FacetOntology: Expressive Descriptions of Facets in the Semantic Web

Daniel A. Smith and Nigel R. Shadbolt

Web and Internet Science Research Group,
Electronics and Computer Science,
University of Southampton,
Southampton, UK
{ds,nrs}@ecs.soton.ac.uk

Abstract. The formal structure of the information on the Semantic Web lends itself to faceted browsing, an information retrieval method where users can filter results based on the values of properties ("facets"). Numerous faceted browsers have been created to browse RDF and Linked Data, but these systems use their own ontologies for defining how data is queried to populate their facets. Since the source data is the same format across these systems (specifically, RDF), we can unify the different methods of describing how to query the underlying data, to enable compatibility across systems, and provide an extensible base ontology for future systems. To this end, we present FacetOntology, an ontology that defines how to query data to form a faceted browser, and a number of transformations and filters that can be applied to data before it is shown to users. FacetOntology overcomes limitations in the expressivity of existing work, by enabling the full expressivity of SPARQL when selecting data for facets. By applying a *FacetOntology definition* to data, a set of facets are specified, each with queries and filters to source RDF data, which enables faceted browsing systems to be created using that RDF data.

1 Introduction

Faceted browsing is a form of information retrieval where results can be filtered based on different properties of the data, known as facets. For example a faceted browser used for a library has facets such as *Author*, *Category*, *Editor* and *Publisher*. A user can select their *Author* of choice, and see only books by that author, as well as the categories those books fall into, their editors and publishers. Faceted browsing is effective under two contrasting scenarios [1]: (a) when users know what they are looking for and can therefore limit by facets which they know the values for, and (b) when users do not know the values they are looking for, but use the facet listings to determine the possible values for a domain; this is unlike keyword searching. Keyword searching is most effective when users know what they are looking for, but not when they do not know the keywords.

The graph structure of information on the Semantic Web, where instances have multiple properties according to numerous ontologies and vocabularies is ideal for faceted browsing. In fact, numerous browsers have been developed to support faceted browsing over Semantic Web data. For example, /facet [2], Rhizomer [3], Exhibit [4], and mSpace [5] have been developed to browse RDF and Linked Data. These systems use their own specialised descriptions of how to query data, typically providing a list of predicates and classes, which are used to generate SPARQL queries over a knowledge base. All existing systems are limited in their expressivity, and therefore limit the filters that can be placed against data. We propose that standardising the method to select facets, using a method that enables the full expressivity of SPARQL, would benefit multiple faceted browsers so that a definition of the human readable facets of a domain can be described once, and then multiple faceted browsers can use that information to configure their framework. In addition, faceted definitions are themselves RDF, and can therefore be published, shared amongst collaborators, and extended, to be used as the basis for new definitions.

In this paper, we present FacetOntology[1], which defines a standard set of queries to define facets in RDF. Specifically, it defines clauses that can be used when generating SPARQL queries to select data, to produce a set of facets from source RDF data. Faceted browsing systems are then configured to use the resulting data and facets. In order to apply the transformations and configure the faceted browser frameworks, a "maker" is created for each supported faceted browsing system. The maker uses the *FacetOntology definition*, which creates a standalone faceted browser based on the facets specified in the definition. The maker harvests the raw data from the Semantic Web, and performs queries to extract the relevant data and relationships between the facets.

We demonstrate the capability of FacetOntology using two faceted browser "maker" systems that configure faceted browsers for a particular dataset and domain from a *FacetOntology definition*. Our approach creates an abstract definition of the data cleanup transformations, which means that data querying can be repeated programatically, when source data is updated. This enables faceted browsers to be up-to-date and reflect changes in the source data automatically, rather than represent the state of data from a single point in time.

We situate our work with exemplar faceted browsers using the data from the BBC iPlayer, Francophone Music Criticism, the Rich Tags bibliographic database and Usability UK knowledge base of usability methods. We demonstrate that FacetOntology is effective by applying it to these exemplars using two different faceted browser frameworks. We advance the state-of-the-art in faceted browsing by providing a standard methodology for configuring faceted browsers that enables faceted definitions to be shared and extended, with a richer expressivity than existing approaches.

In Section 2 we overview existing work in this area. Section 3 presents our FacetOntology, an ontology to define facets over multiple datasets, and in Section

[1] FacetOntology OWL representation: `http://danielsmith.eu/resources/facet/#`

4 we present a user interface for creating *FacetOntology definitions*. Then, in Section 5 we discuss our implementations of FacetOntology browser "makers" for Exhibit and mSpace, and exemplar public browsers that use FacetOntology. Finally, in Section 6 we conclude.

2 Related Work

One of the first Semantic Web browsers, Tabulator [6], enabled users to specify URIs of data, and explore that data directly. It harvested data by keeping a local knowledge base within the user's web browser, and populated that store with more data as the user browsed. Whenever the user explored information that linked to additional data, that data was harvested into the store. This approach suffers from scalability problems when more data is downloaded, and performance problems prevent it from browsing large-scale data sources.

In order to combat the scalability problems, more recent approaches populate server-side triplestores. A data harvesting process is responsible for gathering and asserting data into a server-side triplestore, and a client-side browser issues SPARQL queries to that triplestore in order to populate the facets of a faceted browser. One benefit of using a triplestore is that it does not typically directly retrieve RDF documents; that task is offloaded to the process that populates the triplestore, allowing a static set of data to sit indexed on a server, ready to be queried efficiently. Traditionally, using a triplestore would mean foregoing the gathering of additional linked data, however there is ongoing work by triplestore vendors (such as OpenLink Virtuoso) which automatically dereference URIs and crawl linked data, based on queries used.

One such approach that uses a triplestore is mSpace [7], a faceted browsing framework that runs in a web browser. mSpace supports the querying of server-side triplestores using SPARQL, and therefore does not require a user's computer to load data on-demand, allowing the interface to scale-up to larger datasets than Tabulator. The definition of the facets present in a triplestore are defined in the "mSpace Model". The mSpace model defines the RDF classes of the instances in the triplestore, and the predicate relationships that connect them. mSpace uses this information to generate SPARQL queries used to populate the facets in its interface.

The approaches of /facet [2] and Rhizomer [3] also use a triplestore. However, they use the ontologies of the data as the facets, instead of a model. This approach works by querying the store for all RDF classes, and presenting them to the user as possible facets. Following the user's initial choice, all predicates that connect to the chosen class are then available as facets. This benefit of using this approach over mSpace's model-based approach is that when new data is added to the store that uses additional predicates or RDF classes, they appear as facets automatically, without having to specify their relationships in a model. When RDF is structured and filtered for use in /facet and Rhizomer, the results present facets correctly. They provide an intuitive interface for users, and configurable system for administrators. However, in the case where data contains inconsistent information, for example when RDF data is asserted from the open Semantic Web, the methodology is unsuitable, because non-human-readable information may be exposed. This is because the Semantic Web is a general-case open world system, designed for machine-readability, where a single RDF source file can contain information on anything at all, not limited by domain or resource. Without a definition model to filter the data, extraneous information creates additional sparsely populated facets of limited use, and this problem would grow as the amount of "wild data" is loaded into the triplestore.

In addition to the work above, there have also been work that defines vocabularies to describe facets. Firstly, in the Longwell project [8], the list of facets used by its faceted browser was specified in RDF as a list of `rdf:Property` predicates, according to the Fresnel Facets Ontology[2]. For example a simple bibliographic faceted browser is specified[3] in N3 as:

```
@prefix facets: <http://simile.mit.edu/2006/01/ontologies/fresnel-facets> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix ow: <http://www.ontoweb.org/ontology/1#> .

:publicationFacets rdf:type facets:FacetSet ;
  facets:types ( ow:Publication ) ;
  facets:facets (
    dc:type
    dc:publisher
    dc:contributor
    dc:subject
  ) .
```

The benefit to this approach is that it is in RDF, and thus is machine readable, and can reference any predicate. However, it is limited because it can only list individual predicates, rather than more complex relationships.

Taking this approach further in the ClioPatria faceted browser [9], individual widgets (in particular, individual facets) are formally described using RDF against an ontology. Compared to the above approach of Fresnel in Longwell, the Cliopatria approach is richer, and supports richer facets than individual predicates. It also defines a number of specific types of facet, in order to enable specific features. For example, "Autocomplete" facets that allow users to start typing the name of the items, and "Hierarchy" facet that automatically expands a tree, given the predicate that specifies the hierarchical relationship. Their approach also defines a "facetTarget" which specified the RDF class of the type of result item. Thus, each of their facets specify facets that filter those results. However, as in the Longwell example, more complex filtering is not specified, for example querying against a chain of predicates, rather than a single property.

Hence, in this paper we extend the features and expressivity of these vocabularies, so that facets can be specified, but also enable more complex filtering than single predicates.

[2] SIMILE Longwell Fresnel Facets Ontology: `http://simile.mit.edu/2006/01/ontologies/fresnel-facets`

[3] Fresnel Facet example from: `http://simile.mit.edu/wiki/Longwell_User_Guide#Facet_Configuration`

3 FacetOntology: Expressive Definitions of Faceted Metadata

In this section, we describe our ontology "FacetOntology", which can be used to describe facets and their relationships from multiple RDF sources. It provides a vocabulary for describing a set of facets, and for each facet it defines descriptions of its data source. The descriptions describe RDF classes and predicate relationships, which define links to other facets. These descriptions allow SPARQL queries to be generated, the results of which are used to populate a facet's values. In order to enable the full expressivity of SPARQL to be used when selecting data, we have defined "SPARQL hinting" values to be used in order to further filter data. The description results in a bounded human-readable abstraction of a subset of the RDF data sources, that can be used by a faceted browser.

In order to illustrate how FacetOntology is used, we use a running example with data from the Classical Music domain. Our example dataset is comprised of the details about tracks of music, specifically a piece's title, composer and album. It features the music from the album "Best of Bach" that is composed by the composer "Johann Sebastian Bach", and is serialised in N3, see Figure 1.

```
@prefix : <http://facetontology.example.com/data#> .
@prefix facet: <http://danielsmith.eu/resources/facet/#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

:album1 a :Album .
:album1 :track :piece1 .
:album1 :track :piece2 .
:album1 :composer :composer1 .
:album1 rdfs:label "Best of Bach" .

:piece1 a :Piece .
:piece1 rdfs:label "Cantata BWV 1" .
:piece1 :originalName "Wie schon leuchtet der Morgenstern" .

:piece2 a :Piece .
:piece2 rdfs:label "Cantata BWV 2" .
:piece2 :originalName "Ach Gott, vom Himmel sieh darein" .

:composer1 a :Composer .
:composer1 rdfs:label "Johann Sebastian Bach" .
```

Fig. 1. Example N3 definition of the data used by our running example from the Classical Music domain.

3.1 Facet Definitions

A *FacetOntology definition* is used to describe facets within a dataset, the descriptions are organised into a *FacetCollection* which is a set of facets used to describe a particular dataset. The items contained in a *FacetCollection* are a *FirstOrderFacet* and one or more *ConnectedFacet*s. In more detail:

1. A *FirstOrderFacet*, is described by an RDF class (using the `rdf:type` predicate). For example, the *Piece* facet is comprised of instances of type `:Piece`, as shown in Figure 1.
2. The *ConnectedFacet*s, are described by how they connect in the data graph in relation to the first-order facet. This is modelled as a chain of predicates that are used to generate the SPARQL query that gathers the data for the values of a facet. For example, the *Album* facet relates to the *Piece* facet via the `:track` predicate, as shown in Figure 1.

In our classical music example, there are three classes, *Piece*, which represents a single piece of music with its title, *Album*, which represents a grouping of pieces, and *Composer*, which represents the composer of music. These classes are linked using the `:track` predicate and the `:composer` predicate, and all of these classes have various attributes modelled using different predicates. In our FacetOntology definition for this data, the *Piece* class is used to define the *FirstOrderFacet*, with the predicate `:track` being used as the predicate chain to define the *Album* facet. The predicate chain is used in the two following SPARQL queries to harvest data to populate the facet's values in the classical music's *FacetCollection*, where the variables `?label` and `?uri` gather the label and URI, respectively.

1. Query to gather the first-order facet of *Piece*:

   ```
   SELECT ?label ?uri WHERE {
          ?uri rdf:type :Piece .
          ?uri rdfs:label ?label }
   ```

   This query returns a table of URIs and labels that represent all of the items to populate the *Piece* facet's values.

2. Query to gather connected metadata about the *Album* (where the variable `?firstorder` will contain the URI of the *Piece* in the first-order facet:

   ```
   SELECT ?label ?uri ?firstorder WHERE {
          ?firstorder rdf:type :Piece .
          ?uri :track ?firstorder .
          ?uri rdfs:label ?label }
   ```

   This query returns a table of URIs and labels to populate the values of the *Album* facet, as well as the URI of *FirstOrderFacet* items (*Piece*s, in this case) on the album. A faceted interface can use these URIs to make the link between the *FirstOrderFacet*s and the *Album* facet, so that when users filter on an album, the pieces that are on it can be shown. The predicate that connects a *Piece* to an *Album* is directional from the *Album* to the *Piece*, we must also indicate that the direction of this predicate is reversed, using the `facet:reverse` directive.

In order to define the *Composer* facet, we must define a predicate chain that first joins to *Album*, and then to *Composer*, as the ontology that defines the classical music data describes composers as having composed albums, and there is no direct link to the individual pieces. As such, the same predicate definition that is used for the *Album* facet (see above) is first defined, and then a predicate `composedAlbum` is described, in order to complete the predicate chain.

After using FacetOntology to define the facets in our classical music example, it produces the following facet descriptions in RDF, serialised as N3 in Figure 2.

```
@prefix : <http://facetontology.example.com/data#> .
@prefix facet: <http://danielsmith.eu/resources/facet/#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

:mspace a facet:StandardFacetCollection .
:mspace facet:faceturi :item .
:mspace facet:rdfsource "http://facetontology.example.com/data.n3" .

:mspace facet:faceturi :piece .
:mspace facet:faceturi :album .
:mspace facet:faceturi :composer .

:piece a facet:FirstOrderFacet .
:piece a facet:Facet .
:piece facet:class :Piece .
:piece rdfs:label "Piece" .

:album a facet:ConnectedFacet .
:album a facet:Facet .
:album facet:class :Album .
:album rdfs:label "Album" .
:album facet:nextpredicate :album_predicate .

:album_predicate a facet:Predicate .
:album_predicate facet:predicateuri :track .
:album_predicate facet:reverse "True"^^xsd:boolean .

:composer_facet a facet:ConnectedFacet .
:composer_facet a facet:Facet .
:composer_facet facet:class :Composer .
:composer_facet rdfs:label "Composer" .
:composer_facet facet:nextpredicate :composer_predicate .

:composer_predicate a facet:Predicate .
:composer_predicate facet:predicateuri :track .
:composer_predicate facet:reverse "True"^^xsd:boolean .
:composer_predicate facet:nextpredicate :composer_predicate2 .

:composer_predicate2 a facet:Predicate .
:composer_predicate2 facet:predicateuri :composer .
# nb: facet:reverse is not defined for composer_predicate2
```

Fig. 2. Example N3 definition of a FacetOntology for the running example of Classical Music.

The above example is the typical use case for FacetOntology, however there is a specialised case, when there is only the definitions from literals from the

FirstOrderFacet available. This specialised case requires an additional definition so that a SPARQL query takes the structure into account accurately. In order to specify that a facet's values use a predicate other than `rdfs:label` the facet must be marked up as having a `facet:type` of `facet:TypeLiteral`. The predicate to query must then be specified as the `facet:labeluri`. In our classical music example, a user may wish to have a facet for the "Original Name," as specified by the predicate `:originalName` (see Figure 1). If we use the `nextpredicate` definition by specifying `:originalName`, the query engine will try to find an instance at that predicate, which will fail since there is only a literal. Thus, instead we specify that this Facet has a Facet Type (`facet:facettype`), and that it is `facet:TypeLiteral`. We then add a definition that it also has a label URI (`facet:labeluri`) of `:originalName`. This informs the engine that the facet instances are not RDF instances, and are instead literals. This also means that internally, instance URIs are not used to uniquely identify the instances, their string values are used instead. See Figure 3 for an example definition of the `originalName` Facet.

```
@prefix : <http://facetontology.example.com/data#> .
@prefix facet: <http://danielsmith.eu/resources/facet/#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

:mspace facet:faceturi :originalname .

:originalname a facet:ConnectedFacet .
:originalname a facet:Facet .
:originalname facet:class :Piece .
:originalname rdfs:label "Original Name" .
:originalname facet:type facet:TypeLiteral .
:originalname facet:labeluri :originalName .
```

Fig. 3. Example N3 definition of a Facet for "Original Name," demonstrating the use of the `TypeLiteral` property, using FacetOntology for the running example.

In the next section, we outline all of the data transformation directives that FacetOntology provides so that data can be filtered and modified before inclusion in a faceted browser.

3.2 Data Transformation

One of the areas where existing work is limited is that it assumes that data is under our control, and can be edited freely. However, our approach instead makes the assumption that data is published by others, and make be possibly updated by them. Therefore, our approach is to build in data transformations that modify the data programmatically so that it can be updated repeatedly. Similarly, we support transforming the syntax of data values programmatically so that the relevant aspects of data are used in the facets. For example, removing the time from a time/date value, or applying a regular expression to clean up the data field.

One of the powerful directives in FacetOntology is the `facet:preprocess` directive which allows transformations to be applied to values in a facet, before they are used in the faceted browser. As shown in Table 1, the values of this directive specify the transformation that should be applied. For example, specifying `iso8601:striptime` will remove the time from ISO8601 [10] formatted dates, which is useful if you want to show only dates in a facet, and the times do not matter. We predict that additional directives will be required in the future, in particular with regards to data cleaning (e.g. applying regular expressions), and thus the interpretation of the preprocess value is extensible.

Similarly, we have specified the `facet:regex-PCRE-match` predicate, which expects a perl-compatible regular expressions (PCRE) [11] with a single matched group, used to extract the required information from the facet value. In more detail, Table 1 outlines the predicate directives that can be used against Facets.

Directive	Value	Description
`facet:preprocess`	iso8601:striptime	Remove the time from an ISO8601 formatted date.
`facet:preprocess`	iso8601:year	Extract the year from an ISO8601 formatted date.
`facet:regex-PCRE-match`	PCRE with single match group	Match the regular exression using the first match.

Table 1. FacetOntology directives for transformation of data, applied to each facet.

In the next section we discuss additional selection and filtering flexibility via SPARQL query hinting.

3.3 SPARQL Query Hinting

In addition to data transformation, it can be desirable to compute new data from existing value(s). For example to modify units, such as converting kilometres to miles, or to generate a value based on a combination of multiple properties, such as generating a screen resolution from a width and height. It also may be desirable to specify additional filters and patterns above those built into FacetOntology.

Another requirement is to provide additional context for each facet value, rather than only have a single value, for example to show the birth/death date of a composer, or the URL to a thumbnail of the depiction of a value. To this end, we have devised a set of SPARQL-hinting definitions in order to override the default variable bindings, add more complex "AS" algebra to the SELECT binding, and to inject additional patterns into the WHERE clauses. Through this approach, we allow selection of data to be filtered using SPARQL's full expressivity, for example using complex boolean "FILTER"s, and mathematical algebra using multiple data values in "AS" select expressions. Specifically we define the predicate directives in Table 2.

Directive	Description
`facet:SPARQL-Item-Variable`	Specify the variable to bind to the item URI.
`facet:SPARQL-Context-Variable`	Specify additional variable to select (e.g. to provide extra context on this item).
`facet:SPARQL-Context-Variable-As`	Specify additional binding to get from results (as specified by a `SPARQL-Context-Select-As`).
`facet:SPARQL-Context-Select-As`	Specify additional "AS" definition to use in SELECT, specify the resulting binding with a `SPARQL-Context-Variable-As`.
`facet:SPARQL-Context-Variable`	Specify additional variable to select (e.g. to provide extra context on this item).
`facet:SPARQL-Value-Variable`	Specify the variable to bind to the item value.
`facet:SPARQL-Value-Select-As`	Specify an "AS" definition to use for the SELECT.
`facet:SPARQL-Additional-Pattern`	Add an additional pattern into the WHERE.

Table 2. FacetOntology directives for hinting how to generate the facet query and which variables to read results from.

For example, to calculate a screen resolution by multiplying the width and height, a facet would be defined as follows:

```
:res-facet a facet:ConnectedFacet ;
    facet:class ont:System ;
    facet:type facet:TypeLiteral ;

    facet:SPARQL-Item-Variable "item";
    facet:SPARQL-Context-Variable "res_w";
    facet:SPARQL-Context-Variable "res_h";

    facet:SPARQL-Context-Variable-As "res_name_upper";
    facet:SPARQL-Context-Select-As "UCASE(?res_name) AS ?res_name_upper";

    facet:SPARQL-Value-Variable "val" ;
    facet:SPARQL-Value-Select-As "?res_w * ?res_h AS ?val" ;

    facet:SPARQL-Additional-Pattern "?item res:width ?res_w";
    facet:SPARQL-Additional-Pattern "?item res:height ?res_h";
    facet:SPARQL-Additional-Pattern "?item res:resolution_name ?res_name";

    rdfs:label "Resolution" .
```

results in the following SPARQL query being generated:

```
SELECT ?res_w ?res_h ?item (?res_w * ?res_h AS ?val) (UCASE(?res_name) AS ?
     res_name_upper)
WHERE {
    ?item rdf:type ont:System .
    ?item res:width ?res_w .
    ?item res:height ?res_h .
    ?item res:resolution_name ?res_name }
```

The SPARQL result processor knows from the definition to use the `?item` binding as the item URI and `?val` as the value. Additionally, it will also provide the values of width and height, because `SPARQL-Context-Variable` has been used to enable these bindings, and it will select `?res_name_upper` as an additional context field. Thus, through the SPARQL-hinting mechanism it is possible to provide additional context, and perform functions on the data before it is selected, either through additional patterns (which can contain FILTERs), and by using SPARQL functions on the data, in "AS" expressions.

By extending existing approaches of describing data, one of our aims is to enable FacetOntology to support any possible dataset, and to provide as rich an expressiveness as possible. Through the use of the above SPARQL hinting rules, FacetOntology queries can utilise the full specification of SPARQL functions and operators, and therefore is the full expressivity of SPARQL, which in turn has the expressiveness of relational algebra [12]. We have designed these SPARQL hinting patterns so that they encapsulate at least the full expressiveness of previous facet vocabularies (as discussed in Section 2), and are in fact now more expressive than previous work. In addition, additional context can be included using the SPARQL hinting definitions, to enable a richer faceted browser interface. In the next section we present a user interface to create FacetOntology definitions.

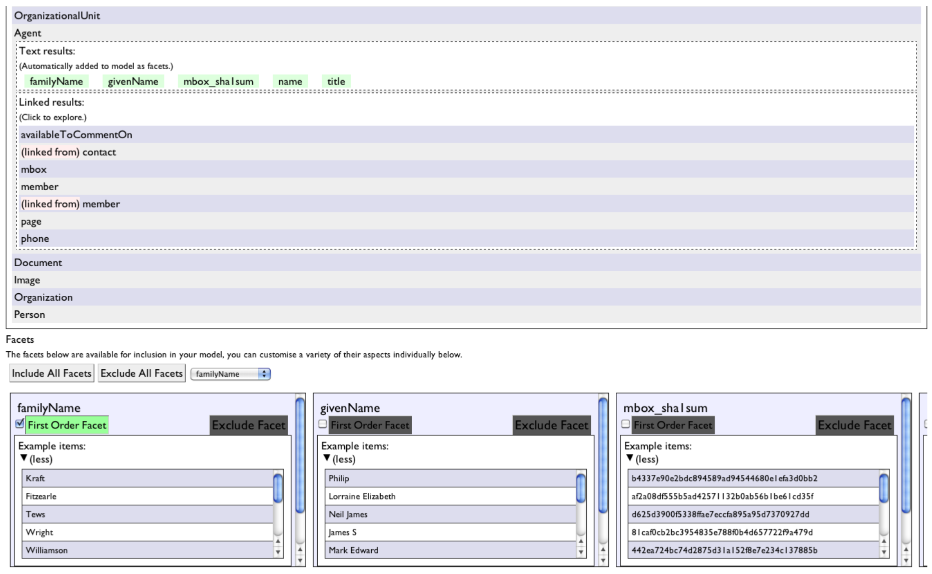
4 A User Interface for creating FacetOntology definitions

A FacetOntology definition is required before a facets browser can be created. This process can be performed manually, which is indeed the way that the data definition is performed in the existing approaches. One of our aims is to make it easier for end-users to create faceted browsers, ideally with a minimal of technical knowledge. To this end, we have created a user interface called "Data Picker" to aid in the creation of FacetOntology definitions.

The "Data Picker"[4] tool provides users with an intuitive interface for picking the metadata that they wish to be included in a FacetOntology definition. The tool works by querying a data set for all RDF classes, and a subset of the labels of individuals of that type. These are displayed, and the user can select a class which they want to explore further. All of the predicates that are joined to individuals of that class are then shown, again with a sample of their values. A user can then select which attributes they wish to be marked up using FacetOntology (see Figure 4). While the user is selected facets, they are presented with a preview of the facets, so that they can verify they are what they wanted, as expected. Finally, the users publish their FacetOntology definition, ready to send to a "maker" tool, which configures a faceted browser framework.

The need to aid manual marking up of facets is not a new problem, even within the domain of the Semantic Web. Work by Oren et. al. [13] looked at ranking facet quality, as an aid to automatically marking up facets, where their technique was formally proven to show an improvement of quality. Similarly, AKTiveRank [14] presents a technique for ranking ontologies, using structural

[4] Data Picker: `http://facetportal.danielsmith.eu/picker/`


OrganizationalUnit
Agent
Text results:
(Automatically added to model as facets.)
familyName givenName mbox_sha1sum name title
Linked results:
(Click to explore.)
availableToCommentOn
(linked from) contact
mbox
member
(linked from) member
page
phone
Document
Image
Organization
Person

Facets
The facets below are available for inclusion in your model, you can customise a variety of their aspects individually below.

familyName
☑ First Order Facet
Example items:
▼ (less)
Kraft
Fitzearle
Tews
Wright
Williamson

givenName
☐ First Order Facet
Example items:
▼ (less)
Philip
Lorraine Elizabeth
Neil James
James S
Mark Edward

mbox_sha1sum
☐ First Order Facet
Example items:
▼ (less)
b4337e90e2bdc894589ad94544680e1efa3d0bb2
af2a08df555b5ad42571132b0ab56b1be61cd35f
d625d3900f5338ffae7eccfa895a95d7370927dd
81caf0cb2bc3954835e788f0b4d657722f9a479d
442ea724bc74d2875d31a152f8e7e234c137885b

Fig. 4. A screenshot of the data picker running on the University of Southampton SPARQL endpoint. The user has selected the "Agent" class and a number of literals have been previewed as facets ("familyName", "givenName" and "mbox_sha1sum"), and data properties are available to explore, labeled as "linked results" (such as "member", "page", and "phone").

metrics. Such metrics could be used to inform a facet creation tool, in order to enable an increased level of automation.

In summary, the cost of creating a FacetOntology is linear with the amount of facets in a domain — if a domain has 4 facets, a single First Order facet is defined by its RDF Class, and 3 Connected Facets are defined by how they link to the First Order facet. In a domain with a larger amount of facets, a larger definition is required, which will take more time to create.

5 Implementations of Faceted Browser Makers

To demonstrate and validate our approach, we have created two different faceted browser "makers" that use different faceted browser frameworks to create interfaces over the same data source. An advantage of this approach is that if a developer is more comfortable with a particular faceted browser framework, then they can use the one they wish. Similarly a developer may wish to migrate from one browser or platform to another, and by having the abstract FacetOntology definition, they do not have to reconfigure a new system. This may occur because the data requirements of a faceted browser change, for example a system may work fine with 1000 items using Exhibit, but as that number increase, then migration to a server-backed browser such as mSpace may be appropriate. Using FacetOntology for this purpose would make that approach more straightforward.

The first implementation we discuss using FacetOntology is our FacetOntology Exhibit maker[5], which reads a FacetOntology definition URI, and outputs a configured Exhibit installation. The core components of this process are generating the JSON data file, and then configuring an Exhibit view that uses that data file. Creation of the data file is performed by loading the data sources into a store, and querying them based on the predicates, filters and SPARQL hints defined in the FacetOntology definition. Following this process, an Exhibit HTML file is created based on a template (to allow styling of the resulting Exhibit) which contains each of the facets that have been defined.

The second implementation using FacetOntology is the “mSpace Maker”[6], which produces mSpaces from FacetOntology definitions. mSpace Maker automates the creation of mSpace faceted browsers, by gathering data into an mSpace database, and configuring mSpace to use this database’s schema. The process to configure an mSpace differs from that of Exhibit above, because rather than holding all data in a single file, the data is stored in a relational database with a schema that has been optimised for the patterns of querying that mSpace performs. Thus, the mSpace Maker works by creating a table for each facet, and uses foreign keys to link them together. The facets and their database table and column names are written into the mSpace configuration, so that the mSpace server can create SQL queries to join the tables together. Upon completion of the data import, a URL to the mSpace is returned to the user (because mSpace cannot be used locally, unlike Exhibit). In the next section we describe public faceted browsers that we have created that use FacetOntology, and were created using our makers.

5.1 Exemplar Uses

We demonstrate the flexibility of FacetOntology by applying it to different domains, where the use of a faceted browser is different. Each of these browsers demonstrates additional support that FacetOntology supports above and beyond configurations that have been previously attempted. It is through developing these configurations that the core requirements for FacetOntology have been developed.

1. **Francophone Music Criticism[7]:** This faceted browser demonstrates faceted filtering of 19th century music reviews from a digital repository. This browser demonstrates basic FacetOntology usage, over five key facets (Collection, Journal, Year, Author and Title), and results in a set of PDFs that the user can then download from a digital repository.

[5] FacetOntology-Exhibit: `https://github.com/danielsmith-eu/facetontology-exhibit`
[6] mSpace Maker: `http://mspacemaker.danielsmith.eu/`
[7] Francophone Music Criticism: `http://fmc.ecs.soton.ac.uk/`

2. **Rich Tags**[8]**:** Faceted browsing over multiple digital repositories. This faceted browser runs over a more sophisticated data collection engine which routinely collects data from multiple institutional repositories into a single store. Thus, the FacetOntology definition is applied to all sources of data, resulting in a single browser over the composite data.
3. **BBC iPlayer**[9]**:** A browser for faceted filtering of television programmes that have been broadcast over the previous 7 days, for streamed viewing online. This browser demonstrates transforming broadcast time data into individual dates for use in a date-picker facet (see Figure 5). The browser provides richer faceted exploration of the data than the BBC site, and includes data from multiple sources (specifically, the Contributor credits data), so that users can filter shows by actor/presenter, which is not possible on the BBC site. In order to demonstrate our Exhibit maker, we have also used the FacetOntology definition of the iPlayer data to create a Exhibit of the same data, as shown in Figure 6.
4. **UsabilityUK**[10]**:** A browser of usability methods and JISC-funded projects that use those methods. UsabilityUK demonstrates a faceted browser where users can search for more than one type of record. Specifically, users can search for projects, experts or usability methods. This contrasts with other typical faceted browsers where users search for a single type of record (such as a TV programme, a song, or document).

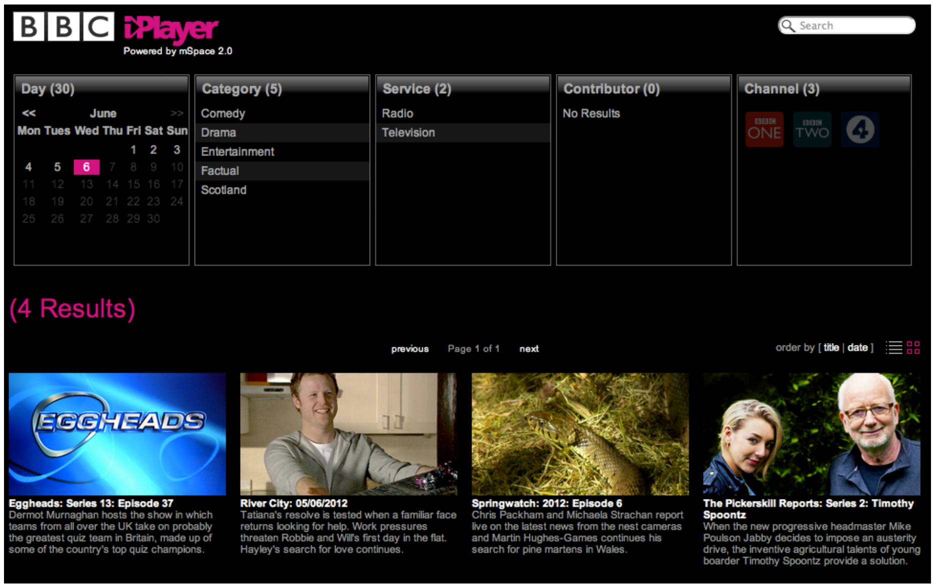
Our approach supports a range of different requirements in a range of different domains, demonstrating that FacetOntology is fit for purpose as a general approach to semantic description of facets, which benefits from the full expressivity of SPARQL.

6 Conclusion

In this paper, we present FacetOntology, an approach to defining facets for faceted browsers. We applied our approach to BBC iPlayer, digital repositories and a usability techniques knowledge base, showing that our approach is effective and can be applied to many domains. In more detail, we verified that our approach works for different types of browsing across these domains, has the full expressivity of SPARQL, and has been applied to different faceted browsing software.

For future work, we plan to investigate appropriate methodologies for enabling additions to the FacetOntology data transformations (as discussed in Section 3.2) to be created by the community, when additional transformations are required.

[8] Rich Tags: `http://www.richtags.org/`
[9] BBC iPlayer mSpace (updates daily): `http://iplayer.mspace.fm/`
[10] UsabilityUK: `http://usabilityuk.org/`



(4 Results)

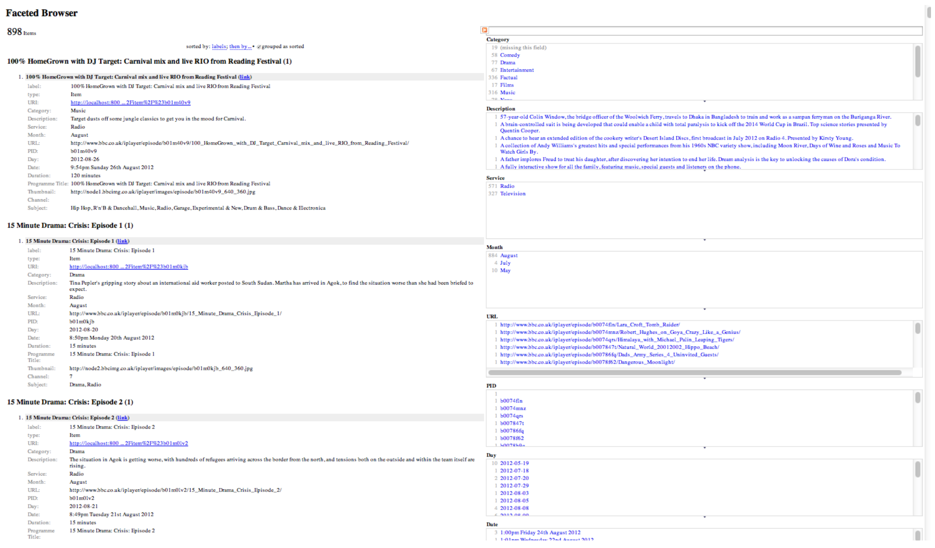
Eggheads: Series 13: Episode 37
Dermot Murnaghan hosts the show in which teams from all over the UK take on probably the greatest quiz team in Britain, made up of some of the country's top quiz champions.

River City: 05/06/2012
Tatiana's resolve is tested when a familiar face returns looking for help. Work pressures threaten Robbie and Will's first day in the flat. Hayley's search for love continues.

Springwatch: 2012: Episode 6
Chris Packham and Michaela Strachan report live on the latest news from the nest cameras and Martin Hughes-Games continues his search for pine martens in Wales.

The Pickerskill Reports: Series 2: Timothy Spoontz
When the new progressive headmaster Mike Poulson Jabby decides to impose an austerity drive, the inventive agricultural talents of young boarder Timothy Spoontz provide a solution.

Fig. 5. The BBC iPlayer mSpace faceted browser created using FacetOntology definitions.



Faceted Browser

898 Items

sorted by: labels; then by... • grouped as sorted

100% HomeGrown with DJ Target: Carnival mix and live RIO from Reading Festival (1)

1. **100% HomeGrown with DJ Target: Carnival mix and live RIO from Reading Festival (link)**

label:	100% HomeGrown with DJ Target: Carnival mix and live RIO from Reading Festival
type:	Item
URI:	http://localhost:800 ... 2Fitem%2F%23b01m40v9
Category:	Music
Description:	Target dusts off some jungle classics to get you in the mood for Carnival.
Service:	Radio
Month:	August
URL:	http://www.bbc.co.uk/iplayer/episode/b01m40v9/100_HomeGrown_with_DJ_Target_Carnival_mix_and_live_RIO_from_Reading_Festival/
PID:	b01m40v9
Day:	2012-08-26
Date:	9:54pm Sunday 26th August 2012
Duration:	120 minutes
Programme Title:	100% HomeGrown with DJ Target: Carnival mix and live RIO from Reading Festival
Thumbnail:	http://node1.bbcimg.co.uk/iplayer/images/episode/b01m40v9_640_360.jpg
Channel:	
Subject:	Hip Hop, R'n'B & Dancehall, Music, Radio, Garage, Experimental & New, Drum & Bass, Dance & Electronica

15 Minute Drama: Crisis: Episode 1 (1)

1. **15 Minute Drama: Crisis: Episode 1 (link)**

label:	15 Minute Drama: Crisis: Episode 1
type:	Item
URI:	http://localhost:800 ... 2Fitem%2F%23b01m0kjb
Category:	Drama
Description:	Tina Pepler's gripping story about an international aid worker posted to South Sudan. Martha has arrived in Agok, to find the situation worse than she had been briefed to expect.
Service:	Radio
Month:	August
URL:	http://www.bbc.co.uk/iplayer/episode/b01m0kjb/15_Minute_Drama_Crisis_Episode_1/
PID:	b01m0kjb
Day:	2012-08-20
Date:	8:50pm Monday 20th August 2012
Duration:	15 minutes
Programme Title:	15 Minute Drama: Crisis: Episode 1
Thumbnail:	http://node2.bbcimg.co.uk/iplayer/images/episode/b01m0kjb_640_360.jpg
Channel:	7
Subject:	Drama, Radio

15 Minute Drama: Crisis: Episode 2 (1)

1. **15 Minute Drama: Crisis: Episode 2 (link)**

label:	15 Minute Drama: Crisis: Episode 2
type:	Item
URI:	http://localhost:800 ... 2Fitem%2F%23b01m0lv2
Category:	Drama
Description:	The situation in Agok is getting worse, with hundreds of refugees arriving across the border from the north, and tensions both on the outside and within the team itself are rising.
Service:	Radio
Month:	August
URL:	http://www.bbc.co.uk/iplayer/episode/b01m0lv2/15_Minute_Drama_Crisis_Episode_2/
PID:	b01m0lv2
Day:	2012-08-21
Date:	8:49pm Tuesday 21st August 2012
Duration:	15 minutes
Programme Title:	15 Minute Drama: Crisis: Episode 2

Category
19 (missing this field)
58 Comedy
77 Drama
67 Entertainment
336 Factual
17 Films
316 Music

Description
1 57-year-old Colin Window, the bridge officer of the Woolwich Ferry, travels to Dhaka in Bangladesh to train and work as a sampan ferryman on the Buriganga River.
1 A brain-controlled suit is being developed that could enable a child with total paralysis to kick off the 2014 World Cup in Brazil. Top science stories presented by Quentin Cooper.
1 A chance to hear an extended edition of the cookery writer's Desert Island Discs, first broadcast in July 2012 on Radio 4. Presented by Kirsty Young.
1 A collection of Andy Williams's greatest hits and special performances from his 1960s NBC variety show, including Moon River, Days of Wine and Roses and Music To Watch Girls By.
1 A father implores Freud to treat his daughter, after discovering her intention to end her life. Dream analysis is the key to unlocking the causes of Dora's condition.
1 A fully interactive show for all the family, featuring music, special guests and listeners on the phone.

Service
571 Radio
327 Television

Month
884 August
4 July
10 May

URL
1 http://www.bbc.co.uk/iplayer/episode/b0074fln/Lara_Croft_Tomb_Raider/
1 http://www.bbc.co.uk/iplayer/episode/b0074mnz/Robert_Hughes_on_Goya_Crazy_Like_a_Genius/
1 http://www.bbc.co.uk/iplayer/episode/b0074qrs/Himalaya_with_Michael_Palin_Leaping_Tigers/
1 http://www.bbc.co.uk/iplayer/episode/b007847t/Natural_World_20012002_Hippo_Beach/
1 http://www.bbc.co.uk/iplayer/episode/b00786fq/Dads_Army_Series_4_Uninvited_Guests/
1 http://www.bbc.co.uk/iplayer/episode/b0078f62/Dangerous_Moonlight/

PID
1
1 b0074fln
1 b0074mnz
1 b0074qrs
1 b007847t
1 b00786fq
1 b0078f62

Day
10 2012-05-19
1 2012-07-18
2 2012-07-20
1 2012-07-29
1 2012-08-03
1 2012-08-05
4 2012-08-08

Date
3 1:00pm Friday 24th August 2012

Fig. 6. The BBC iPlayer Exhibit faceted browser (unstyled) created using FacetOntology definitions.

Bibliography

[1] Hearst, M., Elliott, A., English, J., Sinha, R., Swearingen, K., Yee, K.: Finding the flow in web site search. Communications of the ACM **45**(9) (2002) 49

[2] Hildebrand, M., van Ossenbruggen, J., Hardman, L.: /facet: A Browser for Heterogeneous Semantic Web Repositories. In:. (2006) 272–285 10.1007/11926078_20.

[3] García, R., Gimeno, J., Perdrix, F., Gil, R., Oliva, M.: The rhizomer semantic content management system. Emerging Technologies and Information Systems for the Knowledge Society (2008) 385–394

[4] Huynh, D., Karger, D., Miller, R.: Exhibit: lightweight structured data publishing. Proceedings of the 16th international conference on World Wide Web (2007) 737–746

[5] m.c. schraefel, Wilson, M., Russell, A., Smith, D.A.: mspace: improving information access to multimedia domains with multimodal exploratory search. Commun. ACM **49**(4) (2006) 47–49

[6] Berners-Lee, T., Chen, Y., Chilton, L., Connolly, D., Dhanaraj, R., Hollenbach, J., Lerer, A., Sheets, D.: Tabulator: Exploring and Analyzing linked data on the Semantic Web. Proceedings of the 3rd International Semantic Web User Interaction Workshop (2006)

[7] m. c. schraefel, Smith, D.A., Owens, A., Russell, A., Harris, C., Wilson, M.: The evolving mspace platform: leveraging the semantic web on the trail of the memex. In: HYPERTEXT '05: Proceedings of the sixteenth ACM conference on Hypertext and hypermedia, New York, NY, USA, ACM Press (2005) 174–183

[8] SIMILE: Longwell RDF Browser `http://simile.mit.edu/longwell/`. (2003–2005)

[9] Hildebrand, M., Van Ossenbruggen, J.: Configuring semantic web interfaces by data mapping. Visual Interfaces to the Social and the Semantic Web (VISSW 2009) **443** (2009) 96

[10] International Organization for Standardization (ISO): ISO 8601:2004 Data elements and interchange formats, Information interchange, Representation of dates and times. (2004)

[11] Hazel, P.: PCRE: Perl Compatible Regular Expressions (2005)

[12] Angles, R., Gutierrez, C.: The expressive power of sparql. The Semantic Web-ISWC 2008 (2008) 114–129

[13] Oren, E., Delbru, R., Decker, S.: Extending Faceted Navigation for RDF Data. In:. (2006) 559–572 10.1007/11926078_40.

[14] Alani, H., Brewster, C., Shadbolt, N.: Ranking ontologies with AKTiveRank. Lecture Notes in Computer Science **4273** (2006) 1