UNIVERSITAT POLITÈCNICA DE VALÈNCIA

FINAL PROJECT

Community Based Mappings for the Semantic Web: MappingsTool

Author:
Omar Pera Mira
Computer Science Engineer
Escola Tècnica Superior d'Enginyeria Informàtica

Supervisor:
Antonio Molina
DSIC

March, 2011

Abstract

An extension of BioPortal, an open source ontology repository developed by the UNIVERSITY OF STANFORD, that facilitates the manipulation of mappings between ontologies. We provide a flexible web user interface that facilitate the workflow to create a mapping and the exploration of the relations between ontologies.

Contents

Chapter 1

Introduction

1.1 Presentation

The aim of this document is the report for the Final Project *Community Based Mappings for the Semantic Web: MappingsTool*, for the Escola Tècnica Superior d'Enginyeria Informàtica (ETSINF) of the Universitat Politècnica de València (UPV).

1.2 Overview

The application presented in this document is a mapping editor between concepts of different ontologies. In the context of the Semantic Web, the mappings between controlled vocabularies in a given domain are essential to interoperability between different data systems. In other words, it is important to have a machine-readable way for a computer to understand how terms are related (for example, if one term is equivalent to, more general than, or narrower than another term).

We have developed Mappingstool, an extension of Bioportal[1] that eases the manipulation of mappings across ontologies. We formalize the relationships between ontology concepts by providing a first approach of a specification for representing the mappings involved, and providing a flexible

[1] BioPortal is an open source ontology repository developed by the UNIVERSITY OF STANFORD, more information in the subsection 2.3

user interface that facilitates the generation of mappings and the exploration of the relations. Users can browse the mappings, create new ones, download them, or begin a discussion about the mapping itself.

Each mapping has its own set of metadata that describes who created the mapping and when, application context in which the mapping might be valid, mapping source, the specific mapping relationship, and other properties.

Mappings between ontologies are key to interoperability data in the vision of Semantic Web. They are a fundamental component to achieve the integration and reuse of information. Section 2 will provide some background information, and provide some pointers that show the connection points to our work.

This power users of this extension is a small group of domain experts, who are familiar with the controlled vocabularies. Doing accurate mapping requires an in-depth knowledge of the meaning of the terms in each vocabulary.

1.3 Objectives

- Develop a tool that facilitates the creation of mappings between ontologies encouraging collaboration between domain experts.
- Design a flexible web application on top of Bioportal in which you can see the context of a visual mapping, promoting collaboration and discussion among experts and facilitate the exploration of relations between ontologies.
- Encourage the use of mappings between controlled vocabularies to provide interoperability between different types of systems in a specific domain
- Formalize relationships between ontological vocabulary by creating a well known vocabulary that contains the key relationships as OWL[2]: $same\_as$, $owl : inverse\_of$, etc. In addition to give the possibility to add new relationship types.

[2] More detailed information in the Subsection 2.1.3

1.4 Motivation

Within the context of the project OASIS[3] there was a key requeriment to have an open repository for ontologies and to be able to reuse, create relations between them, collaborate, etc. There are several research and development initiatives at this time aiming at producing 'open ontology repositories'; the most stable and mature of these is the BioPortal platform developed for the Biomedical domain. This platform has been developed by the *National Center for Biomedical Ontology* of the UNIVERSITY OF STANFORD.

We have taken BioPortal and considered the precise extensions and mechanisms necessary for moving the BioPortal framework to the *Assistive Technologies* domain. The result is a new repository, ORATE[4], that is the first instantiation of BioPortal technology outside of the Biomedical domain. ORATE has been implemented by the University of Bremen in the ambit of *Assistive Technologies* (with ontologies such as Device, Sensor, Trip, GPS or Transportation) where the relations are key within the context of the project.

In typical OASIS scenarios, several services and devices have to be in interaction and hence the corresponding software must be highly interoperable. As the software is driven by ontologies, these ontologies must be interoperable or connectable too: if two applications, based on ontologies, interoperate in a well specified way then their underlying ontologies must be related in a meaningful way.

Interoperability or connectivity in the case of ontologies means that two ontologies are connected via a relation with a well defined semantics. In many cases the purpose of such connections is to transport concepts and their implications from the source ontology to the target ontology. A simple example is the import relation where the target ontology imports all knowledge from a source ontology. This interoperability between ontologies has been acquired with Mappingstool, being able to easily formalize relationships between ontology concepts.

[3] Open architecture for Accessible Services, it is an European Project with the scope to revolutionise the interoperability, quality, breadth and usability of services for all daily activities of older people.

[4] Ontology Repository for Assistive Technologies

Chapter 2

Background

2.1 Semantic Web

The World Wide Web is an information resource with virtually unlimited potential. However, this potential is relatively untapped because it is difficult for machines to process and integrate this information meaningfully. The Semantic Web is based on the idea of adding semantic and ontological metadata to the World Wide Web. This extra information - that describes its content, meaning and their relationships - must be provided formally, so that can be evaluated automatically by processing machines.

Researchers have begun to explore the potential of associating web content with explicit meaning. Rather than rely on natural language processing to extract this meaning from existing documents, this approach requires authors to describe documents using a knowledge representation language.

The main obstacle is the fact that the Web was not designed to be processed by machines. Although, web pages include special information that tells a computer how to display a particular piece of text or where to go when a link is clicked, they do not provide any information that helps the machine to determine what the text means. Thus, to process a web page intelligently, a computer must understand the text, but natural language understanding is known to be an extremely difficult and unsolved problem.

With the Semantic Web, the software is able to process your content, reasoning with it, combining and logical deductions for solve everyday problems automatically. All this knowledge is based on ontologies, leading to an intelligence derived from the possibilities of inference implicit in RDF / OWL - languages used to generate ontologies.

At its core, the semantic web comprises a set of design principles, collaborative working groups, and a variety of enabling technologies. Some elements of the semantic web are expressed as prospective future possibilities that are yet to be implemented or realized. On the other hand, other elements of the semantic web are expressed in formal specifications. Some of these include Resource Description Framework (RDF), a variety of data interchange formats (e.g. RDF/XML, N3, Turtle, N-Triples), and notations such as RDF Schema (RDFS) and the Web Ontology Language (OWL), all of which are intended to provide a formal description of concepts, terms, and relationships within a given knowledge domain.

Tim Berners-Lee, inventor of the Web, has coined the term Semantic Web to describe this approach.

2.1.1 What is an ontology

This definition was originally proposed by Tom Gruber.

> In the context of knowledge sharing, I use the term ontology to mean a specification of a conceptualization. That is, an ontology is a description (like a formal specification of a program) of the concepts and relationships that can exist for an agent or a community of agents. This definition is consistent with the usage of ontology as set-of-concept-definitions, but more general. And it is certainly a different sense of the word than its use in philosophy.
>
> What is important is what an ontology is for. My colleagues and I have been designing ontologies for the purpose of enabling knowledge sharing and reuse. In that context, an ontology is a specification used for making ontological commitments. The formal definition of ontological commitment is given below. For pragmetic reasons, we choose to write an ontology as a set of definitions of formal vocabulary. Although this isn't the only way to specify a conceptualization, it has some nice properties for knowledge sharing among AI software (e.g., semantics independent of reader and context). Practically, an ontological commitment is an agreement to use a vocabulary (i.e., ask queries

and make assertions) in a way that is consistent (but not complete) with respect to the theory specified by an ontology. We build agents that commit to ontologies. We design ontologies so we can share knowledge with and among these agents.

2.1.2 Semantic components overview

To represent the Semantic Web, the main technologies are:

XML Provides an elemental syntax for content structure within documents, yet associates no semantics with the meaning of the content contained within.

URIs A global naming scheme

RDF A standard syntax for describing data. An RDF-based model can be represented in XML syntax.

RDF Schema A standard means of describing the properties of that data

OWL A standard means of describing relationships between data items. OWL adds more vocabulary for describing properties and classes: among others, relations between classes (e.g. disjointness), cardinality (e.g. ”exactly one”), equality, richer typing of properties, characteristics of properties (e.g. symmetry), and enumerated classes.

SPARQL It is a protocol and query language for semantic web data sources.

The intent is to enhance the usability and usefulness of the Web through:

- Structured documents with semantic information, this could be machine-understandable information about the human-understandable content of the document.

- Data systems systems using the RDF and SPARQL standards.

- Ontotologies and maps between vocabularies that allow document creators to know how to mark up their documents so that agents can use the information in the supplied metadata.

- Automated agents to perform tasks and infer information using the semantic data.

2.1.3 OWL: Web Ontology Library

The ontologies that are used in Mappingstool are of the type OWL, being able to add relations between classes.

OWL is a W3C specification built upon RDF and RDFS, that defines the types of relationships that can be expressed in RDF using an XML vocabulary to indicate the hierarchies and relationships between different resources. In other words, as we have explained in the previous section, ontologies define data models in terms of classes, subclasses, and properties. This ontologies are expressed with OWL.

Since taxonomies (systems of classification) express the hierarchical relationships that exist between resources, we can use OWL to assign properties to classes of resources and allow their subclasses to inherit the same properties. OWL also utilizes the XML Schema datatypes and supports class axioms such as *subClassOf*, *disjointWith*, etc., and class descriptions such as *unionOf*, *intersectionOf*, etc.

The basic components of OWL include:

- **Classes** are the basic building blocks of an OWL ontology. A class is a concept in a domain. Classes usually constitute a taxonomic hierarchy (a subclass-superclass hierarchy).
- **Properties** have two main categories:
- **Object properties**, which relate individuals to other individuals.
- **Datatype properties**, which relate individuals to datatype values, such as integers, floats, and strings.
- **Individuals** are instances of classes, and properties can relate one individual to another.

We have focus our Mappingstool in establishing relations between classes, not individuals. Bioportal was not supported to retrieve the individuals but they had it in the roadmap, so the main reason was not to reinvent the wheel.

2.1.4 Example: BBC World Cup 2010

In order to show a real example of the use of Semantic Web technologies and mappings, we are going to present what was developed by BBC site to publish information from the World Cup 2010.

The period of the World Cup event is a perfect example of dynamic data, with lots of relationships between Players, Teams, groups, etc. They created a framework that facilitated the publication of automated metadata-driven web pages that required minimal journalistic management, as they automatically aggregate and render links to relevant stories.

The underlying publishing framework does not author content directly, rather it publishes data about the content (metadata). The published metadata described the World Cup content, providing rich content relationships and semantic navigation. By querying this published metadata they were able to create dynamic page aggregations for teams, groups and players.

The origin of these dynamic aggregations is a rich set of ontologies. The ontologies described entity existence, groups and relationships between the concepts that describe the World Cup. For example, ″Frank Lampard″ is part of the ″England Squad″ and the ″England Squad″ competes in ″Group C″ of the ″FIFA World Cup 2010″.

This diagram gives a high-level overview of the main architectural components of this domain-driven framework.

The journalists could tag concepts to content, for example associated the content of ″Puyol″ with the story ″Shark Puyol beats Germany″.

In addition to the manual selective tagging and mapping process, journalist-authored content is automatically analysed against the World Cup ontology. A natural language and ontological determiner process automatically extracts World Cup concepts embedded within a textual representation of a story.

Journalist-published metadata is captured and made persistent for querying using the resource description framework (RDF) metadata representation and triple store technology. A RDF triplestore and SPARQL approach was chosen over and above traditional relational database technologies due to the requirements for interpretation of metadata with respect to an ontological domain model.

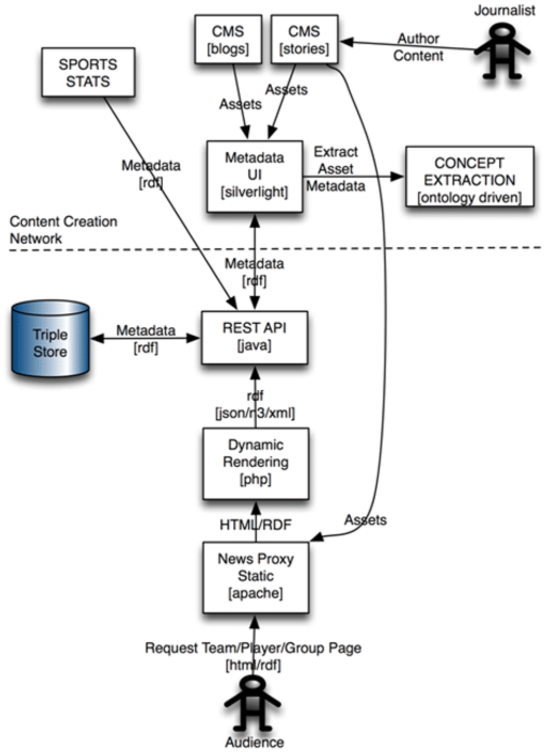


Figure 2.1: Architectural components of the BBC semantic web system

This is a perfect example about what could be the next phase of Internet, Semantic Web or Web 3.0.

2.2 Relationships between ontologies: mappings

The ability to specify semantic mappings between ontologies is an important research agenda in the semantic web community. Several approaches have been proposed for alignment between ontologies ranging from entirely manual (the one explained in the document), to semi-automatic, to fully-automatic mapping techniques.

More recently, with the growing number of ontologies and the increasing requirement for their alignment, community-based approaches to create mappings have been necessary to that allow users and domain experts to specify semantic correspondences in a collaborative manner.

In configurations for which different conceptualizations of the same domain can exist, information systems must respond effectively allowing the contextualization of it. Coordination and the relationship of concepts (mapping) help you share, reuse and integrate different controlled vocabularies.

Similarity

We start with a short definition of similarity from a dictionary: having characteristics in common, being comparable. From our point of view we want to compare two entities to find identity among them. We also give a formal definition of similarity derived from [3]:

- $sim(x,y) \in [0..1]$
- $sim(x,y) = 1 \Rightarrow x = y$: two objects are identical.
- $sim(x,y) = 0$: two objects are different and have no common characteristics.
- $sim(x,x) = 1$: similarity is reflexive.
- $sim(x,y) = sim(y,x)$: similarity is symmetric.5
- similarity and distance are inverse to each other.
- $sim(x,z) \leq (sim(x,y) + sim(y,z))$: The triangular inequation is valid for the similarity measure.

Similarity for Ontologies

As it is pointed out in [4], What is the meaning of similarity in the context of ontologies? The basic assumption is that knowledge is captured in an arbitrary ontology encoding. Based on the consistent semantics the coherences modelled within the ontology become understandable and interpretable. From this it is possible to derive additional knowledge such as, in our case,

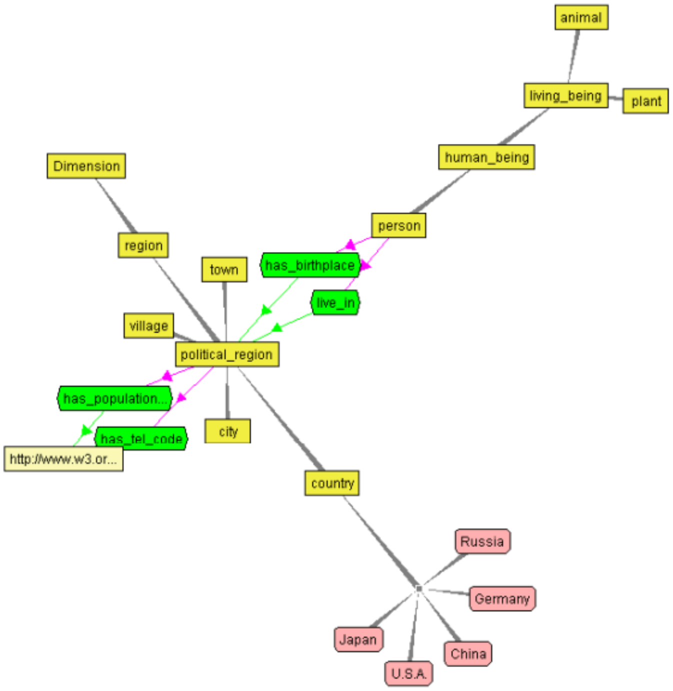


Figure 2.2: Relations between ontologies

similarity of entities in different ontologies. An example shall clarify how to get from encoded semantics to similarity: by understanding that labels describe entities in natural language one can derive that entities having the same labels are similar. This is not a rule which always holds true, but it is a strong indicator for similarity. Other constructs as subclass relations or type definition can be interpreted similarly.

Mapping

Due to the wide range of expressions used in this area (merging, alignment, integration etc.), we want to describe our understanding of the term mapping. The definition for mapping given by [9] is:

> Given two ontologies A and B, mapping one ontology with another means that for each concept (node) in ontology A, we try

> to find a corresponding concept (node), which has the same or similar semantics, in ontology B and viceverse.

We only consider one-to-one mappings between single entities in Mappingstool. Neither do we cover mappings of whole ontologies or sub-trees, nor complex mappings or functional transformation of attributes.

Example within OASIS context

We will illustrate the relations by the following example within the OASIS project context: Suppose there is an ontology describing the functionality of a mobile phone and another ontology describing an application to be plugged in. Then an ontology describing both the mobile phone and its plug-in is most likely given by importing the plug-in ontology into the mobile phone ontology. For instance, if the target ontology of the mobile phone has a notion of telephone numbers in the address book and the source ontology of the plug-in has in addition to this a notion of emergency numbers as a subclass of telephone numbers, then the target ontology will also gain a notion of emergency numbers when it imports the source ontology. An alternative way of describing this is to say that a plug-in is 'integrated' into a target system.

With interoperability we always associate some degree of independence between the participating components. Another relevant example: in particular for the elderly it may be annoying to have for each and every remote controllable device its own remote control with its own specific style of being handled. Different handling for the same purpose is an unnecessary irritation. A remote control that can be used to control several different devices instead may then be desirable. Typically such a remote control should have only few operating elements with a very intuitive operationâfor instance: a wheel that can be used to adjust the volume of the radio or of the television or its brightness depending on the context. With such an example we already see several natural 'modularities' that call for independence of accounts. The basic operation and meaning of the 'wheel' gadget can be defined independently of the particular values that it is used to manipulate. Those values are themselves then defined within the individual 'theories' of TV operation, radio operation, and so on. Interoperability is achieved by defining the necessary relation between the modules.

This very simple scenario already suggests the value of adopting different ontological perspectives to describe the involved objects: this wheel can be just an object that can be turned, it can be a volume controller or a brightness controller, or it can be a controller for a radio or a television. Moreover, volume is a position for the wheel as volume controller, but a certain voltage for the speaker, or decibel for an acoustic sensor, etc.

Concerning modularity we therefore come to at least two conclusions. First, different purposes of a single entity can call for different ontologies reflecting those purposes. It is even possible that different perspectives can lead to incompatible ontologies; for instance, what is called a small symbol in an ontology for visually users with a vision deficit might be called a big symbol in an ontology for users without such a deficit. In fact, this is only one type of the many heterogeneity requirements.

2.3 Bioportal: an ontology repository

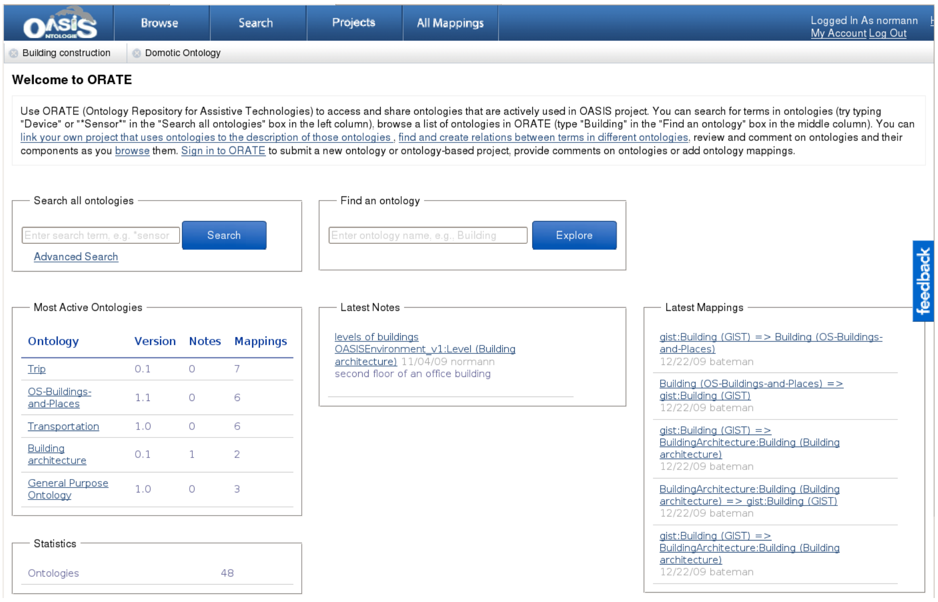
Bioportal[1] is an open source ontology repository where a user can search and browse the ontologies, find resources annotated with concepts from these ontologies, and download ontologies for their use. Provides facilities for browsing, visualizing, and reusing ontologies as well as a basic repository for ontology storage and retrieval.

Researchers in biomedical informatics submit their ontologies to BioPortal and others can access the ontologies through the Bioportal user interface or through web services. The Bioportal users can browse and search the ontologies, update the ontologies in the repository by uploading new versions, comment on any ontology (or portion of an ontology) in the repository, evaluate it and describe their experience in using the ontology,

Understanding how the concepts in different ontologies relate to one another is one of the key requirements of BioPortal users. We refer to the relations between concepts in different ontologies as concept mappings, or simply mappings.

Mappingstool is one of the additions within the BioPortal framework that move the system further towards the OASIS project goals of modularity and re-usability.

[1] `http://bioportal.bioontology.org/`



Welcome to ORATE

Use ORATE (Ontology Repository for Assistive Technologies) to access and share ontologies that are actively used in OASIS project. You can search for terms in ontologies (try typing "Device" or "*Sensor*" in the "Search all ontologies" box in the left column), browse a list of ontologies in ORATE (type "Building" in the "Find an ontology" box in the middle column). You can link your own project that uses ontologies to the description of those ontologies , find and create relations between terms in different ontologies, review and comment on ontologies and their components as you browse them. Sign in to ORATE to submit a new ontology or ontology-based project, provide comments on ontologies or add ontology mappings.

Most Active Ontologies

Ontology	Version	Notes	Mappings
Trip	0.1	0	7
OS-Buildings-and-Places	1.1	0	6
Transportation	1.0	0	6
Building architecture	0.1	1	2
General Purpose Ontology	1.0	0	3

Statistics

Ontologies	48

Latest Notes

levels of buildings
OASISEnvironment_v1:Level (Building architecture) 11/04/09 normann
second floor of an office building

Latest Mappings

gist:Building (GIST) => Building (OS-Buildings-and-Places)
12/22/09 bateman

Building (OS-Buildings-and-Places) => gist:Building (GIST)
12/22/09 bateman

gist:Building (GIST) => BuildingArchitecture:Building (Building architecture)
12/22/09 bateman

BuildingArchitecture:Building (Building architecture) => gist:Building (GIST)
12/22/09 bateman

gist:Building (GIST) => BuildingArchitecture:Building (Building architecture)
12/22/09 bateman

Figure 2.3: ORATE homepage

The Ontology Repository for Assistive Technologies (ORATE) is built upon Bioportal technology. We will use Bioportal when we are referering to the ORATE version of the Bioportal platform that we have extended.

Chapter 3

Previous work

3.1 Existing tools

A brief overview about the existing tools for ontology mapping is provided.

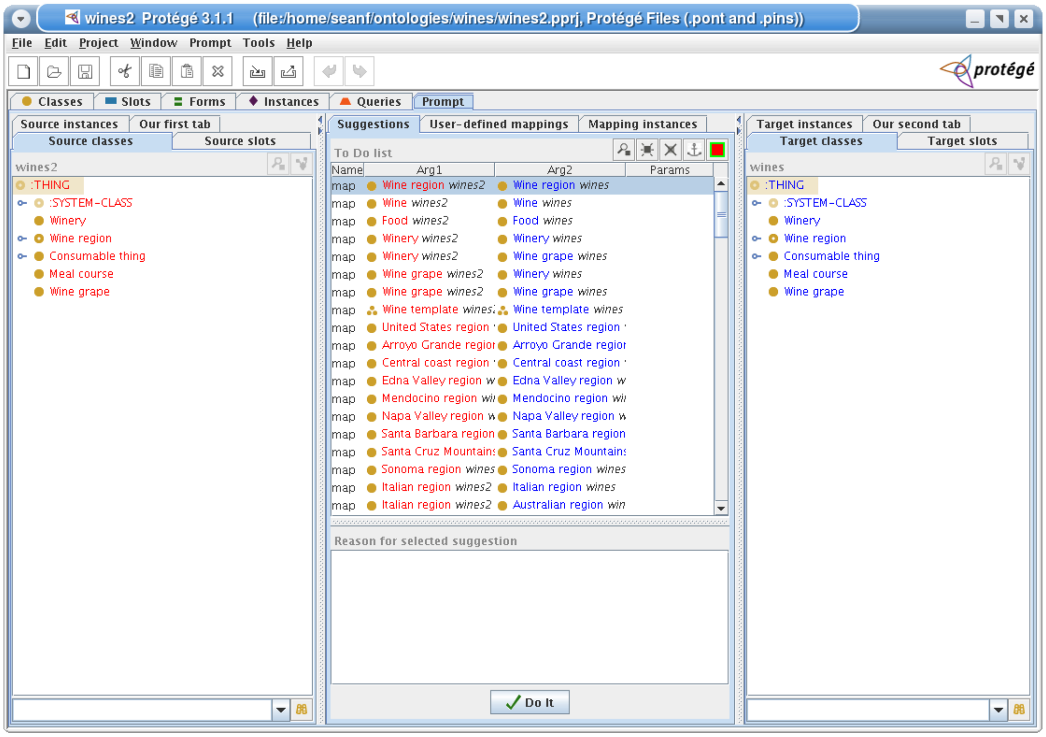
3.1.1 Prompt + CogZ

PROMPT is a Protege[1] plugin that supports various tasks for managing multiple ontologies, including ontology mapping. The ontology mapping process starts by performing an initial comparison of the source and target to be mapped, based on lexical comparison of class names (other algorithms are available). After the comparison, you can create mappings from a set of candidates or user-defined mappings. As you are creating the mappings, Prompt 'monitors' the correspondences that you create and creates instances of its built-in mapping ontology behind the scenes

The PROMPT plugin framework allows developers to plug in your own mapping algorithms for initial comparison of ontologies or your own user interface. At the moment there are some plugins available in Prompt (these plugins are installed when you install PROMPT):

CogZ alternative perspective to Prompt's mapping interface.

[1] Protege is a free, open source ontology editor and knowledge-base framework. More info in `http://protege.stanford.edu/`


wines2 Protégé 3.1.1 (file:/home/seanf/ontologies/wines/wines2.pprj, Protégé Files (.pont and .pins))
File Edit Project Window Prompt Tools Help
protégé
Classes Slots Forms Instances Queries Prompt
Source instances Our first tab
Source classes Source slots
wines2
:THING
:SYSTEM-CLASS
Winery
Wine region
Consumable thing
Meal course
Wine grape
Suggestions User-defined mappings Mapping instances
To Do list
Reason for selected suggestion
Do It
Target instances Our second tab
Target classes Target slots
wines
:THING
:SYSTEM-CLASS
Winery
Wine region
Consumable thing
Meal course
Wine grape

Name	Arg1	Arg2	Params
map	Wine region *wines2*	Wine region *wines*	
map	Wine *wines2*	Wine *wines*	
map	Food *wines2*	Food *wines*	
map	Winery *wines2*	Winery *wines*	
map	Winery *wines2*	Wine grape *wines*	
map	Wine grape *wines2*	Winery *wines*	
map	Wine grape *wines2*	Wine grape *wines*	
map	Wine template *wines2*	Wine template *wines*	
map	United States region '	United States region '	
map	Arroyo Grande regior	Arroyo Grande regior	
map	Central coast region '	Central coast region '	
map	Edna Valley region *w*	Edna Valley region *w*	
map	Mendocino region *wi*	Mendocino region *wi*	
map	Napa Valley region *w*	Napa Valley region *w*	
map	Santa Barbara region	Santa Barbara region	
map	Santa Cruz Mountain:	Santa Cruz Mountain:	
map	Sonoma region *wines*	Sonoma region *wines*	
map	Italian region *wines2*	Italian region *wines*	
map	Italian region *wines2*	Australian region *win*	

Figure 3.1: Prompt tab plugin

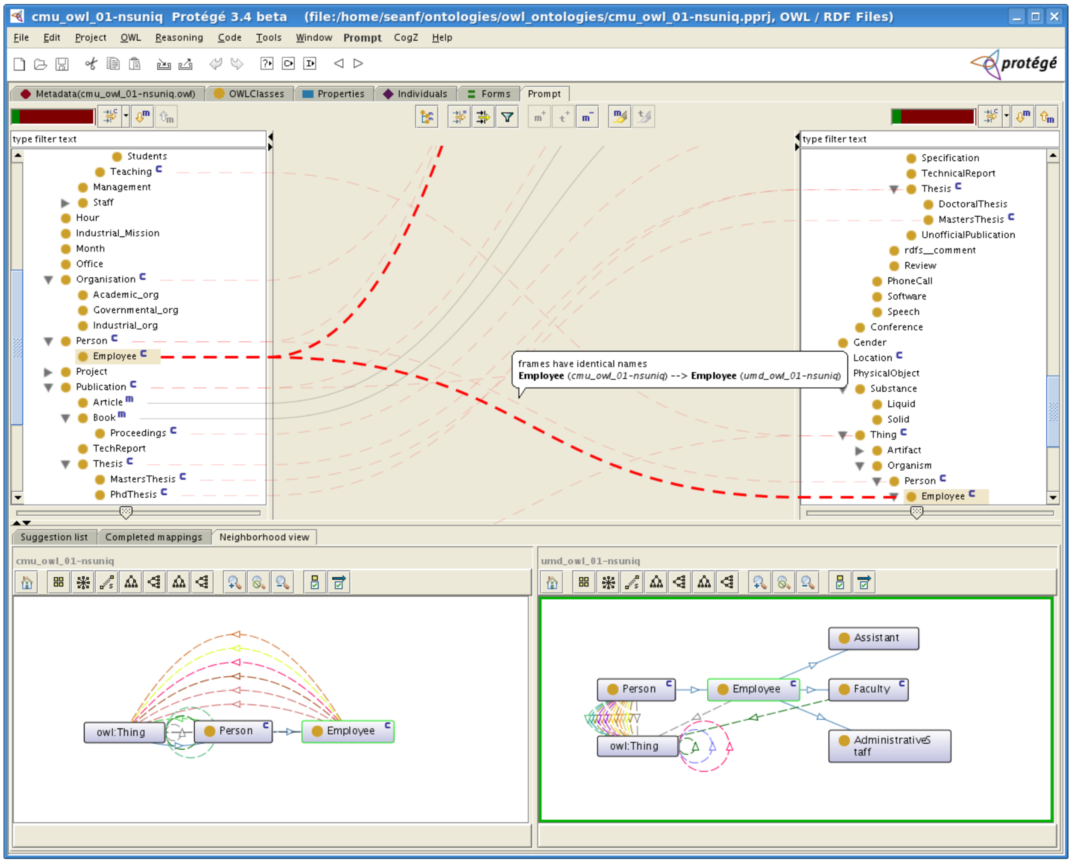
FOAM in Prompt use the FOAM algorithm for mapping

Synonyms if you have synonyms for your terms, have Prompt use synonyms to create lexical mappings

CogZ has been implemented with mapping user support in mind. It supports a variety of filters to aid the mapping process. It also uses a visual representation of mappings rather than a simple list. Finally, it supports a visual graph-based representation of ontology concepts that allow you to inspect the semantics/context of the terms you intend to map.

PROMPT and its CogZ extension are under active development and are included in the full installation of Protege. This work environment manages multiple ontologies and performs automatic and manual mappings, and it is quite full-featured.

The main features of CogZ are:



cmu_owl_01-nsuniq Protégé 3.4 beta (file:/home/seanf/ontologies/owl_ontologies/cmu_owl_01-nsuniq.pprj, OWL / RDF Files)

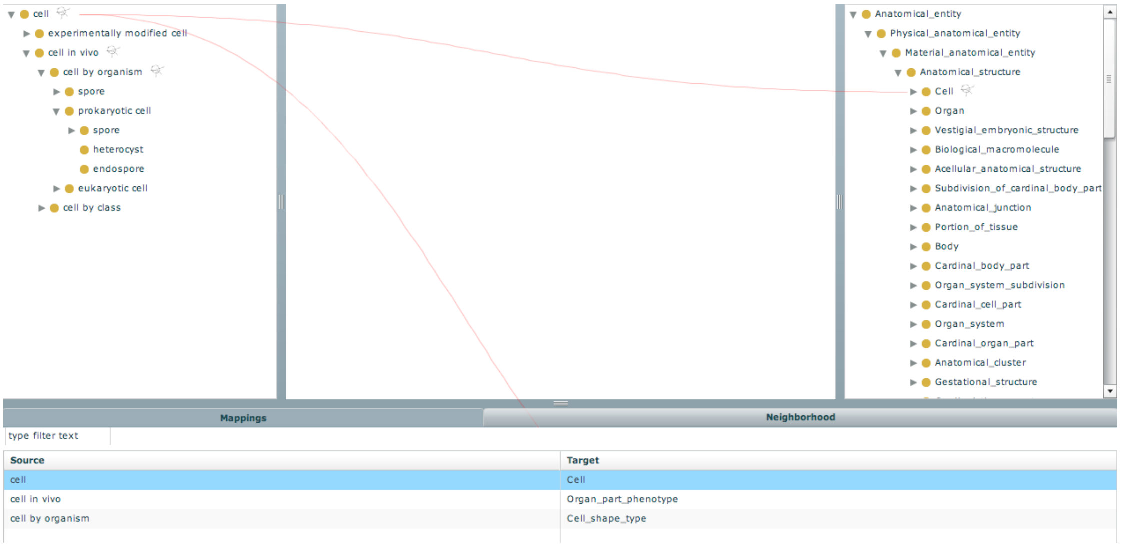
frames have identical names
Employee (*cmu_owl_01-nsuniq*) --> **Employee** (*umd_owl_01-nsuniq*)

Figure 3.2: CogZ user interface

- View structural representation of mapping suggestions in Prompt.
- Identifies candidate rich areas of the ontologies.
- Provides progress indicators about the completion of the mapping.
- Visualizes suggested, created, and temporary mappings and visually supports manual mapping of terms.
- Provides ontology and mapping filtering.
- Mapping status and hierarchy reports.

3.1.2 Mapping view

This experimental tool[2] is under development by Victoria University and is constrained to the visualization of the relationship, the classes, and the corresponding class hierarchies. The list of mappings is provided so that the users can navigate through the mappings.



Source	Target
cell	Cell
cell in vivo	Organ_part_phenotype
cell by organism	Cell_shape_type

Figure 3.3: Mapping view

At the moment there is no way to know the mapping origin at the moment. It is developed in FLEX[3].

3.2 Comparison with Mappingstool

Mappingstool provides a service that allow the user to create and manipulate mappings online, unlike the plugin Prompt + CogZ that is a desktop application. This Protege plugin is extremely powerful providing features such as automatic mapping based on lexic comparison and visual representation of mappings, but that was not the objective of our Bioportal ex-

[2] http://keg.cs.uvic.ca/ncbo/mapping/MappingApp.html

[3] Framework for deployment of cross-platform Rich Internet Applications based on the Adobe Flash, explained in 4.3

tension. We have developed Mappingstool having in mind the domain curators that manually creates mappings.

On the other hand, we have describe a alpha release of a mapping editor developed by Victoria University. They provide a nice visualization for the mappings (a line from the concept source to the target). At the moment, we do not know which is the data source for the mappings, and it does not show enough information about a concept or the mapping itself. Mappingstool provides more information about the ontology, shows mapping information of the whole ontology or a given concept, export features, context visualization, etc.

The main limitation of Mapping view is that you cannot create mappings within the tool, is only for visualization.

Chapter 4

Technology overview

We will present the technologies that have been used to develop this Bioportal extension, with references how we have used them.

4.1 Ruby

Ruby is a dynamically typed programming language similar to Python, Smalltalk and Perl. Its creator, Yukihiro 'matz' Matsumoto, blended parts of his favorite languages (Perl, Smalltalk, Eiffel, Ada, and Lisp) to form a new language that balanced functional programming with imperative programming. In Ruby, everything is an object. Every bit of information and code can be given their own properties and actions.

4.1.1 Ruby and Rails

Ruby on Rails[1] is an open source web application framework for the Ruby programming language. It enables rapid prototyping as well as solid integration with web services.

It is intended to be used with an Agile development methodology that is used by web developers for rapid development. It has became a leading mature UI framework supported by large software development communities.

[1] Official website: `http://rubyonrails.org/`

There are two basic principles that govern the way that Ruby on Rails works. The first is often referred to as DRY, or Don't Repeat Yourself and the second principle is COC or Convention over Configuration. This framework has been growing along the years due to its simplicity and community.

The frontend of Bioportal currently uses the Ruby on Rails technology following the Model-View-Controller (MVC) architectural design pattern. These principle divides the work of an application into three separate but closely cooperative subsystems:

Model It is the system that handles the data representation, handling with validation, association, transactions, etc.

View A presentation of data in a particular format, triggered by a controller's decision to present the data. It renders the Model with the less possible logic. Multiple views can exist for a single model for different purposes (e.g. JSON, XML, HTML, PDF).

Controller Dispatches requests and control flow, centralize access and interacts with Model and View.

4.2 Javascript

JavaScript is an implementation of the ECMAScript language standard and is typically used to enable programmatic access to computational objects within a host environment. It it is primarily used in the form of client-side JavaScript, implemented as part of a web browser in order to provide enhanced user interfaces and dynamic websites.

The very good ideas of Javascript include loose typing, dynamic objects and functions (first class objects with lexical scoping). It is surprising, but it has more in common with Lisp and Scheme than with Java.

Nowadays it is becoming more and more popular thanks to HTML 5, CSS 3, AJAX, Comet and the different stack of technologies that makes the browser as the first target for personal and enterprise applications. And not

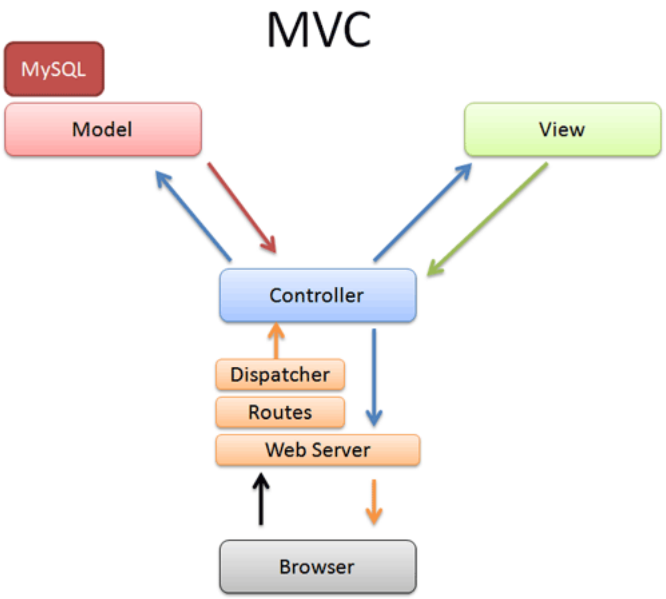


Figure 4.1: MVC arquitectural design pattern with RoR framework

only on the client side, there are some interesting projects around server-side Javascript such as Node.js[2], a toolkit for writing extremely high performance non-blocking event driven network servers in JavaScript.

4.2.1 AJAX

AJAX (Asynchronous JavaScript and XML) is a method of building interactive applications for the Web that process user requests immediately. Ajax combines several programming tools including JavaScript, dynamic HTML (DHTML), Extensible Markup Language (XML), cascading style sheets (CSS), the Document Object Model (DOM), and the Microsoft object, XMLHttpRequest. Ajax allows content on Web pages to update immediately when a user performs an action, unlike an HTTP request, during which users must wait for a whole new page to load. XML is not actually

[2] The author's definition is 'Evented I/O for V8 JavaScript', more info in `http://nodejs.org`

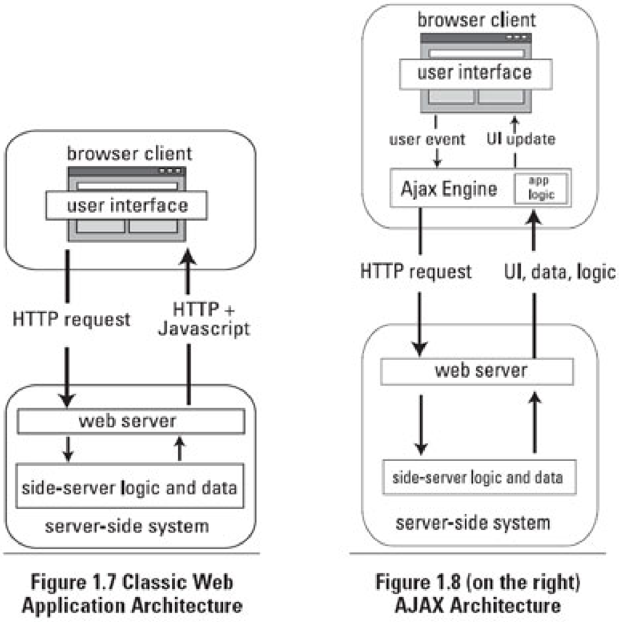


Figure 4.2: AJAX arquitecture

required as the data format, in fact, we make an extensive use of JSON, explained in the next section.

Mappingstool makes extensive use of AJAX, being capable to load different modules asynchronously and retrieving new information without reloading the page.

4.2.2 JSON

JSON (JavaScript Object Notation)[3] is a lightweight data-interchange format. It is easy for humans to read and write, easy for machines to parse and generate and based on a subset of the JavaScript Programming Language.

It has became really popular thanks to AJAX. In our project we make an extensive use of JSON as a data format between the browser and the client.

An example of a JSON object:

[3] Official website: http://www.json.org/

```
bioportal.mappings = {
  sourceTreeviewComp: new Object(),
  sourceConceptInfoComp: new Object(),
  util: {
      isValidObject: function() {},

      ontoSelectElemSource: '',
      ontoSelectElemTarget: '',

      idOntologySource: ['Example 1', 3, 'a']
  }
};
```

4.2.3 jQuery

jQuery is a cross-browser, fast and concise JavaScript Library designed to simplify the client-side scripting of HTML. It facilitates HTML document traversing, event handling, animating, and Ajax interactions for rapid web development. It is most popular Javascript library in use today.

We have make extensive use of this library for manipulating the DOM[4], handle the different events and for the AJAX interation. With this library we have not been worried about the different layout engines of the browsers that translates the HTML into the DOM, being this process the main problem for cross-compatibility.

jQuery also provides capabilities for developers to create plugins on top of the JavaScript library.

Plugins

We make use of some jQuery plugins such as:

jQuery UI.Layout It is page layout manager that can create any UI look you want - from simple headers or sidebars, to a complex application with toolbars, menus, help-panels, status bars, sub-forms, etc. It is

[4] Document Object Model, cross-platform and language-independent convention for representing and interacting with objects in HTML, XHTML and XML documents

really powerful and with a lot of different capabilities and customizations. We use this widget for the whole layout of Mappingstool, being able to resize and show/hide diffent page components.

jQuery Tabs Tabs are generally used to break content into multiple sections that can be swapped to save space, much like an accordion. We make use of the tabs for the information of a ontology concept.

jQuery History It is for Ajax-driven pages to have back/forward navigation and to allow bookmarking. We make extensive use of this plugin to be able to go forward and back throught the process of selecting and ontology, browse a mapping, etc. So we preserve the state modifying the URL as appropiate.

jqgrid It is really powerful grid plugin. It is a client-side solution that loads data with Ajax callbacks, so it can be used with any server-side language, in our case Ruby on Rails for providing its data. These grids are AJAX-enabled, and support sorting, paginating, and data search. In order to facilitate its integration with RoR, the Rails plugin *2dc_jqgrid* has been used.

4.3 FlexViz: ontology visualization tool

Adobe Flex is a software development kit released by Adobe Systems for the development and deployment of cross-platformrich Internet applications based on the Adobe Flash platform. Flex applications can be written using Adobe Flash Builder or by using the freely available Flex compiler from Adobe.

FlexViz is a graph based visualization tool written in Adobe Flex. It has support for browsing a single ontology where the concepts are represented by nodes and the relationships between concepts (e.g. *is_a*, *part_of*) are represented as arcs.

Here is a list of the main features:

- Node and arc type filtering
- Searching
- Many different graph layouts

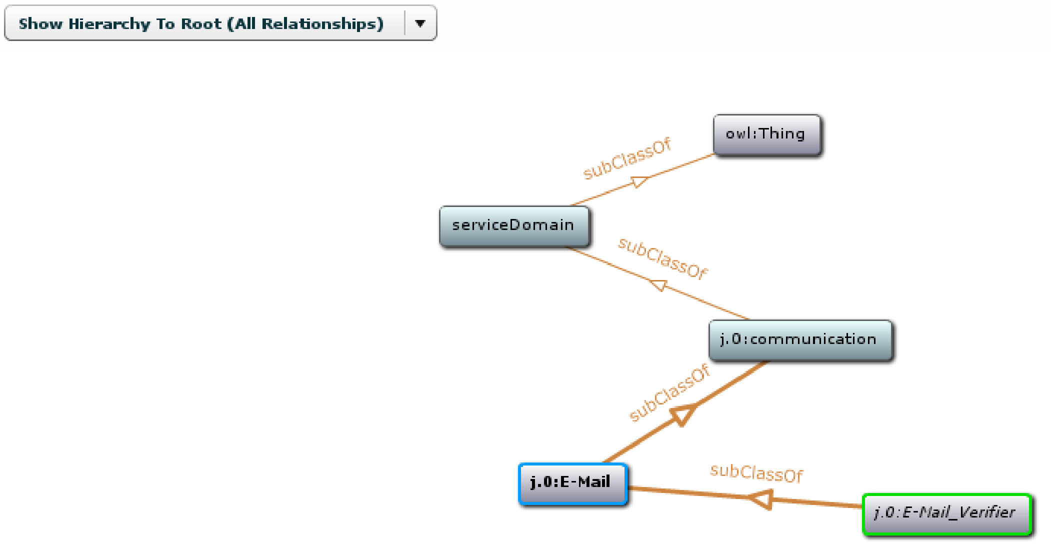


Figure 4.3: Example of visualization using FlexViz

- Customizable node and arc labels
- Customizable node and arc tooltips
- Zooming
- Node and arc type filtering
- Can be displayed as a widget on other (with a fixed ontology)

All communication with BioPortal (ORATE in our case) occurs through the REST services provided by BioPortal for accessing ontology-related data. The asynchronous service calls help reduce performance concerns that are typically found with desktop visualization systems that depend on loading the entire ontology into memory. All expansion, search, and focus operations result in a service call to BioPortal. Results are cached for fast recall.

We will reuse the visualization tool that is already deployed in Bioportal in order to show the context of an ontology concept.

4.4 Web Services

A web service is an Application Programming Interface (API) or that is accessed via HTTP and executed on a remote system, hosting the requested service. There are currently two schools of thought in developing web services: the traditional, standards-based approach SOAP and the conceptually simpler REST.

SOAP[5] was designed to be a platform and language-neutral alternative to previous middleware techologies like CORBA and DCOM. Together with WSDL and XML Schema, SOAP has become the standard for exchanging XML-based messages. It has became less popular during the past years due to its complexity, in favour of REST.

REST[6] is not a standard, there is no W3C specification. REST is just an architectural style. The HTTP methods such as GET and POST, PUT and DELETE are the verbs that the developer can use to describe the necessary create, read, update, and delete (CRUD) actions to be performed. Some may see an analogy to operations in SQL, which also relies on a few common verbs. Similar to SOAP services, RESTful Web services are language and platform independent, having additional advantages in that they are light weight, simple and easier to integrate.

4.5 Bioportal

Bioportal uses a classic architecture layered approach, which decouples the logic and domain object models between each layer. This approach decouples the versioning and changes in one layer from another. There are four differenciate layers within two projects: Bioportal UI and Bioportal Core.

4.5.1 Bioportal UI

Bioportal UI is the Presentation Tier that delivers the user-interface which currently uses the Ruby on Rails technology.

[5] SOAP, Simple Object Access Protocol

[6] REST, Representational State Transfer

4.5.2 Bioportal Core

Bioportal Core it is composed of the following layers:

Interface Tier Consists of both REST and SOAP Web Services[7] that present all BioPortal capabilities to the upper tiers (e.g., upload ontology, download ontology, display concept, administrative functions, etc). Bioportal UI is primarly driven by the REST services. Using this web services another organizations could implement a completely different UI than that currently exposed by the BioPortal. Or, one could simply want to consume a single REST services for integrating into a back-end workflow with no UI at all.

This tier include Web Services to get ontology metadata, to get ontology content, to download an ontology, and to search ontologies. Each ontology in BioPortal is indexed with a stable ontology identifier and is the same for all versions of the same ontology, ontology versions are indicated by a version identifier, which changes from one version to the next. BioPortal Web services provide a list of the latest versions of all BioPortal ontologies, enable callers to find an ontology identifier based on a version identifier, find all version identifiers for a specific ontology, and list ontology categories. Web services to get ontology content include services to get all root concepts, get children or parents of a specific concept, and get details of a concept.

Business Logic Tier Uses the Spring technology which enables a partner to insert any software implementation that abides to the defined interfaces.

Persistence Tier Uses the Hibernate technology as a basic object-relational mapping to the back-end relational database. Hibernate is used for storing administrative (e.g., user information) and external ontology data (e.g., ontology attributes specified at upload time). All ontology content is stored in Protege and LexGrid as shown in the Business Logic layer.

[7] See NCBO REST Web Services documentation to learn about consuming the BioPortal REST services: `http://www.bioontology.org/wiki/index.php/NCBO_REST_services`

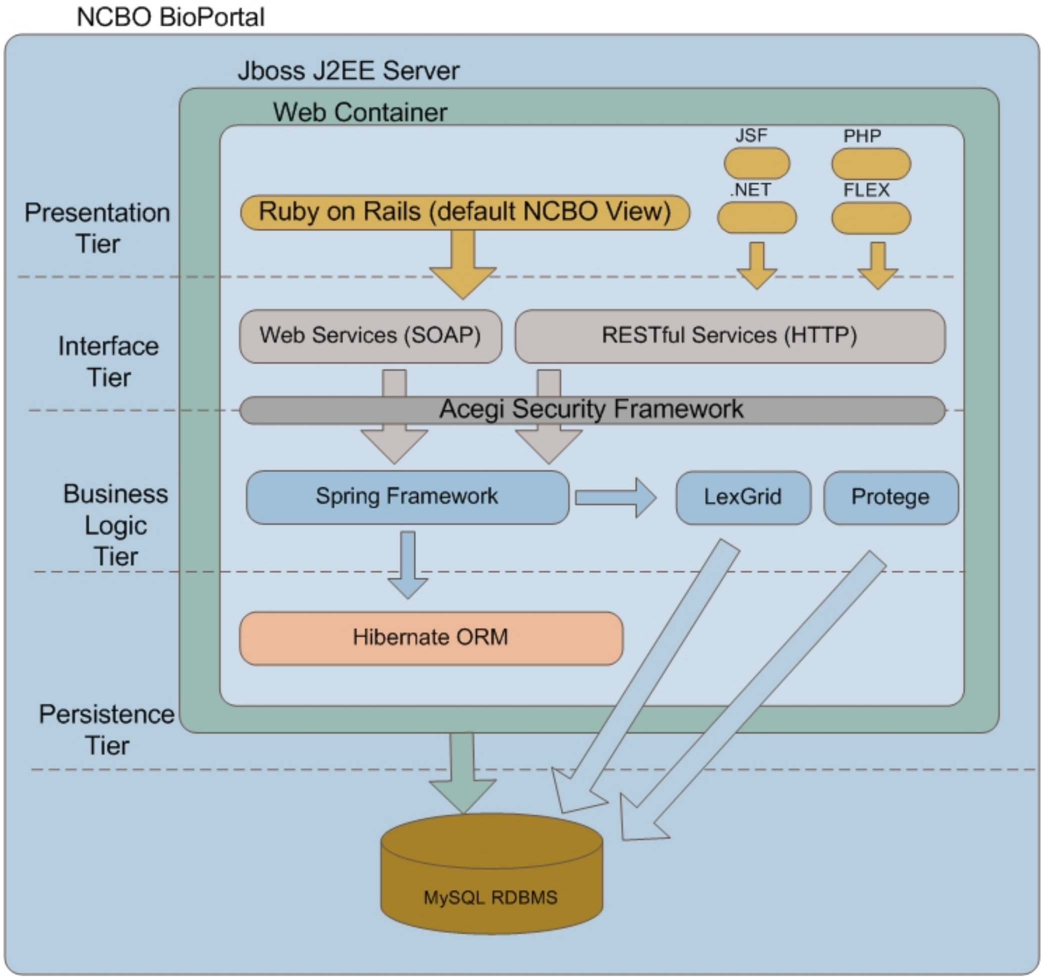


Figure 4.4: Bioportal arquitecture

Chapter 5

Software architecture and design

5.1 Architecture overview

Mappingstool is an extension built on top of Bioportal UI, where the main login of the applications runs on the browser side, using Javascript making extensive use of AJAX. These asynchronous calls that interacts with the server makes the corresponding operations (query, create, update, delete) to the data layer.

We have used the same approach that Bioportal UI uses with the framework Rails to extend it. We have created some controllers on the frontend Bioportal UI (Ruby on Rails) that retrieves information from the Model, having as data sources the external RESTful Web Services (Bioportal Core) and the internal database built in MySQL.

More precisely, here it is a brief overview of the tipical interaction between the browser and Mappingstool:

- A user makes an HTTP request to the application, then the server receives the request and send the response to the browser.
- When the browser finishes to load the page, the browser is *waiting* until an event is fired by the user.

- If the user clicks on an ontology source or target, browse the concept tree, load a mapping, etc. We make AJAX calls to the server.
- This AJAX calls retrieves information from either Bioportal Core Web Services or the MySQL database that Biportal UI handles.

From the point of view where the code is executed, we can clearly differenciate two different approaches in the arquitecture of Mappingstool: server and client side.

5.1.1 Client side

The majority of the logic of Mappingstool is written in Javascript to be executed by the browser. It interacts with the server-side code of Bioportal UI making use of AJAX.

We have tried to modularize the code, so that we produce reusable and maintenable components. As we explained in the last chapter, we make use of several jquery plugins that are used internally in each component, except the jQuery history and jQuery layout. Each component is a class[1] that makes use of the jQuery plugins mentioned in the last chapter, being independent and not accesing the global state of the application (some needed data and dependencies with other components is injected).

The file '/public/javascripts/mappingstool/controller.js' is the starting point of the application, where you can set the components that you want to load and the main configuration for the different components and jQuery plugins. Every component and function is in the scope of *bioportal* object, so that we do not have conflicts with other Javascript libraries.

When the page is loaded we need to initialize the components, the layout and the history:

```
/**
 * Initialize the layout and components, making the appropiate requests
 * according to the current state (hash url)
 */
bioportal.mappings.init = function() {
```

[1] Javascript is commonly considered procedural, but it is still possible to define custom objects that behave, in many ways, like classes in C or Java.

```
  bioportal.mappings.initLayout();

  // Initialize the history
  $.historyInit(bioportal.mappings.util.loadPageFromUrlParams);

  bioportal.mappings.initComponents();
}
```

The main components are:

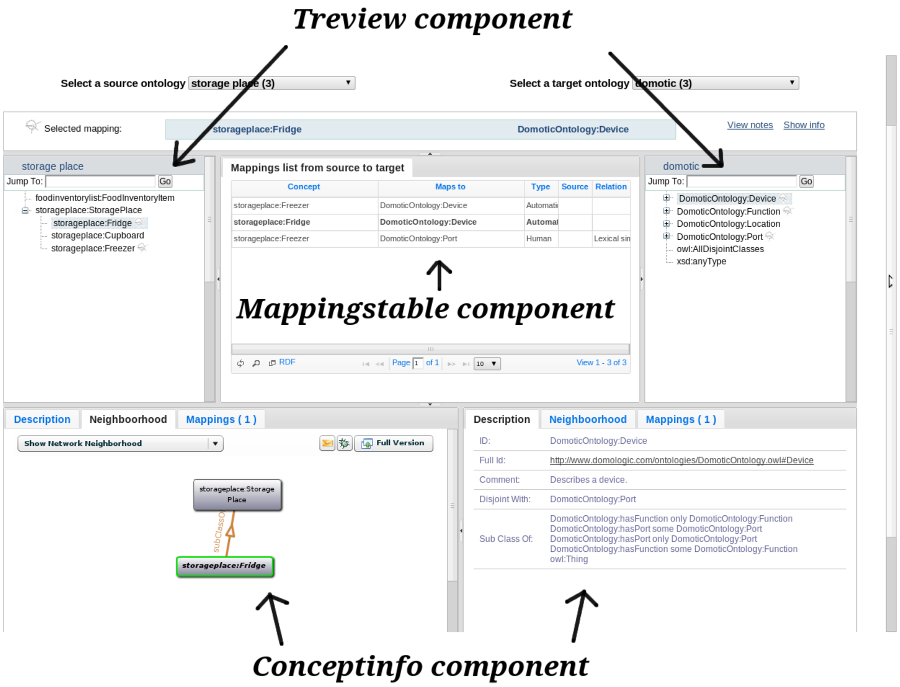
Conceptinfo It is used to show the information about a given concept ontology. It retrieves the concept information (label, id, fullId, etc.), the graph visualization and the mappings that this concept has with other ontologies. It is reused for the source and target ontology.

Mappingstable It is the component that shows a table (using jQgrid) with the mappings of the source and target ontologies. When the ontology source or target ontology changes it fires a event to this component. The data can be sorted, filtered, exported in RDF, etc.

Treeview It retrieves the concept hierarchy of a ontology and display it in a tree. The user can also search a concept withing the current ontology. When you click on a concept it refresh the Conceptinfo component.

As a sample code, in the following function we initialize the components:

```
/**
 * Init the components to use (conceptinfo, mappingstable and treeviews)
 * according to the global state. It is called just once when the page
 * is ready and the layout has been initialized.
 */
bioportal.mappings.initComponents = function () {
    [...]
    M.sourceTreeviewComp = new M.treeviewComponent.treeview(
        M.util.idOntologyVersionSource, M.util.idOntologySource,
        M.util.idConceptSelectedSource, M.util.labelConceptSelectedSource,
        { selectorTree: '.simpleTree',
          selectorTreeWrapper: '.ui-layout-west',
          identifier: 'source',
        init: true }, M.sourceConceptInfoComp);
    [...]
    M.mappingsTableComp = new M.mappingstableComponent.mappingstable(
        M.util.idOntologySource, M.util.idOntologyTarget,
        { selectorMappingsGrid: '#list-mappings',
          selectorMappingsGridPager: '#list-mappings-pager',
          selectorContainer: '#mappings-source-target-tab',
          init: true });
}
```



Concept	Maps to	Type	Source	Relation
storageplace:Freezer	DomoticOntology:Device	Automati		
storageplace:Fridge	**DomoticOntology:Device**	**Automat**		
storageplace:Freezer	DomoticOntology:Port	Human		Lexical sin

ID:	DomoticOntology:Device
Full Id:	http://www.domologic.com/ontologies/DomoticOntology.owl#Device
Comment:	Describes a device.
Disjoint With:	DomoticOntology:Port
Sub Class Of:	DomoticOntology:hasFunction only DomoticOntology:Function DomoticOntology:hasPort some DomoticOntology:Port DomoticOntology:hasPort only DomoticOntology:Port DomoticOntology:hasFunction some DomoticOntology:Function owl:Thing

Figure 5.1: Main components

5.1.2 Server side

The directory structure of the whole Bioportal UI is the convention taken from Ruby on Rails[2].

[2] See Appendix D

We have created and modified some Controllers[3] that handles the different actions needed by Mappingstool. The controller access the Model handling the internal database that has Bioportal UI and also from the REST Web Services from Bioportal Core. Once we get the data, we render it into a View such as JSON, XML, RDF or HTML.

Mappingstool makes extensive use of the REST web services of the Interface tier (Bioportal Core), between others the following actions: authenticate a user, get all ontologies, find all version identifiers for a specific ontology, get all root concepts for a specific ontology, get children or parents of a specific concept, and get details of a concept.

In the following sample code we have an action that retrieves the information of an ontology concept, located in the Controller mappingstool (used in the client-side component *Conceptsinfo*). *DataAccess* object provides an interface to Bioportal Core Web Services, and *Mapping* is an *ActiveRecord* object (belongs to the Model) that access the MySql database of Bioportal UI. Then we render the View *node_info* injecting the variables we have taken from the Model.

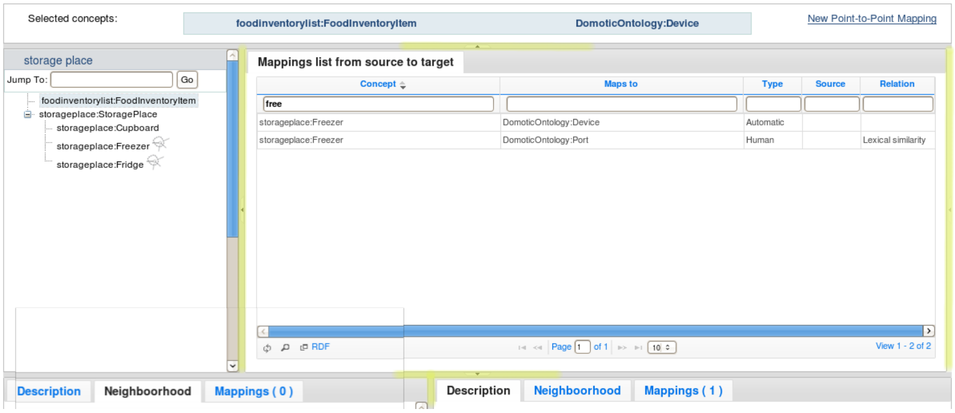
```
def node_info
  @concept = DataAccess.getNode(params[:ontology], params[:id])
  @ontology = DataAccess.getOntology(params[:ontology])
  @mappings = Mapping.find(:all, :conditions =>
    {:source_ont => @concept.ontology_id, :source_id => @concept.id})

  render :partial => 'node_info'
end
```

5.2 UI overview

5.2.1 Flexibility

With the use of the plugin jQuery Layout the tool gains flexibility. You can hide and show sections of the tool or expand or contract them without refreshing the page. This is very convenient for instance when you have an ontology with a lot of concepts and you want to see the entire tree, or when you want to be able to see bigger the concept graph.

[3] See subsection 4.1.1



Selected concepts: foodinventorylist:FoodInventoryItem DomoticOntology:Device

storage place

Jump To:

- foodinventorylist:FoodInventoryItem
- storageplace:StoragePlace
  - storageplace:Cupboard
  - storageplace:Freezer
  - storageplace:Fridge

Mappings list from source to target

Concept	Maps to	Type	Source	Relation
free				
storageplace:Freezer	DomoticOntology:Device	Automatic		
storageplace:Freezer	DomoticOntology:Port	Human		Lexical similarity

View 1 - 2 of 2

Figure 5.2: Flexibility of the main screen, we can expand or collapse the marked zones.

5.2.2 Usability

We have tried to create a rich experience where all the use cases could be done without refreshing the page. Integrating jQuery history, we have all the functionality that could be in a conventional webpage bookmarking the current state or going forward and backward.

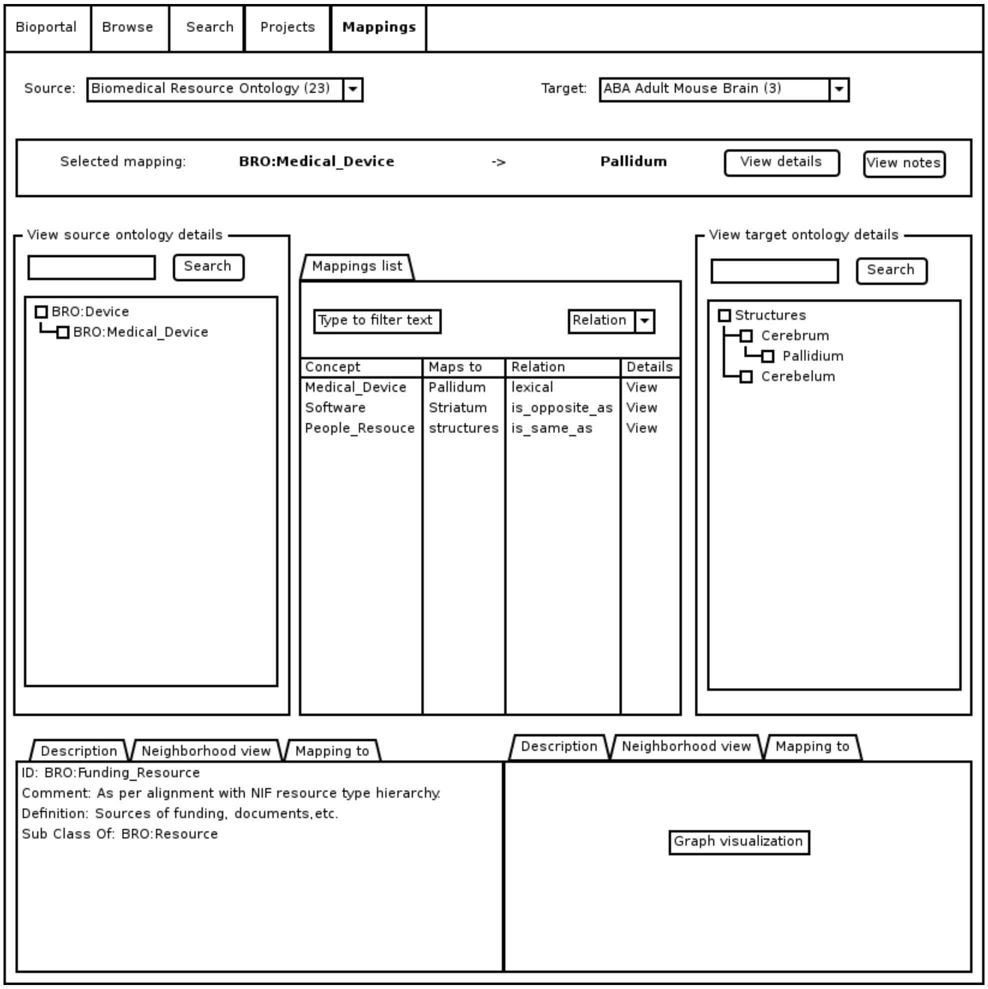
Mappingstool has been reviewed by ontology curators providing feedback about the different components. This enables the mapping curator to look for different ontologies, create a mapping, look at the context, create a note, etc. easily and without losing the current state of your page.

5.2.3 Mockups

We created several mockups with the expected layout of the web application, and its main use cases. We used Wireframe Sketcher[4] to create them.

There are more mockups in Appendix A.

[4] It is an Eclipse plugin to create mockups and wireframes, more information at `http://wireframesketcher.com/`



Source: Biomedical Resource Ontology (23)

Target: ABA Adult Mouse Brain (3)

Selected mapping: **BRO:Medical_Device** -> **Pallidum**

Concept	Maps to	Relation	Details
Medical_Device	Pallidum	lexical	View
Software	Striatum	is_opposite_as	View
People_Resouce	structures	is_same_as	View

ID: BRO:Funding_Resource
Comment: As per alignment with NIF resource type hierarchy.
Definition: Sources of funding, documents, etc.
Sub Class Of: BRO:Resource

Figure 5.3: Main screen

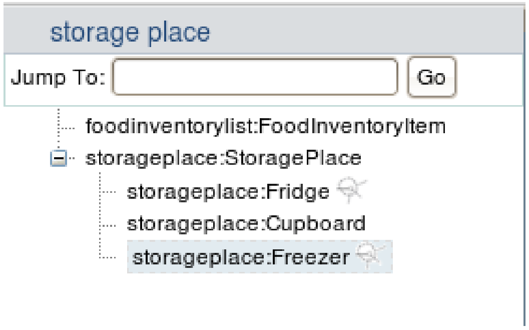
Chapter 6

Main use cases

We will now present the main use cases that the extension covers.

6.1 Browse and search ontology concepts

For this, the user first has to choose an ontology with concepts of interest participating in a mapping, either as the source or the target ontology. Once an ontology is selected, a tree view is presented with the hierarchy of the ontology, having icons when there are mappings in a specific concept. Also the user is able to search a specific concept of the tree.



foodinventorylist:FoodInventoryItem
storageplace:StoragePlace
storageplace:Fridge
storageplace:Cupboard
storageplace:Freezer

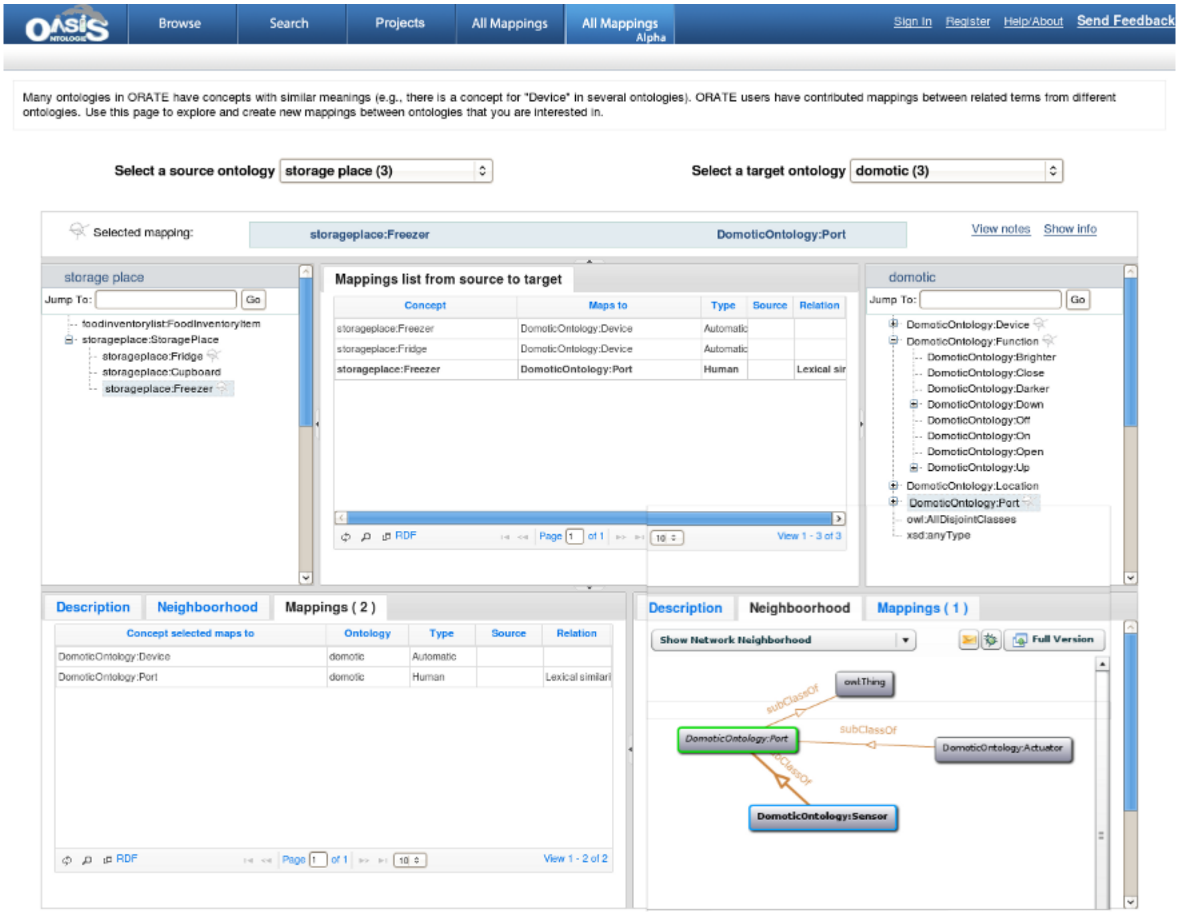
Figure 6.1: Browse and search ontology concepts

6.2 Browse mappings

Users have the option to browse mappings between two ontologies or the ones from a specif concept (to the same ontology or others). After the selection of a source and target ontologies, you have two ontology trees where you completely browse the ontologies, then the list of mappings from the ontology source to the target is displayed. Each mapping has its own set of metadata that describes who created the mapping and when, which algorithm was used to produce the mapping, application context in which the mapping might be valid, the specific mapping relationship, and other properties. There are several options to filter by this options.

The user can also click on a mapping of a specif concept, and the tool will load the target concept. If the mappings has as target another ontology, it will ask if we want to override the current target ontology selected.

We get all mappings associated with a concept attached directly to the current version of this concept or to earlier versions.



Many ontologies in ORATE have concepts with similar meanings (e.g., there is a concept for "Device" in several ontologies). ORATE users have contributed mappings between related terms from different ontologies. Use this page to explore and create new mappings between ontologies that you are interested in.

Select a source ontology: storage place (3)

Select a target ontology: domotic (3)

Selected mapping: storageplace:Freezer DomoticOntology:Port

storage place

Jump To: Go

- foodinventorylist:FoodInventoryItem
- storageplace:StoragePlace
  - storageplace:Fridge
  - storageplace:Cupboard
  - storageplace:Freezer

Mappings list from source to target

Concept	Maps to	Type	Source	Relation
storageplace:Freezer	DomoticOntology:Device	Automatic		
storageplace:Fridge	DomoticOntology:Device	Automatic		
storageplace:Freezer	DomoticOntology:Port	Human		Lexical sir

RDF Page 1 of 1 10 View 1 - 3 of 3

domotic

Jump To: Go

- DomoticOntology:Device
- DomoticOntology:Function
  - DomoticOntology:Brighter
  - DomoticOntology:Close
  - DomoticOntology:Darker
  - DomoticOntology:Down
  - DomoticOntology:Off
  - DomoticOntology:On
  - DomoticOntology:Open
  - DomoticOntology:Up
- DomoticOntology:Location
- DomoticOntology:Port
- owl:AllDisjointClasses
- xsd:anyType

Description Neighboorhood Mappings (2)

Concept selected maps to	Ontology	Type	Source	Relation
DomoticOntology:Device	domotic	Automatic		
DomoticOntology:Port	domotic	Human		Lexical similari

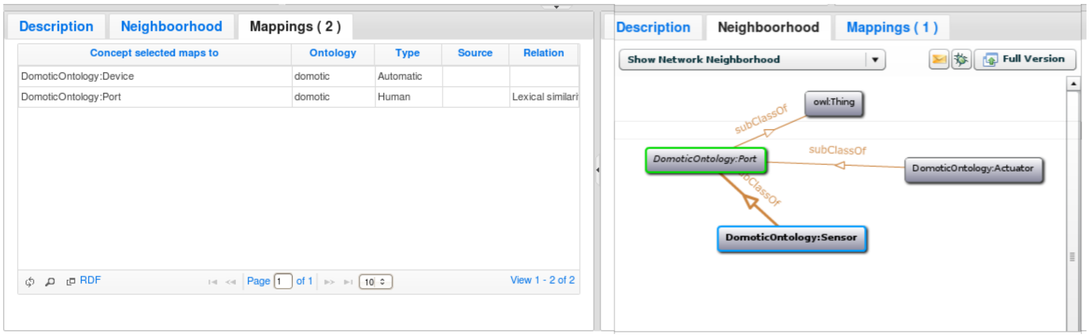
RDF Page 1 of 1 10 View 1 - 2 of 2

Description Neighboorhood Mappings (1)

Figure 6.2: Main screen of the application

6.3 Information about an ontology concept

When users selects a concept from source or target ontologies, you can see the description, the neighboorhood and the list of mappings of this concept with any other concept (concerning target or any other ontology involved in the mapping). If you select a mapping, the target ontology is loaded.



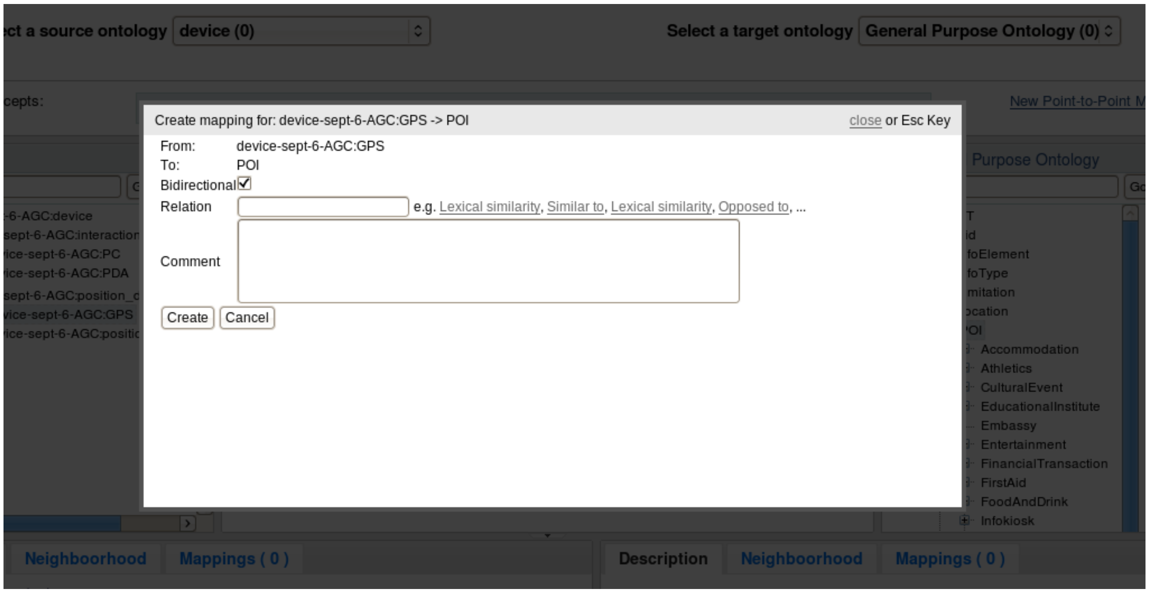
Concept selected maps to	Ontology	Type	Source	Relation
DomoticOntology:Device	domotic	Automatic		
DomoticOntology:Port	domotic	Human		Lexical similari

Figure 6.3: Information about an ontology concept

6.4 Create a mapping

The main use case is to create mappings between two ontologies. The user chooses a source and a target ontologies, select a concept in each ontology tree and create the mapping. If there is already a mapping between these concepts there is an option to view the details (author, date created, relation, map source, map type, etc.).

It creates a new mapping and attach it to the current version of the concept (there could be several versions of the same ontology).



Create mapping for: device-sept-6-AGC:GPS -> POI

From: device-sept-6-AGC:GPS

To: POI

Bidirectional ☑

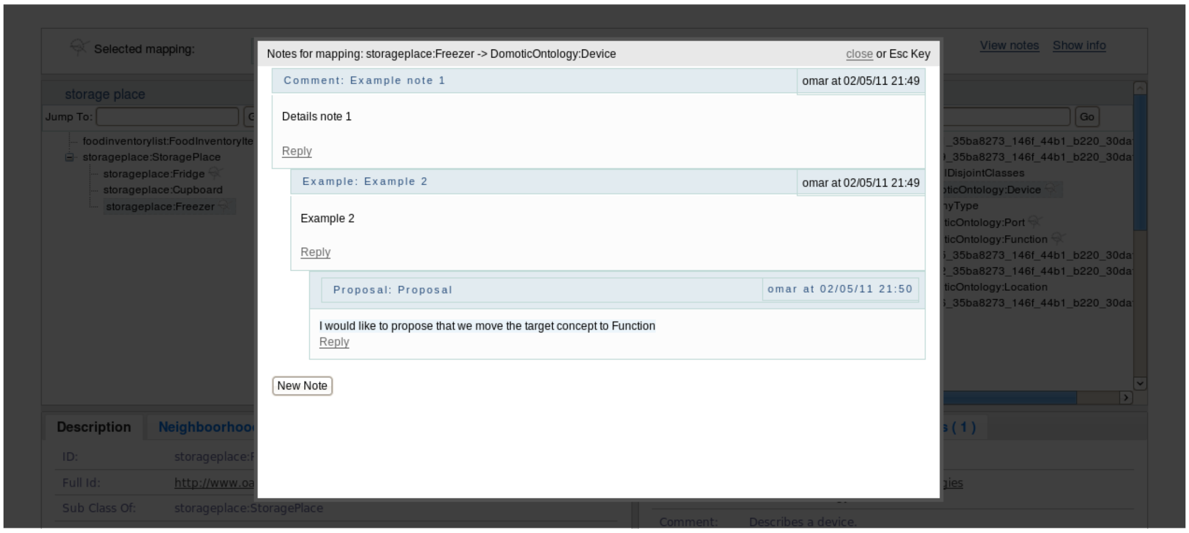
Relation e.g. Lexical similarity, Similar to, Lexical similarity, Opposed to, ...

Comment

Figure 6.4: Create a mapping

6.5 Discussions about a mapping

Registered users can add notes on existing mappings and carry out discussions about the mappings. The curators can create notes, reply and browse notes related to a mapping.



Notes for mapping: storageplace:Freezer -> DomoticOntology:Device close or Esc Key

Comment: Example note 1 omar at 02/05/11 21:49

Details note 1

Reply

Example: Example 2 omar at 02/05/11 21:49

Example 2

Reply

Proposal: Proposal omar at 02/05/11 21:50

I would like to propose that we move the target concept to Function

Reply

New Note

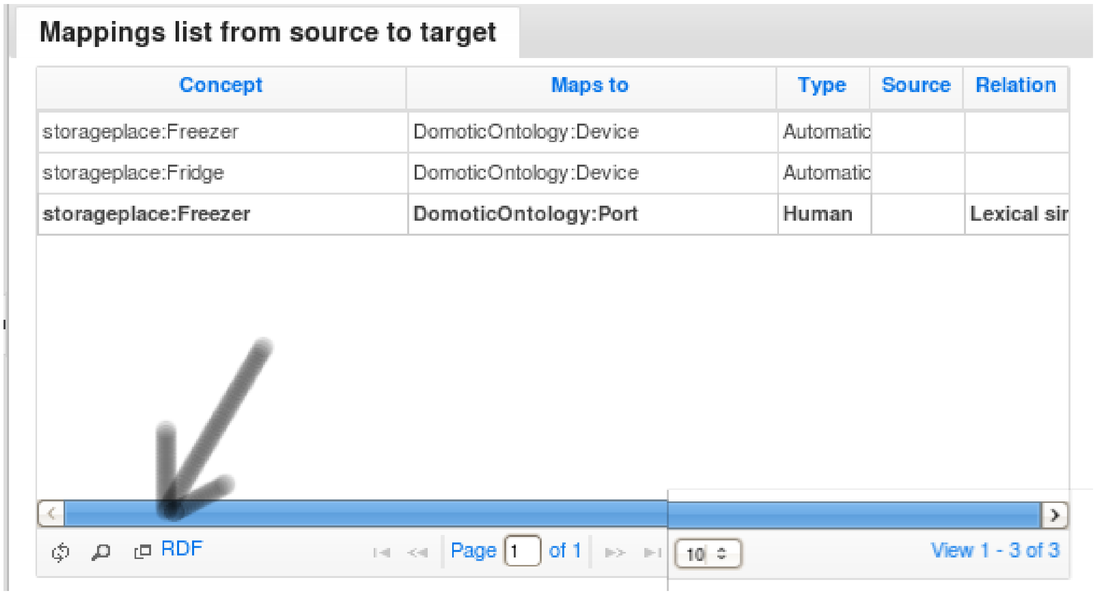
Figure 6.5: Discussions about a mapping

6.6 Visualize the context of a concept

In the concept tab a user could visualize the neighborhood of a concept, being really useful to provide the context of a concept withing the ontology. This can be use to compare the contexts of both source and target ontology concepts selected.

6.7 Export to RDF the mappings between 2 ontologies

Finally these mappings can be exported as RDF-triples and explored and integrated with other tools for employing mappings. See the Appendix B, to see the exact format.



Mappings list from source to target

Concept	Maps to	Type	Source	Relation
storageplace:Freezer	DomoticOntology:Device	Automatic		
storageplace:Fridge	DomoticOntology:Device	Automatic		
storageplace:Freezer	**DomoticOntology:Port**	**Human**		**Lexical sir**

Figure 6.6: Export to RDF the mappings between 2 ontologies

Chapter 7

Conclusions

BioPortal offers investigators 'one-stop shopping' on the Web for important biomedical ontologies. We have adapted this open source tool to our needs, creating ORATE and extending it with Mappingstool. The incorporation of a variety of Web 2.0 features allows the system to behave not only as a comprehensive ontology repository, but also as general infrastructure to support community-based access, peer-review and mapping of ontology content.

We have developed a tool that provides a usable and easy tool to create manual mappings between ontologies, as we gain more experience with mappings in ORATE and as more users start contributing the mappings, we hope that the data that we collect will help us understand the dynamics of ontology mapping as a collaborative and open process. It is interesting to see how curators of a specif domain (Assistive technologies) use the tool and help each other in the process of mapping concepts.

Mappingtool tries to focus in the Bioportal ecosystem, using his mappings and ontologies as the data source. As a future work, a promising task will be to adapt Mappingstool to accept RDF mappings as inputs from Protege or other mapping format. Also, to be able to use different ontology repositories. And the last step for fully integration, would be to be able to change the Bioportal instance to use on the fly, in near future Bioportal will be extended to other domains as we did with Assitive Technologies (ORATE).

Possible features to implement in the future could be a graph visualization of the mappings, automatic mapping and lines between the source and target concepts representing the mappings as it is done in Mapping View (see 3.1.2).

We hope that we have helped researchers in the mapping process, allowing them to know the context of the mapping they try to create and having a better understanding about the whole picture.

Bibliography

[1] Naveen Balani. The future of the web is semantic. 2005. http://www.ibm.com/developerworks/web/library/wa-semweb/.

[2] Tim; James Hendler Berners-Lee and Ora Lassila. *The Semantic Web*. Scientific American Magazine, 2001.

[3] G. Bisso. *Why and how to define a similarity measure for object based representation systems. Towards Very Large Knowledge Bases*. 1995.

[4] Christoph Bussler. *The Semantic Web: research and applications*. Springer, 2004.

[5] Douglas Crockford. *Javascript: the good parts*. O'Reilly Media, Inc, 2008.

[6] T. R. Gruber. *A translation approach to portable ontologies*. Knowledge Acquisition, 1993.

[7] Musen MA Noy NF. Prompt: Algorithm and tool for automated ontology merging and alignment. *AAAI Press*, pages 450–455, 2000.

[8] Jem Rayfield. Bbc world cup 2010 dynamic semantic publishing. 2010. http://bbc.in/d5pM7E.

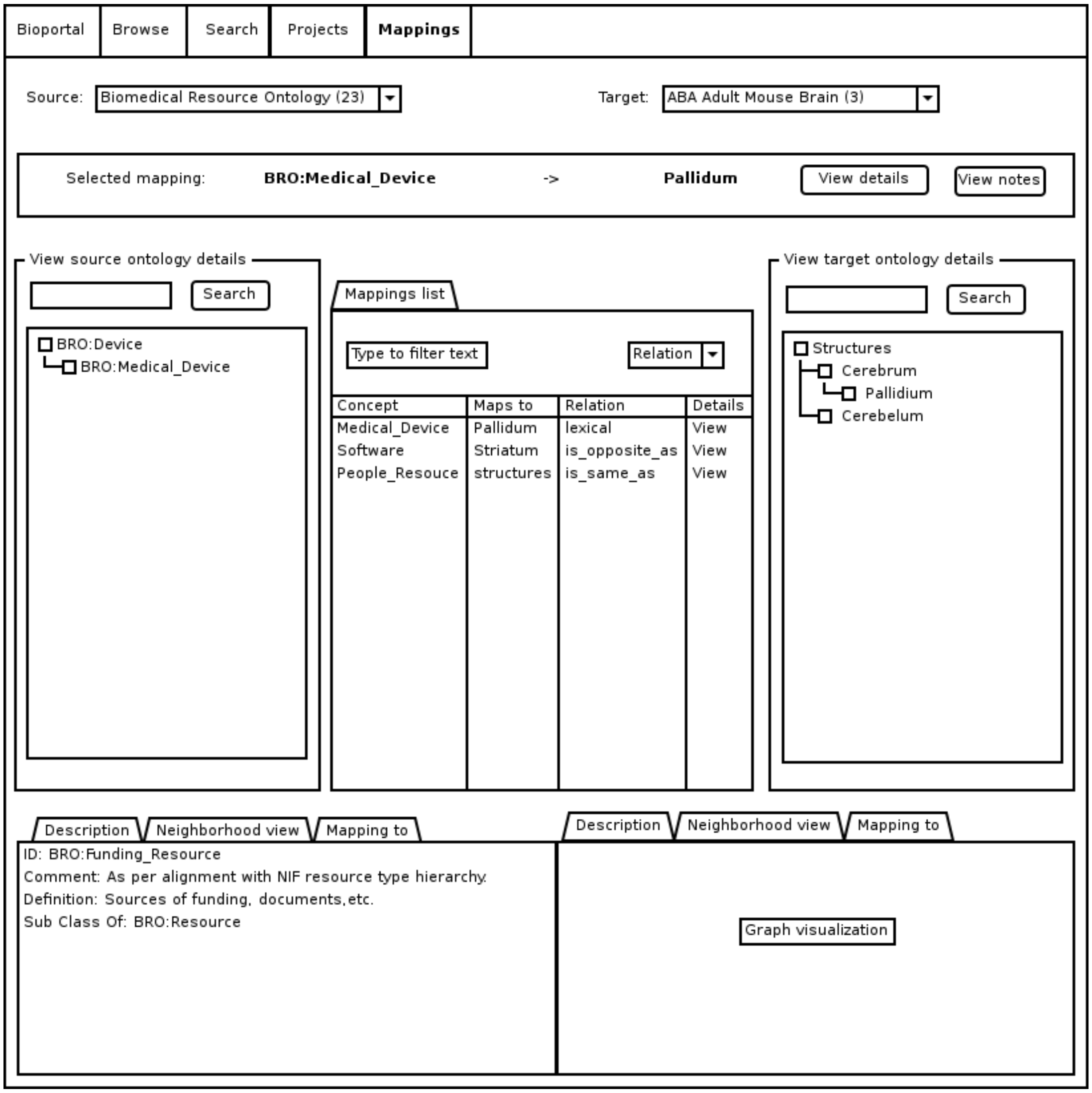
[9] Xiaomeng Su. *A text categorization perspective for ontology mapping*. Norwegian University of Science and Technology, Norway, 2002.

[10] John Bateman; Alexander Castro; Immanuel Normann; Omar Pera; Leyla Garcia; Jose-Maria Villaveces. Oasis common hyper-ontological framework (cof). *OASIS Project report*, 2010.

Appendix A

Mockups

These are the full list of mockups made before developing the application.



Source: Biomedical Resource Ontology (23)

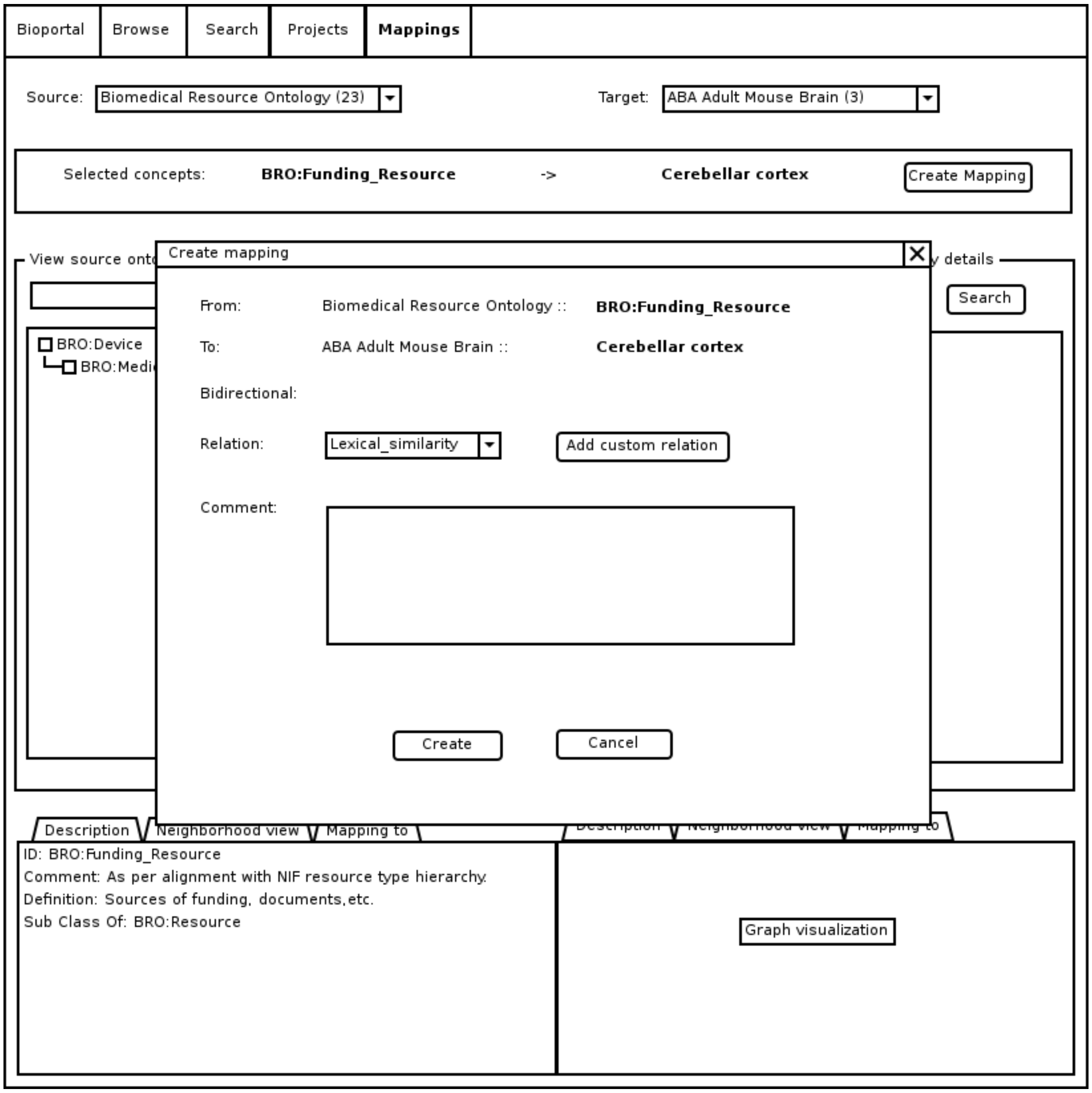
Target: ABA Adult Mouse Brain (3)

Selected mapping: **BRO:Medical_Device** -> **Pallidum**

Concept	Maps to	Relation	Details
Medical_Device	Pallidum	lexical	View
Software	Striatum	is_opposite_as	View
People_Resouce	structures	is_same_as	View

ID: BRO:Funding_Resource
Comment: As per alignment with NIF resource type hierarchy.
Definition: Sources of funding, documents,etc.
Sub Class Of: BRO:Resource

Figure A.1: Main screen



Source: Biomedical Resource Ontology (23)

Target: ABA Adult Mouse Brain (3)

Selected concepts: **BRO:Funding_Resource** -> **Cerebellar cortex**

From: Biomedical Resource Ontology :: **BRO:Funding_Resource**

To: ABA Adult Mouse Brain :: **Cerebellar cortex**

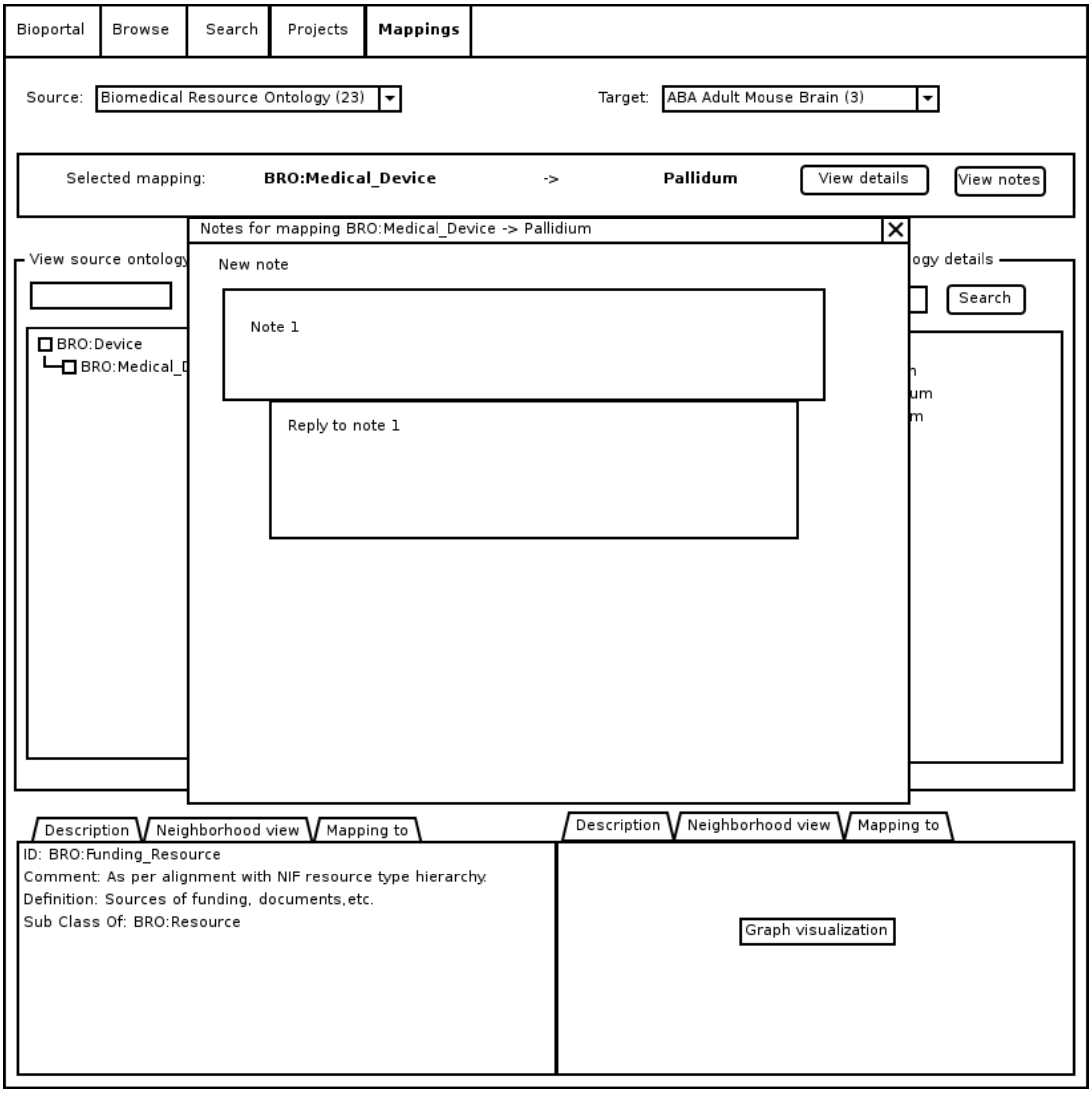
Bidirectional:

Relation: Lexical_similarity

Comment:

ID: BRO:Funding_Resource
Comment: As per alignment with NIF resource type hierarchy.
Definition: Sources of funding, documents,etc.
Sub Class Of: BRO:Resource

Figure A.2: Create a mapping



Source: Biomedical Resource Ontology (23)

Target: ABA Adult Mouse Brain (3)

Selected mapping: **BRO:Medical_Device** -> **Pallidum**

Notes for mapping BRO:Medical_Device -> Pallidium

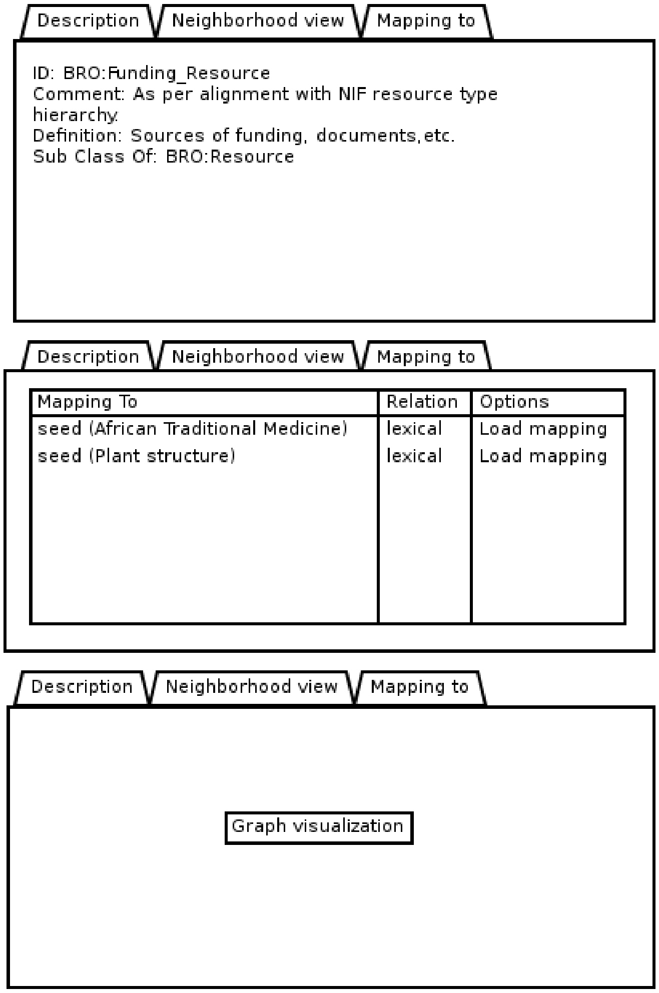
New note

Note 1

Reply to note 1

ID: BRO:Funding_Resource
Comment: As per alignment with NIF resource type hierarchy.
Definition: Sources of funding, documents,etc.
Sub Class Of: BRO:Resource

Figure A.3: Discussion about a mapping



ID: BRO:Funding_Resource
Comment: As per alignment with NIF resource type hierarchy.
Definition: Sources of funding, documents,etc.
Sub Class Of: BRO:Resource

Mapping To	Relation	Options
seed (African Traditional Medicine)	lexical	Load mapping
seed (Plant structure)	lexical	Load mapping

Figure A.4: Concept information

Appendix B

RDF Export

This is a sample of a list of mappings represented in RDF format when you export them from Mappingstool.

```
<?xml version=’1.0’ encoding=’UTF-8’?>


     <!DOCTYPE rdf:RDF [
         <!ENTITY xsd ’http://www.w3.org/2001/XMLSchema#’ >
         <!ENTITY a ’http://protege.stanford.edu/system#’ >
         <!ENTITY rdfs ’http://www.w3.org/2000/01/rdf-schema#’ >
         <!ENTITY mappings ’http://protege.stanford.edu/mappings#’ >
         <!ENTITY rdf ’http://www.w3.org/1999/02/22-rdf-syntax-ns#’ >
     ]>


     <rdf:RDF xmlns="http://bioontology.org/mappings/mappings.rdf#"
          xml:base="http://bioontology.org/mappings/mappings.rdf"
          xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
          xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
          xmlns:mappings="http://protege.stanford.edu/mappings#"
          xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
         <rdf:Property rdf:about="&mappings;author">
             <rdfs:domain rdf:resource="&mappings;Mapping_Metadata"/>
             <rdfs:range rdf:resource="&xsd;string"/>
             <rdfs:label rdf:datatype="&xsd;string">author</rdfs:label>
```

```
</rdf:Property>
<rdf:Property rdf:about="&mappings;comment">
    <rdfs:domain rdf:resource="&mappings;Mapping_Metadata"/>
    <rdfs:range rdf:resource="&xsd;string"/>
     <rdfs:label rdf:datatype="&xsd;string">comment</rdfs:label>
</rdf:Property>
<rdf:Property rdf:about="&mappings;confidence">
    <rdfs:domain rdf:resource="&mappings;Mapping_Metadata"/>
    <rdfs:range rdf:resource="&xsd;string"/>
    <rdfs:label rdf:datatype="&xsd;string">confidence</rdfs:label>
</rdf:Property>
<rdf:Property rdf:about="&mappings;date">
    <rdfs:domain rdf:resource="&mappings;Mapping_Metadata"/>
    <rdfs:range rdf:resource="&xsd;date"/>
    <rdfs:label rdf:datatype="&xsd;string">date</rdfs:label>
</rdf:Property>
<rdfs:Class rdf:about="&mappings;Mapping_Metadata">
    <rdfs:label rdf:datatype="&xsd;string"
        >Mapping_Metadata</rdfs:label>
</rdfs:Class>
<rdf:Property rdf:about="&mappings;mapping_metadata">
    <rdfs:domain rdf:resource="&mappings;One_to_one_mapping"/>
    <rdfs:range rdf:resource="&mappings;Mapping_Metadata"/>
     <rdfs:label rdf:datatype="&xsd;string"
        >mapping_metadata</rdfs:label>
</rdf:Property>
<rdf:Property rdf:about="&mappings;mapping_source">
    <rdfs:domain rdf:resource="&mappings;Mapping_Metadata"/>
    <rdfs:range rdf:resource="&xsd;string"/>
     <rdfs:label rdf:datatype="&xsd;string">authority</rdfs:label>
</rdf:Property>
<rdfs:Class rdf:about="&mappings;One_to_one_mapping">
    <rdfs:label rdf:datatype="&xsd;string"
        >One_to_one_mapping</rdfs:label>
</rdfs:Class>
<rdf:Property rdf:about="&mappings;relation">
    <rdfs:domain rdf:resource="&mappings;One_to_one_mapping"/>
    <rdfs:range rdf:resource="&xsd;string"/>
     <rdfs:label rdf:datatype="&xsd;string">relation</rdfs:label>
</rdf:Property>
```

```
<rdf:Property rdf:about="&mappings;source">
    <rdfs:domain rdf:resource="&mappings;One_to_one_mapping"/>
    <rdfs:range rdf:resource="&xsd;string"/>
    <rdfs:label rdf:datatype="&xsd;string">source</rdfs:label>
</rdf:Property>
<rdf:Property rdf:about="&mappings;target">
    <rdfs:domain rdf:resource="&mappings;One_to_one_mapping"/>
    <rdfs:range rdf:resource="&xsd;string"/>
    <rdfs:label rdf:datatype="&xsd;string">target</rdfs:label>
</rdf:Property><mappings:One_to_one_mapping rdf:ID="1">
      <mappings:mapping_metadata rdf:resource="#2"/>
      <mappings:relation rdf:datatype="&xsd;string"></mappings:relation>
      <mappings:source
       rdf:resource='http://ontologies.informatik.uni-bremen.de/
       virtual/9/storageplace:Freezer'/>
      <mappings:target
       rdf:resource='http://ontologies.informatik.uni-bremen.de/
       virtual/6/DomoticOntology:Device'/>
  </mappings:One_to_one_mapping>
  <mappings:Mapping_Metadata rdf:ID="2">
      <mappings:author rdf:datatype="&xsd;string">omar
      </mappings:author>
      <mappings:mapping_source rdf:datatype="&xsd;string">
      </mappings:mapping_source>
      <mappings:comment rdf:datatype="&xsd;string"><
      /mappings:comment>
      <mappings:date rdf:datatype="&xsd;date">
      Mon Jan 18 22:18:48 +0100 2010</mappings:date>
  </mappings:Mapping_Metadata><mappings:One_to_one_mapping rdf:ID="3">
      <mappings:mapping_metadata rdf:resource="#4"/>
      <mappings:relation rdf:datatype="&xsd;string"></mappings:relation>
      <mappings:source
      rdf:resource='http://ontologies.informatik.uni-bremen.de/
      virtual/9/storageplace:Fridge'/>
      <mappings:target
      rdf:resource='http://ontologies.informatik.uni-bremen.de/
      virtual/6/DomoticOntology:Device'/>
  </mappings:One_to_one_mapping>
  <mappings:Mapping_Metadata rdf:ID="4">
      <mappings:author rdf:datatype="&xsd;string">
    omar</mappings:author>
    <mappings:mapping_source rdf:datatype="&xsd;string">
    </mappings:mapping_source>
    <mappings:comment rdf:datatype="&xsd;string">
    </mappings:comment>
    <mappings:date rdf:datatype="&xsd;date">
    Mon Jan 18 22:18:49 +0100 2010</mappings:date>
</mappings:Mapping_Metadata><mappings:One_to_one_mapping rdf:ID="5">
    <mappings:mapping_metadata rdf:resource="#6"/>
    <mappings:relation rdf:datatype="&xsd;string">
    Lexical similarity</mappings:relation>
    <mappings:source
    rdf:resource=’http://ontologies.informatik.uni-bremen.de/
    virtual/9/storageplace:Freezer’/>
    <mappings:target
    rdf:resource=’http://ontologies.informatik.uni-bremen.de/
    virtual/6/DomoticOntology:Port’/>
</mappings:One_to_one_mapping>
<mappings:Mapping_Metadata rdf:ID="6">
    <mappings:author rdf:datatype="&xsd;string">
    omar</mappings:author>
    <mappings:mapping_source rdf:datatype="&xsd;string">
    </mappings:mapping_source>
    <mappings:comment rdf:datatype="&xsd;string"></mappings:comment>
    <mappings:date rdf:datatype="&xsd;date">
    Tue Feb 23 19:55:27 +0100 2010</mappings:date>
</mappings:Mapping_Metadata></rdf:RDF>
```

Appendix C

Deploying Bioportal Core

The following instructions explains how to build and deploy Bioportal Core.

Building and Deploying BioPortal REST Services (Core)

Table of Contents

System Requirements

1. The Java Development Kit (version 1.5 or higher).
2. Apache Ant (see http://ant.apache.org/ for the latest release).
3. Tomcat Application server (version 5.5 or higher). Although has not been tested, the system should be able to perform equally well on other application servers, such as JBoss, with minimal configuration changes.
4. MySQL 5.0 Database Server.
5. The latest MySQL Connector/J driver (see http://dev.mysql.com/downloads/connector/j/ for the latest release).

Source Structure

build.properties.sample	A template file that contains all build properties with corresponding sample values
build.xml	Ant build file
build	Auto-generated folder that contains BioPortal as it is deployed under an application server
classic_to_core_migration	The folder containing source code, database scripts, and instructions on migrating existing data from BioPortal 1.0
db	The folder containing all relevant database scripts and dumps
dist	The folder containing the generated BioPortal Web Application Archive (WAR) file
src	BioPortal source code
tmpl	The folder containing templates used for generating runtime configuration files and message repository
WebRoot	The web application root folder
WebRoot/WEB-INF/conf/generated	The folder containing runtime configuration files, auto-generated

	using the templates in “tmpl” folder
WebRoot/WEB-INF/lib	The folder containing dependent libraries
WebRoot/WEB-INF/resources	The folder containing resource files, such as stylesheets. Currently, it also houses the auto-generated LexGrid runtime configuration file (config.props) due to a specific LexGrid folder structure requirement

To get the source code from Subversion and successfully use the Ant build file, you must issue the following command:

svn checkout https://bmir-gforge.stanford.edu/svn/bioportal_core

General Installation and Configuration

1. Install the Java Development Kit (version 1.5 or higher). Make sure to set the JAVA_HOME environment variable to the installation directory of your JDK.

2. Install Apache Ant. See Apache's web site to download Ant.

3. Install Tomcat application server. Alternatively, install and configure your own preferred application server.

4. Copy the MySQL Connector/J driver to <your Tomcat installation dir>/common/lib folder. Alternatively, ensure that the Connector/J driver is located in your application server’s CLASSPATH environment.

5. Install and configure MySQL 5 database.

    A. Create three separate databases:

    - bioportal – used for storing ontology metadata, version information, and user data.

    - bioportal_lexgrid – used for storing LexGrid database backend.

    - bioportal_protege – used for storing Protégé database backend.

      The database names are arbitrary, provided they are correctly reference in the build.properties file.

    B. Run the following SQL script: <project root>/db/sql/bioportal_db.sql

C. Run the following SQL script: <project root>/db/sql/bioportal_lookup_data.sql

Ignore the other database scripts. Their artifacts are contained in bioportal_db.sql.

D. Create three dedicated database users, one for each database above:

- bioportal_user – used for accessing bioportal database. Grant this user SELECT/CREATE/INSERT/UPDATE/DELETE rights to bioportal database.

- bioportal_lexgrid_user – used for accessing bioportal_lexgrid database. Grant this user SELECT/CREATE/INSERT/UPDATE/DELETE rights to bioportal_lexgrid database.

- bioportal_protege_user – used for accessing bioportal_protege database. Grant this user SELECT/CREATE/INSERT/UPDATE/DELETE rights to bioportal_protege database.

Installation and Configuration on Ubuntu

These are detailed steps to install and configure the environment required. This is a basic tutorial, for production environment is not recommended. It has been tested on Ubuntu 9.04.

1. Install the Java Development Kit (version 1.5 or higher).

```
$ sudo apt-get install sun-java6-jdk

$ java -version
```

The JVM should provided by Sun NOT gij anymore

2. Install Apache Ant. See Apache's web site to download Ant. Current release is Apache Ant 1.7.1

```
$ tar -xvzf ~/archive/a/apache-ant-1.7.1-bin.tar.gz -C ~/programs

$ cd programs

$ ln -s apache-ant-1.7.1 ant
```

Edit ~/.bashrc and set the ANT_HOME environment variable to the directory where you installed Ant.

```
ANT_HOME=/home/omar/programs/ant

PATH=$PATH:$ANT_HOME/bin
```

```
export PATH
```

Check the basic installation with opening a new shell and typing *ant*. You should get a message like this:

```
Buildfile: build.xml does not exist!

Build failed
```

3. Install Tomcat application server (version 5.5 or higher).

```
$ sudo aptitude install tomcat5 tomcat5-webapps
```

Usually config files are located on */etc/tomcat5.5* , and in *<your Tomcat installation dir>/bin* scripts to startup, restart the server

Tomcat needs JAVA_HOME variable in the path, we can either put it into our environment:

```
    # nano ~/.bashrc

                    JAVA_HOME=/usr/lib/jvm/java-6-sun/

                    JAVA_BIN=$JAVA_HOME/bin

                    PATH=$PATH:$JAVA_HOME:$JAVA_BIN

                    export PATH
```

or in the tomcat init scripts:

```
    # nano ~ <your Tomcat installation dir>/bin/catalina.sh

                    JAVA_HOME=/usr/lib/jvm/java-6-sun/
```

Tomcat is started using a security manager, you can define the permissions for your servlets and JSPs in /etc/tomcat5/policy.d/*. All files in this directory are joined to /etc/tomcat5.5/catalina.policy at startup. If your webapp does not work with the tomcat5.5 Ubuntu/Debian package but works fine with the binary distribution from Jakarta, try to disable the security manager in /etc/default/tomcat5.5 first (TOMCAT5_SECURITY=no). If this works, add the required permissions in a new file in /etc/tomcat5.5/policy.d/ restart and re-enable the security manager. Disabling the security manager is not recommended on production systems since a call to System.exit() in a servlet of JSP page would then stop the whole virtual machine that is running Tomcat.

In order to change the port where tomcat listen:

```
$ sudo <editor_you_want> <your Tomcat installation dir>/conf/server.xml

        # Modify the port int the connector tag:

        <Connector port="8080" maxHttpHeaderSize="8192"

                ....
```

4. Install MySQL 5.0 Database Server.

```
$ sudo aptitude install mysql-server-5.0
```

5. Get the latest MySQL Connector/J driver (see http://dev.mysql.com/downloads/connector/j/ for the latest release).

6. Copy the MySQL Connector/J driver to <your Tomcat installation dir>/common/lib folder. Alternatively, ensure that the Connector/J driver is located in your application server’s CLASSPATH environment.

7. Configure MySQL 5 database.

   A. Set password for a ROOT account on the MySQL5 Server

   ```
       $ mysqladmin -u root password your-new-password

       $ sudo /etc/init.d/mysql restart
   ```

   B. Create three separate databases:

      ❖ bioportal – used for storing ontology metadata, version information, and user data.

      ❖ bioportal_lexgrid – used for storing LexGrid database backend.

      ❖ bioportal_protege – used for storing Protégé database backend.

   The database names are arbitrary, provided they are correctly reference in the build.properties file.

   C. Run the following SQL script: <project root>/db/sql/bioportal_db.sql

   ```
   mysql> SOURCE [<project root>/db/sql/bioportal_db.sql];

   mysql> USE [database_name_bioportal];

   mysql> SHOW tables;   // Just to be sure tables has been created
   ```

   D. Run the following SQL script:

   ```
   mysql> SOURCE [<project root>/db/sql/bioportal_lookup_data.sql];
   ```

E. Create three dedicated database users, one for each database above:

- bioportal_user – used for accessing bioportal database. Grant this user SELECT/CREATE/INSERT/UPDATE/DELETE rights to bioportal database.

- bioportal_lexgrid_user – used for accessing bioportal_lexgrid database. Grant this user SELECT/CREATE/INSERT/UPDATE/DELETE rights to bioportal_lexgrid database.

- bioportal_protege_user – used for accessing bioportal_protege database. Grant this user SELECT/CREATE/INSERT/UPDATE/DELETE rights to bioportal_protege database.

```
mysql> CREATE USER <user> IDENTIFIED BY PASSWORD <password>;

mysql> GRANT SELECT,CREATE,INSERT,UPDATE,DELETE ON <database_name>.*
    TO '<user>'@'localhost';
```

Build and Deploy BioPortal REST Services

1. Rename “build.properties.sample” to “build.properties”. The “.sample” extension has been added so that each time the source code is downloaded, the build.properties file is not overwritten (which causes user defined settings to be lost).

2. In the file build.properties there is a property called *lexgrid.single.db.mode*, check what it means and then, if you decide to set it to true be sure to give all privilegies to bioportal_lexgrid user related to bioportal_lexgrid database.

3. Modify build.properties to reflect your environment settings. Below are the most commonly set properties:

bioportal.environment	An environment within which this installation is run. Possible values: [dev, prod]. Setting the value to “dev” will enable the use of local datasource, which is convenient for running JUnit tests and other development tasks. Setting the value to “prod” will enable a container-based datasource, which will use connection pooling, maximum active

	connections, and other configurable settings (see tmpl/context.xml.tmpl for complete list)
bioportal.resource.path	Location of resource files on the server, such as ontology files, lucene indices, and other filesystem artifacts required at runtime
appserver.home	The root directory of your application server
obo.pull.scheduler.enabled	A flag that enables/disables a scheduler job that pulls ontologies from OBO Sourceforge repository. Possible values: [true/false].
obo.pull.scheduler.cronexpression	A cron expression that defines the time and frequency of the OBO Sourceforge scheduler runs (provided the obo.pull.scheduler.enabled property is set to “true”). Uses standard cron expressions (see http://quartz.sourceforge.net/javadoc/org/quartz/CronTrigger.html for examples)
ontology.parse.scheduler.enabled	A flag that enables/disables a scheduler job that parses ontologies to enable searching/visualization. Possible values: [true/false].
ontology.parse.scheduler.cronexpression	A cron expression that defines the time and frequency of the ontology parse scheduler runs (provided the ontology.parse.scheduler.enabled property is set to “true”). Uses standard cron expressions (see http://quartz.sourceforge.net/javadoc/org/quartz/CronTrigger.html for examples)
bioportal.smtp.server	The SMTP server to be used by the application
bioportal.datasource.name	The name of the BioPortal datasource (used only if the bioportal.environment flag is set to “true”)
bioportal.jdbc.url	JDBC url of the “bioportal” database (see 5A above)

bioportal.jdbc.username	Username of the dedicated db user for accessing “bioportal” database (see 5D above)
bioportal.jdbc.password	Password of the dedicated db user for accessing “bioportal” database (see 5D above)
bioportal.encryption.key	Encryption key used for encrypting user passwords
protege.jdbc.url	JDBC url of the “bioportal_protege” database (see 5A above)
protege.jdbc.username	Username of the dedicated db user for accessing “bioportal_protege” database (see 5D above)
protege.jdbc.password	Password of the dedicated db user for accessing “bioportal_protege” database (see 5D above)
lexgrid.db.url	JDBC url of the “bioportal_lexgrid” database (see 5A above)
lexgrid.db.user	Username of the dedicated db user for accessing “bioportal_lexgrid” database (see 5D above)
lexgrid.db.password	Password of the dedicated db user for accessing “bioportal_lexgrid” database (see 5D above)
lexgrid.email.to	An email address for sending LexGrid errors. Only applicable if lexgrid.email.errors property is set to “true”

4. Execute the following ant commands to build and deploy BioPortal REST services:

    a. ant clean

    b. ant deploywar

Accessing BioPortal REST Services

1. Start application server.

2. Access BioPortal RESTful URLs. (See http://www.bioontology.org/wiki/images/7/71/BioPortal2.0_User_Guide_v2.doc for details).

Appendix D

Deploying Bioportal UI

The detailed instructions about how to deploy Bioportal UI with Mappingstool.

Deploying BioPortal Ruby On Rails GUI

Table of Contents

System Requirements

1. Ruby Programming Language (1.8.5 or greater)
2. RubyGems
3. Memcache
4. Apache
5. MySQL 5.0 Database Server.

Source Structure

/config/database.yml	File that contains database connection properties
/config/mongrel_cluster.yml	File for configuring the mongrel server cluster
/config/environment.rb	File containing environment variables such as mail server
/config/environments/production.rb, /config/environments/development.rb	Files that contain environment variables that are specific to a environment (e.g. memcache settings)
/app	The folder containing all the core logic to the ruby on rails application
/db	The folder containing database migrations
/lib	The folder containing libraries used for talking to services
/public	The folder containing public files such as stylesheets and images
/vendor	The folder containing ruby plugins

To get the source code from, you must issue the following command:

```
svn checkout https://bmir-gforge.stanford.edu/svn/bioportalui
```

Installation and Configuration

1. Install the Ruby Core Library (version 1.8.5 or higher

2. Install Ruby Gems

    A. actionmailer (2.1.0, 2.0.2)

    B. actionpack (2.1.0, 2.0.2)

    C. activerecord (2.1.0, 2.0.2)

    D. activeresource (2.1.0, 2.0.2)

    E. activesupport (2.1.0, 2.0.2)

    F. capistrano (2.4.3)

    G. cgi_multipart_eof_fix (2.5.0)

    H. daemons (1.0.10)

    I. fastthread (1.0.1)

    J. gem_plugin (0.2.3)

    K. highline (1.4.0)

    L. hoe (1.7.0)

    M. httpclient (2.1.2)

    N. memcache-client (1.5.0)

    O. mongrel (1.1.5)

    P. mongrel_cluster (1.0.5)

    Q. mysql (2.7)

    R. net-scp (1.0.1)

    S. net-sftp (2.0.1)

    T. net-ssh (2.0.3)

    U. net-ssh-gateway (1.0.0)

    V. rails (2.1.0, 2.0.2)

    W. rake (0.8.1)

    X. rubyforge (1.0.0)

    Y. soap4r (1.5.8)

3. Install Apache Web Server

4. Install Memcache

5. Install and configure MySQL 5 database.

    A. Create a database to contain metadata

    B. Point the config/database.yml file to the newly created database.

    C. Run the following command in the main ruby code directory: rake db:migrate

Installation and Configuration on Ubuntu

These are detailed steps to install and configure the environment required. It has been tested on Ubuntu 9.04.

1. To install the Ruby base packages, you can enter the following command in the terminal prompt:

```
$ sudo aptitude install ruby rdoc ri irb libopenssl-ruby ruby-dev

ruby = An interpreter of object-oriented scripting language Ruby

ri = Ruby Interactive reference

rdoc = Generate documentation from ruby source files

irb = Interactive Ruby
```

At the moment of this tutorial Ruby version: ruby 1.8.7 (2008-08-11 patchlevel 72) [i486-linux]

2. Test the correct instalation:

```
 $ ruby -ropenssl -rzlib -rreadline -e "puts :Hello"

     >Hello
```

3. Next we need to install the Ruby gem package manager. You can download the latest Ruby gems from http://rubyforge.org/projects/rubygems/

```
    $ cd ~

    $ wget http://rubyforge.org/frs/download.php/45905/rubygems-1.3.2.tgz

    $ tar xzvf rubygems-1.3.2.tgz
    $ cd rubygems-1.3.2

    $ sudo ruby setup.rb

    $ sudo ln -s /usr/bin/gem1.8 /usr/bin/gem
```

Once it’s done you can delete the .tgz file and rubygems directory.

4. (Optional) We can to create a set of simlinks, otherwise it will be a tedious task to type commands with the version (1.8).

```
$ sudo ln -s /usr/bin/gem1.8 /usr/local/bin/gem

$ sudo ln -s /usr/bin/ruby1.8 /usr/local/bin/ruby

$ sudo ln -s /usr/bin/rdoc1.8 /usr/local/bin/rdoc

$ sudo ln -s /usr/bin/ri1.8 /usr/local/bin/ri

$ sudo ln -s /usr/bin/irb1.8 /usr/local/bin/irb
```

5. Install Rails via RubyGems and the gems mentioned on the previous section.

```
    $ sudo gem install rails
```

6. Install MySQL 5.0 Database.

By default, Rails includes the sqlite3 gem and is automatically configured to use it. To use MySQL instead, follow the following steps:

```
$ sudo apt-get install mysql-server-5.0 mysql-client-5.0

$ sudo apt-get install libmysql-ruby libmysqlclient-dev

$ sudo gem install mysql
```

Rails assumes that the MySQL socket file will be found in /tmp/mysqld.sock. In Debian/Ubuntu, this is not the case. Be sure to change database.yml to reflect the actual location of the socket file. (it will be located on /var/run/mysqld/mysqld.sock or /var/lib/mysql/mysql.sock

7. To build the necessary databases and tables (development, test and production configured on config/database.yml):

```
$ cd <path_rails_app>

$ rake db:create

$ rake db:migrate
```

8. Installing Memcache system

1. Introduction

Bioportal UI uses MemCached, a distributed memory object caching system intended for use in speeding up dynamic web applications by alleviating database load. (http://www.danga.com/memcached/)

MemCached is the server library which runs the caching daemon, a client library that interact with the server is required.

Bioportal has choose Ruby MemCache client (http://www.deveiate.org/projects/RMemCache/)

2. Memcached installation

Memcached is packaged for Ubuntu Jaunty, so:

```
    $ sudo aptitude install memcached
```

We then started a memcached instance:

```
    $ memcached -d -l 127.0.0.1 -p 11299 -m 1024

  * -d = run as a daemon

  * -l <ip address> = bind to ip address

  * -p <port> = run on this port

  * -m <num> = use num Mb of memory for the store

  * -P = where to put the pid file

    * -vv = extra verbose and doesn't daemonize
```

3. Ruby memcached binding installation: Ruby-Memcache

```
# sudo gem install Ruby-MemCache
```

It install an old version (currently 0.0.1), in order to install the last version download from http://www.deveiate.org/projects/RMemCache/ the last gem.

At this point we suggest to make all the changes in order to deploy Bioportal correctly (Memcache options, database, Bioportal REST URL, etc.) before the next step. Read the next section and returns to this point.

Check that everything works OK with the built-in server of Ruby (WEBrick)

```
    $ ./script/server
```

9. Installing the production server

There are now a huge number of options available for deploying a Rails application in a production environment: Apache + FastCGI, Lighttpd + FastCGI, SCGI, Apache 2, Litespeed, Mongrel, etc.

We have decided to use Mongrel + Mongrel Cluster + Apache 2.

1. Installing Mongrel + Mongrel Cluster

```
$ sudo gem install mongrel –include-dependencies
```

In order to check everything is OK

```
$ cd <path_rails_app>

$ mongrel_rails start

// Or just run ./script/server and check is booting with Mongrel
```

We will be using a cluster of three mongrel processes, you need to decide how many will be suitable for you. To run a cluster of mongrel processes we need to install the mongrel_cluster gem, for details on this gem see the Mongrel web site.

```
$ sudo gem install mongrel_cluster
```

2. Testing Mongrel

Go to your Rails application directory and execute:

```
$ mongrel_rails cluster::configure -p 18000 -N 3
```

This allows mongrel to spawn 3 instance to handle the load. Each instance will open on port 18000, and succeeding. So that is 18000, 18001, and 18002.

Check cluster:configure help to get more information about the options.

This will create the a mongrel_cluster configuration file in *<path_rails_app>/config/mongrel_cluster.yml*.

To test everything is OK, execute:

```
$ mongrel_rails cluster::start
```

Now open up http://localhost:18000/, http://localhost:18001/, and http://localhost:18002/ to ensure that your site is working perfectly fine.

3. Installing Apache 2

```
$ sudo apt-get install apache2
$ sudo a2enmod proxy_balancer
$ sudo a2enmod proxy_http
$ sudo a2enmod rewrite
```

4. Configure Apache and the Balacing Proxy

You have X mongrel servers running, ready to handle incoming requests. But, you want your visitors to use ‘myapp.com’ or just an IP address and not an IP address with different port numbers. This is where Apache comes in.

Create a new file in /etc/apache2/sites-available named 'bioportal' and add the following:

```
<proxy balancer://mongrel_cluster>
  BalancerMember http://127.0.0.1:18000
  BalancerMember http://127.0.0.1:18001
  BalancerMember http://127.0.0.1:18002
</proxy>
Listen 10000
# Set the port you want to listen the requests for your application
<VirtualHost *:10000>
  #ServerName myapp.com
  #ServerAlias www.myapp.com
  DocumentRoot "path_to_public_dir"
  <directory "path_to_public_dir">
    Options FollowSymLinks
    AllowOverride None
    Order allow,deny
    Allow from all
  </directory>
 RewriteEngine On
 RewriteCond %{DOCUMENT_ROOT}/system/maintenance.html -f
 RewriteCond %{SCRIPT_FILENAME} !maintenance.html
 RewriteRule ^.*$ /system/maintenance.html [L]

 RewriteRule ^/$ /index.html [QSA]

 RewriteRule ^([^.]+)$ $1.html [QSA]

 RewriteCond %{DOCUMENT_ROOT}/%{REQUEST_FILENAME} !-f

 RewriteRule ^/(.*)$ balancer://mongrel_cluster%{REQUEST_URI} [P,QSA,L]

 #ErrorLog /path/log/errors_log

#CustomLog /path/log/custom_log combined

</virtualhost>



Listen 9999

<VirtualHost *:9999>

  <Location />

    SetHandler balancer-manager

    Deny from all

    Allow from localhost

  </Location>

</VirtualHost>
```

The first VirtualHost (port 10000) is our rails application, the second one enables a front end for load balancing, available only to localhost on port 9999.

Lets enable the virtual host:

```
$ sudo a2ensite myapp

$ sudo /etc/init.d/apache2 restart
```

Note 1: If you can’t access your site, its because the default virtual host has a higher precedence. Execute sudo a2dissite default. Similarly ensure that you are access your site via the domain you specified in the ServerName directive.

Note 2: In some cases you may need to make a small change to /etc/apache2/mods-enabled/proxy.conf and swap

```
Order deny,allow
```

```
Deny from all
```

for

```
Order allow,deny

Allow from all
```

5. Start Mongrel Cluster on Boot

Create a *mongrel_cluster* conf directory in /etc/mongrel_cluster.

In the Mongrel Cluster gem, there is an init.d script you need to copy.

```
$ sudo cp /usr/lib/ruby/gems/1.8/gems/mongrel_cluster-
    0.2.1/resources/mongrel_cluster/mongrel_cluster /etc/init.d
```

Now we need to add the init.d script to startup:

```
$ sudo chmod +x /etc/init.d/mongrel_cluster

$ sudo update-rc.d mongrel_cluster defaults

$ sudo ln -s <path_rails_app>/config/mongrel_cluster.yml /etc/mongrel_cluster/railsapp
```

Deploy BioPortal GUI

1. Configure the mongrel_cluster.yml to contain the amount of servers you want to run and the port you want them to run on (default is 5)

2. Modify the /config/environments/production.rb file to point to the correct location of your memcache server.

3. Modify the /config/environment.rb file to point to the correct SMTP server.

4. Configure apache to point to the mongrel services

Detailed deployment of BioPortal GUI

1. *config/database.yml* → Set your database settings

2. config/environment.rb

   A. Comment RAILS_VERSION in order to use the last Rails version installed on your machine.

   B. Set REST_URL var with the Bioportal core URL

3. app/views/ontologies/show.html.erb

    A. Change the FlexViz tool path to the local one (/flex/FlexoViz.swf)

    B. Change Bioportal URL webservice → <%=$REST_URL%>

    C. Change Bioportal URL webservice in diff row.

4. *app/views/concepts/images.html.erb* → check path to FlexViz

5. *app/views/search/index.html.erb* → choose between built-in or Flex search:

    A. Flex search:

        i. Change path to Bioportal REST webservice and the current server holding Bioportal UI in *app/views/search/index.rhtml, app/views/search/results.rhtml*

        ii. This search is hosted on a remote server, so we must create a crossdomain.xml in the root directory of our Bioportal webservice server. Put the following lines in <tomcat_dir>/webapps/ROOT/cross-domain.xml:

```
          <?xml version="1.0" encoding="UTF-8"?>

          <!DOCTYPE cross-domain-policy SYSTEM
    "http://www.macromedia.com/xml/dtds/cross-domain-policy.dtd">

          <cross-domain-policy>

                  <site-control permitted-cross-domain-policies="all"/>

                  <allow-access-from domain="*" secure="false"/>

                  <allow-http-request-headers-from domain="*" headers="*"
    secure="false"/>

          </cross-domain-policy>
```

    B. Built-in search:

        i. Descomment '@ontologies = DataAccess.getActiveOntologies()' in *app/controllers/search_controller.rb* in the action 'index'.

6. Set your own server and port specified in MemCached daemon in *config/environments/*.rb*, also note that we must include the class before the creation of the MemCache object:

```
--> require 'memcache'

memcache_options = {

  :c_threshold => 10_000,
  :compression => true,
  :debug => true,
  :namespace => 'BioPortal',
  :readonly => false,
  :urlencode => false
}
CACHE = MemCache.new memcache_options
CACHE.servers = '<server>:<port>'
```

7. *lib/BioportalRestfulCore.rb* → it is hard-coded the port of Bioportal REST service URL, so in case is not 80, change it in '*`def self.postMultiPart(url,paramsHash)`*'

Accessing BioPortal GUI

1. Start Memcache daemon.
2. Start the mongrel cluster
3. Access BioPortal UI.