome/skin/classic/browser.css
```

```

/* ::::: tagging toolbar ::::: */
#piggyBank-tagToolbar-closebutton {
    list-style-image: url("chrome://global/skin/icons/Close.gif");
}

#piggyBank-tagToolbar {
    background-image: url("chrome://global/skin/icons/white-gray-gradient.gif");
    background-repeat: repeat-x;
    background-position: bottom right;
    background-color: rgb(246, 246, 246);
    border-top: 1px solid #b3b3b3;
    border-bottom: none;
}

```

5.7.2 The tagging code

The Javascript files

As stated in section 5.7.1, the Javascript files are stored in the content subdirectory of the Chrome. The methods that are called for each event are specified in the XUL files in the attributes of the user interface elements. Let us follow the code path where a user clicks the “Tag with Concept” button. As we can see in the XUL listings above, the “PB.TagBar.onConceptTag()” method is called, which closes the TagBar and in turn calls the private `_conceptTag()` method.

```
chrome/content/scripts/tag-bar.js
```

```
PB_TagBar.onConceptTag = function() {  
    PB_TagBar.close();  
    PB_TagBar._conceptTag(false);  
}
```

The `_conceptTag()` Javascript method gets all the needed information from the user interface elements to tag a web page and passes them as arguments to the `conceptTagWebPage()` Java method.

```
chrome/content/scripts/tag-bar.js
```

```
PB_TagBar._conceptTag = function(publish) {  
    var textbox = PB_TagBar._getConceptTextbox();  
    var listbox = PB_TagBar._getConceptListbox();  
    var definitionTextbox = PB_TagBar._getDefinitionTextbox();  
    var glossTextbox = PB_TagBar._getGlossTextbox();  
    var text = "::CONCEPT::" +  
        listbox.selectedItem.value;  
    var browser = document.getElementById("content");  
    var url = browser.contentDocument.location.href;  
    var title = browser.contentDocument.title;  
  
    PB_Extension.getPiggyBank().resetTags(url);  
    var s = PB_Extension.getPiggyBank().conceptTagWebPage(url,  
        title, text, publish, definitionTextbox.value, glossTextbox.value);  
}
```

The Java files

The `conceptTagWebPage()` method is stored in the `PiggyBank.java` file. We included the complete listing of this file in Appendix C on page 125.

Calling a Java method from within a Javascript method is made possible by the `ClassLoader` that comes with Piggy Bank. Note that this `ClassLoader` relies on core Java methods that have a bug in all current Windows implementations of the Java Virtual Machine. Tracking down this bug took up a serious part of our valuable development time. Luckily, we have found a work-around, otherwise DogmaBank would have been available for Linux and Mac OS only. This work-around is documented in the comments of the `DogmaProxy.java` file, of which we included the complete listings in Appendix D on page 133.

To make a connection to the DOGMA server we use a Business Delegate object. The connection to the DOGMA server is initialized when the Firefox extension is loaded. This way the connection can be reused, which, after testing, turned out to be much more efficient than re-establishing the connection for each query.

5.7.3 The browsing interface

When a web page is tagged, it is saved on the local disk drive in the RDF store. As stated in section 5.4.2 on page 90, the web pages can be browsed by clicking on the little piggy button in the button toolbar in the top right corner. These pages are actually generated by a Jetty web server that runs on the local machine (`localhost`). The Jetty web server runs completely stateless, which means no session data is stored and all the information is encoded into the URL.

The Jetty web server is in fact part of the Longwell project, which is a part of the Piggy Bank project. Longwell is a Semantic Web browser for browsing RDF stores. We had to adapt parts of Longwell and recompile the entire project into Piggy Bank. The fact that small changes have to be made in different places and using different technologies adds to the complexity of the DogmaBank project. One of the adapted files is `TagModel.java`, which we included fully in Appendix E on page 135. As we added a few new methods, we also had to adapt the interface for the `TagModel` class stored in the file `ITagModel.java`.

We used the Velocity templating engine for generating the dynamic pages. We

added the code below to the tag-start.vt template file. The code generates the top left part of the user interface as can be seen in figure 5.3 on page 90.

```
ui-modules/html/tag/server-side/templates/commands/tag-start.vt
```

```
#set($tagModel = $profile.getStructuredModel("edu.mit.simile.longwell.tag.ITagModel"))
$tagModel.loadDefinitionsFromFile()
#set($tagLabels = $utilities.sort($tagModel.getTagLabels()))
#set($conceptTagLabels = $utilities.sort($tagModel.getConceptTagLabels()))
#if ($tagLabels.size() > 0)
<div id="lw_concepttags" class="lw_box"
  style="width: 20em">
  <div class="lw_title">Browse Data by Concept (DogmaBank)</div>
  <div class="lw_filter"><input
    onkeyup="filterConceptTags(event, 'lw_concepttags_body')"
    onfocus="if(this.value=='Type here to filter')this.value=''"
    onblur="if(this.value=='')this.value='Type here to filter'"
    type="text"
    value="Type here to filter"/></div>
  <div class="lw_body" id="lw_concepttags_body">
  #foreach($conceptTagLabel in $conceptTagLabels)
    #if ($tagModel.countObjects($conceptTagLabel) > 0)
      #set($conceptTagUrl = $url.changeCommand("browse"))
      #set($param = "1")
      #set($param = $param.concat($conceptTagLabel))
      $conceptTagUrl.getQuery().
        addRestriction("edu.mit.simile.longwell.query.project.TagProjector",
          "", "edu.mit.simile.longwell.query.bucket.DistinctValueBucketter", $param)
      <div class="lw_item" class="break">
        
        <a href="$conceptTagUrl.toURLString()">
          $tagModel.getDefinition($conceptTagLabel)
        </a> ($tagModel.countObjects($conceptTagLabel))
        <br />
        <div class="gloss">
          $tagModel.getGloss($conceptTagLabel)
        </div>
      </div>
    #end
  #end
</div>
<p align="center">
  
</p>
</div>
```

For the “filter while you type” functionality we use some Javascript code that updates the DOM dynamically without reloading the entire page, as can be seen in the code included below.

```
ui-modules/html/tag/client-side/scripts/commands/tag-start.js
```

```
function filterConceptTags(e, valueListID) {
    var inputElement = getTarget(e);
    var list = document.getElementById(valueListID);
    var text = inputElement.value;
    var regexp = new RegExp(text, "ig");
    var values = list.getElementsByTagName("div");
    for (var i = 0; i < values.length; i++) {
        var value = values[i];

        if (text.length > 0) {
            if (value.getAttribute("class") == "lw_item") {
                try {
                    var tagLabel = value.firstChild.nextSibling.firstChild.nodeValue;
                    var divs = value.getElementsByTagName("div");
                    var div = divs[0];
                    var gloss = div.firstChild.nodeValue;
                    if (tagLabel.match(regexp)
                        || gloss.match(regexp)
                    ) {
                        if (value.style.display != "block") {
                            value.style.display = "block";
                        }
                    } else {
                        if (value.style.display != "none") {
                            value.style.display = "none";
                        }
                    }
                } catch (e) {
                    if (value.style.display != "none") {
                        value.style.display = "none";
                    }
                }
            } else {
                value.style.display = "block";
            }
        }
    }
}
```

5.8 Testing

In this section we talk about three kinds of tests. We have run technical tests to see if our Firefox extension functions properly under different operating systems and Java Virtual Machine versions as reported in section 5.8.1. Further, we discuss the qualitative analysis and peer review process we went through in section 5.8.2. Finally, we discuss what we learned from the user testing in section 5.8.3.

5.8.1 Technical tests

We tested our DogmaBank Firefox extension in a number of different environments. We tested the extension on several Linux distributions, on two G4 Mac OS X machines and on several Windows XP installations.

We also tested the extension in different networking environments, behind firewalls, (wireless) routers, and from within and outside STAR.Lab and the Vrije Universiteit Brussel.

We always used the latest Firefox version 1.5.0.4, because we experienced some trouble with Piggy Bank and the second Release Candidate of Firefox 1.5. We filed a bug report, which was followed up by the MIT Piggy Bank developers team.

We developed DogmaBank on the latest Java Virtual Machine version 1.5.0_06, but also tested DogmaBank with 1.4 versions. We include an overview of the technical tests in table 5.1.

Java Virtual Machine version	1.4.2	1.5.0_06
Linux Debian	X	
Mac OS X	X	
Windows XP	X	X

Table 5.1: Overview of technical tests, all run on Mozilla Firefox version 1.5.0.4.

These tests revealed the stubborn ClassLoader bug we talked about in section 5.7.2 on page 105. This bug applies only to Windows implementations of the Java Virtual Machine beginning with version 1.4. The bug still has to be fixed by Sun Microsystems, but we managed to dig up a working work-around.

5.8.2 Qualitative analysis and peer review

ODBASE 2006 paper

We submitted a paper about DogmaBank to the ODBASE¹² 2006 conference. The paper was accepted. We included a draft in appendix A, as the final version of the paper is not yet finished at the time of writing.

Below, we include the full review of one of the reviewers and include some quotes of the other reviewers.

“Summary: This is an “idea” paper that presents a solution to the problem of bridging informal and formal ontological specifications in the real world. This is an important problem and a useful result.

Details: I like this paper very much. It is one of the rare “idea” papers that manages to bring out a focused, novel concept and then show an application of it.” (Reviewer 1)

“A number of people have proposed hybrids like this, but it’s a clean example and well articulated.” (Reviewer 2)

“The paper addresses a very interesting topic.” (Reviewer 3)

“The authors have presented an interesting characterization of emergent semantic systems and present a prototype system implementation.” (Reviewer 4)

BruDISC

We presented our system at a BruDISC meeting of researchers to prepare for the Call for Proposals in ICT research by the Brussels-Capital Region. DISC is a coordination and support centre for ICT research financed by the Institute for the encouragement of Scientific Research and Innovation of Brussels. Partners in this project proposal are (among others) three research groups of the Vrije Universiteit Brussel, namely MOSI¹³ with their KnoSoS project, STAR.Lab and WISE, the OSC¹⁴ with their PointCarré project, the Erasmus Hogeschool, the Mediatheek Geel with a

¹²Ontologies, DataBases, and Applications of Semantics

¹³Mathematics, Operational research, Statistics and Information systems

¹⁴Onderwijs Service Centrum at the Vrije Universiteit Brussel

project to classify theses, the center for PhD students of Antwerp and also the one of the Vrije Universiteit Brussel. Our DogmaBank idea was received in a positive way by these partners and may end up being part of the final project proposal.

5.8.3 User testing

At the BruDISC meeting, some people approached us with the question to download the DogmaBank Firefox extension and play with it. For security reasons, we could not set up a public web site with a download link to the extension, but we did send the extension to a few trusted researchers who are already familiar with tagging.

From the user feedback we received from Céline Van Damme of the MOSI research group, we learned that DogmaBank is user friendly and practical to use, but we still need to implement two missing functionalities that are technically related: tagging one web page with several concepts, and combining tagging with freely chosen keywords with tagging with concepts. Céline posted an internal blog entry to the MOSI blogging system about DogmaBank, but reactions from the other MOSI members are still pending at the time of writing.

Frederik Questier of the OSC wrote us an encouraging e-mail after having read a draft of our thesis text and having tried our DogmaBank tool. We include his reaction here in full:

“The work presented is in a very interesting research field.

The DogmaBank tool is a very useful and user friendly tool for socially bookmarking sites with concepts from an ontology.

The use of concepts instead of tags for (social) bookmarking seems indeed superior.

I might have missed something, but a way to make tagging in DogmaBank less effort would be the possibility to suggest to users the tags or concepts that other users assigned with DogmaBank or even other popular bookmarking sites.

From professional experience with the development and administration of e-learning platforms I can testify that getting author (teacher) supplied meta data for learning objects is difficult. Additional user tags

from consumers (students while they consult learning objects or teachers while they reuse learning objects) would be easier.

In Archemedes, an e-learning knowledge management system for architect students, we experimented already with simple ontologies, author tagging and tagging creation by domain experts. The here presented idea of seeing these tags as concepts with a definition would bring it to a higher level. I see use for combining these concept definitions with the glossary functionality which is already available in the Archemedes System, and which is generally useful in e-learning systems. I've got the impression that the defining of the glossary terms in the Archemedes projects happened less than planned. The reuse of these definitions for the concepts could possibly be an extra motivation for defining them.

We planned already to implement all functionality from our Archemedes project into our Open Source Dokeos / PointCarré e-learning platform, including a similar meta data engine. It would be useful to have the DogmaBank ideas implemented as well.

It would be useful to see how to get scaling from the DogmaBank concept tool to an e-learning system which is used by 10.000 users in our university and by circa 1000 institutions worldwide. Especially scaling from one domain to diverse overlapping domains is a matter which still deserves further investigation." (Frederik Questier)

Frederik is preparing a blog post to the OSC blog¹⁵ based on the above e-mail. His blog post will undoubtedly generate more user feedback from the e-learning community.

5.9 Conclusions

In this section we have presented the DogmaBank system we implemented for our thesis. We looked at DogmaBank from different angles. We started with the high level design decisions and characteristics, described the user interface of the system

¹⁵<http://osc.vub.ac.be/blog/>

and zoomed in to the architecture, UML diagrams , and even the code level. We ended with a discussion on the testing of the system.

We believe the system does what it promises to do and inspires to go even further. DogmaBank is just the start, a proof of concept that clearly shows how the idea of tagging with concepts can be implemented in a real working system. Seeing and using such a system allows researchers to fine tune their ideas on the subject. During our literature study we have encountered many theoretical observations about what a tagging system on the conceptual level *could* look like, but we never saw any actual implementation. We have provided an implementation in order to focus the discussion and point out possible further extensions.