UNIVERSITY OF AMSTERDAM

UvA-DARE (Digital Academic Repository)

End-user support for access to heterogeneous linked data

Hildebrand, M.

Link to publication

Citation for published version (APA):
Hildebrand, M. (2010). End-user support for access to heterogeneous linked data

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.

UvA-DARE is a service provided by the library of the University of Amsterdam (http://dare.uva.nl)

Download date: 19 Aug 2017

Chapter 4

Case study II: Faceted browsing

In this chapter we investigate the exploration of multiple heterogeneous linked data sources through faceted browsing. We explore solutions for the required search functionality and presentation methods by the implementation of a prototype system. We focus on the use of semantics present in the data to extend traditional facet browsing functionality i) to support the formulation of structured queries on linked data and ii) to improve presentation methods to organise the large number of navigation paths. The study shows that generic faceted browsing functionality can be defined directly on top of the RDF triple model. The extra semantics available in the data provide a means to extend this functionality and to organise the navigation paths.

This chapter was published as "/facet: A Browser for Heterogeneous Semantic Web Repositories" in the Proceedings of the International Semantic Conference 2006 (Hildebrand et al. 2006) and was co-authored by Jacco van Ossenbruggen and Lynda Hardman.

4.1 Introduction

Facet browser interfaces provide a convenient and user-friendly way to navigate through a wide range of data collections. Originally demonstrated in the Flamenco system (Yee et al. 2003), facet browsing has also become popular in the Semantic Web community thanks to MuseumFinland (Hyvönen et al. 2005) and other systems (SIMILE 2005). An individual facet highlights one dimension of the underlying data. Often, the values of this dimension are hierarchically structured. By visualising and navigating this hierarchy in the user interface, the user is able to specify constraints on the items selected from the repository. To use an example from the art domain: by navigating the tree associated with a "location created" facet from the root "World", via "Europe" to "Netherlands", the results set is

constrained to contain only paintings that have been painted in the Netherlands. By combining constraints from multiple facets, a user is able to specify relatively complex queries through an intuitive Web navigation interface. All values of a dimension that would lead to an over-constrained query are dynamically removed from the interface, preventing the user from running into frustrating dead ends containing zero results.

We are working on repository exploration in the context of a national e-culture project (Schreiber et al. 2006). Our project's goals are similar to those of MUSEUMFINLAND, and aim at providing a syntactic and semantic interoperable portal to on-line collections of national museums. A major difference, however, is that we work with each museum's original metadata as much as possible. This means, for example, that we do not map all metadata relations of the various museums to a common set of ontological properties, nor do we map all metadata values to terms from a common thesaurus or ontology.

Initially, we experimented with traditional facet browsers, which assume a fixed set of facets to select and navigate through relatively homogeneous data. This, however, conflicts with our approach for the following reasons. First, our data set is too diverse to use a single set of facets: facets that make perfect sense for one type of object are typically inappropriate for other types. A related problem is that we cannot fix the facets at design time. When new data is added, the system should be able to add new facets at run time. This requires an extension of the facet paradigm to cater for objects of multiple types, to associate a set of appropriate facets to each type dynamically and to navigate and search larger sets of facets. Second, we use a rich and extensive set of art-related background knowledge. As a result, users expect to be able to base their selection not only on facets of museum artefacts, but also on facets from concepts from the background knowledge, such as artists and art styles. This requires two other extensions: one that allows users to switch the topic of interest, for example from artworks to art styles; and another one that allows selection of objects of one type based on the facets of another. For example, a set of artworks can be selected based on the properties (facets) of their creators.

This article discusses these extensions as they are realised in /facet, the browser of the project's demonstrator[1]. The article is structured as follows. The next section introduces a scenario to illustrate the requirements for enhanced facet-browsing across multiple types. Section 4.3 discusses the requirements in detail and section 4.4 explains our design solutions in a Web-based interface. Section 4.5 discusses related work and open issues.

[1] See (Schreiber et al. 2006) for a more detailed description and http://e-culture.multimedian.nl/art/facet for an on-line demo of /facet.

4.2 Example scenario

Throughout the chapter, we use examples from the art domain. The system itself, however, is domain independent and used on several other domains[2]. The scenario is divided in two parts: the first part illustrates typical usage of facet browsers; the second part illustrates search tasks that go beyond the current state of the art and introduce new requirements for facet browsers.

Our protagonist is Simon, a high school student who recently visited the Dutch Kröller-Müller museum. The museum's collection features several works from Vincent van Gogh. Back at home, Simon has to write an essay on post-impressionism. He remembers seeing a particular post-impressionist painting but can no longer remember the name of the painter nor the title of the painting. The only thing he remembers is that the painting depicted a city street at night time. He uses a facet browser to restrict the search space step by step. He selects the current location of the painting (Kröller Müller), the art style of the painting (post-impressionist), its subject type (cityscape), and the subject time (night). He finds the painting he was looking for among the few results matching his constraints (Vincent van Gogh's "Cafe Terrace on the Place du Forum").

He now wants to further explore the work of Van Gogh, and selects this painter from the creator facet, and resets all previous selections. The interface displays the 56 paintings from Van Gogh that are in the repository. The facets now only contain values of the remaining paintings. For example, the create location facet instantly shows Simon that van Gogh made paintings in the Netherlands and in France, and how many in each country. Simon asks the system to group the results on create location and notices the significant difference in the color palette Van Gogh used in each country. By selecting "France" he zooms in further to explore potential differences on the city level.

In addition to the types of browsing possible in typical facet browsers, Simon also wants to explore works from painters born in the area of Arles. Unfortunately, the artworks in the repository have not been annotated with the birthplace of their creator. Simon uses multi type facet browsing and switches from searching on artworks to searching on persons. The interface now shows the facets available for persons, which include place of birth. Searching on Arles, he sees that four painters with unfamiliar names have been born here, but that the repository does not contain any of their works. Expanding his query by navigating up the place name hierarchy, he selects artists from the Provence-Alpes-Côte d'Azur, the region Arles is part of. He quickly discovers that Paul Cézanne, a contemporary of Van Gogh, was born in Aix-en-Provence in the same region.

Simon reaches his original goal by switching back from searching on persons to searching on artworks. Despite this switch, the interface allows him to *keep* his

[2] Demos on various domains are available at the /facet website http://slashfacet.semanticweb.org/

constraint on Provence-Alpes-Côte d'Azur as a place of birth. It thus shows only artworks created by artists that were born in this region.

Backstage area . The experimentation environment in which /facet was developed, contains sufficient data to cover the scenario above. It uses a triple store containing three different collections with artwork metadata: the collection of the Dutch National Museum of Ethnology[3], the ARIA collection from the Rijksmuseum[4], and Mark Harden's Artchive collection[5]. RDF-versions of WordNet[6] and the Getty AAT, ULAN and TGN thesauri[7] are also included. For the annotation schema, we use Dublin Core[8] and VRA Core 3[9].

In total, the store contains more than 10.8 million RDF triples. Artwork images are served directly from the websites of the museums involved. All the collection metadata has been converted to RDF, with some minimal alignment to fit the VRA Core 3 schema. In addition, explicit links were created from the art works to the Getty thesauri: literal names of painters and other artists were automatically converted to a URI of the ULAN entry; literal names of art styles and art materials to a URI of the AAT entry; and literal place names to a URI of the TGN entry. For example, in the scenario, some `vra:Works` have a `dc:creator` property referring to the painter `ulan:Person` Paul Cézanne, born in `tgn:Place` Aix-en-Provence in the `tgn:Region` of Provence-Alpes-Côte d'Azur. Additionally, some artworks have been manually annotated using concepts from the Getty thesauri and WordNet. In the remainder of this chapter, we will use the following prefixes for the corresponding namespaces: `wn`, `aat`, `tgn`, `ulan`, `vp`, `dc` and `vra`.

4.3 Requirements for multi-type facet browsing

While the second half of the scenario sketches a seemingly simple means of accessing information, a number of issues have to be addressed before it can become a reality. Most facet browsers provide an interface to a single type of object. Including multiple types, however, leads to an explosion in the number of corresponding properties and thus the number of available facets. A facet browser still needs to be able to present instances of all the types and allow a user to select a particular type of interest. In addition, the relations between the types also need to be made

[3] http://www.rmv.nl/

[4] http://www.rijksmuseum.nl/collectie/, thanks to the Dutch CATCH/CHIP project (http://chip-project.org/) for allowing us to use their translation of the dataset to RDF.

[5] http://www.artchive.com/

[6] http://www.w3.org/2001/sw/BestPractices/WNET/wn-conversion.html

[7] http://www.getty.edu/research/conducting_research/vocabularies/, used with permission

[8] http://dublincore.org/documents/dcq-rdf-xml/

[9] http://www.w3.org/2001/sw/BestPractices/MM/vra-conversion.html

explicit and selectable by the user. To a large extent, the requirements we discuss are a direct consequence of these two key points.

4.3.1 Dynamically selecting facets

Fortunately, a first way to deal with the increased number of facets lies in the facet paradigm itself. One of the key aspects of all facet browsers is that, while constraining the dataset, all links that would lead to an over-constrained query are automatically removed from the interface, thus protecting the user against dead ends. As a consequence, if no instance in the current result set has a particular property, the facet associated with this property is removed from the interface. In our multiple type scenario, this means that if two types have no properties in common, the entire set of facets displayed is replaced when the user switches from one type to the other.

Facets in context of the `rdfs:subClassOf` hierarchy For most classes that have no subclasses, just hiding facets of properties that have no corresponding instances will result in an interface with a set of facets that intuitively belong to instances of that class[10]. For the super-classes, however, it is not immediately obvious what this "intuitive" set of facets is.

A first possibility is to associate with a specific class the *union* of the facets of its subclasses. This has the advantage that users can immediately start browsing, even if they have selected a class too high up in the hierarchy. By selecting a facet that only applies to instances of one of the subclasses, the result set is automatically constrained to instances of the intended class. A major drawback is that the number of facets displayed rapidly grows when moving up the class hierarchy, culminating in the complete set of all facets for `rdf:Resource`.

An alternative is to use the *intersection* of the facets of the `rdfs:subClassOf` hierarchy. This has the advantage that the user only sees facets that are common to all subclasses, and in practice these are, from the perspective of the super-class, often the most important ones. A drawback is that when moving up the hierarchy, one quickly reaches the point where the intersection becomes empty, leaving no facet to continue the search process. This forces the user to navigate down the class hierarchy to return to a usable facet interface.

A final possibility is to view the association of a set of facets with a certain class as an aspect of the personalisation of the system. While personalisation is one of the key aspects in our project, it is beyond the scope of this thesis.

Facets in context of the `rdfs:subPropertyOf` hierarchy As described above, the `rdfs:subClassOf` hierarchy helps to reduce the number of facets by only showing

[10] For simplicity, we ignore the question of whether or not to show sparsely populated properties, of which only a few instances have values.

facets that are relevant to a particular class. A similar argument applies to the `rdfs:subPropertyOf` hierarchy. On the one hand, the property hierarchy worsens the problem by introducing even more facets: in addition to the facets corresponding to the "leaf node" properties, their super-properties also become facet candidates. On the other hand, the property hierarchy also provides an opportunity for an interface to organise and navigate the property (and thus the facet) hierarchy, allowing the user to select facets as part of the interaction.

4.3.2 Search in addition to navigation

While the beauty of facet browsing lies in the ease of constructing queries by navigation, an often heard critique is that navigating deep tree structures is complex, in particular for users who are not expert in the domain modelled by the hierarchy. A second critique is that facet browsers become complex in applications with many facets and when users do not know what facets to use for their task. Multi-type facet browsing only makes this problem worse, by radically increasing the number of facets in the system. A search interface in addition to the navigation interface is thus required, and the two interaction styles should be well integrated and complement each other.

4.3.3 Creating multi-type queries

The example of selecting artworks created by artists born in a particular region requires a facet on a object of one type (`ulan:Person`) to be applied to find objects of another type (`vra:Work`). This is just one example of how such combinations can be used to exploit background knowledge in the selection process.

Using facets across types only makes sense if the objects involved are semantically related so the browser is required to know which relation to use. For the end user, the power of the facet interface lies in the ease of combining multiple facets to construct a complex query. This should be no different in a multi-type browser. So in addition a transparent interface needs to be available to easily constrain a dataset of one type, based on facets of another type.

4.3.4 Run-time facet specification

Manual definition of relevant facets and hard-coding them in a facet browser might be feasible for homogeneous data sets. This approach does not scale, however, to heterogeneous data sets, where typically each type of object has its own set of associated facets. Even for simple applications, the total number of facets might rapidly grow to several hundreds. Instead of hard-coding the facets in the browser software, some means is needed to externalise the facet definitions and make them configurable independently from the software. This simplifies maintenance and, by simply reloading a new configuration, allows adding and changing facets at

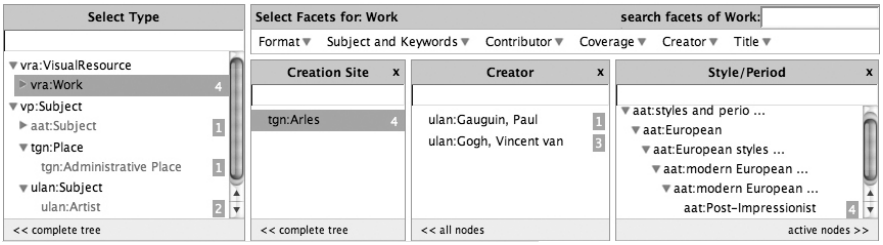

Figure 4.1: Snapshot of the /facet GUI. `vra:Work` has been selected in the type facet on the left, so only facets applicable to artworks are visible. Simon has restricted the results to have `tgn:Arles` as the place of creation. The interface shows Simon that the four matching paintings are created by either `ulan:Gauguin` or `ulan:Van Gogh` (picked from a flat list of artists), and that all four have `aat:post-impressionist` as the art style (shown in the context of its place in the AAT hierarchy).

runtime. The system also needs to be able to derive facet configurations from the dataset automatically. This allows the facet browser to run instantly on any dataset without prior manual configuration, while also allowing later manual refinement of the generated configuration. The latter is important, since it allows developers to tune the interface for specific end users, who might not be best served by a generic tool that gives access to all data.

4.3.5 Facet-dependent interfaces

A typical facet browser visualises the possible values of the facet either as a hierarchy or as a flat list. Related interfaces, such as those of mSpace (m.c. schraefel et al. 2005) and Piggybank (Huynh et al. 2005), have shown that some facets are better shown using a more specialised visualisation or interaction technique, such as geographical data displayed in an interactive map. To be able to tune a generic, multi-type facet browser to a tool for end-users that have a specific task in a specific domain, we require a mechanism for supporting visualisation and interaction plug-ins.

4.4 Functional design for multi-type facet browsing

We have explored the design consequences of these requirements in /facet. This section explains and motivates our design decisions in the prototype.

4.4.1 Browsing multiple object types

To support facet search for all object types, the /facet user interface needs a way to search for objects other than artworks[11]. A natural and convenient way to integrate such functionality is by regarding the `rdf:type` property as "just" another facet. The facet applies to all objects and the values from its range are typically organised by the `rdfs:subclassOf` hierarchy, allowing navigation just as for any other facet. Since the semantics of this facet is derived directly from that of `rdfs:type`, by making a selection users indicate the type of object they are interested in. This constraint automatically selects which other facets are also active.

This is illustrated in Figure 4.1, which shows the upper half of the /facet interface. On the left is the type facet with a part of the domain's class hierarchy. Simon has already selected artworks (e.g. objects of `rdf:type vra:Work`) and, as a result, only facets applicable to artworks are available from the facet bar at the top. Simon has expanded three of these from sub-menus of the facet bar: Creation Site, Creator and Style/Period. He selected "Arles" in the Creation Site facet. Apparently, the dataset contains only four objects of type `vra:Work` that were painted in Arles, indicated next to the selected type and location `tgn:Arles`. Simon has made no selections in the Creator and Style/Period facets, indicating that all four paintings are "post-impressionist" and that one painting is by Gauguin and three are by Van Gogh.

4.4.2 Semantic keyword search

In Figure 4.1, the art style's full path in the AAT concept hierarchy is automatically unfolded because all paintings with "Arles" as the Creation Site share the same style "post-impressionist". Showing the tree structure has the advantage that Simon could quickly select related art styles by simply navigating this hierarchy. This illustrates a well-known disadvantage of navigating complex tree structures: if Simon had instead started by selecting the art style, he would need to have known the AAT's art style classification to navigate quickly to the style of his choice, which is hidden six levels deep in the hierarchy.

To overcome this problem, we added a keyword search box to each facet, with a dynamic suggestion facility, (b) in Figure 4.2. This allows Simon to find the style of his choice based on a simple keyword search. This interface dynamically starts suggesting possible keywords after Simon has typed a few characters. Note that the typical "no dead ends" style of facet browsing is retained: only keywords that produce actual results are suggested. Backstage, this means that in this case the suggested keywords are picked from the (labels of) concepts under the AAT "Style and Periods" subtree that are associated with art works in the current result set. In practice, the intended keyword is typically suggested after only a few keystrokes.

[11] We still use artworks as the default type to give users a familiar interface when starting up the browser.

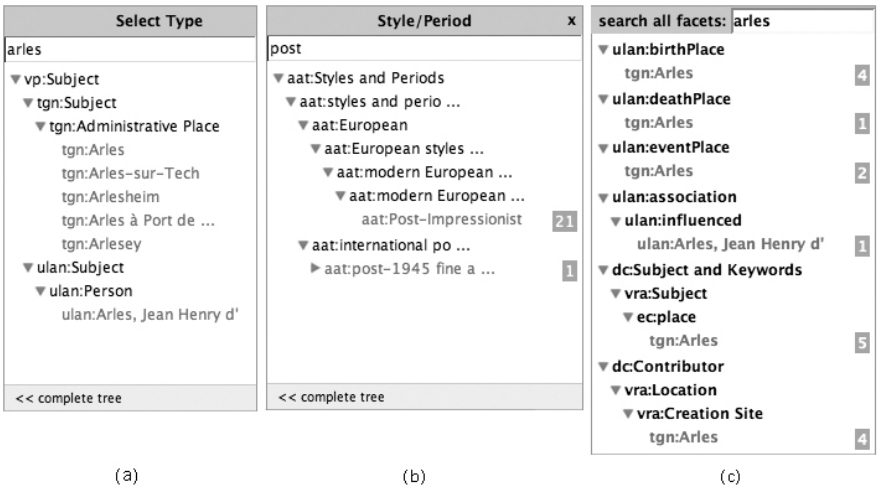

Figure 4.2: Three types of keyword suggestion and search. (a) show search on all instances, helping to select the right type. (b) shows search within a single facet, helping to move in complex facet hierarchies. (c) searches across all active facets, showing the user the different uses of the keyword "Arles" in different facets.

This makes the interaction often faster than navigating the tree, even for expert users who know the tree structure by heart.

The keyword search discussed above addresses the problem of navigation difficulties within the hierarchy of a single facet. Another problem could be picking the right facet in the first place. The keyword search box shown in (c) of Figure 4.2 addresses this problem. It provides the same search as the facet keyword search in (b), only across all facets of the selected type. For the figure, no type was selected and all facets have been searched. Arles is suggested as a TGN concept used in the facet corresponding to the `vra:location.creationSite` property (for paintings created in Arles), but also as the place used in the facet of the `vra:subject` property (for paintings that depict Arles), the birth and death place of Persons, etc. As a result, this search box can be used to find the right facet, but also to disambiguate keywords that have different meanings or are used in different ways.

A final problem can be that the user does not know the type to select to start with. This is addressed by adding also a keyword search box to the type facet, as shown in (a) in Figure 4.2. This searches over all literal properties of all instances and highlights matching instances and their types in the context of their location

Figure 4.3: Facet search on type `vra:Work`, but with a still active constraint on ulan:Person (birthplace Provence-Alpes-Côte d'Azur). Also note the timeline in the bottom, visualising multiple time-related facets. Images courtesy of Mark Harden, used with permission.

in the class hierarchy.

4.4.3 Specifying queries over multiple object types

We strive to support selection of facets from objects with different types in a transparent way, without further complicating the interface. In the example scenario, Simon searched on objects of `ulan:Person`, selecting
`ulan:Provence-Alpes-Côte d'Azur` as the place of birth.

After making this selection, Simon can just switch back to searching on art-

works by selecting `vra:Work` in the type facet. In /facet, this would yield a page such as the one shown in Figure 4.3. Note that under the facets, the currently active constraints are shown, including the `ulan:Provence-Alpes-Côte d'Azur` constraint on the `ulan:birthPlace` facet of `ulan:Person`. For comparison, also a facet on `vra:Work` has been selected, in this case `ulan:Paul Cezanne` as the `dc:creator`.

To realise the example above, the facet browser needs to know the relation that can be used to connect a set of `vra:Works` with a set of `ulan:Persons` born in `ulan:Provence-Alpes-Côte d'Azur`. The current prototype searches for such properties at run time, and in this case finds the `dc:creator` property, as intended. To keep the user interface simple, we only support one property (that is, the first suitable candidate found by the system) to connect the different sets. Properties with the same domain and range can be used for normal facet browsing within a single type, but not for relating instances of different types.

4.4.4 Run-time facet specification

The facets that are shown in the interface can be configured in a separate file. Because a facet is defined in terms of RDF classes and properties, the configuration file itself is also in RDF, using a simple RDF vocabulary.

The vocabulary defines instances of `Facet` by three key properties. For example, the `birthday` facet is modelled by the `hierarchyTopConcept` and `hierarchyRelation` properties defining the hierarchy to be shown in the interface, by specifying the top of the tree (`tgn:World`) and the `rdf:Property` used for the hierarchical relation (in this case `vp:parent`, the universal parent relation that is used across the Getty vocabularies). The `resourceProperty` defines how places are related to the painters, in this case by the `ulan:birthPlace` property.

Some other properties are optional and used to speed up or improve the user interface. The explicit definition of the type of objects the facet applies to, for example, makes it much more efficient to quickly switch to the right set of facets when users move from one type to the other. The `rdfs:label` property can be used to specify the name of the facet, which defaults to the label or name of the corresponding property.

To generate a first configuration file (that can later be hand edited), /facet analyses the dataset and generates a set of RDF facet definitions similar to the birthPlace facet example given above. For each property, the current algorithm search for a hierarchical relation in the set of related values to find the top concepts. If this relation is not found, or if the values are literals, it generates a facet with a flat list of values. For the scenario dataset, 22 hierarchical facets, 84 literal facets and 154 facets with a flat list of terms were found.

4.4.5 Facet-specific interface extensions

The values of a facet are typically presented in a list or a tree structure with textual labels. However, some structures are more easy to understand when presented differently. In particular, data which can be ordered linearly can be presented as points on an axis. Time, in particular, is a quantity that is often associated with objects, not only in the cultural heritage domain. It is useful to give a timeline representation of date data where this is appropriate. We have developed a timeline plug-in to visualise time-related facets (such as `dc:date` and its sub-properties).

Not only artworks have associated dates, but also related objects such as the life-span of the artist, Van Gogh, and the period associated with the `aat:post-impressionist` art style. Since the temporal information is related to the set of objects, this can be displayed together on a single timeline, as shown on the bottom of Figure 4.3.

A timeline interface could also be extended to not only show the temporal information, but also allow it to be used as part of the facet constraint mechanism. A similar facet dependent interface extension would be to relate geographical information together and display it on a two-dimensional spatial-axes interface such as a map.

4.5 Discussion and Related Work

Initial development of /facet has been heavily inspired by the facet interface of the MUSEUMFINLAND portal (Hyvönen et al. 2005). Where MUSEUMFINLAND is built on a strongly aligned dataset, we focus on supporting heterogeneous, loosely coupled collections with multiple thesauri from external sources. They provide mapping rules that hide the peculiarities of the underlying data model from the user interface. We have sacrificed this abstraction level and expose the underlying data model, but with the advantage that the software is independent of the data model and requires no manual configuration or mapping rules.

In comparison with mSpace (m.c. schraefel et al. 2005), /facet retains the original facet browsing principle to constrain a set of results. In a visually oriented domain such as ours, this leads to an intuitive interface where, after each step, users can see a set of images that reflect their choices: even users who do not know what "post-impressionist" paintings are, can immediately see from the results whether they like them or not. Also note that a heterogeneous dataset, such as ours, would lead to an m-dimensional space with $m > 250$, which would make the mSpace interface unusable. Alternatively, we could split up the data in multiple smaller mSpaces, but would then have no way of connecting them.

Unlike /facet, the Simile project's Longwell (SIMILE 2005) facet browser requires to be configured for a specific dataset and its interface provides no solutions for dealing with large numbers of facets. An advantage of Longwell over /facet is

that the display of the results is fully configurable using Fresnel (Bizer et al. 2005).

While Noadster (Rutledge et al. 2005) is not specifically a facet browser, it is a generic RDF browser. Noadster applies concept lattices to cluster search results based on common properties. It clusters on any property, but ignores "stop properties" that occur too frequently. The resulting hierarchy forms a Table of Contents, with the original search results typically as leaf nodes, and common properties as branches. An advantage of Noadster is that its clustering prioritizes facets by placing those occurring more frequently in the matches higher in the tree. A disadvantage is the occasional "noisy" excess of properties in the clustering.

While we claim that /facet has some key advantages over the systems discussed above, the current prototype also suffers from some limitations. First, the algorithm for determining the facet configuration automatically needs further refinement. We now treat every RDF property as a potential facet. We then filter out many "schema level" properties from the RDF, RDFS and OWL namespace, and from our own internal namespaces (including the namespace we use for our facet specifications). It is still possible that a certain type of object will be associated with so many facets that the interface becomes hard to use. The techniques discussed in this chapter only partially address this problem, and they are highly dependent on the structure of the `rdfs:subpropertyOf` hierarchy. More research is needed to classify facets into a hierarchy that is optimised for usage in a user interface. In addition, we currently generate facets for all literal properties. On the one hand, this has the disadvantage that facets are generated for properties such as comments in RDFS, gloss entries in WordNet and scope notes in the AAT. The values of such properties are unlikely to become useful for constraining the dataset. On the other hand, other properties with literal values, such as labels in RDFS or titles in Dublin Core, provide useful facets. More research is needed to provide heuristics for determining the type of literal values that are useful in the facet interface.

In the type hierarchy, we display all classes from the underlying domain, filtering out only the classes from the RDF(S) and OWL namespaces and the system's internal namespaces. Still, this often leaves classes that are not helpful for most users. Examples include the abstract classes that characterise the top levels of many thesauri, such as the `vp:Subject`, `aat:Subject` and `aat:Concept` classes in our domain. The prototype's current facet-mining algorithm is unable to deal with multiple hierarchies within a single facet. For example, many of our paintings have subject matter annotations referring to concepts from WordNet. There are, however, several different relations that can be used to organise these into a hierarchy, such as hypernym/hyponym and holonym/meronym. For the user interface, it may be appropriate to merge the different hierarchies in a single tree or to keep them separate.

On the implementation side, the current prototype is developed directly on SWI-Prolog. The server side is a Prolog module built on top of the SWI-Prolog

Semantic Web Server (Wielemaker et al. 2003; Wielemaker 2005a). The client side is a standard Web browser that uses AJAX (Paulson 2005) for the dynamics of the suggestion interface. A drawback of this implementation is that users have to upload their data into /facet's triple store. We are planning to make future versions of the browser using the SPARQL (W3C 2006) API, so that /facet can be used to browse any RDF repository that is served by a SPARQL compliant triple store. The large amount of RDFS and OWL-based reasoning at run time slowed down the system and gave the impression of an unresponsive interface. To address this, we reduced the amount of run-time reasoning by explicitly deriving the triples needed for calculating the result set when starting up the server so we can quickly traverse the expanded RDF-graph at run time. For example, we compute and add closures of transitive and inverse properties to the triple set at start up or when new data is added.

4.6 Conclusion and Future Work

We have discussed the requirements for a fully generic RDFS/OWL facet browser interface: automatic facet generation; support for multiple object types; cross-type selection so objects of one type can be selected using properties of another, semantically related, type; keyword search to complement hierarchical navigation; and supporting visualisation plug-ins for selected data types.

We developed the /facet Web interface to experiment with facet browsing in a highly heterogeneous semantic web environment. The current prototype meets the requirements discussed, it fulfils the described scenario in a cultural heritage domain and similar scenarios in other domains. A number of drawbacks remain, which we would like to address in future work. First, determining the facets automatically needs further refinement. Second, we are still fine tuning which classes from the class hierarchy we want to show and which facets we want to associate with the super-classes. Third, the prototype's current facet-mining algorithm is unable to deal with multiple hierarchies within a single facet. Finally, we need to develop a version of /facet that is independent of a particular triple store implementation and runs on any SPARQL compliant triple store.