Getting Started with OntoViews Toolkit - Brief Tutorial for Creating a Semantic Portal

Teemu Sidoroff (tsido@cc.hut.fi)

March 8, 2005

0 Crash Course

Very brief installation instructions for the impatient. For more detailed instructions on using the OntoViews-toolkit read the rest of this document.

1. Download `ontodella.zip` and `ontoviews.zip` from `http://www.cs.helsinki.fi/group/seco/museums/dist/`.
2. Unzip `ontodella.zip`.
3. Install SWI-Prolog with HTTP-support, available from `http://www.swi-prolog.org`.
4. Create and edit `$HOME/.ontodella`.
5. Configure `<ontodella>/conf/prolog_config.pl`.
6. Define category rules and create the categories (`<ontodella>/bin/run-cache-categories.sh`).
7. Define recommendation rules.
8. Start Ontodella (`<ontodella>/bin/run-onto.sh`).
9. Unzip `ontoviews.zip`.
10. Configure `<semcocoon>/webapp/fms/sitemap.xmap`.
11. Build OntoViews (`ant <semcocoon>/build.xml`).
12. start the portal (`<semcocoon>/build/semcocoon/cocoon/cocoon.sh`)
13. Modify the XSLT stylesheets to fit your needs (located at `<semcocoon>/webapp/fmsweb/transformation/`).

1 Introduction

This document gives an introduction to the Ontoviews-toolkit. The toolkit supplies a framework for building a semantic portal. As the toolkit is created mainly as a research tool, its ease of use is not of commercial standard. Therefore this document tries to help the user to avoid the most common pitfalls of deployment of the toolkit. This document also serves as a brief tutorial in creating a new semantic portal using arbitrary content material.

Brief overview of the functionality and interoperability of separate components of the system is described in section 2. Section 3 describes the installation of the components. The rest of the document describes the deployment of different components of the toolkit. Sections 4 and 5 deal with the usage of Ontodella, the semantic recommendation engine, and section 6 describes the architecture user interface of the OntoViews system.

The software described in this document can be downloaded from the software distribution page of Semantic Computing Research Group[1]. Two packages from the page are required for creating a semantic portal: the Ontodella package (`ontodella.zip`) and OntoViews package (`ontoviews.zip`).

2 The Components of the OntoViews-toolkit

OntoViews consists of three major components: Ontodella, Ontogator and the graphical user interface (GUI). Ontodella acts as the recommendation engine that provides the links recommended in the GUI (eg. in MuseoSuomi[2] the links served at the left side of the window). Ontodella is also used to generate the categories that are given to Ontogator, the search engine of the OntoViews system. The GUI is based on the open source web publishing platform, Apache Cocoon[3]. Overview of the architechture is given in [1].

The different components of Ontoviews are implemented in myriad of different programming languages. Ontodella is created on top of SWI-Prolog[4], Ontogator is written in Java, and the GUI is generated by a series of XSLT-transformations. However, just basic knowledge of XSLT and Prolog is sufficient in order to generate a functioning portal with on OntoViews. Knowledge of Java is not required, unless more complex modifications of the system are needed (cf. section 6).

Ontodella is separate component from the rest of the OntoViews system,

[1] http://www.cs.helsinki.fi/group/seco/museums/dist/
[2] http://museosuomi.cs.helsinki.fi
[3] http://cocoon.apache.org/
[4] http://www.swi-prolog.org

namely Ontogator and the GUI components. The component has dual functionality. First, it is used to generate the categories that are used in the faceted search interface of the to-be-created portal. Secondly, Ontodella provides the recommendations of information items that are displayed in the user interface. The category generation is done offline, prior to the initialization of the portal. The recommendations, however, are generated on the fly.

Ontogator is tightly embedded into the Cocoon-side of the OntoViews and it requires virtually no attention or modification when creating a new portal. Ontogator is included in the OntoViews distribution package, that also includes the Cocoon-based portal platform.

3 Setting up the Components

The OntoViews-toolkit consists of two separate packages that need to be installed, Ontodella and the OntoViews package. These packages can be installed in separate systems.

3.1 Ontodella Installation and Configuration

Ontodella requires SWI-prolog, version 5.2.11 or higher, to function. However, Ontodella does not function with SWI-Prolog 5.4.3. Another important thing to notice is that SWI-Prolog must be installed with HTTP support. For detailed installation instructions of SWI-Prolog consult the `README` document of the distribution package.

The installation instructions of Ontodella are given in the `INSTALLATION` document of the package. Two important things to notice is to create an configuration file called `.ontodella` in your `$HOME` directory with proper settings and modify the configuration to load your data and rule files. The configuration of the Ontodella is defined in `<ontodella>/conf/ontodella_config.pl` file. Config file defines where the data and the rule files are located that are used for the category creation and recommendations. The port used by Ontodella can also be changed by modifying the configration file.

3.2 Semcocoon Installation and Cofiguration

The Ontoviews package contains the Cocoon based portal infrastructure and Ontogator search engine. The installation instructions are located in the `<semcocoon>/readme.txt` file of the package.

Before launching the portal application, some basic settings need to be adjusted to conform with the local portal setup. Like all Cocoon applications, the portal settings as defined in the sitemap. Sitemap is located at `<semcocoon>webapp/fmscocoon/sitemap.xmap`. The data flow of the Cocoon pipelines is defined in the sitemap. For more detailed account of configuring and using Cocoon, please consult Cocoon documentation or a book, such as [2].

As mentioned in the `readme.txt` file of the Ontoviews package, the location of the Ontodella server must be defined in the sitemap. Also, the location of the `category.rdf` file generated by Ontodella (cf. section 4) must be defined so Ontogator can find the categories and send the correct data to the GUI. Defining these two locations is enough to get your portal application up and running.

Before the Semcocoon can be launched, it first needs to be built with ant (`ant <semcocoon>/build.xml`). The application only needs to be rebuilt when the source code of a Java based component is modified.

The `<semcocoon>/build/semcocoon/cocoon/cocoon.sh` script launches the portal application. The script starts a Jetty Servlet container in which the portal application is run. Once launched the portal is accessible at `http://localhost:8081/`. The default port can be changed by modifying the launch script.

4 Creating Categories with Ontodella

The categories that are used as the basis of the faceted browsing interface of the soon-to-be portal are generated with Ontodella. The output of this phase is actually the whole content of the portal, the categories for the facets and the information items, or *bookmarks* in Ontodella jargon, that are to be displayed in the portal. The information items can be almost anything: informative web pages, museum items, multimedia clips etc.

The category rules are written in Prolog. However, the rule creation usually requires only the rudimentary knowledge of the language. Rules consist of set of Prolog predicates that define the resources of the original ontologies and data set that are to be projected to the category structure of the portal.

The categories are projected from the ontologies used for describing the data that is to be published in the portal. Each category rule defines one facet for the portal interface. The facets are a set of hierarchical categories, for example a of time periods, locations or topic areas.

Each of these rule consists of four parts (ie. four predicates). First part

defines the root resource of the category to be created and names of the other predicates that form the body of the rule. The other three parts are label rules, the sub-category rule, and the leaf rules.

Label rule is trivial and only defines the name that is to be used for the category. The sub-category rule defines how the hierarchical structure of the facet is formed. Typically the hierarchy is created either by using some subclass structure or some part-of type property. The leaf rule defines what type of bookmarks are connected to the category.

Here is an example of a complete rule for creating a facet. The rule creates a hierarchical structure of resources from the domain ontology that are of type `PageContent` [5]. The hierarchy is based on part-of (`partOfContent`) / has-part (`hasSubContent`) type relations between the resources.

```
%The main rule predicate that defines the root resource to be used in
%the category creation and the other parts of the rule.
sewehgrius_category(
   'http://www.cs.helsinki.fi/group/iwebs/ns/sf-page#PageContent', %and
   pagecontent_root_labels,
   pagecontent_sub_category,
   pagecontent_leaf
).

%The label of the facet in different desired languages.
pagecontent_root_labels(_, LabelList) :-
   LabelList = [en:'By content:' fi:'Sisältötyyppeittäin:'].

%Predicate that defines which resources from the domain ontology
%are linked together to form the hierarchical structure of the facet.
pagecontent_sub_category( URI, CatPartURI) :-
   URI = 'http://www.cs.helsinki.fi/group/iwebs/ns/sf-page#PageContent',
   rdf_type(CatPartURI, URI),
   not( rdf(CatPartURI,
        'http://www.cs.helsinki.fi/group/iwebs/ns/sf-page#':partOfContent,
        X))
    ; %or
   rdf(URI,'http://www.cs.helsinki.fi/group/iwebs/ns/sf-page#':
       hasSubContent, CatPartURI).

%The final part of the rule that defines what kind of resources are
%attached to the categories of the facet. Here 'page' is a predicate
%that defines what types of resources are displayed as bookmarks in
%the portal.
pagecontent_leaf( BookmarkURI,URI) :-
   rdf( BookmarkURI,
```

[5] For brewity only local names of the resources are presented in the text.

```
        'http://www.cs.helsinki.fi/group/iwebs/ns/sf-page#':containsContent,
         URI),
   page( BookmarkURI ).
```

The previous example shows a `page` predicate that is defined as follows:

```
page( SubjectURI ) :-
  rdf_type( SubjectURI,
            'http://www.cs.helsinki.fi/group/iwebs/ns/sf-page#Webpage')
 ; %or
 (rdf_type( SubjectURI, B),
 rdfs_subClassOf(B,
            'http://www.cs.helsinki.fi/group/iwebs/ns/sf-page#Webpage')).
```

The predicate matches all the resources of the domain ontology and data set that are of type `Webpage` or its direct subclass. The predicates `rdf_type`, `rdf_subClassOf` are defined in Ontodella's source code. Note that these predicates are not from SWI-Prolog's standard RDF-library, so the naming convention and available predicates are bit different. The available predicates are located at `<ontodella>/lib/prolog/sewehgrius_rdfs_core_rules.pl`.

In the previous example the `page` predicate defines which resources form the bookmarks of the portal. Before the categories can be created, Ontodella must be explicitly told that the bookmarks connected to the categories must match the wanted predicates. This is done by modifying the `<ontodella>/src/prolog/ontodella_request_handler.pl` file and by adding the wanted predicates to the list that defines which resources are to be treated as bookmarks. The list is defined in `get_categories` predicate.

The category file (`<ontodella>/categories/categories.rdf`) is created by running the `<ontodella>/bin/run-cache_categories.pl` script. Depending on the size of the domain ontologies and data sets, and the complexity of the category creation rules the category creation may take as long as a day. However, with small data sets the creation is usually done in matter of seconds. Before running the script it is often helpful to try out the rules by hand in the Ontodella shell. You can launch the shell with the `<ontodella>/bin/run-onto.sh` script. From the shell you can give direct commands to the Prolog interpreter.

Testing a rule happens simply by giving the predicate to the Prolog interpreter with unassigned variables. Eg. executing the command `pagecontent_sub_category(X,Y).` should output the resources that fulfill the predicate, viz. those resources that form the category hierarchy. By default, the Prolog interpreter only outputs only a single variable set at time. To receive more results, press `;` (semicolon) in the interpreter after the first result. To stop listing the answer set, press `.` (period).

5 Defining Recommendation Rules

Another function of the Ontodella service is to act as the recommendation engine of the portal. Ontodella delivers recommendations to the portal's user interface based on set of recommendation rules. The recommendation rules are defined as predicates that describe which two resources are to be linked together by a virtual property. In essence, the recommendations are shortcut properties that are generated on the fly from the original data set.

The recommendations are transferred over HTTP protocol. Whenever user views a single item in the user interface, a request accompanied with the URI of the displayed item is sent to the Ontodella server. Ontodella server then generates all the recommendations and sends the reponse in RDF/XML back to the Cocoon based GUI. The selected item with the accompanying recommendations are then displayed to the user.

In a simple case the recommendation rule can simply connect two resources that share a property with the same value. However, the recommendation rules can be complex and they can connect two resources that are connected by an arbitarily long chain of properties.

The following is an example of a simple recommendation rule that connects `SubjectURI` and `TargetURI` which both have the same value of the property `coversTopic`:

```
%The rule setup defines the URI assigned to the rule, label of the rule,
%and the name of the predicate that actually defines the rule.
sewehgrius_relation_rule(
    'http://www.cs.helsinki.fi/group/iwebs/2004/12/deduced_rel#same_topic',
    [en:'Related to the same topic: ', fi:'Tietoa samalta aihealueelta:'],
    same_topic
).


same_topic( SubjectURI, RelationDescription, TargetURI ) :-
 PropertyURI = 'http://www.cs.helsinki.fi/group/iwebs/ns/sf-page#':
               coversTopic,
 page( SubjectURI ),
 page( TargetURI ),
 rdf( SubjectURI, PropertyURI, SameTopic ),
 rdf( TargetURI, PropertyURI, SameTopic ),
 SubjectURI \= TargetURI,
 list_labels( [SameTopic], TopicLabel ),
 RelationDescription=[commonResources( SameTopic ), label(fi:TopicLabel)].
```

Similarly to the category rules, every relation rule is defined initially as

a new `sewehgrius_relation_rule` predicate, that contains the basic predicates that are used in the creation of the rule.

The Ontodella must be running and accessible by the Cocoon component of OntoViews for the recommendations to work. Ontodella can be started with the script `<ontodella>/bin/run-onto.sh` or started as a background process running inside a screen with `<ontodella>/bin/screen-run-onto.sh`. As with the category rules, it is wise and time saving to debug the recommendation rules by using the prolog interpreter.

6 Displaying the Content to the User

The portal application is built on top of modified Apache Cocoon, Semcocoon framework. According to the Cocoon desing principles, the portal content is displayed to the user as the result of consecutive transformations of data through Cocoon pipelines.

After getting the portal running and displaying the data you can start customizing the portal to correspond your requirements. A great deal of the customizations can be done by modifying the XSLT stylesheets (located at `<semcocoon>/webapp/fmsweb/transformation/`) that are used to transform the RDF results generated by the Ontogator search engine to human readable HTML-content (stylesheets `webItem.xslt` and `webItems.xslt`). The stylesheets used to create an Ontogator query (`ogt*.xslt` stylesheets) need modifications if different set of properties attached to the categories and bookmarks are required as a result of a query. Straightforward modifications to these stylesheets do not require any reconfiguration the cocoon pipelines, but new features can be created by extending the pipelines.

The series of transformations form the backbone of a Cocoon pipeline. A pipeline consists of various of components, most interesting in our case being the transformers. Each transformer

- takes the output stream of the previous component as its input stream,
- proceses the SAX-events received and
- gives the output to the next component,

thus the set of transformers create a pipeline. Besides transformers the pipelines also require the initial content generators and a serializer at the end of the pipeline that renders the output of the pipeline into form understandable by the end user. The components and their execution order in the pipeline are defined in the sitemap.

The two important pipelines of OntoViews are the search pipeline and the item pipeline. The search pipeline generates the multifaceted search interface based on the selected categories and key words given to Ontogator and the item information displays a single information item that is selected by the user as a result of a search.

In addition to modifying the existing transformers, the pipelines can be extended by adding an additional transformer to the pipeline. The transformer can be a small Java program, that picks out interesting SAX events and generates new content based on the received event.

An example of a such transformer is a component that is added to the pipeline that delivers an information page to the user. The new transformer listens to the SAX event stream and expands received resource that is of type `rdf:list` to contain all the items of the list to the outgoing stream. This way the expanded list can be processed in the remaining transformers and the content of the list can be displayed to the user.

These custom transformers can be created under the `<semcocoon>/src/` directory. The portal application needs to be rebuilt after creating or modifying such transformer. For more details about creating your own transformers consult Cocoon documentation.

7 Conclusion

This document discussed the deployment a semantic portal using the OntoViews-toolkit. The toolkit consists of three major components, of which two need to be configured during the deployment.

Ontodella is the recommendation engine that is also used for generating the categories and the data for the portal application from the original data set and ontologies. Ontodella functions as a separate HTTP-service that is queried by the Cocoon based portal application. The portal application includes Ontogator, the search engine, that generates the results based on the user's interaction. The results are then displayed to the user through the Cocoon pipelines that can be modified to customize the displayed data to correspond the desired information content of the portal.

References

[1] E. Mäkelä, E. Hyvönen, S. Saarela, and K. Viljanen. OntoViews - A Tool for Creating Semantic Web Portals. In *Proceedings of the Third*

Internation Semantic Web Conference (ISWC2004), Hiroshima, Japan, November 2004. Springer Verlag.

[2] C. Ziegeler and M. Langham. *Cocoon: Building XML Applications*. Pearson Education, 2002.