Configuring Semantic Web Interfaces by Data Mapping

Michiel Hildebrand
CWI Amsterdam
Michiel.Hildebrand@cwi.nl

Jacco van Ossenbruggen
CWI/VU University Amsterdam
Jacco.van.Ossenbruggen@cwi.nl

ABSTRACT
We demonstrate how to develop Web-based user interfaces for Semantic Web applications using commonly available, off-the-shelf Web widget libraries. By formally defining the underlying data model that is assumed by these widgets, Semantic Web application developers can use familiar RDF constructs to map their own data to the model implemented by the Widgets. As an example, we briefly describe the interface model underlying our own framework, and provide concrete examples showing how it has been used to create Semantic Web applications in two different domains. We conclude by discussing the advantages and limitations of our approach.

Author Keywords
Interface design, heterogeneous data, ClioPatria, YUI

ACM Classification Keywords
H.5.2 Information Interfaces and Presentation: Miscellaneous

INTRODUCTION
Semantic Web data is typically rich in interconnections and highly heterogeneous. Designing user interfaces for applications that use this type of data is intrinsically hard. Designing interfaces for highly diverse data tends to lead to overly generic interfaces that do not communicate the richness of the data to the end user. On the other hand, interfaces that communicate this richness effectively tend to work well for only a set of fixed schemas and not for the entire dataset. Finding a sweet spot that balances these two forces is not trivial, especially if one takes into account that most Semantic Web developers are not UI specialists, and often have even little affinity with UI design. The problem is even deepened because many Semantic Web developers tend to build UIs from scratch, as the fixed data model that is assumed by many off-the-shelf UI toolkits seems to conflict with the heterogeneity of their data.

In this paper, we argue that for a wide range of applications, such a sweetspot can be identified and formally modeled in RDFS or OWL. By building standard interface components on top of this model, building an initial interface can be as easy as mapping the application's data model to this interface model.

This approach has the advantage that it leverages the significant amount of design, implementation and testing effort already invested in today's Web UI toolkits, and we believe that reuse of these commonly available toolkits will, in general, lead to better interfaces than interfaces that are designed and implemented from scratch by a (small) Semantic Web research team.

In addition, it replaces the traditional configuration and tailoring that is needed to adapt a generic interface to a local dataset by a straight forward RDF data mapping task, a skill most Semantic Web developers will have. By providing default mappings, it is even possible to provide a no-configuration, first crude version of a user interface, very early in the application development life cycle. This will give RDF data developers the "instant gratification" that has made many Web 1.0 and 2.0 applications so popular. It also encourages them to, during their data modeling and data development tasks, think about how their data will be used in the end-user applications, and how their modeling decisions may impact the interface.

Finally, while our approach provides pre-packaged solutions for common tasks, it does not prohibit application developers to go beyond those solutions in order to add more advanced or more application specific interface components. It is built to be extended or to build other layers on top of it.

This paper is structured as follows. In the next section, we discuss related work and compare it to the approach proposed in this paper. As an example user interface model and its binding to a Web UI toolkit, we discuss the interface model upon which the ClioPatria [12] interface components are built, and how these components are implemented on top of the Yahoo! User Interface (YUI) library. We show how this model can be used to easily create two interfaces, one in the cultural heritage domain and one in the news domain. We provide some details of the current implementation and in the last section, we discuss the pros and cons of our approach.

RELATED WORK
Modeling part of the user interface in RDF is in itself not new. Fresnel [7] is a good example of an RDF vocabulary

Workshop on Visual Interfaces to the Social and the Semantic Web (VISSW2009), IUI2009, Feb 8 2009, Sanibel Island, Florida, USA. Copyright is held by the author/owner(s)

that can be used to specify the presentation details of RDF data. Fresnel provides vocabularies for the selection and formatting of the data. A Fresnel engine generates a tree structure with selected resources and formatting information. The actual visualization of this tree is left to the application or widget. Our interface widgets depend on configurable search functionality on the server, for example, vocabulary-based prefix search for autocompletion, faceted queries and graph search. This functionality goes beyond Fresnel's selection vocabulary. Our approach targets the configuration of the server-side search algorithms instead of providing a vocabulary for selection. Furthermore, the visualized data often becomes part of the widget's interactive behavior, for example, extra information for a result needs to be shown in a popup, trees need to be dynamically extended and facets may be interactively removed or added. Fresnel's formatting vocabulary is not suited for describing this type of interaction. We propose a method that takes the widget's functionality as the starting point and provides a means to configure the widget's selection, organization and visualization methods. An interface model captures these configuration dimensions and by providing mappings to domain specific data a developer can fully exploit the widget's behavior for a specific task.

Simile's faceted browser Longwell supports Fresnel for the visualization of the results [9]. In addition, the set of facets displayed in the interface can be defined in a configuration file in RDF. The individual facets are, however, not configurable. Simile's Exhibit [3] provides a more extended vocabulary to define the visualization and organization of the facet values. The content of a facet in Exhibit is defined by a property that refers to a key in the native JSON format. This prevents the use of RDFS reasoning facilities, such as rdfs:subPropertyOf. Our approach combines the reasoning facilities of a server-side triple store with a lightweight client-side widget.

Web interface widgets have become a standard in web development. The choice among JavaScript libraries is numerous and all provide a convenient abstraction of low-level issues, such as cross-browser support. Interface widgets for semantic content are also available. Eetu Mäkelä et. al. provide several interface widgets that work on top of their ontology service infrastructure ONKI [5]. Example widgets are autocompletion and faceted navigation. They also so seem to strive for easy configuration of the widgets, but have not presented a clear model for this.

The approach of semantic widgets is also used by the Semantic Web company Mondeca [1].

COMBINING THE YAHOO! USER INTERFACE LIBRARY WITH THE CLIOPATRIA INTERFACE MODEL

In many domains there is a central role for persons, locations, times and artifacts. Sometimes these are modeled together as an event. For example, in the cultural heritage domain works of art are created by persons at a specific time

[1] http://www.mondeca.com/index.php/en/intelligent_topic_manager/applications/semantic_portal_semantic_widgets

Figure 1. Autocompletion suggestions of historical persons. Underneath the name a short biography is displayed. This contains the nationality, role/profession and birth/death date. Note that for the first person listed, only the profession is available in the data. The abbreviation RMA, shown to the right, indicates the thesaurus source.

and location. In a figurative art it may also be important to know which persons, times and locations are depicted. News images are also made on a specific time and place by a specific photographer and depict an event often involving persons, times and locations. Persons, locations and times are thus good candidates for a central model which is sufficiently generic, while sufficiently specific to answer the classic who, where and when questions to the end user. Man-made artifacts also play a central role in many domains and their specific properties can often be abstracted from by using, for example, a general vocabulary such as Dublin Core. In addition, different domains often deploy their own set of specific thesaurus concepts that describe the properties of events. We found that SKOS provides a sufficiently rich and abstract model to describe these concepts and their relations.

Within the semantic search and annotation framework ClioPatria [12] we have developed several interface widgets. When deploying ClioPatria in a specific application domain we use persons, locations, times, artifacts and thesaurus concepts as an intermediate model between the interface model and the domain specific details of the underlaying RDF data. In the following paragraphs we explain the configuration dimensions of two interface widgets, autocompletion and faceted navigation, and show how this can be captured in an RDF interface model. In the next section we show how the intermediate model sketched above can be mapped to these widget's interface models.

Example 1: Autocompletion

Autocompletion is an interface feature that allows users to type only a few characters instead of a full word or phrase. After the user has entered the first characters, the system responds by completing the word or phrase. If the characters typed in so far can be completed in more than one way, most interfaces present a list of multiple options. The user can than either select one of the options from the list, or continue typing to narrow down the number of options.

In context of the Semantic Web autocompletion is useful to quickly find a resource by one of its labels[2]. In previous work we argued that for different tasks and data sets autocompletion widgets require a different configuration [1]. The screenshots in Fig. 1 and Fig. 2 show autocompletion

[2] For sake of simplicity we do not consider finding resources by a label of a related resource.

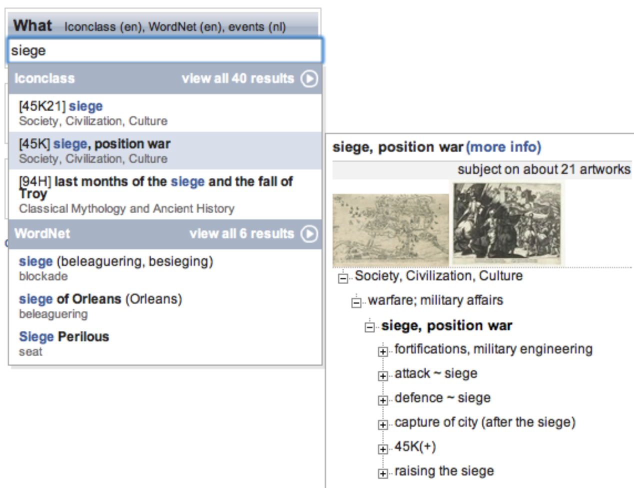

Figure 2. Autocompletion suggestions of thesauri concepts. Results from both `IconClass` and `WordNet` are shown, each presented in a separate group. A secondary panel shows more information for the highlighted term ("[45K] siege, position war"), including the hierarchical structure the term is part of. The hierarchy contains the term itself in bold, its ancestors and the direct children.

suggestions of historical persons and thesauri concepts. In the next section we discuss the configurations of these two widgets, here, we focus on the main differences between these two widgets. First, the widgets suggest a different type of resource (e.g. persons and concepts), thus, requiring a different *selection* of the right RDF resources. Second, the persons are *organized* in an alphabetically ordered list, while the concepts are grouped by different thesauri and ranked according to popularity. Finally, the individual suggestions are *visualized* differently in each widget. The suggested persons are shown with extra biographical information, whereas, the concepts are shown within their hierarchy.

The two examples are built on top of the YUI autocompletion widget. The YUI widget contains several client side configuration parameters, it supports custom functions for result formatting, construction of remote data requests and it provides many event handlers. Although this is sufficient to configure the widget for an RDF data source, we experienced that it required extensive JavaScript programming to obtain the appropriate configurations. For example, visualizing different types of information requires the configuration of the server request as well as the client side JavaScript formatting functions.

An interface model for an autocompletion widget provides a single focal point for the configuration of a widget and only requires Semantic Web modeling skills. Note, we do not claim that this is a complete model for autocompletion, we merely want to illustrate that it is possible to define an interface model for an autocompletion widget in RDF. The model we present is for an extended version of the YUI autocompletion widget. We added support for clustered presentation of search results, a secondary display that is shown when the user hovers over a suggestion, a single configurable result formatting function and support for easy configuration

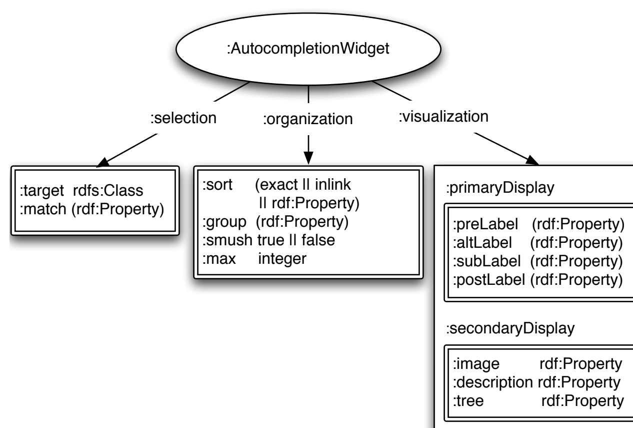

Figure 3. Interface model for ClioPatria's thesaurus concept autocompletion widget. All organization and visualization properties are optional.

of the server side search algorithm.

An interface model for the *RDF concept autocompletion* widget is shown in Fig. 3. The widget contains three main configuration properties that correspond with the three phases of the search process: selection, organization and visualization. For the selection of the appropriate resource it should be known what type of resources should be selected and which literals should be used to find these resource? The first is configured by providing an rdfs:Class for `:target`[3]. The second requires the definition of a collection of RDF label properties for `:match`. The order of the properties indicates which property has preferences in case the same resource is found by multiple labels. The selected resources can then be organized in a list or in groups of different lists. The grouping is performed on the values of the RDF properties provided for `:group`. The resources in the list can be ordered according to several criteria and are defined in a collection as a value of `:sort`. The built in constant, *exact* puts all resources with an exact matching label before resources with partial matching labels. Another built in constant is *inlink* that sorts the resources by the number of incoming links they have in the graph. Further sorting criteria are the display labels (explained in the next paragraph) or any rdf:Property. The number of results that are returned can be limited by defining `:max`. In a grouped organization the maximum applies to the number of items within a group. Finally, resources that are defined as equivalent (owl:sameAs or skos:exactMatch) are shown as a single suggestion when `:smush` is set to true.

The results are visualized in a primary and secondary display panel. The primary display contains all suggestions, while the secondary display is shown when the user hovers over one of these. The default display format of these panels, as shown in Fig. 4, is fixed. The primary display contains

[3]The properties and classes used in the interface models are contained in our own namespace `http://e-culture.multimedian.nl/ns/interface/`. In this paper we omit this namespace and simply write a colon.

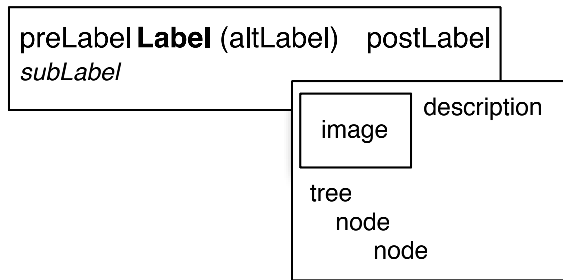

Figure 4. Layout of an autocompletion result. Primary display with a preLabel, the label itself, an alternative label between brackets, a postLabel aligned on the right side and a subLabel on a second line. The secondary panel provides additional space for larger content, such as images, descriptions and trees.

place markers for four labels besides the matching label itself. The developer defines which labels are shown by mapping domain specific properties to the properties of the interface model (e.g. mapping skos:broader to `:subLabel`). If the default display format is not sufficient a new formatting function can be created in the JavaScript widget, which might also require updates to the interface model.

Example 2: Faceted browsing

Facet browser interfaces provide a convenient way to navigate through a wide range of data collections. Originally demonstrated in the Flamenco system [13], facet browsing has also become popular in the Semantic Web community thanks to MUSEUMFINLAND [4] and other systems [3, 6, 9]. An individual facet highlights one dimension of the underlying data. Often, the values of this dimension are hierarchically structured. By visualizing and navigating this hierarchy in the user interface, the user is able to specify constraints on the items selected from the repository.

The facet browser we developed within ClioPatria, /facet, could be applied to any RDF dataset [2]. By considering the Class and Property hierarchy as special facets the user could configure the interface to her needs. In the Class facet the user selects the target resources (e.g. documents or persons) and from the Property facet she selects the facets she wants to navigate (e.g. creator and subject for documents or birthplace and birthdate for persons). This approach provides an instant interface for Semantic Web engineers. Presenting the raw data is, however, not suited for end user applications. In the project HealthFinland it was demonstrated that through careful user studies a more user-friendly configuration of the facets can be achieved [10].

Consider a faceted interface on a collection of documents. Each individual facet contains the values within one dimension. For example, one facet might display all the creators, whereas, another might display the subject categories. On an RDF data source this type of value *selection* corresponds to the values of a particular RDF property (e.g. dc:creator and dc:subject). Other selection criteria are also possible, such as all instances of a particular Class. In a similar fashion

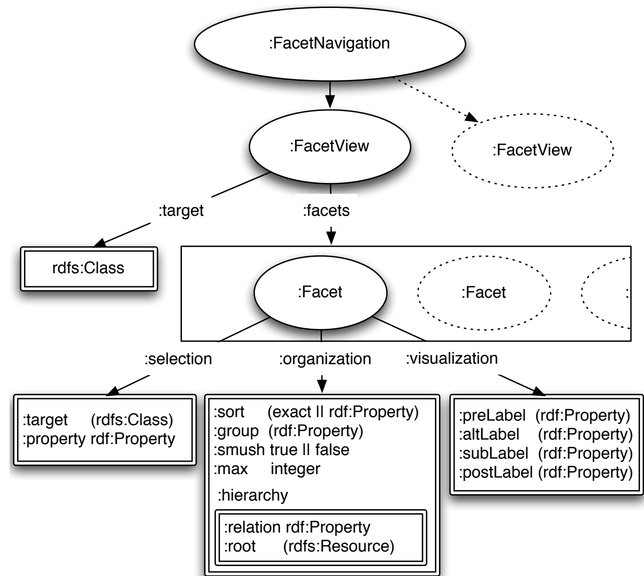

Figure 5. Interface model for ClioPatria's faceted navigation widget.

as the autocompletion suggestions, different types of facet values require different methods of *organization*. The creators might be best organized in an alphabetically ordered list, while the hierarchical structure is important for the subject categories. Also the *visualization* of facet values shows similarities with the autocompletion widget. Adding extra information in the display may help to disambiguate similar values. In addition, specific types of values (e.g. geographical locations and dates) are suited for alternative visualization (geographical map and timeline).

When the number of defined facets is too large to be shown in the interface, it has to be defined which facets are shown. In Longwell a *facet view* can be defined as a collection of facets for a particular target. The facets defined in this view are shown, while all other facets are collapsed and available on requested. An alternative method is to allow multiple views and allow the user to select the most appropriate view. For example, the creation view contains all facets that cover the creation of a document, whereas, the content view contains the facets about the topic. In either solution, a view defines which facets are selected for display.

As an example we describe a possible model for a faceted navigation widget. A screenshot of this widget, used for a collection of news items, is shown in Fig. 6. The widget displays multiple facets that are defined in a facet view and allows the selection of alternative views. The individual facets are built on top of the autocompletion widget, which allows re-use of its organization and visualization methods. In addition, it allows autocompletion within each facet. Again, we make the disclaimer that our purpose is not to provide a complete model for faceted navigation, but merely to illustrate that it is possible to define an interface model for a faceted navigation widget in RDF.

An interface model for a faceted navigation widget is shown

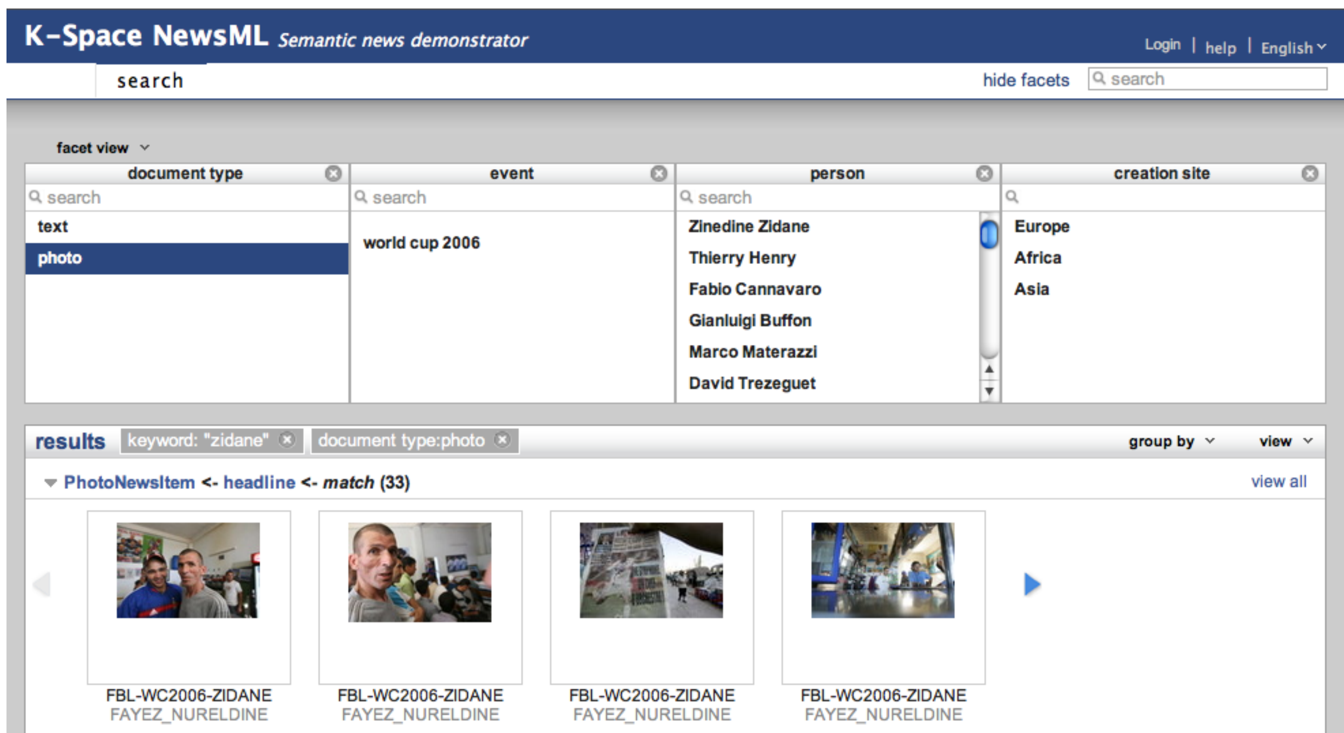

Figure 6. Faceted interface of the NewsML demonstrator. Four facets are active: `document type`, `creation site`, `event` and `person`. The value "photo" is selected from the `document type` facet. The full query also contains the keyword "Zidane", as is visible in the header above the results.

in Fig. 5. The widget can contain multiple facet views, that each apply to the resources of a particular Class. Each facet view contains an RDF list of facets. A facet is configurable in the selection of facet values, the organization and visualization of these values. At the moment our widget only supports the selection of facet values from an rdf:Property. The configuration of the organization and visualization is similar to that of the autocompletion widget. In addition, the facet values can be organized hierarchically, meaning that initially only the root values of the hierarchy are shown and after selection of one of these its children become available.

CONFIGURING INTERFACE WIDGETS: A MAPPING TASK

Given an interface model the configuration of a Semantic Web application becomes a task of mapping the right properties and classes to this model. In practice, this often means first finding a suited intermediate model for the domain. For example, in terms of persons, locations and times. Second mapping this intermediate model to the widget's interface model. We illustrate the mapping task with two use cases: configuring autocompletion components for a cultural heritage annotation application and configuring faceted navigation for a news application.

Use case 1: Rijksmuseum annotation user experiment

Within the MultimediaN E-Culture project [8] we developed a prototype interface for the subject annotation performed at the Rijksmuseum in Amsterdam, the Netherlands. The professional annotators of the Rijksmuseum describe thousands of artworks a year by assigning terms from controlled vocabularies. Finding the right term is complicated because the vocabularies used are large, very detailed, contain similar terms (or even duplicates) and often the annotator does not know exactly how to spell a term. We experienced that autocompletion helps professional annotators to find the right terms, but only when the widget is properly configured.

In an extensive study with these professionals we gathered the requirements for term search from multiple thesauri. During an iterative process of prototyping and discussion we tested several configurations of autocompletion widgets. A screenshot of the interface of the final prototype is shown in Fig. 7. On the right side the interface contains three autocompletion fields to lookup terms from thesauri and a free text field to input dates. One of the results of the study is that the three autocompletion fields all required a different configuration.

The interface model for the autocompletion widget, as described in the previous section, is based on our findings at the Rijksmuseum. We acknowledge that a single study might not be sufficient to determine a complete interface model that applies to other domains. On the other hand, all three autocompletion fields required different features and configurations. Furthermore, the three fields cover generic types of resources (persons, thesaurus concepts and locations) that are very likely to be used in other domains as well.

We first introduce the vocabularies used in the annotation interface, before describing the configuration of the person and

Figure 7. Interface of the Rijksmuseum subject annotation interface. The four annotation fields in the right column are configured to support effective search in different thesauri.

concept autocompletion fields. We used three thesauri with persons: Getty's United List of Artist Names[4] (`ULAN`), DB-Pedia's RDF version of person data[5] from Wikipedia (`WP`) and the Rijkmuseum's own people thesaurus. All three thesauri were mapped to the generic "Person" scheme of ULAN. For places we also used, Getty's Thesaurus of Geographic Names[6] (`TGN`) and aligned it with the Rijksmuseum's place thesaurus. We used SKOS for the geographical containment relations in combination with location-specific properties from TGN. The concepts used in this domain were also modeled or mapped to SKOS. In addition to the Rijksmuseum's events thesaurus we added the RKD `IconClass`[7] thesaurus and, as a source for more general terms, W3C's RDF version of Princeton's `WordNet`[8].

Autocompletion on persons

Listing 1 shows the configuration of the autocompletion widget in the `Who` field. The selection is restricted to resources of type ulan:Person. Note, the class of persons in the Rijkmuseum thesaurus and DBPedia people are subclasses of ulan:Person. We only consider literal values of skos:prefLabel and rdfs:label, where preference is given to the skos:prefLabel as this is first in the list. The results are organized alphabetically on the label and first showing all resources with an exactly matching label. The professionals at the Rijksmuseum explicitly indicated that they expect alphabetical ordering for a list of person names. As the autocompletion field gives access to the resources from different overlapping vocabularies it turned out essential to smush equivalent results to a single suggestion.

The primary display contains three labels in addition to the matching label. The `:altLabel` is only shown in case the match was not found an a skos:prefLabel. Thus, when a hit is found by a skos:altLabel it's skos:prefLabel is also shown. The `:endLabel` contains the value of the skos:inScheme property. Thus indicating the thesaurus the term comes from. The professional annotators requested this information as terms are suggested from their own as well as other thesauri. The `:subLabel` shown beneath the main label is composed out of the values of four properties. Together these compose a short biography of the person. The annotators use this information to disambiguate similar persons from one another. The secondary display contains an image depicting the person and a longer biography.

[4] `http://www.getty.edu/research/conducting_research/thesauri/ulan/`
[5] `http://dbpedia.org/`
[6] `http://www.getty.edu/research/conducting_research/thesauri/tgn/`
[7] `http://www.iconclass.nl/`
[8] `http://www.w3.org/2006/03/wn/wn20/`

```
:PersonAutocomplete
    a :Autocomplete ;
    :label "search_person"@en;
    :label "zoek_persoon"@nl ;
    :selection [
       :target ulan:Person ;
       :match (skos:prefLabel rdfs:label)
    ] ;
    :organization [
       :sort ("exact" "label");
       :smushing "true"
    ] ;
    :primaryDisplay [
       :subLabel (
         ulan:role
         ulan:nationality
         ulan:birthDate
         ulan:deathDate
       ) ;
       :altLabel skos:prefLabel ;
       :postLabel skos:inScheme
    ] ;
    :secondaryDisplay [
       :description ulan:biography ;
       :image vra:subject
    ] .
```

Listing 1. Person autocompletion allows autocompletion on instances of ulan:Person. The results are sorted first on exact matches and then alphabetically on the matching label. Results that are defined as equivalent (skos:exactMatch or owl:sameAs) are smushed. Each result is displayed with extra information. The primary display contains a short biography composed out of the values different properties and it contains the thesaurus source. The secondary display contains a full description and an artwork that depicts the person.

```
:ConceptAutocomplete
    a :Autocomplete ;
    :label "search_concept"@en;
    :label "zoek_concept"@nl ;
    :selection [
        :target [
           owl:unionOf (
                 ic:Concept ;
                 wn:Synset ;
                 rma:Event
             )
        ] ;
        :matchLabel (skos:prefLabel rdfs:label) ;
    ] ;
    :organization [
       :sort ("exact" "inlink") ;
       :group skos:inScheme
    ] ;
    :primaryDisplay [
       :subLabel skos:broader
    ] ;
    :secondaryDisplay [
       :description skos:note ;
       :image vra:subject
    ] .
```

Listing 2. Concept autocompletion allows autocompletion on instances of skos:Concept. The results are sorted first on exact matches and then on the number of inlinks. The suggestions from the same thesaurus are grouped together. In the secondary display a tree is shown with the all ancestors and direct children of the result.

Autocompletion on thesaurus concepts

Listing 2 shows the configuration of the autocompletion widget in the `What` field. We only describe the configurations that are different from the `Who` field. The target is defined as an owl:union of three classes, iconclass and wordnet terms and the events from the Rijksmuseum thesaurus. The resources from the three thesauri are each shown in a separate group. The Rijksmuseum wanted to give preference to terms from IconClass and only use WordNet as a backup. Organizing the results in different groups allowed the annotators to easily compare terms from the different thesauri to one and other. Within each group the results are ordered by the number of links that are pointing to the resource. Intuitively, this means that the popular resources are shown first.

Use case 2: K-Space Semantic News Browser

ClioPatria is used to support search and browsing of news items [11]. These news items are described with multimedia standards, news codes from the IPTC standard and additional metadata from various thesauri modeled (mapped) to SKOS. The additional metadata is acquired through extraction of named entities such as persons, organizations and locations, from the textual stories. The extracted named entities are mapped to existing resources available on the Web, such as locations from Geonames, and persons from DBPedia. The data set in this use case consists of news items from 2006, including the World Cup football.

A screenshot of the faceted interface from ClioPatria is shown in Fig. 6. The top part contains four facets: `document type`, `creation site`, `event` and `person`. The result viewer, visible below the facets, contains news items related to the keyword "zidane". The current query is shown in the header of the result viewer. The user can extend the query by selecting values from the facets. In this case the value "photo" is selected from the `document type` facet. The other facets only contain values that correspond with the current result set. Note, this prevents the user from constructing queries that lead to an empty result set.

```
:CreationSite
    a :Facet ;
    :label "creation_site"@en ;
    :target (newsml:newsItem) ;
    :property newsml:locCreated ;
    :hierarchy [
        a :Hierarchy ;
        :label "Geonames_location_hierarchy"@en ;
        :relation geo:parentFeature ;
        :root geo:World
    ].
:DepictedPerson
    a :Facet ;
    ...

:ContentView
    a :FacetView ;
    :label "content"@en ;
    :label "content"@nl ;
    :target (newsml:newsItem ;
    :facets
    (    :DocumentType
         :DepictedEvent
         :DepictedPerson
         :CreationSite
    ) .
```

Listing 3. Excerpt of the facet and facet view configuration for a news demonstrator

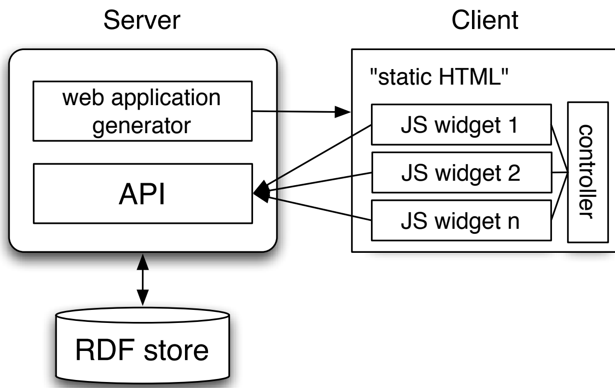

Figure 8. Simplified overview of the ClioPatria architecture, based on server-side generation of application pages with parameterized JavaScript widgets. Widgets request data from the server using AJAX and communicate with other widgets through a central controller.

Listing 3 shows an excerpt of the facet and facet view configuration for a news demonstrator. The creationSite facet applies to instances of the class NewsItem, as indicated by the value of the `:facetTarget` property. This facet will display values from the property newsml property locCreated. As the values are part of a geographical containment hierarchy, this is used for the organization. In the screenshot of Fig. 6 the creationSite facet it is visible that initially only the children of the hierarchy root (e.g. World) are shown (e.g. Europe, Africa and Asia). When one of these values is selected, the children available through the hierarchyRelation, geo:parentFeature, become available. Four facets are grouped into a facet view that covers the content of news items. In a similar fashion other facets can be grouped into views on the production and document characteristics. The facet view menu shown in the screenshot on the top left allows the user to select one of these facet views.

IMPLEMENTATION

The interface models have been implemented within the open source semantic search and annotation ClioPatria framework [9]. The overall architecture of ClioPatria is described in [12], here we focus on the parts that are relevant for the interface model. Figure 8 shows a simplified overview of the ClioPatria architecture. Server-side the system has a single triple store that contains all RDF data including the interface models and the mappings between the domain specific data and these models. In response to a request from the user the server generates an HTML page with static HTML and parameterized JavaScript widgets. ClioPatria uses the interface mappings to generate the appropriate parameter settings for the widgets. For example, the JavaScript initialization function of the Who field in the annotation application would be generated as:

```
<script type="text/javascript">
var WhoField = new YAHOO.mazzle.AutoComplete(
    "elInput",
    "elContainer",
    "/api/autocomplete",
    {   callback: {fn:addTag, args:["person"]},
        filter:[type:"ulan:Person"],
        match:[skos:prefLabel, rdfs:label],
        sort:["exact", "label"],
        smushing:true,
        info:{
            altLabel:"skos:prefLabel",
            subLabel:[
                "ulan:role", "ulan:nationality",
                "ulan:birthDate", "ulan:deathDate"
            ],
            postLabel:"skos:inScheme",
            description:"ulan:biography",
            image:"vra:subject"
        }
    });
</script>
```

The widgets populate themselves by requesting JSON data from the server through one of the HTTP APIs. The configuration of the widget determines what information is requested. Widgets within a single web application communicate with each other through a controller, which is itself a JavaScript component. For example, selecting a value in one facet sends an update message to the controller that updates the internal state and tells the registered components to update themselves. A client-side widget can also be used independent of ClioPatria's web application generator and the RDF interface model by manually defining the JavaScript function in a static HTML page.

CONCLUSION AND FUTURE WORK

We have shown how we can use RDF to model the interface widgets of a specific Web application, an abstract intermediate data model, and the mapping between these two models. We argue that this approach can provide developers with an interface early in the development cycle of a Semantic Web application. As long as the chosen widgets, associated interface model and intermediate model prove to be sufficiently rich, all the developer needs to do is to provide the mappings (in RDF) between his own data model and the intermediate model, using skills that Semantic Web developers can be safely assumed to possess. This approach also allows Semantic Web UIs to be built on top of existing Web tool kits, without sacrificing the heterogeneity and semantic richness of the underlying data.

A first drawback of our approach is that our interface models are typically specific for a given interface widget or toolkit. If the same RDF data needs to be displayed the same way in multiple interfaces, a vocabulary such as Fresnel, that abstracts from the interface technology used, might be a better alternative. In our applications, we have aimed to fully exploit the functionality of the interface widgets, and have traded the advantages of extra functionality against generality. Other developers might make a different trade off.

A second drawback surfaces when a given set of widgets and the associated interface model provides insufficient function-

[9] `http://e-culture.multimedian.nl/software/ClioPatria`

ality. Then, extensions will require traditional Web scripting skills to develop extensions to widget set, typically involving a mix of HTML, CSS and JavaScript. But it also requires skills to be able to model these extensions in RDF or OWL, and this combination of skills might be hard to find.

For future work, we would improve upon our current interface model and its implementation. The current implementation is realized as an integral part of the ClioPatria server framework, and we are investing ways to be able to apply the same approach to create interfaces on top of arbitrary SPARQL endpoints.

ACKNOWLEDGEMENTS

The dataset used in the NewsML demonstrator has been kindly provided by AFP. We like to thank all members of the MultimediaN E-Culture Project for creating an environment that allows experimenting with Semantic Web technologies in a real life setting. In particular, we like to thank Jan Wielemaker for the cooperative development on ClioPatria and the inspiring feedback. We also like to thank our colleagues Raphaël Troncy and Michiel Kauw-A-Tjoe for the contributions to ClioPatria and setting up the news demonstrator.

This research was supported by the MultimediaN project funded through the BSIK programme of the Dutch Government and by the European Commission under contract FP6-027026, Knowledge Space of semantic inference for automatic annotation and retrieval of multimedia content — K-Space.

REFERENCES

1. M. Hildebrand, J. van Ossenbruggen, A. Amin, L. Aroyo, J. Wielemaker, and L. Hardman. The design space of a configurable autocompletion component. Technical Report INS-E0708, CWI, November 2007.

2. M. Hildebrand, J. van Ossenbruggen, and L. Hardman. /facet: A Browser for Heterogeneous Semantic Web Repositories. In *The Semantic Web - ISWC 2006*, pages 272–285, November 2006.

3. D. Huynh, D. Karger, and R. Miller. Exhibit: Lightweight structured data publishing. In *16th International World Wide Web Conference*, Banff, Alberta, Canada, 2007. ACM.

4. E. Hyvönen, M. Junnila, S. Kettula, E. Mäkelä, S. Saarela, M. Salminen, A. Syreeni, A. Valo, and K. Viljanen. MuseumFinland — Finnish museums on the semantic web. *Journal of Web Semantics*, 3(2-3):224–241, October 2005.

5. E. Mäkelä, K. Viljanen, O. Alm, J. Tuominen, O. Valkeapää, T. Kauppinen, J. Kurki, R. Sinkkilä, T. Känsälä, R. Lindroos, O. Suominen, T. Ruotsalo, and E. Hyvönen. Enabling the semantic web with ready-to-use web widgets. In *Proceedings of the First Industrial Results of Semantic Technologies Workshop, ISWC2007*, November 11 2007.

6. m.c. schraefel, D. A. Smith, A. Owens, A. Russell, C. Harris, and M. L. Wilson. The evolving mSpace platform: leveraging the Semantic Web on the Trail of the Memex. In *Proceedings of Hypertext 2005*, pages 174–183, Salzburg, 2005.

7. E. Pietriga, C. Bizer, D. R. Karger, and R. Lee. Fresnel: A browser-independent presentation vocabulary for rdf. In I. F. Cruz, S. Decker, D. Allemang, C. Preist, D. Schwabe, P. Mika, M. Uschold, and L. Aroyo, editors, *International Semantic Web Conference*, volume 4273 of *Lecture Notes in Computer Science*, pages 158–171. Springer, 2006.

8. G. Schreiber, A. Amin, L. Aroyo, M. van Assem, V. de Boer, L. Hardman, M. Hildebrand, B. Omelayenko, J. van Ossenbruggen, A. Tordai, J. Wielemaker, and B. J. Wielinga. Semantic annotation and search of cultural-heritage collections: The multimedian e-culture demonstrator. *J. Web Sem.*, 6(4):243–249, 2008.

9. SIMILE. Longwell RDF Browser. http://simile.mit.edu/longwell/, 2003-2005.

10. O. Suominen, K. Viljanen, and E. Hyvönen. User-centric faceted search for semantic portals. In *Proceedings of the European Semantic Web Conference ESWC 2007, Innsbruck, Austria*. Springer, June 4-5 2007.

11. R. Troncy. Bringing the iptc news architecture into the semantic web. In A. P. Sheth, S. Staab, M. Dean, M. Paolucci, D. Maynard, T. W. Finin, and K. Thirunarayan, editors, *International Semantic Web Conference*, volume 5318 of *Lecture Notes in Computer Science*, pages 483–498, Berlin Heidelberg, November 2008. Springer.

12. J. Wielemaker, M. Hildebrand, J. van Ossenbruggen, and G. Schreiber. Thesaurus-based search in large heterogeneous collections. In A. P. Sheth, S. Staab, M. Dean, M. Paolucci, D. Maynard, T. W. Finin, and K. Thirunarayan, editors, *International Semantic Web Conference*, volume 5318 of *Lecture Notes in Computer Science*, pages 695–708, Berlin Heidelberg, November 2008. Springer.

13. 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, Ft. Lauderdale, Florida, USA, 2003. ACM Press.