Automated Generation of a Faceted Navigation Interface Using Semantic Models

Tim Hussein and Daniel Muenter
University of Duisburg-Essen
Lotharstr. 65, 47057 Duisburg, Germany
{tim.hussein, daniel.muenter}@uni-due.de

ABSTRACT
In this paper, we introduce a concept for automated generation of faceted navigation widgets. These widgets are generated on the fly depending on the type of data to be displayed. For this purpose, we use semantic models for data representation and apply generic SPARQL queries, which makes the navigation creation completely independent from the content and structure of the underlying models.

Author Keywords
Model-driven UI Generation, Faceted Browsing

ACM Classification Keywords
D.1.2 Software: Programming Techniques—*Automatic Programming*

INTRODUCTION

An intelligent navigation structure can help the user find things faster and with less effort. Often, the information flood is just overwhelming for the user – especially in environments with lots of items to choose from (which is very typical for online e-commerce portals or news sites). There are several ways to overcome this problem, each with its own benefits and drawbacks. Keyword search, (fixed) navigation bars, or recommender systems are often used for this purpose.

FACETED BROWSING

A relatively novel approach of navigating structured data is called *faceted browsing* [2, 6]. The basic idea is, to filter items by their attributes (e.g. shoes by size and color). If the items that are supposed to be explored can be classified by certain characteristic features, faceted browsing is a suitable way of narrowing alternatives. This is especially useful, if the user does not look for a particular item, but for alternatives meeting certain requirements.

Based on the facet theory [8], the information space is partitioned using conceptual dimensions of the data. Faceted browsing is used to narrow the search space gradually by means of so called *facets*, until the user finds what he or she is looking for. This theoretical concept has been adopted to the semantic web scenarios in the last years: There have been various approaches of browsing semantic datasets modeled in OWL or RDF [3, 4, 7, 10] by using facets.

Each facet is able to filter the relevant items in a different way [5]. An important advantage of facets is the flexible exploration of the data space from various entry points reflecting the features of the items. The user does not have to know the underlying structure or the objects itself. Instead, he uses the navigation structure automatically generated from the objects and is able to narrow the search space until he finds what he is looking for. As a convenient side effect, the user implicitly learns about the items' features, which might help him to find them even more efficiently in the future.

In order to classify objects, we need some kind of metadata about them. Usually, the objects features are used for that purpose. We want to illustrate this technique by using the example of an electronic product catalog for books, cds, dvds, and other items. Each of these products can be described in a certain manner: A book has specific features like "title", "author", "publisher", "year of publication" and more. In this fashion, other products can be classified as well: Dvds by using actors, directors, and so on. These features then can be grouped to categories like "author", "title", "publisher", etc., which are used as facets. The user can use an arbitrary facet as the entry point for the navigation. After he selects a certain facet, for instance "author", all possible values are listed for filtering the items. In this case, all authors of all books would be displayed.

DATA SET

We implemented a faceted navigation structure on top of a predefined set of semantic data. In this case, we set up a virtual (fictitious) web portal for shopping and entertainment purposes, in which the user can browse information about more than 500 different articles, including (but not limited to):

- Books, DVDs, Videogames, and CDs
- Actors, Artists, and Authors
- Clubs and Pubs

SEMAIS 2010.

- Concerts and Sport-Events

We entitled the web portal "Discovr" and Figure 1 shows a screenshot of it.

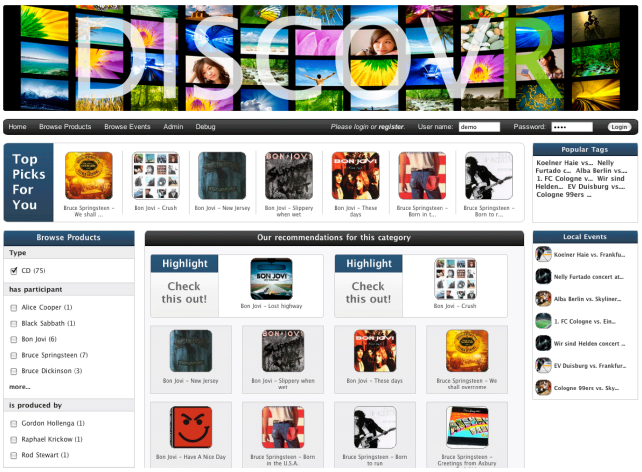


Figure 1. The Discovr Web Portal including a faceted navigation structure on the left. The center area shows a selection of items that meet the current exploration criteria.

All information about the items in Discovr are encoded in RDF models. Listing 1 shows a small example of how the data is modeled. They do not have to provide further information for the navigation widgets; just a semantic description of the data and how the items are related to each other.

```
<DVD rdf:about="#dvd_big_fish">

 <rdfs:label
  rdf:datatype="&xsd;string">
  Big Fish</rdfs:label>

 <genre
  rdf:resource="#fantasy"/>

 <participant
  rdf:resource="#ewan_mcgregor"/>

 <participant
  rdf:resource="#steve_buscemi"/>

 <produced_by
  rdf:resource="#tim_burton"/>

</DVD
```

Listing 1. Data is encoded in RDF models

These models are handcrafted, but in principle it is possible to use existing RDF Data Sets or SPARQL-Endpoints like `http://www.dbpedia.org`[1]. In the next section, we explain how we implemented a mechanism to automatically generate a faceted navigation structure on models like this.

[1]We used our own data set, because we thought that sources like dbPedia might contain to many items for a controlled experiment.

NAVIGATION WIDGET GENERATION

In the previous section, we mentioned that one advantage of faceted browsing is, that it supports multiple entry points for navigation. To illustrate the facet creation process, we assume that the user is looking for a particular DVD. So we use "DVD"as the entry point.

As all items in our database are semantically structured, we can use SPARQL queries to select a certain subset of them. SPARQL is a SQL-like query language that can be used to filter OWL or RDF data by semantic queries. Listing 2 shows a simple example how to select all items that are DVDs. A query like this would be triggered to start the facet creation process.

```
SELECT DISTINCT ?item
WHERE {
  ?item rdf:type domain:DVD .
}
```

Listing 2. SPARQL query for selecting all DVDs

The result of the listing would be a list of all DVDs. From an architectural point of view, it looks like illustrated in Figure 2. The client clicks on a certain button that triggers a first request. In this case, he wants to receive all items that have "DVD" as the type. The request is encoded in SPARQL by the server application behind the web portal and this SPARQL request is applied to the semantic data set. The result of the query, in return, is delivered to the browser. In this case, a list of all DVDs.

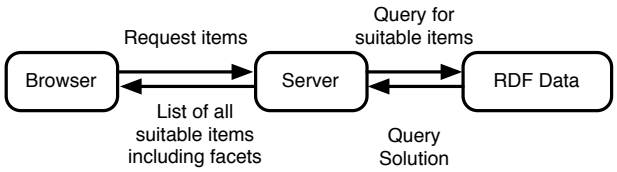


Figure 2. A basic request for certain items. The browser request is translated into a SPARQL query. For instance a query to retrieve all items of a certain type (DVD). The query solution, in return, is translated into appropriate HTML widgets by the server application. These widgets are send to the browser.

Upon the query solution, the server automatically analyzes the data and creates facets for further exploration. Figure 3 shows an example for facets that are automatically generated based on a subset of all DVDs. This analyzation process is relatively simple: All relations of all items returned by the query are examined and grouped by type, for instance "Producer", "Participant", etc. and a facet is created for each group.

Of course, most real web shops would now produce a vast amount of DVDs. This is a typical case where faceted browsing makes sense to incrementally narrow the results. Now,

we can make use of the features that are encoded in the RDF data set.

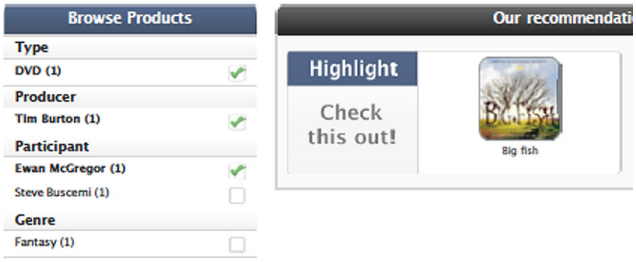


Figure 3. Facets for DVD browsing. This is an example for nominal data. In this case, a selection box is a compatible navigation widget. Facets for other datatypes follow.

In this example, the user can browse a collection of DVDs by filtering by producer, genre, participating actors, etc. Upon each click on a certain element in a facet, the selection is narrowed. A typical workflow could look like this:

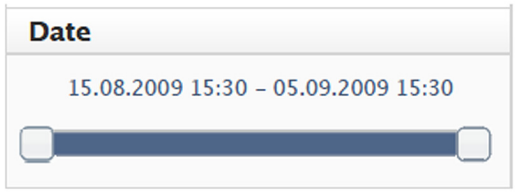
1. The user clicks on a (pre-defined) "Browse DVDs" Button.
2. This triggers a request like the one shown in Figure 2.
3. A navigation structure like the one from Figure 3 is created on the fly.
4. The user can filter the content by selecting one or more options, in this case he selects *Tim Burton* as a producer.
5. The selection (Tim Burton) is encoded into a SPARQL-query that is now used to narrow the selection. The process is similar to step 2.
6. The original selection (all DVDs) is constrained to only those items, which fulfill all conditions: They have to be DVDs and they have to be produced by Tim Burton.
7. In a third step, the user could, for instance, select *Ewan McGregor* as an actor that has to appear in the movie, which is another constraint that is treated in the same process as above.

At any step, the user can release a facet condition and the selection is expanded. In this way, the user is able to explore the item space in many different ways.

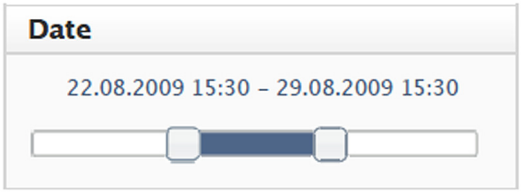
WIDGET DECORATION

The example illustrated in the previous section entirely used *nominal* data, a form of categorical data where the order of the categories is not significant [9]. The categories are, for instance "Movies with Ewan McGregor", "Movies with Matt Damon", and so on. A selection box like the one in Figure 3 is a good form of representation. In our case, only the top n values are displayed with the option to show the other entries by using a "more"-button (or constraining the option by using other facets).

Showing all possible alternatives, however, is not suitable in case of *ordinal* or *interval* data. Figure 4 is an example for an automatically generated facet based on dates. As dates follow a certain order and can be restricted by intervals, a navigation widget using a slider is a better approach to restrict the selection.



(a) The range is automatically set



(b) Applying date restrictions

Figure 4. An automatically generated facet for filtering by date. This facet uses a slider instead of a selection box. Depending on the type of data to be displayed, certain navigation widgets are more useful than others.

In this case, the elements can be ordered in a linear way, which makes it possible to use a slider to constrain the items to a subset. So, it makes sense to choose a meaningful display widget depending on the type of data. Fortunately, we can use SPARQL for this purpose. Using a query like the one presented in Listing 3, we can automatically detect the type of data to be created to select the appropriate widget as a representation.

```
SELECT DISTINCT ?facetType ?facetLabel
WHERE {
  ?individual rdf:type ?type ;
  ?facetType ?restriction .
  ?facetType rdfs:label ?facetLabel .
}
ORDER BY ?facetType
```

Listing 3. Obtaining the type of data via generic SPARQL query

In 2, we used only one variable ($?item$). Now, we query the data that we want to create a facet for, and use two variables: $?facetType$ and $?facetLabel$. We use the label just for sorting the elements within the facet. The *facetType*, on the other hand, is the key for the widget selection process.

Now, all we have to do is create a mapping for the possible types of facets to certain types of widgets. We retrieve the type of data as a XML-Schema Datatype, like *xsd:integer*, *xsd:string*, *xsd:date*, and so on. In our case, we mapped string-data to a selection box and dates, for instance, to a slider widget.

For that purpose, we make use of the *Decorator* design pattern [1] illustrated in Figure 5 to automatically create a suitable navigation widget that is intuitive to use for that kind of data. The Figure 5 illustrates the process of *decorating* the facet.

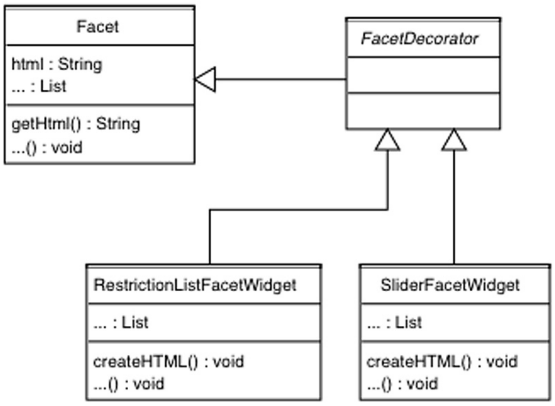


Figure 5. Depending on the type of faced, a suitable navigation widget is used to *decorate* the data. For that purpose, we make use of the *Decorator* design pattern.

A *FacetDecorator* is assigned to each facet. This FacetDecorator "knows" how to display the facet content. This approach separates the content of the facet from its representation. By default, every facet can be displayed as a selection box. But, if appropriate, other interaction widgets can be used.

The facet representation for the browser is created by calling the *facet.getHTML()* method. The facet delegates the call to its decorator's *createHTML()* method. This method returns valid HTML code that can be displayed on the client to realize a HTML facet.

SUMMARY

In this paper, we presented a generic method for automatically generating faceted navigation widgets based on semantic models to filter items by their characteristic features. This approach is independent from the actual items to be displayed and exploits the type of feature to automatically create a widget that is suitable for the type of data. The approach is generic and can be re-implemented for any scenario in any type of application.

Our approach iteratively constraints the search space to a subset of suitable items. After that, a compatible display widget is selected to *decorate* the data. So, the process of data retrieving and data display is divided into distinct processes.

Both processes are independent of the underlying ontological representation. The underlying source code can (and has been) used for different scenarios without custom adjustments. But even, if a totally different kind of data model is used, the concept itself can be adapted, because the basic principles are independent from the languages used.

ACKNOWLEDGEMENTS

The research presented in this paper is part of the CONTICI project, in which the Universities of Duisburg-Essen, Siegen, Hagen, and Aachen take part. CONTICI is funded by the German Research Foundation (Deutsche Forschungsgemeinschaft).

The authors thank Werner Gaulke and Timm Linder for fruitful discussions and implementation support.

REFERENCES

1. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design patterns: Abstraction and reuse in object-oriented designs. In *Proceedings of the 7th European Conference on Object-oriented Programming (ECOOP '93)*. Springer, 1993.

2. N. Gibbins, S. Harris, A. Dix, and M. C. Schraefel. Applying mspace interfaces to the semantic web. Electronics and Computer Science EPrint 8639, University of Southampton, 2003.

3. P. Heim, J. Ziegler, and S. Lohmann. gfacet: A browser for the web of data. In S. Auer, S. Dietzold, S. Lohmann, and J. Ziegler, editors, *Proceedings of the International Workshop on Interacting with Multimedia Content in the Social Semantic Web (IMC-SSW'08)*, pages 49–58, 2008.

4. M. Hildebrand, J. R. van Ossenbruggen, and L. Hardman. /facet: A browser for heterogeneous semantic web repositories. In *Proceedings of the 5th International Semantic Web Conference (ISWC '06)*, pages 272–285. Springer, 2006.

5. E. Oren, R. Delbru, and S. Decker. Extended faceted navigation for rdf data. In *International Semantic Web Conference*, pages 559–572, 2006.

6. C. Plaisant, B. Shneiderman, K. Doan, and T. Bruns. Interface and data architecture for query preview in networked information systems. *ACM Transactions on Information Systems*, 17(3):320–341, 1999.

7. D. Quan, D. Huynh, and D. R. Karger. Haystack: A platform for authoring end user semantic web applications. In *Proc. 2nd International Semantic Web Conference, 2003 (ICSW '06)*, pages 738–753. Springer, 2003.

8. S. R. Ranganathan. *Elements of library classification*. Asia Publishing House, 1962.

9. S. S. Stevens. On the theory of scales of measurement. *Science*, 193(2684):677–680, JUN 1946.

10. K.-P. Yee, K. Swearingen, K. Li, and M. Hearst. Faceted metadata for image search and browsing. In *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 401–408, New York, NY, USA, 2003. ACM.