SWAD-Europe deliverable 12.1.4: Semantic Blogging - Lessons Learnt

Project name:
Semantic Web Advanced Development for Europe (SWAD-Europe)
Project Number:
IST-2001-34732
Workpackage name:
12.1 Open Demonstrators
Workpackage description:
☞http://www.w3.org/2001/sw/Europe/plan/workpackages/live/esw-wp-12.1.html
Deliverable title:
Semantic blogging for bibliographies - Lessons Learnt
URI:
☞http://www.w3.org/2001/SW/Europe/reports/demo_1_report/
Author:
☞Steve Cayzer, HP Laboratories, Bristol, UK
Abstract:
This document reports on our experiences building one of the open demonstrators contributed by Hewlett-Packard to the *SWAD-E* programme. The chosen prototype is a semantic blog, where we enhance blogging with semantic web tools and ideas, moving it beyond communal diary browsing to a rich information sharing scenario. Specifically, our semantic blog is used for informal knowledge management within the bibliography management domain. Our design was, broadly, to augment a blog with a metadata pipeline, with import, export and storage/access mechanisms. Three semantic behaviours (view, navigation and query) were built over this base. We have aimed with this demonstrator to develop a tool that is simple, useful, extensible and illustrative. Our objectives include providing an illustrative demonstrator, gathering useful development and deployment experience, providing outreach and publicity and creating the groundwork for a useful tool. In addition to discussing the three semantic behaviours (view, navigation and query) we discuss platform choices, information modelling, metadata storage, and import/export mechanisms. Overall lessons include the tension between *RDF* the model and *RDF* the syntax, the use of *RDF* for configuration and personalisation, and the importance of rich and interesting metadata. Semantic web values are covered partly by the existing demonstrator, partly by stories one can tell around it, and partly by extensions that we (and others) are planning to build. In terms of outreach, the demonstrator has proved to be an exemplary base. We have had overwhelming, and positive, international interest from individuals, corporations and journalists. We therefore believe that semantic blogging is an excellent way for people to see (and assess) the benefit of using *RDF*. We conclude by noting that perhaps semantic blogging is a technology whose time has come. We look forward to the semantic blogging theme maturing in many, perhaps even unexpected, ways in the future.
Status:
Initial release.
Comments on this document are welcome and should be sent to ☞Steve Cayzer or to the ☞public-esw@w3.org list. An archive of this list is available at ☞http://lists.w3.org/Archives/Public/public-esw/

Contents

Executive Summary

The semantic web promises to make the web more useful by endowing metadata with machine processable semantics. The SWAD-Europe project [☞SWADE] provides targeted research, demonstrations and outreach to help semantic web technologies move into the mainstream of networked computing.

This report is concerned with workpackage 12.1; one of the open demonstrations contributed by Hewlett-Packard to the *SWAD-E* programme. We took an existing and popular platform, blogging, and applied semantic web principles to it. While it is merely a prototype, the demonstrator [☞SEMBLOG-DEMO] shows that the use of semantic metadata can enhance the blogging experience. In addition, the process of building this demonstrator was educational, providing a useful context for 'best practice' suggestions regarding the development and deployment of semantic web applications. Finally, the demonstrator has been very useful in initiating dialogues with communities who might not otherwise have interacted with the semantic web.

The aim of this report is to reflect on the lessons learnt from the work undertaken during the analysis, design and implementation of the *SWAD-E* semantic blogging demonstrator. We discuss our design choices, evaluate the illustration of semantic web values, and describe the reaction of the wider community to this project.

Semantic blogging builds upon the success and clear network value of blogging by adding additional semantic structure to items shared over the blog channels. In this way we add significant value allowing view, navigation and query along semantic rather than simply chronological or serendipitous connections. Our semantic blogging demonstrator demonstrates metadata driven views, new navigation modalities and richer query. It shows how a semantic blog can be used for informal knowledge management, and is set in the bibliography management domain.

We have aimed with this demonstrator to develop a tool that is simple, useful, extensible and illustrative. We seek to show that semantic blogging is a way to move blogging beyond communal diary browsing to rich information sharing scenarios.Our objectives include:

- Provide a demonstrator for semantic web, easily understood by current developers.
- Provide a war story for application of semantic web techniques.
- Assist outreach and publicity.
- Provide the groundwork for a useful, deployable tool.

Our design was, broadly, to augment a blog with a metadata pipeline, with import, export and storage/access mechanisms. The three semantic behaviours (view, navigation and query) were built over this base. We grounded our demonstrator in the bibliography management domain, and implemented ontologies for bibliographic metadata, topic hierarchies and *UI* configuration. Our lessons learnt can be thought about both separately (per component) and overall.

- Bibliographic management is an excellent choice of domain - an appropriate grounding for our system, a demonstrable need for semantic web technologies, and a great opportunity to spread the semantic web meme.
- Bibliographic metadata requirements were captured well using BibTeX [☞BibTeX], though we note that a richer schema such as MODS [☞MODS] might be required for more ambitious tasks. *TIF* [☞TIF] (now replaced by *SKOS* [☞SKOS]) worked well for us too, and ongoing work should make this a generally useful resource for many information modelling needs. The use of *RDF* for *UI* configuration worked well for us - we adopted opt-in and opt-out styles which were particularly useful.
- Jena [☞JENA] (for *RDF* support) and blojsom [☞BLOJSOM] (for blogging platform) were both good platform choices.
- For metadata storage, we used a single *RDF* store for the whole blog. This brings consistency and performance issues, though a simple file backed solution worked satisfactorily for the demonstrator. The proposed NetAPI protocol [☞NETAPI] would appear to form a good basis for future extensions.
- Metadata can be viewed and edited either locally, or by piping it to an external resource. For flexibility, we need to generate metadata 'on the fly' and at an externally retrievable location.
- Importing metadata can be semantically assisted, and we show one simple way to achieve this
- Use of *RDF* in an export feed leads to consumer issues. Extra metadata may be useful to semantically enabled consumers, but for others it causes at least unnecessary bandwidth load. In addition, the arbitrary serialization of *RDF* could potentially cause problems for non *RDF*-aware consumers. Syntax constraints and accepted *XML* profiles can be used to circumvent these issues but such measures can pose a limitation for *RDF* expressiveness.
- Semantic view is a good way to show how a blog can look very different with a little effort. Semantic navigation is potentially useful but requires richer metadata for a full exploration of the possibilities (eg facet browsing). Semantic query enables one to expose the rich structure of a semantic blog (eg "show me all blog entries about this paper").
- Overall lessons include the tension between *RDF* the model and *RDF* the syntax (as in the export component), the use of *RDF* for configuration and personalisation, and the importance of rich and interesting metadata.

In terms of the semantic web values that we wish to demonstrate (data representation, semantics and webness), we have covered these partly in the existing demonstrator, partly by stories one can tell around it, and partly by extensions that we (and others) are planning to build. In terms of outreach, the demonstrator has proved to be an exemplary base. We have had overwhelming, and positive, international interest from individuals, corporations and journalists. We therefore believe that semantic blogging is an excellent way for people to see (and assess) the benefit of using *RDF*.

We expect the future to bring further semantic blogging developments. Not only are we looking at ways to apply the techniques to other areas, but others are picking up on the ideas too. It seems semantic blogging is a technology whose time has come. We look forward to the semantic blogging theme maturing in many, perhaps even unexpected, ways in the future.

1 Introduction

This report is part of ☞SWAD-Europe ☞Work package 12.1: Open demonstrators. This workpackage covers the selection and development of two demonstration applications designed to both illustrate the nature of the semantic web and to explore issues involved in developing substantial semantic web applications.

The aim of this report is to reflect on the lessons learnt from the work undertaken during the analysis, design and implementation of the first demonstrator. We also reflect on how well the demonstrator has achieved our aims. We discuss our design choices, evaluate the illustration of semantic web values, and describe the reaction of the wider community to this project. We believe that the experience gained during this project has been useful, and seek to pass this experience on the semantic web development community. In particular, we feel that aspects of the chosen domain, details of our dissemination activities and design choices which worked well (or badly) for us might all be of interest. We provide additional design and implementation details in the appendices.

Throughout this report, the plural form "we" is used to refer variously to the author (Steve Cayzer), the implementers (author plus William Kong), the designers (implementers plus Paul Shabajee), or the entire project team (designers plus Dave Reynolds and Ian Dickinson). We also take this opportunity to acknowledge the myriad contributions from others in Hewlett Packard, our *SWAD-E* partners and the wider community.

2 Background and objectives

2.1 *SWAD-E* Objectives - The semantic web promises to make the web more useful by endowing metadata with machine processable semantics. Much of the base work for this endeavour has already been laid. We have a language, *RDF* [☞RDF-LANGUAGE] backed by a model theory [☞RDF-MODEL] and a series of logical formalisms [☞OWL] which allow us to bring increasing expressivity and power to inferencing over this metadata. We have a number of mature frameworks [☞JENA, protege] and some commercial activity [☞NETWORK-INFERENCE, ☞SEMAVIEW]. Yet there is still much misunderstanding and resistance, antagonism even, from the developer and wider community [☞SHIRKY-SYLLOGISTIC]. Therefore, while much vital foundational work remains to be done, there is clearly an equally important need. A need for education and outreach. A need for tools which make it easy for *XML* developers to work with *RDF*. And a need to make what is sometimes called (tongue-in cheek) 'The Semantic Web (version 1.0)' a reality. It is with this need in mind that Semantic Web Advanced Development, SWAD [☞SWAD] was chartered. The Semantic Web Advanced Development Europe project, *SWAD-E*, [☞SWADE] aims to support this initiative in Europe, providing targeted research, demonstrations and outreach to ensure Semantic Web technologies move into the mainstream of networked computing.

This report is concerned with workpackage 12.1; one of the open demonstrations contributed by Hewlett-Packard to the *SWAD-E* programme. We took an existing and popular platform, blogging, and applied semantic web principles to it. While it is merely a prototype, the demonstrator [☞SEMBLOG-DEMO] shows that the use of semantic metadata can enhance the blogging experience. In addition, the process of building this demonstrator was educational, providing a useful context for 'best practice' suggestions regarding the development and deployment of semantic web applications. Finally, the demonstrator has been very useful in initiating dialogues with communities who might not otherwise have interacted with the semantic web. For example, we grounded the demonstration in the bibliography management domain. Subsequent discussions [☞DARCUS] have not only confirmed that this was an excellent choice, but have strengthened the links between the semantic web and bibliographic metadata communities. In fact the whole semantic blogging metaphor has proved a useful framing for discussion about and around the semantic web.

2.2 Semantic web values - In our application survey [☞SWADE-ANALYSIS] we characterised the semantic web as an attempt to turn the web from a large hyperlinked book into a large interlinked database. Viewed like this, we identified the characteristics which seem to capture the value of a semantic web approach:

- **Data representation** The foundation of the semantic web is a common format, *RDF* [☞RDF-LANGUAGE], to represent data. A critical feature of *RDF* is the use of web URIs to provide items with a well-defined place in the global namespace. This allows many sorts of data (property values of objects, relationships between objects, value annotations) to be represented uniformly and allows data from multiple locations to be combined without accidental clashing of property names or structure mismatches.
- **Semantics** The aspiration of the semantic web is to be able to express meaning. The schema [☞RDF-SCHEMA] and ontology [☞OWL] layers of the semantic web begin to do this; for example, to say whether two terms are distinct, equivalent or whether one term is a subset of another. This capability allows a data source to expose its conceptual model explicitly in machine processable form thus providing the automation of more sophisticated processing and better integration of data from different sources. Of course, the semantic web does not require some standardized global upper ontology to function, remaining decentralized so that data sources are free to mix and match terms from different ontologies.
- **Webness** The critical innovation of the semantic web is to put both these values into a web

framework. This is manifested in deceptively simple ways such as the use of URIs, enabling an agent to discover the ontology associated with a data source, the development of decentralized ontologies and the combination of terms from different ontologies.

In this report, we shall assess to what extent these values are apparent in the semantic blogging demonstrator.

2.3 Why did we choose semantic blogging? - In our applications survey [☞SWADE-ANALYSIS], we noted that semantic web technologies are well suited to tasks where a user community is incrementally publishing structured and semantically rich (categorized and cross-linked) information. We also noted that blogging is a very successful paradigm for lightweight publishing, providing a very low barrier to entry, useful syndication and aggregation behaviour, a simple to understand structure and decentralized construction of a rich information network.

The notion of semantic blogging builds upon this success and clear network value of blogging by adding additional semantic structure to items shared over the blog channels. In this way we add significant value allowing view, navigation and query along semantic rather than simply chronological or serendipitous connections. Our semantic blogging demonstrator design emphasises three key behaviours:

- **Semantic View**: Semantically enriched blog metadata enables context sensitive, metadata driven views of the blog content (over and above fixed templates).
- **Semantic Navigation**: Semantically enriched blog metadata enables new blog navigation modalities (over and above unlabelled links).
- **Semantic Query**: Semantically enriched blog metadata enables richer query and discovery mechanisms (over and above free text search).

We should note here that the use of the term 'semantic' emphasises the use of semantic web technology to enable these behaviours. It is true that in the current instantiation of the demonstrator, the new capabilities are enabled primarily by using rich metadata, and require little in the way of actual semantic machinery. However, in each case it is easy to see how the behaviour can be further extended by adding inferencing over a semantic model. A simple example would be subcategory inferencing for semantic query. We prepare our blog for such possibilities by encoding its metadata in *RDF*.

In summary then, the rich structure and query properties enabled by the semantic web greatly extends the range of blogging behaviours, and allows the power of the metaphor to be applied in hitherto unexplored domains. One such domain (and the one explored here) is bibliography management.

2.4 Why did we choose bibliography management? - It is important to chose a specific domain because without it there is not enough application feedback to enable focus on core values and key technical challenges. Bibliographic management is one such domain. Although traditional bibliographic management deals mainly with static categorisations, the needs of a small group collectively exploring a domain exhibit a more dynamic, community based flavour. Here is a task which is characterised by a need to share small items of information with a peer group in a timely, lightweight manner. This information should be easily publishable, easily discoverable and easily navigable. It should be simple to enrich the information with annotation, either at the point of delivery or later. The information should be archived in a commonly understood way for effective post-hoc retrieval. It should be possible to be notified, in a timely way, of new items of interest. We believe that a combination of blogging and semantic web technologies offers an ideal solution to this problem. Blogging provides low barrier publishing, a simple shared conceptual model, and a mechanism for natural, dynamic community formation. The rich structure provided by semantic metadata enables improved view, navigation and query capabilities.

In addition, the need we are exploring is not one well served by current tools [☞SWADE-USER-STUDY]. Although it is, of course, easy enough to integrate a personal bibliography and (say) a word processor, the current tools do not facilitate the sharing of metadata between small groups. This is, in part, due to the lack of weaknesses of current bibliography standards when it comes to representing rich community annotations, and in part due to the lack of tools to make use of these annotations. The semantic blogging demonstrator is largely an attempt to answer the second need, but it also recognises the need to small groups to 'roll their own' classification and annotation schemes and yet make these annotations available for use both within and without their community. We have built our demonstrator with the ultimate aim of creating a community sharing tool.

Whilst bibliography management is an important task in the research community, it could be seen as a niche application in the wider community. However, the semantic blogging tools and approaches are just as applicable to dissemination and management of other content such as business documents or news items. Generalizing the results to related areas should be straightforward.

2.5 Demonstrator Objectives - We have aimed with this demonstrator to develop a tool that is simple, useful, extensible and illustrative. Simple, because it should be easy to learn and to use. Useful, because it should do something that users actually want, efficiently and reliably. Extensible, because although we ground the requirements in the bibliographic domain, we expect it to be reusable for other semantic blogging applications. And illustrative, because we wish to incorporate features that demonstrate the values of the semantic web approach without losing the attractive features of blogging.

We seek to show that semantic blogging is a way to move blogging beyond communal diary browsing to rich information sharing scenarios.

We finish this section with a list of objectives for the demonstrator:

- Provide a demonstrator for semantic web, easily understood by current developers.
- Provide a war story for application of semantic web techniques.

- Assist outreach and publicity.
- Provide the groundwork for a useful, deployable tool.

3 Choices and lessons learnt

In this section, we describe how we went about building our demonstrator and what we learned along the way. We start with an evaluation of the domain. We then look at the choice of platforms on which to build. Finally, we step through the main design components of the semantic blogging demonstrator, evaluating our choices at each step.

3.1 Domain choices - We grounded the demonstrator in the bibliography management domain. Initially, as described above, this was simply because it (intuitively) seemed a useful grounding point. Certainly it acted as a good focus for development, providing a concrete base for requirements. But in fact closer study [☞SWADE-USER-STUDY] revealed that there is actually a real need in this area. In particular, current bibliographic tools are inadequate for the purposes of shared management, annotation and discovery. In addition, the choice of domain has lead to discussions with a number of people involved in bibliographic metadata standards [☞DARCUS, ☞OCLC]. In addition to confirming our intuitions, these contacts are in themselves a useful outcome of the project. It is probably important to emphasise that while there is currently disagreement over the most appropriate bibliographic metadata standards, it is not our intention to take sides in this debate. Indeed, part of the idea of semantic blogging is to allow different communities to use their own preferred vocabularies while allowing these communities to share, mix and exchange metadata.

Lessons learnt

Bibliographic management is an excellent choice of domain - an appropriate grounding for our system, a demonstrable need for semantic web technologies, and a great opportunity to spread the semantic web meme.

3.2 Architecture and platform choices - In building the semantic blogging demonstrator, we made a number of platform choices.

Firstly, we opted to use Jena [☞JENA] - a premium toolkit for semantic web applications. Clearly the fact the Jena was created (and is maintained) by HP Labs in Bristol was a factor in our decision. But there are also more objective reasons for our choice - it is fully standards compliant, with an active support list and more than adequate cover for the functionality we expected to use.

Secondly, we chose to build our demonstrator over an existing blogging platform. We could have opted to implement our own blogging platform, building in semantic capabilities from the ground up. However, not only does an existing platform save us effort, it also creates awareness amongst a blogging development community. One of the more respected blog platforms is MovableType (*MT*) [☞MT]. Indeed, at the start of the project we evaluated *MT*. We found that it is well designed and actively supported. *MT* also supports a number of semantic extensions, for example FOAF [☞FOAF] and blog entry *RDF* metadata [☞MT-RDF]. However there were two key limitations from our point of view. Firstly, *MT* is perl based and therefore requires some work to integrate with Jena. Secondly, and more seriously, *MT* has a template based extension mechanism. Templates are rebuilt from time to time, creating static HTML pages which are available to be accessed through any webserver. While this system is certainly elegant and extensible, its template based system makes it difficult to encode dynamic information (like user/session data). It is not impossible to do such things but they may require changes to the *MT* code and/or extensions to the database. Thus, *MT* is well suited for static (rebuild time) customizations but not for dynamic, context sensitive (run time) content repurposing.

We decided instead to use the newer blog platform blojsom [☞BLOJSOM]. Blojsom offers us three key benefits. Firstly, being Java based it integrates well with Jena. Secondly, it is actively supported and its key architects are semantic web enthusiasts. Thirdly, its plug-in architecture and Java Server Page (*JSP*) environment offer clean extensibility and run time flexibility. This last point deserves some clarification. Blojsom uses a series of *JSP* templates to build pages on the fly. Blojsom plug-ins can be inserted into the chain before dispatching to *JSP* so can be used to enrich the served page in a number of ways. This means that arbitrary data can be used to customize the blog entries at access time, but it also mean that access to the blog entries is usually through the blojsom interface. Anything changing the top level blog entry array (like semantic query) requires an out of band mechanism. And in any case, formatted entries are not available as static pages. Still, notwithstanding these disadvantages, we think that blojsom was the right choice. Semantic blogging is an application that requires frequent, often major, access time customisation, and so a dynamic platform is a boon. A happy side effect of using blojsom was that we didn't suffer from the recent spate of *MT* spamming [☞MT_SPAM].

One point that is relevant whatever platform is chosen is that of resource availability. For example, *MT* requires a web server on which you can run CGI scripts, and it also requires a database and a number of perl modules. Blojsom requires a Java enabled server with a servlet container (like Tomcat) and, usually, shutdown/restart access to the servlet engine. Similar considerations would apply to any home grown solution too. Not all users will have access to all these components, however in principle a hosted platform could be provided [☞MT_TYPEPAD].

The other area in which we made 'platform' choices was that of the metadata schema. Our choices are more fully explored in ☞the appendix but here we note two examples. Firstly, we used BibTeX [☞BibTeX] for representing our bibliographic entries. Bibtex is a simple standard, yet reasonably rich and one that can be mapped to *RDF*. It is often provided on homepages, in journals and via online resources like Citeseer [☞CITESEER]. A more sophisticated solution, like MODS [☞MODS] is probably an obvious extension. Certainly, it is not our intention to constrain users to one particular schema. There is no reasons why semantic bloggers could not describe their resources using a different standard, or even their own ontology.

We also experimented with the use of Easy News Topics (ENT) [☞ENT] to represent categories. In order to support this, we created a proposal for ENT in *RDF* (see ☞appendix). In fact, this representation was unnecessary for the prototype demonstrator, however the possibility still interests us and may form the basis for a future extension.

Lessons Learnt

The use of an *RDF* framework is, we feel, a *sine qua non* which deserves no further discussion. Jena is but one of a number of options available, one which nevertheless worked well for us. We also feel that the use of a preexisting blogging platform was the right one - both for efficiency and for dissemination. The choice of such a platform should be made not just technically but also on the basis of its support community. Using these criteria, we feel that blojsom was an excellent choice.

The choice of ontology used can, of course, be controversial. However the choices we made were pragmatic, for the benefit of the timely implementation of the prototype. The design should make it easy to plug in new ontologies as the tool matures and as user requirements change.

3.3 Information Model Choices - Using a blog as a bibliographic management tool immediately forces us to take a novel viewpoint. Blog entries are separate entities. Bibliographic items are separate entities. *They are not the same*. For example, an entry might be my comment on a particular paper. The author of the entry (the comment) is different to the author of the item (the paper). Similarly, it is possible that two entries on a blog (or even different blogs) might reference the same paper. This enables us to link the two entries while retaining their identities and provenance. In order to model this within the demonstrator we took the view that ***(blog) entries contain (bibliographic) items***

This relationship was modelled by a custom-made 'contains' property in our own semblog namespace. We did consider the re-use of a pre-existing term, however the closest that we found was the RSS annotate module [☞RSS-ANNOTATE] which seemed a more natural fit to blog comments. While inter-vocabulary mappings can be performed, this does not totally obviate the problem. For example, we considered the use of the Annotea threads language [☞ANNOTEA-THREADS]. One problem here is that `partOf` property has a domain of `Post`, constraining the semantics of any mapped property and binding the result perhaps too tightly to the Annotea schema.

We took a number of other less dramatic information modelling choices. These include:

- For the blog entries, the standard RSS properties were used [☞RSS10]. Some Dublin Core [☞DC] additions were included, as well as the RSS content module [☞RSS-CONTENT] (for full, escaped content in the RSS feed). The exact choice of properties is not that significant; we put in a placeholder for friend of a friend [☞FOAF] information and have drafted (see ☞appendix) an *RDF* version of Easy News Topics [☞ENT]. Other properties, drawn from different vocabularies, could easily be added to the metadata. Such is the nature of *RDF*.
- Perhaps of more significance is what we chose to leave out. One such example is the Dublin core 'title' property whose value would be a simple duplicate of the rss 'title'. This does mean that rss-unaware consumers might not pick up the title correctly, however setting up an equivalence class would solve this problem for inference-enabled consumers. Alternatively, we could simply duplicate the property where required. A slight variation on this theme is the use of similar properties for different purposes. For example, both RSS and Dublin Core have a description element. Although this element should perhaps hold a *summary* of the content, many RSS feeds use the element to contain the *full* content. This use, though controversial, is of undoubted use to an RSS aggregator. So one could make one description (say Dublin Core) the abbreviated content, and one (RSS) the full content.
- Bibliographic metadata is modelled using BibTeX [☞BibTeX]. This standard has all the fields we are likely to need and yet is fairly simple to implement. In addition, a BibTeX to *RDF* translator [☞BIBTEX2RDF] is already available, with its own ontology [☞ONTOWEB]. There are, however, some limitations with BibTeX. A more complicated model, FRBR [☞FRBR], has a notion of works themselves and instantiations of that work (for example, the Italian translation of 'Othello', or the 2nd edition of 'Java Servlets'). While BibTeX can differentiate between such items (for example, by ISBN number, or year of publication) there is no reliable way to group such related items together (Title and author will work in some, but not all, cases). A second limitation is that BibTeX does not constrain fields. For example, there is no imposed formatting for year or page number (and although there is a formatting rule for authors, the field itself is unconstrained). It is possible that datatypes, or even controlled vocabularies, might be a useful future extension to an *RDF* encoding of the BibTeX schema.
- As mentioned, the bibliographic items are annotated using the ontoweb schema [☞ONTOWEB]. This is largely because this ontology is used by the BibTeX to *RDF* converter [☞BIBTEX2RDF] that we use. In addition, the schema has made some, in our view, sensible modelling choices. For example, people are modelled as separate resources, giving a framework for asking queries such as "Who has blogged a paper written by X?". On the downside, the ontology is a little large and monolithic, covering many more things than bibliographies. Yet the bibliographic metadata itself, being based on BibTeX, is necessarily limited. An alternative to explore would be an *RDF* encoding of MODS [☞MODS] or some suitable subset thereof [☞MODS-DARCUS].
- The topics are modelled using Thesaurus Interchange Format (tif: [☞TIF]). This is a schema primarily designed for thesauri, yet one that maps well to our domain. The central idea is that a topic tree can be built using topics (called concepts in *TIF*) and 'broader'/'narrower' relationships. Each topic is identified by a set of indicator terms, one of which is the preferred term. This allows us to build in a (very simple) natural language mapping from blog entries to our concept tree (this mapping is implemented in our category chooser). An alternative would be to maintain a simpler topic tree representation, but to tie the topic names to a natural language ontology such as

WordNet [☞WORDNET]. Although intuitively appealing, such an approach has two limitations. Firstly, it is somewhat more complex than the approach we took. Secondly, a general purpose resource such as WordNet fails to capture the precise and highly individual nature of topics in our blog. Using *TIF*, an output of *SWAD-E* workpackage 8 [☞SWADE-THESAURUS] had another beneficial effect, which was increased interaction between the *SWAD-E* workpackages. The interested reader should be aware that *TIF* has now been replaced by *SKOS* [☞SKOS], a deliverable of the same activity.

- We generally eschewed two-way links. For example, a blog entry contains a bibliographic item. By the same token, a bibliographic item is contained in a blog entry. This second relationship could be encoded explicitly, or it could be inferred. We took the view that (in Jena at least) it is trivial to ask the question "What blog entries contain this item?" and so the reverse connection is not required.
- Schemata were designed to control the *UI*. Our general intention was to make these schemata as simple as possible. That is, rather than trying to anticipate all future needs, or architecture in very fine grain control of presentation, the ontologies would only govern simple and coarse grain characteristics, such as properties to show/hide, view type (eg table, record card) and so on. One subtlety that is worth mentioning is that we use a choice of 'opt-in' or 'opt-out' styles for displaying properties. The opt-in style means that only properties explicitly named in the configuration file will be shown. This style works well for display types like tables, where one would like to specify in advance the number of columns and their contents. The opt-out style, on the other hand, shows all properties except those specifically named. This is good for views like record card (and edit) where one wants to display all the metadata for an entry (except perhaps the 'internal use only' metadata). Such an approach ties in well with the 'open world' nature of *RDF* - that is, one does not have to anticipate all metadata that might be relevant for an entry. Finally, a design-time decision was to hide statements whose object is a resource in edit view. This is because editing a resource URI in a free text field is not generally an option we want to provide for the user. More detailed comments on the individual *UI* schemas themselves can be found in the appropriate design sections.

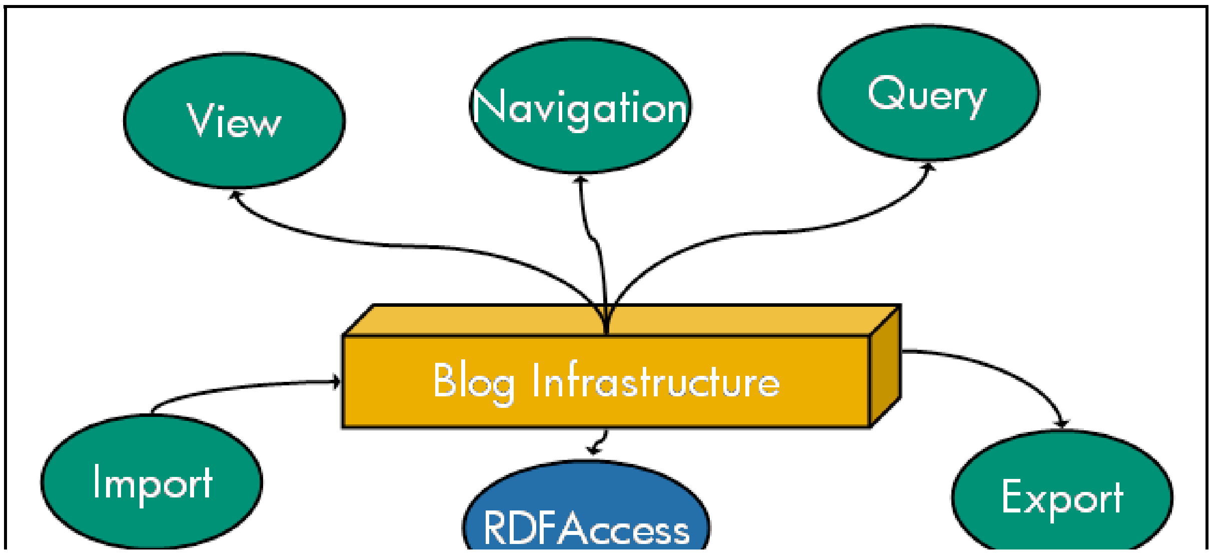
Lessons Learnt

- The `semblog:contains` property was a useful device. Defining one's own property however always brings with it possible future interoperability issues.
- Use of multiple vocabularies for blog (and bibliography) metadata worked seamlessly, as one would expect. The issue here is deciding which properties to choose (and to leave out), in order to avoid bulking up the RSS feed unnecessarily.
- Bibliographic metadata requirements were captured well using BibTeX, and future refinements should be possible. In this context, we note that the BibTeX to *RDF* translator [☞BIBTEX2RDF] is a useful resource, and that Michel Klein deserves credit for making this available.
- *TIF* worked well for us too, and ongoing work should make this a generally useful resource for many information modelling needs.
- The configuration schema worked well for us - the opt-in/opt-out styles were particularly useful.

Other more detailed information modelling considerations are discussed in ☞the appendix.

3.4 Design Choices - Recall that our aim is to demonstrate three capabilities: semantic view, semantic navigation and semantic query. Within the context of bibliographic management we have demonstrated these capabilities in the following ways.

- **Semantic View**: Metadata driven blog views include 'record card' format (particularly appropriate for bibliographic items) and summary tables.
- **Semantic Navigation**: A metadata driven navigator window provides a category tree browser and a simple facet-type interface.
- **Semantic Query**: A metadata driven query window allows queries over user-selected metadata.



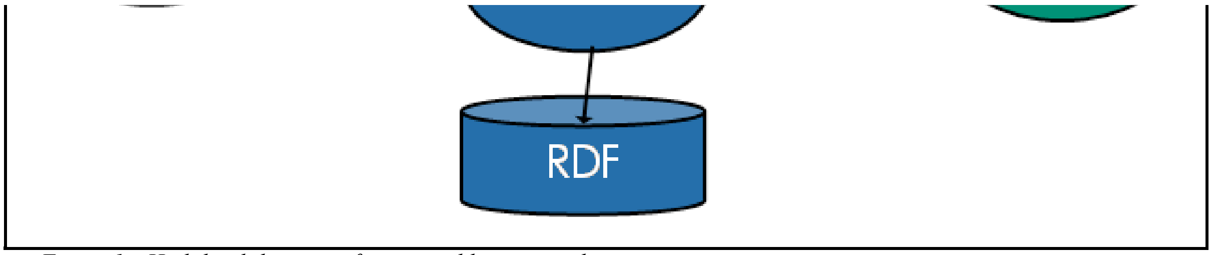
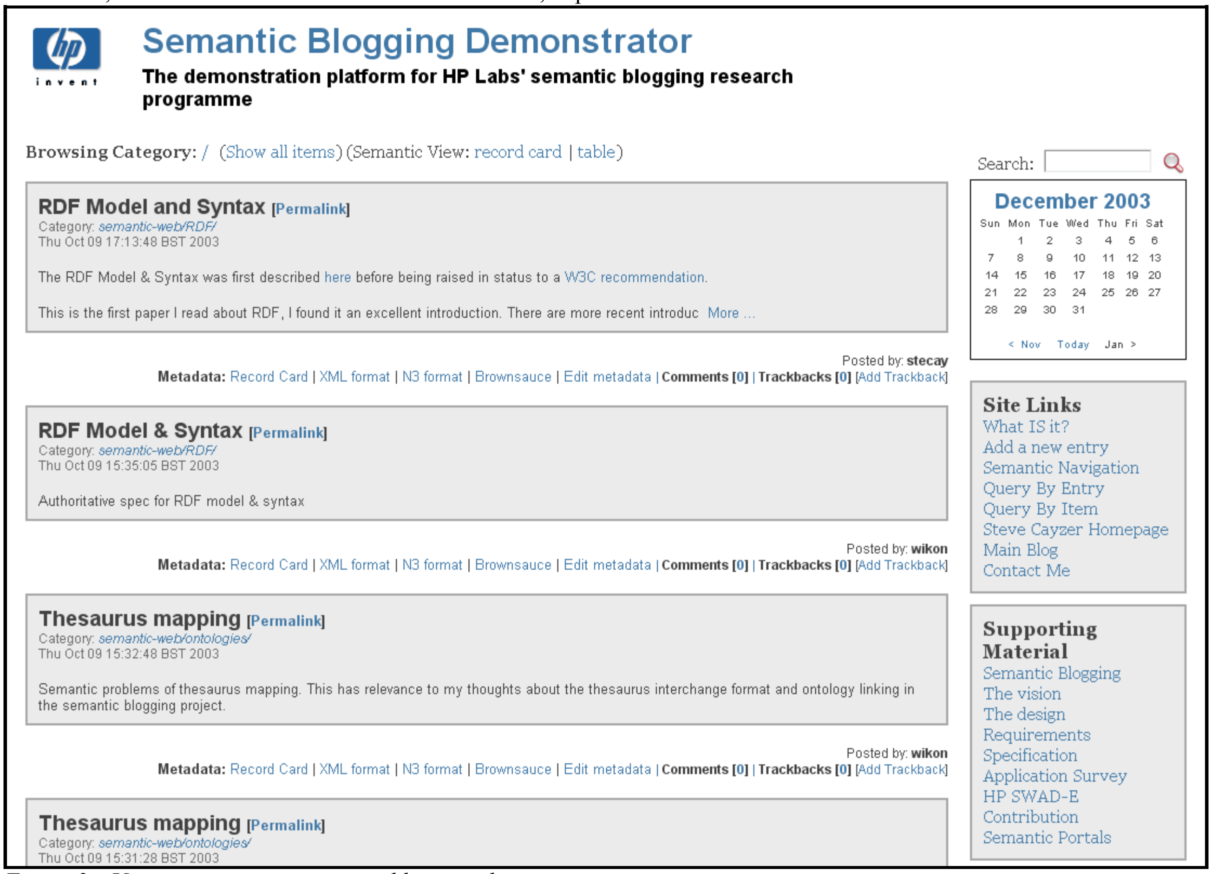


Figure 1 - High level diagram of semantic blogging architecture

We have built these capabilities over a semantically enabled blog. Figure 1 shows the basic architecture (more design details can be found in [☞SEMBLOG-DESIGN], and the javadoc is at [☞SEMBLOG-JAVADOC]). In the bottom part of the figure, the blog is semantically enabled by providing it with an *RDF* backend. An *RDF*Access component provides an interface to this metadata store, while import and export complete the blog metadata pipeline. Metadata view and edit functionalities (not shown in the diagram) are also provided. The semantic view, navigation and query functionalities, which demonstrate the semantic web values, exploit this basic infrastructure.



Semantic Blogging Demonstrator

The demonstration platform for HP Labs' semantic blogging research programme

Browsing Category: / (Show all items) (Semantic View: record card | table)

Search:

December 2003

Sun	Mon	Tue	Wed	Thu	Fri	Sat
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

< Nov Today Jan >

RDF Model and Syntax [Permalink]
Category: *semantic-web/RDF/*
Thu Oct 09 17:13:48 BST 2003

The RDF Model & Syntax was first described here before being raised in status to a W3C recommendation.

This is the first paper I read about RDF, I found it an excellent introduction. There are more recent introduc More ...

Posted by: **stecay**
Metadata: Record Card | XML format | N3 format | Brownsauce | Edit metadata | **Comments [0]** | **Trackbacks [0]** [Add Trackback]

RDF Model & Syntax [Permalink]
Category: *semantic-web/RDF/*
Thu Oct 09 15:35:05 BST 2003

Authoritative spec for RDF model & syntax

Posted by: **wikon**
Metadata: Record Card | XML format | N3 format | Brownsauce | Edit metadata | **Comments [0]** | **Trackbacks [0]** [Add Trackback]

Thesaurus mapping [Permalink]
Category: *semantic-web/ontologies/*
Thu Oct 09 15:32:48 BST 2003

Semantic problems of thesaurus mapping. This has relevance to my thoughts about the thesaurus interchange format and ontology linking in the semantic blogging project.

Posted by: **wikon**
Metadata: Record Card | XML format | N3 format | Brownsauce | Edit metadata | **Comments [0]** | **Trackbacks [0]** [Add Trackback]

Thesaurus mapping [Permalink]
Category: *semantic-web/ontologies/*
Thu Oct 09 15:31:28 BST 2003

Site Links
What IS it?
Add a new entry
Semantic Navigation
Query By Entry
Query By Item
Steve Cayzer Homepage
Main Blog
Contact Me

Supporting Material
Semantic Blogging
The vision
The design
Requirements
Specification
Application Survey
HP SWAD-E
Contribution
Semantic Portals

Figure 2 - User entry point to semantic blogging demonstrator

The blog itself is presented to the user as follows in figure 2. This figure shows a standard blog interface - it is important that the user introduction to semantic blogging should be as familiar as possible. So the blog entries are displayed as separate boxes on the page, you can browse them individually or by category just as you would expect from a normal blog. The semantic capabilities are accessed by links on the sidebars ("Query by Entry/Item") and on the items themselves ("Record Card", "Edit metadata").

We now describe the basic design of each blog component, evaluating our design choices at each stage. Full design details for the components are available in the appendix.

3.4.1 *RDF*Access

*RDF*Access is the control point to the *RDF* data store which backs the semantic blog. This is effectively an API which allows access to various storage implementations (memory, file access and database). Queries, additions and modifications to *RDF* data are handled by this component. Tied into this component is the RSS generation (ie RSS feeds should query this database to get the metadata for each blog item rather than generating it themselves, separately from and possibly in conflict with the *RDF* store). In our demonstrator, the implementation of the *RDF* store was a single file containing metadata for all blog items. The key functionality exposed by *RDF*Access includes:

- Generic methods for adding, removing and updating metadata in a store, configuring properties (eg store location and attributes) and performing queries (of various types, we return a model with the results).
- Specialized "blog entry"-level operations; for example, get and remove entries. Both deal with all metadata relevant to an entry (this will do transitive searches to find, for example, metadata on

the underlying bibliographic item. Simple caching is employed to deal with potential cycles). You can also do context sensitive updates - eg update metadata only where relevant to a particular entry

- Specialized query operations; for example, RDQL queries returning a list of (string) variable bindings. This is useful to avoid the need for any *RDF* model handling by the caller (eg a *JSP*).

In our initial experiments, we used the approach of storing a separate *RDF* file for each blog entry. This seems intuitively reasonable, as most blogs are file based (ie one file per blog entry). But the approach will not work for the blog metadata, since the user will often want to query over the entire blog metadata. Storing aggregate metadata in a central *RDF* store seems essential to support this. We did find one problem with this approach however, which is that quality control becomes increasingly important. Using a single file, the store becomes vulnerable to small metadata errors (which are quite easy to generate during development). Presumably an *RDF* database will be less vulnerable to such errors, but brings its own drawbacks, such as infrastructure requirements and performance hit. However, clearly the single file approach, quite apart from its fragility, is not scalable to very large blogs. We note however that each blog entry carries with it only a modest amounts of metadata, and that the blog has been growing for over 6 months without any noticeable performance degradation. In addition, *RDF* database solutions necessarily imply a deployability hurdle.

It is instructive to compare our design choices with those laid out by the recent NetAPI proposal [☞NETAPI] and implemented by Joseki [☞JOSEKI]. Here, abstract methods are proposed for operations such as get, insert, remove and query. The last method is the most relevant here, since our design is very similar. In both NetAPI and *RDF*Access, the query is given a model reference and a query language. The query is then carried out in a server specific way (for example, the use of inferencing is up to the server). In the case of *RDF*Access, a 'chaining' rule is employed to return all metadata relevant to a blog entry, whether directly (eg title) or indirectly (eg author of an enclosed bibliographic item) linked. Clearly this mechanism needs protection against cyclic data (*RDF*Access does this through a simple caching mechanism), and would be poorly suited to highly interlinked data. But, equally, different local mechanisms could be employed to deal with different data models, all without changing the API.

*RDF*Access was designed without reference to Joseki. It is interesting then to see how similar the two designs are. Clearly there is some common underlying need here. One minor difference is that NetAPI allows a choice of results format, a requirement that is sidestepped in *RDF*Access by simply returning the satisfying model. A more significant difference is that *RDF*Access was designed for local access rather than networked access. Given the similarity of underlying design, it would seem reasonable to recommend basing any future networked implementation (ie a semantic blog aggregator) on Joseki.

Lessons Learnt

- One *RDF* file per blog entry is the wrong way to go. We need an *RDF* store for the whole blog.
- A file backed solution works quite well for the scale of our demonstrator. Various approaches could be investigated in future ranging from database solutions to caching - the *RDF*Access abstraction is a good way to enable this.
- The NetAPI protocol would appear to form a good basis for future extensions.
- Deployability and performance are important considerations for a blog.

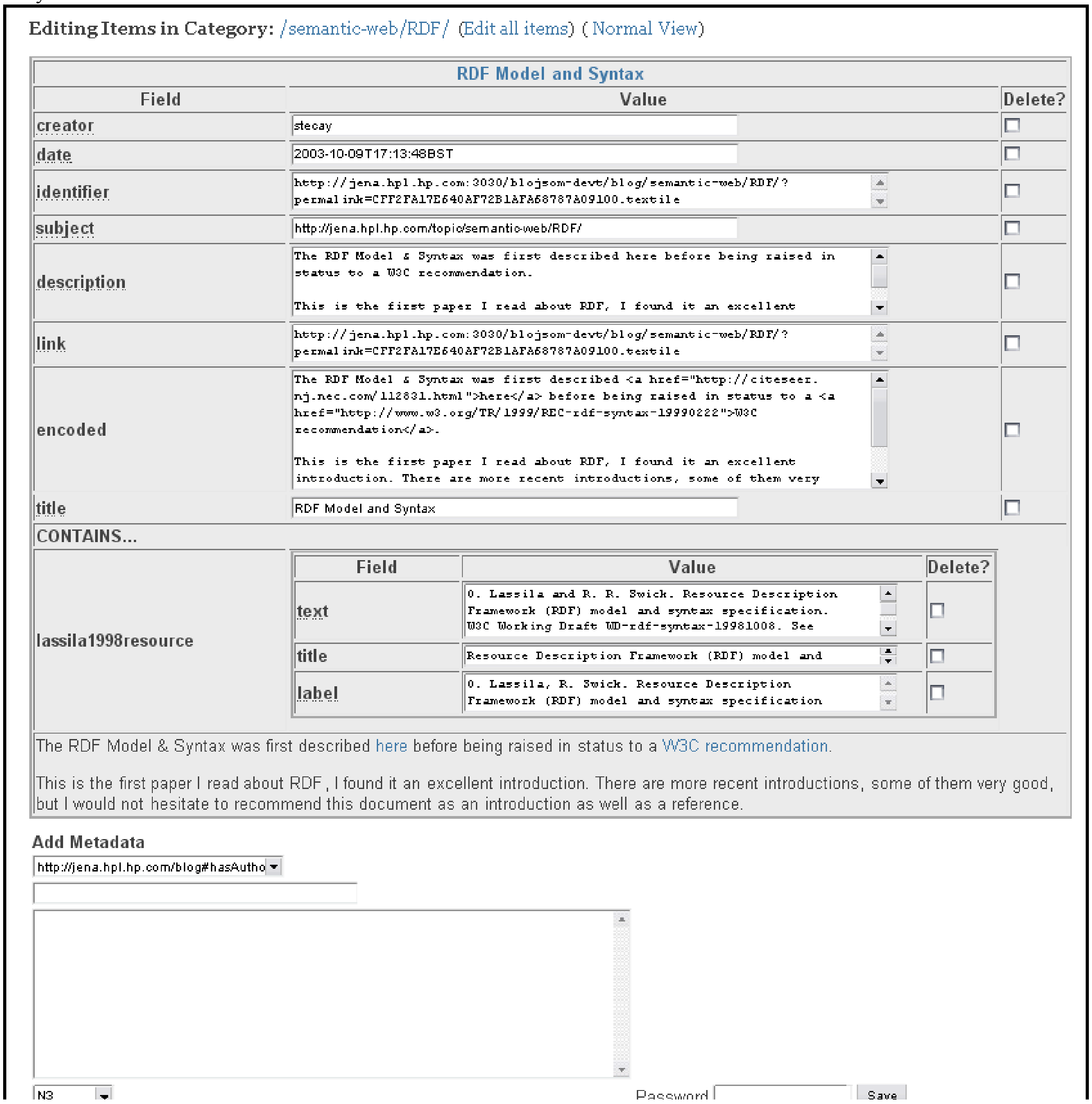
3.4.2 Metadata edit and view

One of the first components to be built was a metadata viewer. It is important at least to be able to look at the metadata behind each blog entry. This is accomplished fairly easily in Jena by using a specialised writer (eg "N3" or "*RDF/XML*"). In addition, third party metadata viewers such as brownsauce [☞BROWNSAUCE] can easily be invoked, since given the *RDF*Access functionality described above, we can use a simple HTTP GET to retrieve the *RDF* relevant to any individual blog entry. An (abbreviated) example of the metadata behind a single blog entry is shown in below; note that this (blog) entry metadata includes (bibliographic) item metadata. This is data that would not normally be present in an RSS feed, yet is relevant and useful for a bibliographic blog.

```
   @prefix tif: <http://www.limber.rl.ac.uk/External/thesaurus-iso.rdf#> .
@prefix semblog: <http://jena.hpl.hp.com/semblog#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix : <http://purl.org/rss/1.0/> .

<http://jena.hpl.hp.com/~stecay/downloads/semblog.bib#o_lassila>
   a <http://www.ontoweb.org/ontology/1#Person> ;
   <http://www.w3.org/2000/01/rdf-schema#label> "O. Lassila" ;
   <http://www.ontoweb.org/ontology/1#name> "O. Lassila" .

<http://jena.hpl.hp.com/~stecay/downloads/semblog.bib#lassila1998resource>
   a <http://www.ontoweb.org/ontology/1#Misc> ;
      <http://www.w3.org/2000/01/rdf-schema#label>
      "O. Lassila, R. Swick. Resource Description Framework (RDF) model and syntax
specification" ;
   <http://www.ontoweb.org/ontology/1#author>
      <http://jena.hpl.hp.com/~stecay/downloads/semblog.bib#o_lassila>              ,
<http://jena.hpl.hp.com/~stecay/downloads/semblog.bib#r_swick> ;
   <http://www.ontoweb.org/ontology/1#title>
      "Resource Description Framework (RDF) model and syntax specification" ;
   <http://www.ontoweb.org/ontology/1#url>
      <http://citeseer.nj.nec.com/article/lassila98resource.html> .
```

```
<http://jena.hpl.hp.com:3030/blojsom-devt/blog/semantic-web/
RDF/?permalink=CFF2FA17E640AF72B1AFA68787A09100.textile>
   a rss:item , semblog:blogItem ;
   <http://www.w3.org/2000/01/rdf-schema#seeAlso>
<
http://jena.hpl.hp.com:3030/blojsom-devt/blog/semantic-web/
RDF/?permalink=CFF2FA17E640AF72B1AFA68787A09100.textile&flavor=RDFXML> ;
   semblog:contains <
http://jena.hpl.hp.com/~stecay/downloads/semblog.bib#lassila1998resource> ;
   semblog:hasConcept <http://jena.hpl.hp.com/topic/semantic-web/RDF> ;
   dc:creator "stecay" ;
   dc:date "2003-10-09T17:13:48BST" ;
   dc:identifier         "http://jena.hpl.hp.com:3030/blojsom-devt/blog/semantic-web/
RDF/?permalink=CFF2FA17E640AF72B1AFA68787A09100.textile" ;
   dc:subject "http://jena.hpl.hp.com/topic/semantic-web/RDF/" ;
   rss:description "The RDF Model & Syntax was first described here before being
raised in status to a W3C recommendation ";
   content:encoded    "The    RDF    Model    &    Syntax    was    first    described    <a
href=\"http://citeseer.nj.nec.com/112831.html\">here</a>    before    being    raised    in
status    to    a    <a    href=\"http://www.w3.org/TR/1999/REC-rdf-syntax-19990222\">W3C
recommendation</a>.

This is the first paper I read about RDF, I found it an excellent introduction. There
are more recent introductions, some of them very good, but I would not hesitate to
recommend this document as an introduction as well as a reference. " ;
   rss:title "RDF Model and Syntax" .
```

It is important to be able to change the metadata once it has been entered. We provide a password-controlled metadata edit form to do this. This edit form is invoked by pressing the 'edit this metadata' link below each blog entry. The edit metadata dialog, shown in figure 3, has three main elements. Firstly, a list of fields relevant to that blog entry *and any associated bibliographic item*. Secondly, a combo-box used for adding certain fields to this entry. Finally, a free text box for adding arbitrary metadata in N3 or *RDF/XML* format.



Editing Items in Category: /semantic-web/RDF/ (Edit all items) (Normal View)

RDF Model and Syntax

Field	Value	Delete?
creator	stecay	☐
date	2003-10-09T17:13:48BST	☐
identifier	http://jena.hpl.hp.com:3030/blojsom-devt/blog/semantic-web/RDF/?permalink=CFF2FA17E640AF72B1AFA68787A09100.textile	☐
subject	http://jena.hpl.hp.com/topic/semantic-web/RDF/	☐
description	The RDF Model & Syntax was first described here before being raised in status to a W3C recommendation. This is the first paper I read about RDF, I found it an excellent	☐
link	http://jena.hpl.hp.com:3030/blojsom-devt/blog/semantic-web/RDF/?permalink=CFF2FA17E640AF72B1AFA68787A09100.textile	☐
encoded	The RDF Model & Syntax was first described <a href="http://citeseer.nj.nec.com/112831.html">here</a> before being raised in status to a <a href="http://www.w3.org/TR/1999/REC-rdf-syntax-19990222">W3C recommendation</a>. This is the first paper I read about RDF, I found it an excellent introduction. There are more recent introductions, some of them very	☐
title	RDF Model and Syntax	☐

CONTAINS...

lassila1998resource

Field	Value	Delete?
text	O. Lassila and R. R. Swick. Resource Description Framework (RDF) model and syntax specification. W3C Working Draft WD-rdf-syntax-19981008. See	☐
title	Resource Description Framework (RDF) model and	☐
label	O. Lassila, R. Swick. Resource Description Framework (RDF) model and syntax specification	☐

The RDF Model & Syntax was first described here before being raised in status to a W3C recommendation.

This is the first paper I read about RDF , I found it an excellent introduction. There are more recent introductions, some of them very good, but I would not hesitate to recommend this document as an introduction as well as a reference.

Add Metadata

http://jena.hpl.hp.com/blog#hasAutho

N3

Password

Save

Figure 3 - Screenshot of metadata editing on the semantic blogging demonstrator

The 'add metadata' combo box is schema controlled. That is, a simple *RDF* file controls the properties that are available. In fact, the schema simply lists the properties that populate the combo box. For the rest of the metadata , the decision is made simply to display all the properties of the blog item - with the exception of resources, which cannot be sensibly edited. One subtlety is that editing metadata is different to editing the blog entry itself. So changes to the metadata will not change the entry text. Such changes are outside the scope of the demonstrator and in any case are handled well by conventional blogging functionality like w.bloggar [☞WBLOGGAR] and bledit [☞BLEDIT].

Lessons Learnt

Using an *RDF* toolkit made it easy to provide basic view functionality. It is also nice to link to third party functionality, but to do so we need to generate metadata 'on the fly' and at an externally retrievable location. Editing metadata is important, but the functionality shown here is probably just a start. For example, editing of *RDF* statements whose object is a resource would be a 'nice to have', though it would complicate the *UI*. Similarly, showing the difference between editing the metadata and editing the entry might be achieved through a more sophisticated *UI*. An alternative would be to abolish the difference by simply abolishing non-metadata content (there is only metadata; the blog entry is created directly from the metadata).

3.4.3 Import

Blog entries are easily created using our SemBlogIT! utility. The idea in is not novel; similar functionality is provided by MovableType [☞MT] bookmarklets, and a 'blog this page' utility is available on the Google toolbar [☞GOOGLE-TOOLBAR]. So, if you want to blog the URL you are browsing, one click produces an entry form populated with a title, a link to that URL, and any text you have highlighted. You then add the details you want, choose a category and post the entry. The posting process is password controlled.

We have added two semantic capabilities to SemBlogIT!

- Firstly, a way to attach items to blog entries. So, if you wanted to blog about a bibliographic item (say, a paper), you can enter metadata about that paper. An 'add bibliographic item' popup provides space for bibliographic fields such as author and title. Alternatively, prepared metadata (such as that produced by an online BibTeX to *RDF* converter [☞BIBTEX2RDF]) can be entered directly. In either case, the blog entry will be marked as containing the bibliographic item. Incidentally, the popup dialog is itself schema-controlled, so that the choice of fields to enter is determined by the user. The intention here is to show how *RDF* powering user interfaces provides another advantage of semantically enabling a blog.
- The **Category Chooser** is a form of assisted metadata creation. After submission, the demonstrator suggests a number of categories for a blog entry. This is implemented using an ontology, represented in *TIF* [☞TIF] format. This ontology essentially represents a hierarchy of concepts, each with a set of indicator terms (think of keywords), one of which is the preferred term for that concept. The category chooser client goes through the text for a blog entry, stemming words and matching them with the indicator terms. If a term is matched, then that concept is suggested. This is a form of ontology driven assisted metadata creation.

Lessons Learnt

Import processes provide a good way to demonstrate machine assisted metadata creation. The assistance can itself be semantically assisted (eg categorization backed by *TIF* hierarchy). The challenge here is to enable semantic enrichment of blog entries without increasing the complexity of the user task. In addition, schema controlled forms provide a perhaps more flexible way of customisation than do conventional templates. Future work should be focused around the integration of other tools (for example, automatically extracting metadata from a Citeseer page, or integrating the BibTeX to *RDF* converter into SemBlogIT!)

3.4.4 Export

Most blogs support metadata export in the form of RSS. The demonstrator uses the *RDF* flavour of RSS, RSS1.0. This format provides the usual *RDF* mechanisms for extension and vocabulary mixing. Thus, the RSS feed contains information over and above what is specified by the RSS1.0 specification; for example, relevant bibliographic metadata linked to the requested blog entries. Although we have not implemented an aggregator, it is simple to imagine that such enriched feeds could be collected and that community versions of the semantic services described below could be provided.

Rather than using a static template (which most blogs do), our export consults the *RDF* store. Hence, we can retrieve arbitrary metadata at feed request time, metadata whose existence was not anticipated by any template. This can be both a boon and a curse. Clearly it is a more flexible way to generate blog metadata. On the other hand, this very flexibility might lead to problems for RSS consumers. Most RSS1.0 consumers are not full *RDF* parsers (an exception being NewsMonster [☞NEWSMONSTER]). Thus, *RDF* that doesn't fit the consumers 'template' for RSS1.0 is at risk of being rejected. One obvious risk is that the arbitrary ordering of *RDF* statements, and the graph structure of the *RDF* model, both allow for multiple, equally valid serializations. Such serializations can fall foul of aggregator rules. For example, the channel metadata is normally expected to precede the list of items which in turn precedes the item metadata. Similarly the item metadata is not expected to be serialized within the item list itself. While it is feasible to constrain the serialization syntax to some extent, Jena itself provides no out of the box functionality to do this.

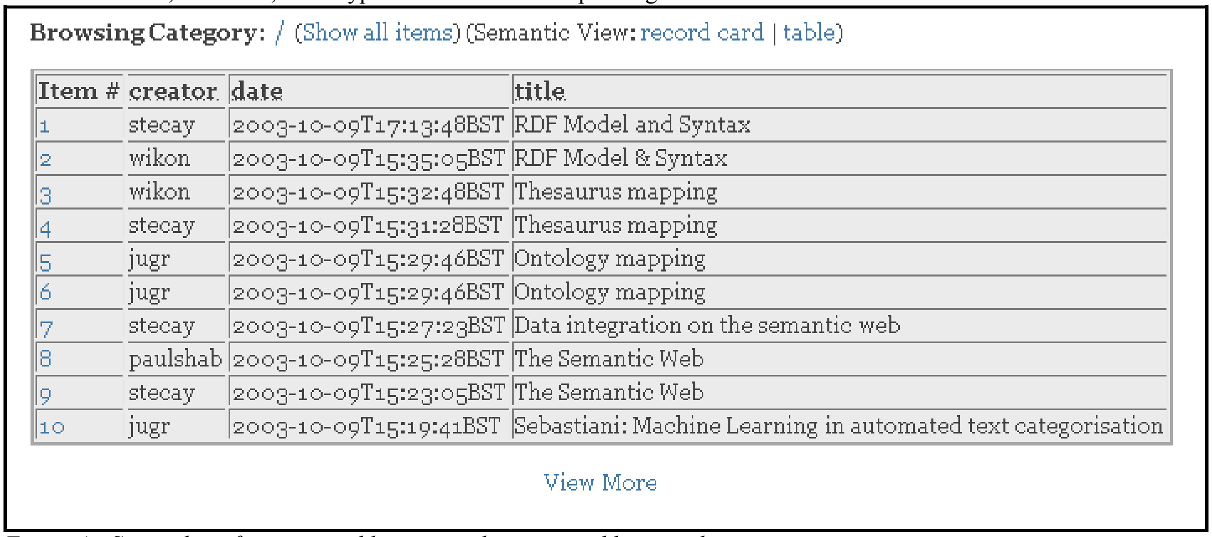
Lessons Learnt

Use of *RDF* in an export feed leads to consumer issues (the term 'consumer' is taken here to mean feed consumers, including RSS aggregators).

- Firstly, how much do you enrich your feed? Extra metadata may be useful to semantically enabled consumers, but for others it causes at least unnecessary bandwidth load. For consumers that are using fragile *XML* schemas to validate the feed, it may even cause parsing errors.
- Secondly, can we overcome consumer issues by using a syntactically constrained version of *RDF*? Techniques like *XML* to *RDF* mapping are being investigated by *SWAD-E* [☞SWADE-XML]. RSS1.0 [☞RSS10] and Atom [☞ATOM] have an accepted *XML* profile. And *XML* syndication languages can be converted to *RDF* [☞SSR]. However, any form of syntactic constraint, though it might make the feed more acceptable to a wider range of consumers, might also constrain its usefulness.

3.4.5 Semantic View

This is the first of the semantic capabilities that we are trying to demonstrate. The idea is that, given we have bibliographic items in the blog, we would like to be able to view these in a useful way. We provide two views. The first is a 'record card' view (which appears like a readonly version of the edit dialog described earlier). In this view, the bibliographic items are described in a sort of indented record card. This enables one to distinguish between the blog entry and the bibliographic items that the entry is about. An alternative way of viewing blog entries is a summary table (figure 4). This shows a list of entries contained in this particular category, with summary information (creator, date, concept and title) for each. The list entries, of course, have hyperlinks to their corresponding full view.



Browsing Category: / (Show all items) (Semantic View: record card | table)

Item #	creator	date	title
1	stecay	2003-10-09T17:13:48BST	RDF Model and Syntax
2	wikon	2003-10-09T15:35:05BST	RDF Model & Syntax
3	wikon	2003-10-09T15:32:48BST	Thesaurus mapping
4	stecay	2003-10-09T15:31:28BST	Thesaurus mapping
5	jugr	2003-10-09T15:29:46BST	Ontology mapping
6	jugr	2003-10-09T15:29:46BST	Ontology mapping
7	stecay	2003-10-09T15:27:23BST	Data integration on the semantic web
8	paulshab	2003-10-09T15:25:28BST	The Semantic Web
9	stecay	2003-10-09T15:23:05BST	The Semantic Web
10	jugr	2003-10-09T15:19:41BST	Sebastiani: Machine Learning in automated text categorisation

View More

Figure 4 - Screenshot of summary table view on the semantic blogging demonstrator

Like semantic edit, semantic view is metadata driven. There is a semview preferences file (expressed in *RDF*) which, while simple, does allow for named views. An excerpt from the semview preferences file we use can be found in ☞the appendix. We used two named views (record and table) but one could also define different views for each user. The required view is selected at runtime through a URL parameter.

Lessons Learnt

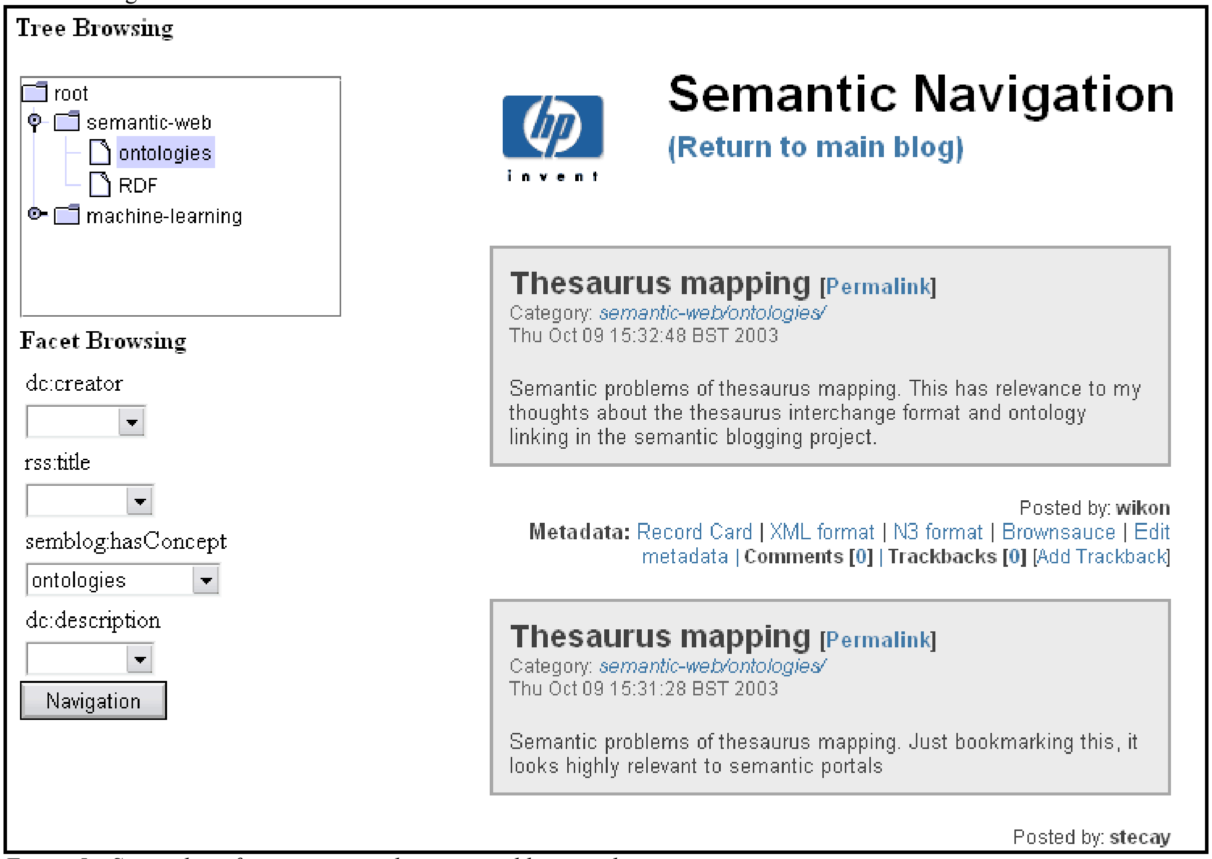
The semantic view component is a good way to show how a blog can look very different with a little effort. We made a deliberate decision not to get carried away with the presentation ontology. That is, we made it as simple as possible for the functionality we required. The idea is that more refined functionality can be added by simple schema extensions. In principle, this seems feasible. On the other hand, we felt that the personalisation solution was a little brittle (this is explored in more detail below). Finally we note that a mechanism to present metadata in a human readable way is a key requirement of the *SWAD-E* semantic portals [☞SWADE-SEMPORTALS] work. The lessons from this section are useful input to that project, and we expect further learnings in this area.

3.4.6 Semantic Navigation

The demonstrator supports two forms of navigation (over and above standard blog navigation modalities):

- **Tree Browsing** Here, a Java applet renders a tree representation of the hierarchical categories. The tree itself is generated from an *RDF* schema (the blog category hierarchy is represented in *TIF* [☞TIF]). An excerpt from the *TIF* metadata for the demonstrator blog can be found in ☞the appendix.
- **Facet browsing** Although this is not strictly facet navigation, the idea here is that data can be selected by choosing an intersection of features. For example, one might choose all entries written by "stecay" on a concept (or subconcept) of the semantic web.

The navigation interface is controlled by an *RDF* formatted configuration file. So one can choose which properties appear in the facet browser, and which values can be chosen for them. Thus, navigation is really a constrained query. Once the user has chosen a set of navigation options (or equivalently submitted a query) the relevant blog entries are displayed in a results frame. An example of navigation can be seen in figure 5.


Tree Browsing

Facet Browsing

dc:creator

rss:title

semblog:hasConcept

ontologies

dc:description

Navigation

Semantic Navigation

(Return to main blog)

Thesaurus mapping [Permalink]
Category: semantic-web/ontologies/
Thu Oct 09 15:32:48 BST 2003

Semantic problems of thesaurus mapping. This has relevance to my thoughts about the thesaurus interchange format and ontology linking in the semantic blogging project.

Posted by: wikon
Metadata: Record Card | XML format | N3 format | Brownsauce | Edit metadata | Comments [0] | Trackbacks [0] [Add Trackback]

Thesaurus mapping [Permalink]
Category: semantic-web/ontologies/
Thu Oct 09 15:31:28 BST 2003

Semantic problems of thesaurus mapping. Just bookmarking this, it looks highly relevant to semantic portals

Posted by: stecay

Figure 5 - Screenshot of navigation on the semantic blogging demonstrator

Lessons Learnt

Semantic navigation works reasonably well on our demonstrator, but it is not the most compelling aspect, This is for a number of reasons. Firstly, the tree browser simply uses hierarchical categories, which are a feature of many blogs anyway. A mild form of inferencing ('show me items at or below this category) would be useful. More compelling would be a tree browser which used a different hierarchical property. Unfortunately our metadata (and arguably our domain) doesn't have such useful examples. Clearly one can construct artificial examples ('authors starting with A', 'blog entries for November 2003'...) but more useful would be true hierarchies such as geography, species, genealogy and the like. Such features would overcome our second limitation, and provide the facet browser with a rather more interesting interface. Another limitation is that the navigation is based on blog entries rather than bibliographic items (a limitation that we overcame for query, see below). Thus, the semantic navigation in the demonstrator doesn't really expose the rich structure of the blog. More positively, the navigation experience (browsing) can be made to feel quite different to query, even though the underlying mechanism is the same. This is useful and interesting, although in fairness it is probably not wholly ascribable to the use of *RDF*!

3.4.7 Semantic Query

The demonstrator implements two forms of semantic query. The first ("Query By Entry") presents a set of fields, corresponding to blog entry metadata. The dialog looks like the facet navigator, except that the user can enter unconstrained text in each field. Again, the exact form of the display is schema driven. Once the user submits the query, the relevant entries are retrieved from the blog database on the basis of their metadata. The results are piped into semantic view - hence, one can choose a preferred format (such as record card or table form).

The second form of query ("Query By Item") allows a search for blog entries about a certain paper (or about papers on a certain topic, or by a certain person...). The query works exactly the same as before, except that the entries returned are those about the items of interest.

Lessons learnt

Semantic Query again worked well within the scope of the demonstrator. It is quite different to navigate by item or entry, and there are probably better ways to indicate this to the user. More generally, it is important to have a simple way of presenting the results of a query. We used semantic view, but other approaches such as *RDF* Objects[☞RDF-OBJECTS] or brownsauce [☞BROWNSAUCE] are also possible. Again, this is an ongoing *UI* issue.

3.4.8 Overall Design Lessons

There are a number of key lessons to be learnt from our design choices. Clearly, these are only demonstrably relevant to our prototype, but we hope that some might be generalizable to other semantic web applications.

Platform Dependence

We started this project with a goal of making the architecture, as far as is practical, pluggable, so that in principle the component (which we'll call semblog) can be taken and plugged into a different blogging system. One way to implement an interface would be to have an blog platform plugin calling out to semblog utilities directly. This allows the blogging templates to be used in the normal way, but creates considerable tie in to the blogging tool. A more generalizable approach is to invoke semblog directly as a servlet, and to communicate with the underlying blogging tool using *XML*-RPC (and a standard API such as Blogger [☞BLOGGER-API] or Atom [☞ATOM-API]). However, this generalization comes at a high cost. In particular, input/output is limited to that supported to the API, and seamless integration with the blogging environment is more difficult. We found that such a purist approach was untenable for a rich semantic blogging experience. We used a mix of approaches. For instance, SemBlogIT! is a semblog component which uses an *XML*-RPC interface to communicate with blojsom. Metadata view is implemented as a blojsom plugin which calls out to semblog functionality.

RDF for Machines

RDF processing is not trivial. True, there are many specification-compliant parsers such as ARP [☞JENA-ARP] and Raptor [☞REDLAND-RAPTOR] readily available. And the *XML* serialization of *RDF* means that a basic level parsing can be done by any *XML* parser. But this second point causes a potential issue. While the *XML* serialization is an essential component of *RDF*, and has undoubtedly contributed to the success of *RDF* so far, the interoperability it brings carries with it a risk. The risk is that people will depend on some 'canonical' *XML* serialization of an *RDF* model. This has happened for RSS1.0. Many RSS aggregators rely on a profile that the semantic blogging demonstrator is not guaranteed to provide (for example, the channel elements first). It is possible (with some work) to provide a syntactically constrained *RDF*, which make the feed more acceptable to a wider range of consumers. It is also possible to use a more liberal *XML* schema notation, such as RelaxNG [☞RELAXNG]. Transformations also exist to bridge between *RDF* and *XML* [☞SWADE-MAPPING] and vice versa [☞SSR]. All these methods help. Treating *RDF* as *XML*, however, always carries with it a risk that the constraints might limit its usefulness. It is probably important to say in this context that our extensions to the RSS feed did not *in themselves* cause problems to the aggregators we tried. And the extensibility of RSS1.0 (through the modules) seems to indicate that there is no fundamental barrier to semantic enrichment, rather that there are purely syntactic hoops to jump through. However, our tests were far from exhaustive and the extensibility issue is an ongoing issue, both for RSS1.0 and the emerging Atom [☞ATOM] standard.

RDF for Humans

A related topic is how you present the *RDF*. *RDF*/*XML* is often criticised for being verbose and difficult to understand (by humans). At the (perhaps) trivial level, the way it looks affects how easily people can understand it, and thus accept it. In addition, there is an argument that "developers learn [a specification] primarily through the process of viewing and copying existing implementations" [☞VIEWSOURCECLAN]. Applications such as brownsauce [☞BROWNSAUCE] are designed to hide the minutiae of machine readable syntax and present a human friendly face to the *RDF*. However, this is of little help to the developer, or user who wants to 'hand-roll' RSS. Such people can be assisted by syntaxes such as N3 [☞N3], which are designed to aid both efficiency and legibility. Of course, such syntaxes are certain to be unacceptable to many RSS consumers (machines). On the other hand an N3 encoding makes the *RDF* nature apparent, which could arguably be a good thing. That is, it could be argued that semantic RSS consumers ought to be aware that the data model is RDF, and ought to be able to handle this. Thus the RSS producer can extend the metadata in arbitrary ways, confident that the RSS consumer is *RDF*-aware. One could imagine a semantic blog offering both semantic and non-semantic RSS feeds to satisfy both needs.

RDF everywhere

In the demonstrator, we experimented with using *RDF* for internal configuration options. In general, this worked well. Simple schema allowed base level customisation of query, navigation, view and entry user dialogues. We deliberately chose as simple schema as possible, because these would be easy to customise, and because they would be extensible. We also provided a configurable query to access these configuration files. For example, to select the format preference for a view, the `view.properties` file contains a property `formatQuery` with an associated `bindingVariable` and `viewnameVariable`. An example follows:

```
   contains=http://jena.hpl.hp.com/semblog#contains
prefFile=/home/stecay/programs/jakarta-tomcat-4.1.24/webapps/blojsom/semview.n3
prefLanguage=N3
formatQuery=SELECT ?x, ?y WHERE (?a <http://jena.hpl.hp.com/semview#formatPreference>
?x), (?a <http://jena.hpl.hp.com/semview#viewName> ?y)
filterQuery=SELECT ?x, ?y WHERE (?a <http://jena.hpl.hp.com/semview#filterPreference>
?x), (?a <http://jena.hpl.hp.com/semview#viewName> ?y)
showPropsQuery=SELECT ?x WHERE (?a <http://jena.hpl.hp.com/semview#displayPreference>
"show"), (?a <http://jena.hpl.hp.com/semview#controls> ?x), (?a
<http://jena.hpl.hp.com/semview#viewName> ?y)
hidePropsQuery=SELECT ?x WHERE (?a <http://jena.hpl.hp.com/semview#displayPreference>
"hide"), (?a <http://jena.hpl.hp.com/semview#controls> ?x), (?a
<http://jena.hpl.hp.com/semview#viewName> ?y)
bindingVariable=x
viewnameVariable=y
```

The intention behind this is to reduce the need for schema dependence in the code, and to allow people to choose (in principle) their own configuration schemata. With hindsight, this approach didn't really work. The problem here is that the specification of 2 RDQL variables actually constrains the form of the query (and hence the way of expressing preferences) to uninteresting variants. Thus, the intention (which was to generalize the preference mechanism by use of a run time configurable `formatQuery`) is largely thwarted. We did experiment with other methods. For example, the use of (an arbitrary number of) constraints, each with its own binding variable and URL parameter. However, we felt that such an approach was adding too much complexity for the advantage gained.

Therefore, while we would certainly advocate the use of *RDF* for *UI* configuration, we would urge caution in the mechanisms used to promote generality, plugability and customisation of the configuration schemata.

Choose interesting metadata

One of the downsides to our navigation is that there is only one usefully navigable, hierarchical field - category. It is useful to query by author or by date (eg month) but these fields are by their nature flat. A more rewarding navigation experience would be using truly faceted fields (a canonical example is 'grape, region, price/age' for wine). We hope to explore this possibility in our next project, semantic portals [☞SWADE-SEMPORTALS].

3.5 Demonstrating semantic web values - Let us now examine the three semantic web values that we wished to demonstrate. They were data representation, semantics and webness. The first value is quite well demonstrated here. The common data representation afforded by *RDF* has allowed us not only to encode and extend the blog metadata, but also to integrate information (for example, between two blog entries about the same paper). One could also imagine taking this extension and integration further - for example, by adding metadata about entry or item authors (using a vocabulary like *foaf* [☞FOAF]). The second value, semantics, is not really apparent in the demonstration. However, the idea of inferencing is the logical next step for semantic query. Even in the simplest case, returning entries at *or below* a topic node might be useful. Moving to community-based aggregations, the need becomes more obvious. In particular, inferencing is essential if we use some mechanism for sharing categorization schemes between peers. A related topic, thesaurus linking, is being discussed under the *SWAD-E* umbrella [☞SWADE-THESAURUS]. Finally, the webness aspect is only partially captured by this demonstrator. Certainly, the enriched RSS feeds are a prerequisite. However the full benefit of the semantic web approach will only be implemented once aggregators and community services allow variously provenanced metadata to be aggregated, linked and searched.

It is important to note that the demonstrator is only a prototype. Therefore some of the semantic web values are evident from its current implementation, others form part of stories that one can tell around the demonstrator and may be implemented in future instantiations. For example, the use of ontology linking, which forms part of the vision and is enabled by the use of *TIF*,is not yet implemented. But the utility of such an extension is quite easy to extrapolate from the demonstrator, and is also being separately pursued by related *SWAD-E* activities [☞SWADE-THESAURUS, ☞SWADE-SEMPORTALS].

4 Outreach and community feedback

This is probably the most successful aspect of this project. Not only is the demonstrator visited by hundreds of people each month, it has provided a good basis for talks with individuals, academia, industry and the press. We have had useful conversations with people from UK, France, Italy, Canada, Germany, the United States and Mexico. Early in the project, we presented the semantic blogging vision [☞SEMBLOG-VISION] at Blogtalk [☞BLOGTALK]. This generated a lot of positive interest from people like David Weinburger [☞WEINBURGER-SEMBLOG] as well as from the press (the Guardian). We have also had ongoing talks with industry over the course of the project. Innogy [☞INNOGY] is an example of a company looking at the idea of semantic blogging in the context of their internal knowledge management. We (Hewlett-Packard) are also evaluating various possibilities in this context. Reuters [☞REUTERS] are interested in the semantic blogging idea to encourage a community around their articles. Of course, one of the challenges for semantic blogging is to discern whether two people are talking about the same paper (or article). One way is to agree on some definitive URI. Citeseer URLs might work for (some) papers, In the same way, Reuters identifiers might work for (some) news articles. Another encouraging development is that two UK based educational institutes (Warwick [☞WARWICK] and Ravensbourne [☞RAVENSBOURNE]) are gearing up to deploy semantic blogging technology on site. Finally, both New Scientist [☞NEWSCIENTIST] and Communications of the ACM [☞CACM] have (or plan to) run articles on semantic blogging.

We choose bibliographic metadata as a test domain for semantic blogging. As we have noted, that this is merely an example - we expect the idea to generalize successfully to any area which requires the decentralized sharing of information snippets. However, the domain choice is a good one. Early in the project, we validated our choice with a short user study [☞SWADE-USER-STUDY]. We note that there are standards for encoding bibliographic metadata in *XML* [☞MODS] and *RDF* [☞OCLC-DC-RDF, ☞BIBTEX2RDF]. Finally, we are starting to explore relationships with the open source bibliographic metadata community [☞DARCUS, ☞OCLC].

Lessons Learnt

The semantic blogging demonstrator has proved to be an excellent base for outreach. There have been positive reactions to the demonstrator too. In the future, it would be nice to push a bit harder on the inferencing and community aspects of semantic blogging. These are necessarily difficult to explore in a short prototype like this but it is possible that the demonstrator provides a base for us (or others) to build on in the future. Also, some of the themes left unexplored here are tackled in our other demonstrator. In summary, semantic blogging is a good way for people to see the benefit of using *RDF* - and to weigh up the potential cost.

5 Future Work

There are two key areas that we would like to explore with this demonstrator. Firstly, the use of an aggregator to provide a view over a community's semantic blogs. This would enable us to explore some of the added value functionality that we have been discussing above. Secondly, the use of ontology linking to allow the decentralized creation and merging of different peers' conceptual models (eg categorisation scheme). We have concrete plans for the former, in that we plan to build an internal bibliographic semantic blog that will enable us to test our assumptions about the day to day utility of such an approach. Some of the ideas discussed here will also be explored in a different context, the semantic community portal, which forms the second of Hewlett Packard's *SWAD-E* open demonstrators [☞SWADE-SEMPORTALS]. Finally, as mentioned above there are a number of groups that have expressed interest in using semantic blogging technology and we look forward to working with these groups as semantic blogging technology evolves.

6 Conclusions

We conclude that semantic blogging is a useful metaphor, both to illustrate semantic web values and to provide a hook for a wider community to interact with the semantic web. We believe that our semantic blogging prototype is of interest for this reasons. In this report, we have provided an overview of its architecture and evaluated our design choices, for the benefit of anyone building similar systems. In particular, we emphasise the dynamic nature of semantic blogging, the risk of over generalizing solutions (especially prototypes), the flexibility of *RDF* for configuration, and the importance of 'interesting' metadata. We have also evaluated semantic blogging as a semantic web illustration, and shown that while the prototype goes some way to demonstrating the principles, the wider semantic blogging picture (and future scenarios) give more complete coverage. We have also recorded our positive interactions with the wider community, and shown that bibliography management is an excellent choice of domain for semantic web explorations.

In ☞Section 2, we set out a series of objectives for the demonstrator:

- Provide a demonstrator for semantic web, easily understood by current developers.
- Provide a war story for application of semantic web techniques.
- Provide the groundwork for a useful, deployable tool.
- Assist outreach and publicity.

We have at least partially achieved the first two aims, building a successful prototype which demonstrates semantic web values and which has provided us with a large body of useful development experience. We expect the demonstrator to mature into a useful tool, fulfilling another aim. With regard to the final objective, we have exceeded out expectations, attracting unsolicited interest from industry, academia and the press. It seems semantic blogging is a technology whose time has come. We look forward to the semantic blogging theme maturing in many, perhaps even unexpected, ways in the future.

A Design documentation

This appendix contains a precis of the design documents produced during the course of the project. We have summarized and amended these documents to more accurately reflect the current state of the prototype. However, important changes from earlier plans are noted and commented on. This is another useful lesson learnt - how perspectives changed over the course of the project as we accumulated experience. The information presented here is intended to provide a high level design overview, rather than exhaustively covering every detail of the current implementation. In addition, certain aspects of the design may become outdated as the demonstrator continues to evolve beyond the project. For a more detailed and up to date picture, the reader is directed to [☞SEMBLOG-JAVADOC].

A.1 Vision - Here we provide the details behind the vision presented in [☞SEMBLOG-VISION]. Note that some of the terminology has changed and that not all aspects of the vision are currently implemented. A more up to date, high level view of the design can be found in [☞SEMBLOG-DESIGN].

We started from the observation that blogging as it stands offers compelling functionality for the end user. Of particular interest are these values:

- **Easy Publishing**; it is very easy (and quick) to create, manipulate and propagate web content.

- **Aggregation mechanisms**; Web content from different blogs can be combined in arbitrary ways for convenient access.
- **Community formation**; Aggregation, blogrolling, commenting and trackback mechanisms provide for a simple, natural way to encourage communities.

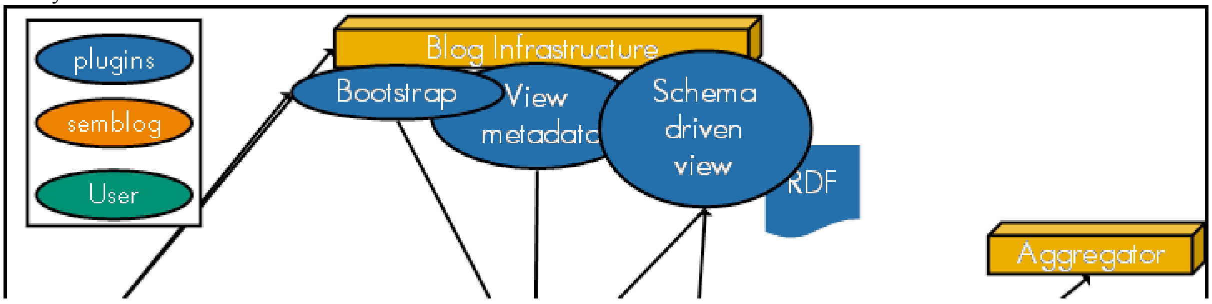
We wanted the semantic blogging demonstrator to build on these capabilities and to make blogging even more compelling. In our original plan, we identified four high level use cases:

- **Assisted Import and Markup**. A user can import useful (eg bibliographic) data to a blog. This data may come from an external source, or it may come from somebody else's blog. The imported data may have metadata already available for it, or this metadata can be inferred automatically at import time. In addition, the user may wish to enrich the content by adding simple metadata or performing natural actions (like filing the item in a topic hierarchy) .
- **Customise View**. Rather than viewing plain text, it might be useful to incorporate the metadata into the standard view. For example, viewing each blog item as a structured record showing author, title and topic. Summary tables would be another option, leading to...
- **Semantic Navigation** How to find the item of interest? The standard calendar view of blog entries is good for finding the most recent entry. Most blogging platforms nowadays also support category-based browsing. Other paths can easily be provided using metadata-assisted mechanisms. For example, a view of bibliographic items sorted by author or by rating. These views could also be filtered, leading to...
- **Community Query**. The idea here is to find items of interest using a metadata based query. It is easy to see how this could apply to your own blog (and the results semantically navigated and viewed as explained above). But we can take the query idea further and ask metadata based queries to a blogging community. For example, 'who else has blogged this paper' or 'who is currently blogging on this topic'. Note that the queries require some kind of shared conceptual structure: the first requires a unique identifier for the paper, the second requires a common understanding of 'this topic'. The extensions mentioned in the requirements document (semantic linking, name by property, disparate ontologies) will not be considered in this document.

Having outlined the approaches, we then outlined some more detailed use cases.

- **Import Options** The user wants to blog something. This can be achieved in one of two ways. Firstly, an import utility that takes some set of data (eg bibliographic file) and produces a set of semblog entries. Secondly, a 'SemBlogIt!' utility that enables the user to blog an item of interest. In both cases, the blog text is transferred but in the first case, metadata is also generated for the item. In the second, any available metadata is obtained from the blog item of interest (which may after all be on another semantic blog) and some may also be generated.
- **Assisted Markup** The user clicks on an entry and is provided with a dialog to update the metadata for that entry. Optionally, the user can customize the appearance of the metadata edit dialog (as MovableType does for its regular entry dialog).
- **Customize View** The user is presented with a form allowing the choice of fields to show (or hide) for each blog item (record). Individual or table format can be selected, as can field(s) to order the records by. On completing the form, she is shown the items presented in the metadata driven view as required. Each item is provided with a permalink that links back to the original blog.
- **Semantic Navigation** Much of the needs of semantic navigation can be met by customize view (above) or query (below). There is another utility, 'More Like This' which is useful. Here a user can click on the metadata field of any item (in a metadata driven view) and get more items with matching metadata fields. In effect, this is a very simple canned query. Another possibility is "More entries on this item". Both options should be back-ended onto the query mechanism, that is, there is a way to perform the query on the whole community.
- **Query** The user is presented with a form which can be populated with metadata fields (drop down dialogs) and metadata values against which to filter (free text or more complex, eg drop down list or tree selection). This query can be local or can be a community query. Optionally, the user can select from a number of canned queries. Some simple extensions can be provided ("generalise this query") as well as a more extended query capability, and the ability to record query preferences (and further canned queries).

These strands can be pulled into one framework. The following picture explains it. The bottom circles (green) represent use cases (ie actions initiated by the user). The middle circles (red) are semblog components, which are in principle transferable from one blogging platform to another. The top circles (blue) are blog-platform specific modules which need to be provided in order to obtain the necessary flexibility.



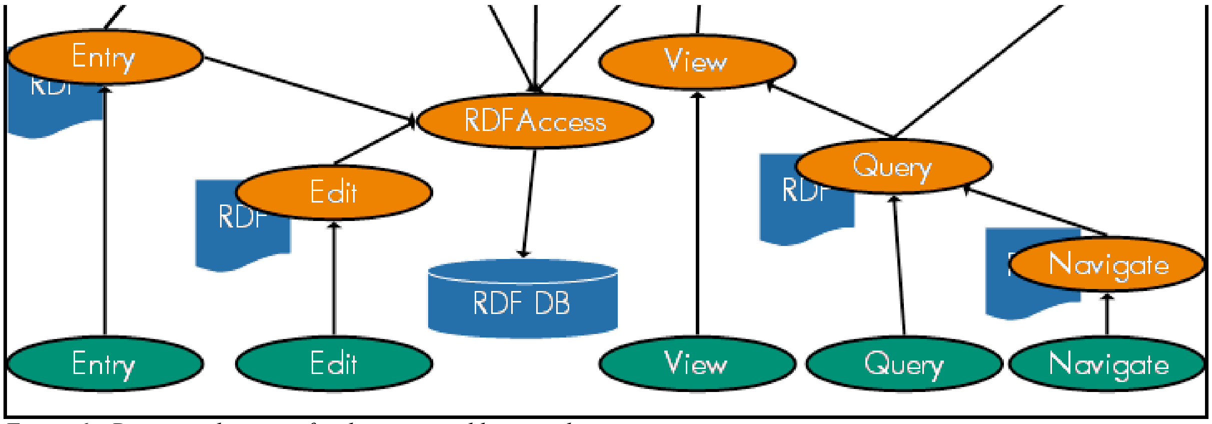


Figure 6 - Design architecture for the semantic blogging demonstrator

This diagram shows a number of user actions (in green) which correspond to the use cases covered above. The 'semblog' components (red) are envisaged to be dealt with by a servlet, with configuration parameters (blog name/URL, ontologies, rdf datastore format and so on) provided at startup. Dynamic parameters (name/password) can be passed to the servlet at run time. The semblog components are invoked by user actions. Let's deal with them in turn.

- ***RDF* Access**. The only semblog component that is NOT called directly by the user. Provides access to the *RDF* database for a local blog. This is expected (in the first instance) to be a single file with all *RDF* data for all blog items.
- **Entry**. The SemBlogIt! User action invokes this component. This is quite similar to the existing BlogIt! utility provided by many popular blogging platforms. Like BlogIT! the URL, title, & entry are extracted from the selected web page, and the user adds a category and further entry text. The blog entry is written to the blog using *XML*-RPC bridge. In addition, SemBlogIT! creates metadata (obtained from the user at point of entry, as well as from the Bootstrap *RDF* module). This metadata is passed to the *RDF* Access component to add to the *RDF* database. In addition, this component will detect and transfer any available metadata from blogged items (which may, in principle, come from another semantic blog)
- **Edit**. The Edit component consults an *RDF* preference file to build an edit metadata dialog. The *RDF* Access module is used to retrieve the necessary metadata to populate the dialog. New and changed metadata is written back to the *RDF* store using the *RDF* Access mechanism.
- **View**. This component provides a semantic view based on an *RDF* based preferences file. The basic options are table view vs entry view, which attributes to show and in which order. The schema driven view (a blog specific component) is then invoked to show an appropriate view of the blog.
- **Navigation**. This module deals with a navigation via hierarchical facets. The navigation required is transformed into a suitable query, which can be handled by the query component.
- **Query** (RDQL, QBE) interrogations of the *RDF* data are handled by this component (currently, only RDQL is supported). Query preferences are used to provide the metadata fields that can be used for building up queries, and/or to provide canned queries. So the component builds a dialog which will have metadata fields, required values, and probably a 'Go' button and 'CommunityQuery' checkbox. Tied into this component is RSS generation (ie RSS feeds should query this database to get the metadata for each blog item rather than generating it themselves, separately from and possibly in conflict with the *RDF* store)
- **Other**. There are other components still on the extensions list. For example, ideally we should have an import mechanism, which would handle import of new data using (for example) a file and a parser utility.
  Also on the list is a 'more like this' mechanism. Here, the metadata is used to seed a simple query (find me items where field F has value V).
  Finally, there is a notion of *community queries*, which are routed to the aggregator blog. The aggregator simply performs the query over its own data and displays the result.

In addition to these components, there are some blog-specific elements (shown in blue). For example, a **Bootstrap** process, which takes an entry and infers metadata for it. Much of the data it needs is in blog specific format so this component has to be a 'plugin' to a blogging platform. It invokes the semblog *RDF* Access component to add the metadata to the database. This includes scraping for metadata, eg hyperlink extraction and representation as xlinks. Also a **View Metadata** plugin that enables a user to view the metadata associated with an entry (or set of entries). And a **Schema Driven View** which provides an alternate view of a blog. The initial options would be summary table views, and structured record views.

Important Changes

In reviewing this document, we can see several differences between our previous perspectives and our current view. Firstly, and least interestingly, there were some elements that we did not have time to implement. These include a "more like this" navigation option, an aggregator and a community query tool. Secondly, we did not implement the semantic blogging functionality as a single servlet, but rather as a series of servlets. This was a more flexible approach. Thirdly, our terminology has changed from "customise view, semantic navigate, community query " to "semantic view, navigate and query", which makes it easier to explain but perhaps lays one open to the over use of the term "semantic" (we have addressed that issue in ☞section 2).

All of these are minor differences however, and it is encouraging to see how much of the original framework is still in place. Certainly we can see how to build further aspects of our vision on what we already have. For example, an aggregator blog (based on Joseki as we have said above) could be implemented. The aggregator would simply perform a query on its own, aggregate data. The results would be shown in a 'metadata only' view. Some design issues remain however. For example, it should be clear that this 'aggregate' view is not the local view of the user's blog. Moreover, the view is slightly specialised in that it shows the provenance of each attribute. Links could be provided back to the blogs of the included items. Other issues include choice of similarity measures (for 'more like this' functionality) and generalization options (for query expansion). Some of these design issues are tackled in our semantic portals project [☞SWADE-SEMPORTALS].

A.2 *RDF*Access - The *RDF*Access module is a semantic blogging component that controls access to the metadata store. An abstraction API allows common access to memory, file, database or other implementations. The exact nature of the store is driven by a configuration file.

*RDF*Access is conceptually simple. It is just an access point to the *RDF* store. It is called by semblog (and by blog plugin) components, and is not directly invoked by the user. It provides the following access points:

```
addMetadata(Model m)
```
Add supplied metadata to *RDF* store.
```
Model getMetadata(String uri)
```
Get all the metadata related to a particular blog item.
```
updateMetadata(String uri, Model add, Model remove)
```
Update metadata in the store, relating to a particular blog entry.
```
Model performQuery(int method, String query)
```
Perform a query over the metadata store.
```
removeItem(String uri)
```
Remove a blog item's metadata.

*RDF*Access is driven from a configuration file which contains a set of properties. The exact properties required depend on the *RDF*Access component. For example, a file store might have the following settings:

```
   # RDF Access properties file
# 1= memory, 2=file, 3=db
store-type=2
filename=/absolute/path/to/rdfAccess.n3
lock-filename=/absolute/path/to/rdfAccess.lock
language=N3
retries=4
waitTime=500
```

Important changes

The *RDF*Access was the first component to be implemented. In general, the design served us well. We did implement further specializations however. For example, we found a need for utility functions such as `getModel` and `setModel`, which act on the entire metadata store. Also, `performRDQLQuery` allows a specialized (RDQL) query. This is useful if, for example, you want to specify a binding variable and a return a (String) array of all bindings to that variable. Such an approach is invaluable for semantic view configuration, but it does of course tie the caller in to a particular implementation of query. Finally, the functions `getConfiguration` and `setConfiguration` were used to provide access to the *RDF*Access configuration properties.

Elements like database access and caching are not tackled in the current prototype (although concurrent access is addressed). Such issues remain for future iterations. The specialized RDQL query operation is perhaps the most weakly defensible design choice, but it arises due to the configuration behaviour of semantic view/query, which has been discussed elsewhere.

A.3 *RDF* view and edit - These two modules allow a user to easily view or edit the metadata behind a blog entry. The required uses cases are as follows:

- **View metadata** The user presses a 'view metadata' button on the blog entry display. A view of the metadata is presented. N3 and *XML*/ *RDF* versions (different buttons) are available.
- **Edit blog entry metadata** The user presses an 'add or change metadata' button on the blog entry display. A metadata edit panel is provided for the user to enter metadata, which is added to the blog item. Internally, the system builds the dialog by consulting the item's metadata, and a set of (*RDF* encoded) configuration options. A number of things are worth noting here. Firstly, each metadata field may be free text (of varying sizes) or may be a different modality. Drop down lists and hierarchical selection widgets are two possibilities. Secondly, there is a hook for validation of the entries, though this is not yet implemented. Thirdly, this metadata edit dialog is not hard coded. Indeed, the user may change its format (in conjunction with a supplied schema) as described below.

For metadata view, our original design was based on static *RDF* files, one per blog entry (see below). We rejected this option for reasons discussed ☞elsewhere, and instead built the *RDF* dynamically by consulting the blog metadata store. Rendering of this metadata is a trivial task in Jena.

The metadata edit dialog is controlled by an *RDF* configuration file. A suitable *RDF* schema is required for a simple view parameters, eg visible, maxSize, literal, resource, fromFile etc. A first proposed example is shown below:

```
   Bib:title
 sb:isVisible Yes;
 sb:maxSize 100.
Bib:topic
 sb:isVisible Yes;
 sb:getValuesFromFile Yes;
 sb:useFile topics.rdf.
Bib:keywords
 sb:isVisible No;
```

This shows a set of options for the 'title', 'topic' and 'keywords' fields. The metadata edit dialog box will be built according to these settings. A customise dialog box would allow a user to change these settings. Thus, the current configuration of the metadata dialog box could be persisted as a separate, per user, *RDF* file (or a single file with different user preferences modelled appropriately).

However, we actually used a different approach for modelling user preferences. An example is shown below:

```
   @prefix semview: <http://jena.hpl.hp.com/semview#> .
@prefix semblog: <http://jena.hpl.hp.com/semblog#> .
@prefix myPrefs: <http://jena.hpl.hp.com/stecay#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix blog: <http://jena.hpl.hp.com/blog#> .
@prefix bib: <http://jena.hpl.hp.com/bib#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix rss: <http://purl.org/rss/1.0/> .

blog:myBlog
   a blog:Blog ;

   blog:hasPreference [ a semview:BlogView ; semview:viewName "table" ;
semview:formatPreference "table" ; semview:filterPreference "optin" ] ;
   blog:hasPreference [ a semview:PropertyView ; semview:viewName "table" ;
semview:displayPreference "show" ; semview:controls dc:creator ] ;
   blog:hasPreference [ a semview:PropertyView ; semview:viewName "table" ;
semview:displayPreference "show" ; semview:controls rss:title ] ;
   blog:hasPreference [ a semview:PropertyView ; semview:viewName "table" ;
semview:displayPreference "show" ; semview:controls dc:date ] ;
   blog:hasPreference [ a semview:BlogView ; semview:viewName "record";
semview:formatPreference "record" ; semview:filterPreference "optout" ] ;
   blog:hasPreference [ a semview:PropertyView ; semview:viewName "record" ;
semview:displayPreference "hide" ; semview:controls dc:identifier ] ;
   blog:hasPreference [ a semview:PropertyView ; semview:viewName "record" ;
semview:displayPreference "hide" ; semview:controls semblog:contains ] ;
   blog:hasPreference [ a semview:PropertyView ; semview:viewName "record" ;
semview:displayPreference "hide" ; semview:controls bib:abstract ] .
```

On the one hand, this approach is weaker in that we do not specify attributes such as size of *UI* field, and controlled vocabularies (as in the topic field in the first example) are not supported. Such extensions could of course be added quite easily. On the other hand, we adopted an opt-in and opt-out style (this has been discussed already) and provided a way of specifying different user preferences in the same file.

Given that we have an *RDF* file specifying, at a high level, the metadata edit dialog settings, we need to use these settings to generate a dialog box. The intention is to provide as simple a schema as possible, not to do fine grained *UI* control. We use a servlet, built over jena, to parse the *RDF* and build the dialog accordingly. The implementation of all this is Java, partly for tight integration with Jena, and partly to enhance deployability of this solution to other blog installations, and potentially to other platforms altogether. Hence, we package this utility as a servlet.

During the design process, we discussed a number of possible extensions to this core functionality. These extensions include:

- **Customize metadata entry.** In this scenario, the user can customize the 'edit metadata' (EM) dialog. This customisation is performed by selecting the properties with which to populate the EM dialog. This selection can be done by ticking a series of check boxes (the properties that already exist on the EM dialog are pre-checked, they can of course be unselected). The customisation process, just like the customisation of a MovableType entry form, is user specific. Different users can choose customisations that make sense to them, the results being saved in the user's configuration file.
  We have not yet said anything about how the customisation dialog itself is built. Clearly, the ticks in the boxes are governed by the user's current configuration settings. But what about the other fields? For example, how does semblog know that 'keywords' & 'media type' are allowable (but currently unselected) fields? How can the user control this? In fact, the customisation dialog can be populated from an underlying schema (eg bibliography schema) which is provided by the user. This is considered next.
- **Schema components** A user may introduce extra schema components to the blog (eg a bibliography ontology). All schema components are supplied in the form of *RDF* (hence *RDF*S, DAML or OWL) files which must be notified to the system by including their names in a configuration file. This allows these schemata to specify the contents of the customise metadata dialog box. The files are expected to include: Allowable metadata fields for blog entries; Constraints for these metadata fields (eg datatypes, cardinality); Preferred display options for these metadata fields (user visible, literal (and expected size string), resource, default file where instances are defined).
  These presentation elements can also be included in the user supplied schema to specify preferred settings. So, for example, a user might provide a bibliography ontology, specifying which of

these elements would be user editable by default, what size they should be, and the preferred file to read permissible values from. These preferences may of course be modified by the user at run time.

- **Multiple metadata edit forms**. A single user might want different edit forms for bibliography items, news articles and so on. These edit forms would be accessed by making the 'add/edit metadata' button a drop down list. Further, the 'customise this form' option would give the user to save the metadata edit form to a user supplied location.
- **Autopopulate metadata from existing item**. I am not talking here about import filters, such as those which would parse a CiteSeer [☞CITESEER] page for relevant metadata (these would be covered by a separate component). Rather, this is an extension to the import functionality, an extension which copies across the blog item metadata file when a user clicks on the 'SemBlogIT!' bookmarklet.
- **Hierarchical selection widget**. The above functionality implements topics as a flat list rather than a hierarchy. It would be rather useful to allow a user to select a topic from a tree browser rather than a flat list.

Important changes
In the initial design, the 'edit metadata' was built over an 'edit this entry' form. That was before we switched from *MT* to blojsom. In the latter system, there is no such form, and editing of the entry is handled in a different manner to editing of the metadata (mechanisms for the former include bledit [☞BLEDIT] and w.bloggar[☞WBLOGGAR]). The advantage of this is that the difference between editing the blog entry and its metadata is made clear, the disadvantage is that the user is forced to learn two different mechanisms for editing. A radical way to solve this would be to abolish the difference between metadata and data, a distinction that could be done for small, textual items such as blog entries.

One crucial difference is that this design was drafted when we were experimenting with the idea of one *RDF* file per blog entry. As we have discussed elsewhere, this was not a good option, because it made querying over the entire blog metadata problematic. We build our *RDF* dynamically at request time by consulting the metadata store. Having said that, a static file does have some advantages, notably for caching and speed of access.

A.4 Semantic Import - The import module is a semantic blogging component that allows a user to enter new blog entries with associated metadata. Users can blog a particular web page, entering a bibliographic item by cutting and pasting a BibTeX entry, filling in a form, or adding arbitrary metadata in N3 or *RDF*/*XML* form.
A web form (SemBlogIT) provides the basic entry point (other import components have been dubbed BibImport, FreeImport and CategoryChooser). This form can be created automatically by clicking on a 'SemBlogIT!' bookmark. The form will then be pre-populated with a title, hyperlink and highlighted text taken from the current web page. The user then adds more details by

1. Changing or adding to the title and body text (SemBlogIt)
2. Persisting a user name and password in a cookie or session (SemBlogIt)
3. Enter bibliographic metadata by filling in a form (BibImport)
4. Enter bibliographic metadata by cutting and pasting a bibtex entry (BibImport)
5. Entry arbitrary metadata (FreeImport) in N3 or *RDF*/*XML* (the format is selected using a combo box).

On selecting 'OK', a simple ontology-backed algorithm (CategoryChooser) is invoked to present the user with a list of options for the category for this item. All the metadata gleaned from the import components is augmented by a metadata bootstrap plugin. This is responsible for generating a basic *RDF* profile for a blog entry (information like created date etc).The resulting metadata is passed to *RDF*Access, which adds it to the store.

Let us now examine the import components in more detail

SemBlogIT! is invoked by navigating to (or invoking javascript that creates) the SemBlogIT form (implemented as, for example, a *JSP*). The SemBlogIT component is responsible for driving the user interaction. The basic dialog is intended to be as simple as possible, with title, author, password and entry fields only. The BibImport and FreeImport forms are presented as separate (popup) forms driven from the main SemBlogIT form.

The BibImport component parses a bibtex file, or accepts form based input, transforming this to *RDF*. It does this by constructing a dialog form with space for common bibliographic fields (author, title, journal, issue, date). There is also space to cut and paste a bibtex entry for automatic parsing. Once entered, BibImport creates the appropriate metadata and passes it back to SemBlogIT for adding to the blog entry's metadata. The bibliographic item will be linked to the entry via a `semblog:contains` link. The bibliographic import form is schema-driven (and hence customisable). The schema reuses that from the edit component (see ☞above). It would be possible to extend this customisation to the other forms. The preferences could be user specific and themselves controllable through the *UI*.

FreeImport is similar to BibImport, except that it is not tailored for bibliographic items. So one needs to enter a URI for an item together with the raw metadata (N3 or *RDF*/*XML*) about that item. The item will again be linked to the entry via a `semblog:contains` link.

The user input from SemBlogIT is passed to CategoryChooser. CategoryChooser consults a *TIF* ontology for the available topics. The ontology conforms to the *TIF* schema [☞TIF].An excerpt from an example ontology is presented below.

```
   # Base:
@prefix : <#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix tif: <http://www.limber.rl.ac.uk/External/thesaurus-iso.rdf#> .
```

```
@prefix blojsom-hp: <http://jena.hpl.hp.com/tif/blojsom-hp#> .

# Root node
blojsom-hp:cRoot
   a tif:TopConcept ;
   tif:NarrowerConcept blojsom-hp:cblogging ;
   tif:PreferredTerm blojsom-hp:tRoot .

blojsom-hp:tRoot
   a tif:Term ;
   rdf:value "/" .

# Category blogging
blojsom-hp:cblogging
   a tif:Concept ;
   tif:NarrowerConcept blojsom-hp:cklogging ;
   tif:NarrowerConcept blojsom-hp:csemblog ;
   tif:PreferredTerm blojsom-hp:tblogging .

blojsom-hp:tblogging
   a tif:Term ;
   rdf:value "blogging" .

# Category klogging
blojsom-hp:cklogging
   a tif:Concept ;
   tif:PreferredTerm blojsom-hp:tklogging .

blojsom-hp:tklogging
   a tif:Term ;
   rdf:value "klogging" .

# Category semblog
blojsom-hp:csemblog
   a tif:Concept ;
   tif:PreferredTerm blojsom-hp:tsemblog .

blojsom-hp:tsemblog
   a tif:Term ;
   rdf:value "semblog" .
```

CategoryChooser consults this topic ontology and suggests to the user a (set of) topic(s) for the blog item. To do this, CategoryChooser uses a simple algorithm:

choose(word)

```
   Stem all words in blog entry text
   choices := empty list

   for each stemmed word
      for each concept in TIF ontology
         if match (word, concept) then choices <- choices + word
      next concept
   next word

   return choices

end choose
```

match(word, concept)

```
   if (word equals preferredTerm(concept))
      return true
   else if (word equals indicatorTerm(concept))
      return true
   else if (word equals URI(concept))
      return true

   return false

end match
```

This is a very simple algorithm. More could be done to distinguish matches (eg give words matching preferred terms higher weightings than those matching indicator terms). But it is a useful starting point for ontology guided matching.

This ontology could be created by hand, but in our case we used the blog category hierarchy. CategoryChooser thus consults the category hierarchy (available using the MetaWeblog [☞METAWEBLOG] XML-RPC call `getCategories`) and constructs a TIF ontology from it. Since blojsom categories are implemented as a directory structure (into which the blog entries are placed) the algorithm can also be run offline, and we wrote a perl script to accomplish the same task. The basic algorithm is as follows:

```
   print TIF headers
create Root category
FOR each category
```

```
   create TIF concept for category
   create TIF preferred term for category
   find supercategory
   create TIF NarrowerTerm relationship between supercategory and this category
NEXT category
```

The results of CategoryChooser are presented in a further form. On dismissing that, the bootstrap plugin is invoked.

The bootstrap plugin is responsible for creating core metadata using the available information from the blog entry. Typical metadata generated includes: title, link (entry location), description, date, creator, subject (Dublin core), concept (*TIF* ontology), content (escaped), seeAlso (link to machine readable *RDF* location). Others are possible; eg (copy)rights, (MIME) format, updatePeriod, hyperlinks etc.

The resulting metadata, from all the import components plus the bootstrap plugin, is passed to *RDF*Access, which adds it to the store.

Important Changes

There are no major changes to the import component. However, there are some desirable extensions. For example, a useful utility would be a one that could, for example, easily scrape bibtex records from a web page such as Citeseer [☞CITESEER]. Also, it would be nice to allow the user to choose the order of the metadata fields in the bibtex dialog. In fact, the concept of personalisation is easy to add to this framework, by making the EditViews have viewNames. The idea of a *UI* to customise dialogs is a 'nice to have'.

Other enhancements, more specific to semantic import, include:

- Improving the CategoryChooser, in particular allowing weighting between the different suggestions according to their level of semantic match.
- Persisting a user name and password in a cookie or session
- Using an ontology to control the format of the SemBlogIT! form just as the bibtex entry form is controlled. The need for this is less pressing though as there are not so many degrees of freedom.

A.5 Semantic Export - The export module is a very simple semantic blogging component. Essentially it just generates an enriched RSS feed. Typically (both in *MT* and blojsom, for example), RSS feeds are created by a hard coded template. That is, the metadata properties are fixed, it is just their values that change between blog entries. In the semantic blogging demonstrator, we use a utility that consults the metadata store before creating our RSS feed. Specifically, the algorithm works like this:

```
   create empty model
add bootstrap 'channel' metadata to model
for each requested blog entry
   get all metadata for this blog entry from RDFAccess
   add metadata to model
   add channel-entry link
next blog entry
serialize model
```

Note that the metadata for a blog entry will, of course, also include metadata about any contained items (such as bibliographic metadata). Repeated metadata (for example two blog entries referencing the same bibliographic item,) is handled naturally by Jena.

Important Changes

There are no important changes to what is, after all, a fairly simple component. However, we do note that the arbitrary serialization of *RDF* could potentially cause problems for non *RDF*-aware RSS aggregators; this issue has been discussed ☞above.

A.6 Semantic View - The view module is a semantic blogging component that allows a user to view their blog in a schema driven way. Initially 'record card' and 'summary table' views are supported. The format of the blog is described using an *RDF* 'preferences' file which conforms to a simple presentation ontology. A possible extension would be a *UI* that allows the user to change the preferences file. However, a nearer term workaround is to simply edit the preferences file by hand as required. These capabilities are supported in this current iteration:

- **Record card view**. Each blog item is represented as a 'record card'. That is, certain metadata fields will be displayed (eg 'author' and 'title') with their value next to them. In addition, the metadata of any contained object (eg author of the blogged article) may also be displayed.
- **Summary table view**. A summary table is provided, with the columns being selected metadata fields, the rows being blog items and the cells being the values for those fields.

Each item is also provided with a permalink that links back to the original blog. In addition, we need a special purpose ontology to control the format of the preferences file.

The semblog 'view' component is invoked from a blog plugin (a *JSP*). The semblog component consults an *RDF* preferences file, which provides information about which properties to display. The plugin is then responsible for rendering the blog. This division of labour aids portability between blogging platforms.

The user invokes the semantic view functionality by specifying (or clicking on a link that specifies) a special URL parameter

```
   ?flavor=semview
```

The plugin is responsible for calling out as necessary to the semblog component and formatting the blog page accordingly. The plugin (in blojsom's case) is actually be implemented by a *JSP*. In *MT*'s case this would be a template. The pseudocode looks like this (bold items are the semblog functions described below):

```
   formatPref = getFormatPref()
```

```
if (formatPref is table)

   /* write out table heading row */
   props = getPropertiesToShow()
   for each p in props
      print p
   next p

   /* Now write out one row for each item */
   for each blog item bi
      m = getMetadata(bi)
      for each p in props
         print getPropertyValue(bi,p,m)
      next v
   next bi

else if (formatPref is record card)

   for each blog item bi
      m = getMetadata(bi)
      properties = getProperties(bi, m)
      for each p in properties
         pname = getLabel(p)
         pvalue = getPropertyValue(bi, p, m)
         print pname : pvalue
      next p

      /* and now the contained items - eg bibliographic records */
      containedItems = getContainedItems(bi, m)
      for each ci in containedItems
         print "contains " + getLabel(ci, m)
         properties = getProperties(ci, m)
         for each p in properties
            pname = getPropertyLabel(p)
            pvalue = getPropertyValue(ci, p, m)
            print pname : pvalue
         next p
      next ci

   next bi

end if
```

The semblog component consults the preferences file and retrieves relevant metadata using the *RDF*Access component.The first time this is called , the component reads the preferences file and creates a list of 'properties to show' and 'properties to hide'. We also set up a format preference (record card or table) and a filter preference (opt-in or opt-out). Then you can call the following functions:

`Model getMetadata(String itemURI)`
Gets all the metadata for a given item (including metadata about contained items)

`String [] getContainedItems(String rootURI, Model model)`
Gets a list of URIs for items 'contained' by the rootURI in this model

`String getLabel(String itemURI, Model model)`
Gets a human readable label for this particular URI, using data available in the model

`String getLabel(String itemURI)`
Convenience method - gets a human readable label for this particular URI

`String [] getFormatPref(String viewName)`
Gets the format preference (table or recordCard) specified by the preferences file for the specified view. Note that in the table case, the filter preference is implicitly opt-in.

`String [] getFilterPref(String viewName)`
Gets the filter preference ('opt-in' or 'opt-out') specified by the preferences file for the specified view. The table summary format preference assumes (and imposes) opt-in.

`String [] getPropertiesToShow()`
Gets the 'opt-in' properties specified by the preferences file

`String [] getPropertiesToHide()`
Gets the 'opt-out' properties specified by the preferences file

`String [] getProperties(String rootURI, Model model)`
Gets all the properties specifically associated with this resource in this model. Note that the properties will be filtered either on an 'opt-in' or 'opt-out' basis depending on the settings in the preferences file

Gets a single value specifically associated with this resource and this property in this model. In the case of duplicate properties, an arbitrary choice is made.

`String [] getPropertyValues(String rootURI, String propertyURI, Model model)`
Gets all the property values specifically associated with this resource and this property in this model.

We need a special purpose ontology to define certain presentation components. There is little option here but to create our own. Certainly we advocate using as simple an ontology as possible. We expect to specify some broad preferences, which the *UI* can then intelligently use to allocate display resources as appropriate. The abbreviated schema definition is as follows:

```
   # View of the whole blog
vp:BlogView
   a rdfs:Class .

# View of a single blog item property
vp:PropertyView
   a rdfs:Class .

# Property to which a PropertyView pertains
vp:controls
   a rdf:Property;
   rdfs:domain vp: PropertyView;
   rdfs:range rdf:Property .

# display preference for a property view
# (expected values: "show" and "hide")
vp:displayPreference
   a rdf:Property;
   rdfs:domain vp: PropertyView;
   rdfs:range rdfs:Literal .

# format preference for a blog view
# (expected values: "table" and "record")
vp:formatPreference
   a rdf:Property;
   rdfs:domain vp: BlogView;
   rdfs:range rdfs:Literal .

# filter preference for a blog view
# (expected values: "optin" and "optout")
vp:filterPreference
   a rdf:Property;
   rdfs:domain vp: BlogView;
   rdfs:range rdfs:Literal .

# name for a blog view
# (use whatever identifier you like)
# Maybe we should allow this to pertain to PropertyViews too.
vp:viewName
   a rdf:Property;
   rdfs:domain vp: BlogView;
   rdfs:range rdfs:Literal .
```

An example configuration file might look like this:

```
   @prefix semview: %lt;http://jena.hpl.hp.com/semview#> .
@prefix semblog: %lt;http://jena.hpl.hp.com/semblog#> .
@prefix myPrefs: %lt;http://jena.hpl.hp.com/stecay#> .
@prefix rdf: %lt;http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix blog: %lt;http://jena.hpl.hp.com/blog#> .
@prefix bib: %lt;http://jena.hpl.hp.com/bib#> .
@prefix dc: %lt;http://purl.org/dc/elements/1.1/> .
@prefix rss: %lt;http://purl.org/rss/1.0/> .

blog:myBlog
   a blog:Blog ;

   blog:hasPreference [ a semview:BlogView ; semview:viewName "table" ;
semview:formatPreference "table" ; semview:filterPreference "optin" ] ;
   blog:hasPreference [ a semview:PropertyView ; semview:viewName "table" ;
semview:displayPreference "show" ; semview:controls dc:creator ] ;
   blog:hasPreference [ a semview:PropertyView ; semview:viewName "table" ;
semview:displayPreference "show" ; semview:controls rss:title ] ;
   blog:hasPreference [ a semview:PropertyView ; semview:viewName "table" ;
semview:displayPreference "show" ; semview:controls dc:date ] ;

   blog:hasPreference [ a semview:BlogView ; semview:viewName "record" ;
semview:formatPreference "record" ; semview:filterPreference "optout" ] ;
   blog:hasPreference [ a semview:PropertyView ; semview:viewName "record" ;
semview:displayPreference "hide" ; semview:controls dc:identifier ] ;
   blog:hasPreference [ a semview:PropertyView ; semview:viewName "record" ;
semview:displayPreference "hide" ; semview:controls rss:link ] ;
   blog:hasPreference [ a semview:PropertyView ; semview:viewName "record" ;
semview:displayPreference "hide" ; semview:controls semblog:contains ] ;
   blog:hasPreference [ a semview:PropertyView ; semview:viewName "record" ;
semview:displayPreference "hide" ; semview:controls bib:abstract ] .
```

Major changes The semantic view component has been implemented pretty much as originally conceived. Various extensions are possible. For example, it might be desirable to allow the user to change their view preferences without editing the preference file. In this case, the user would be presented with a form allowing the choice of fields to show (or hide) for each blog item (record). Individual or table format can be selected, as can filter preferences (opt in or out) and field(s) to order the records by. However, the user defined blog views allowed by the schema do go some way to addressing this need.

A.7 Semantic Navigation - The navigation module is a semantic blogging component that allows a user to navigate their blog in a schema driven way. There are two main elements to this component:

- **Tree Browsing** Here, a Java applet renders a tree representation of the hierarchical categories. The tree itself is generated from an *RDF* schema (the blog category hierarchy is represented in *TIF* [☞TIF]).
- **Facet browsing** Although this is not strictly facet navigation, the idea here is that data can be selected by choosing an intersection of features. For example, one might choose all entries written by "stecay" on a concept (or subconcept) of the semantic web.

For tree browsing, the selection is used to construct a URL of the type used by blojsom. For example, selecting the 'semantic web/ontologies' tree node on the demonstrator [☞SEMBLOG-DEMO] leads to a URL of the form
`http://jena.hpl.hp.com:3030/blojsom-devt/blog/semantic-web/ontologies/`
This is then passed to blojsom, which displays a normal blojsom page based on that category.

The facet browsing behaviour is slightly different. The features are used to create a query, which is passed to *RDF*Access to retrieve the relevant blog entries. These entries are drawn from different categories (and potentially, in the future, different blogs). The query results are displayed on a specialised 'navigation' page. This page is invoked with a URL of the form
`http://jena.hpl.hp.com:3030/blojsom-devt/blog/?queryType=navigate&`
`flavor=semnav&http%3A%2F%2Fpurl.org%2Fdc%2Felements%2F1.1%2Fcreator=stecay`

In addition, we need a special purpose ontology to control the format of the navigation preferences file. Again, we advocate using as simple an ontology as possible. We expect to specify some broad preferences, which the *UI* can intelligently use to allocate display resources as appropriate. An abbreviated preferences file is given below:

```
   # Base:
@prefix rss: <http://purl.org/rss/1.0/> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix semblog: <http://jena.hpl.hp.com/semblog#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix : <#> .

[] a semblog:navigationPreference ;
   semblog:hasMetadata
      [ a dc:creator ;
       semblog:hasValue "admin" , "jugr", "paulshab", "stecay", "wikon"
      ] ;
   semblog:hasMetadata
      [ a rss:title ;
       semblog:hasValue "TIF" , "semantic" , "RDF" , "learning" , "thesaurus"
      ] ;
   semblog:hasMetadata
      [ a semblog:hasConcept;
       semblog:hasValue "semantic-web", "ontologies" , "machine-learning"
      ] .
```

Major changes There are no major changes, though there are some things we might have done differently with hindsight. One example is the schema, which as it stands is inelegant and would benefit from being changed to a form more in keeping with ☞metadata edit. Another change would be the use of semantic query to power the hierarchical browser. This is an essential prerequisite to generalizing the browser to properties other than category.

Another, more subtle point, is that we chose a different strategy here for the results page than we did for semantic query. The URL constructed by semantic navigation is passed into the main blojsom chain. Hence, all entries on the blog are processed, and the query (encoded in the URL parameters and implemented by a blojsom plugin) is used to filter these entries. While inefficient in a sense (all blog entries for a blog are processed), the approach does allow us to make use of the generic blojsom machinery (anything in the generic plugin chain, eg calendars, search interface, trackback metadata etc). A different approach is taken for semantic query (see below).

A.8 Semantic Query - The query module is a semantic blogging component that allows a user to query the blog according to its metadata. There are 3 separate query components: **semquery_*UI***, **semquery_servlet** and **semquery_results**. A 'semantic query' link provides a query view (**semquery_*UI***). The query view can be blog based (about the blog entries) or item based (about the items of information which the blog entries describe). The view is two paned. In the left hand pane there is a set of metadata fields (schema controlled) into which the user enters a query. The query itself is performed using **semquery_servlet**. The results of the query are displayed in the right hand pane (**semquery_results**), in a format (eg record card/table) specified by the user.

semquery_*UI*

The user invokes the semantic query functionality by specifying (or clicking on a link that specifies) a special URL parameter

```
    ?flavor=semquery&queryType=entry|item&resultview=<name>
```

This will bring up the appropriate (entry or item) query view. Semquery_*UI* builds the appropriate *UI*, the exact format of which is driven by the preferences file. This preferences file provides information about the properties to be made available for query. Thus, semquery_*UI* does a number of things:

- It reads a semquery properties file (to find out, for example, the location of the preferences file, and the name of the template *JSP*s to use)
- It constructs some appropriate request parameters based on the semquery preferences for the required query type (item or entry). For example, parameters
  `<PREFIX>+<NUMBER> + <ABBREVIATION>`
  `<PREFIX>+<NUMBER> + <FULL_NAME>`
  query and result view types
- It passes the result to the *JSP* that constructs the appropriate query form (entry or item) from the parameters.

Semquery_*UI* requires the following generic piece of functionality (supplied by *RDF*Utilities) to read its preferences file and hence to construct the *UI*.

```
   String[] readConfiguration(configFile, String language, String query,
String bindingVariable)
```

This utility reads configuration preferences from an *RDF* encoded file (in the specified language), using an RDQL query and reading the binding(s) to the supplied binding variable. binding variable.

This utility gave us a fair amount of flexibility with only moderate complexity. It did tie us to using RDQL, but this tradeoff was worth it in our opinion. For sake of comparison, we present an alternative, more refined example :

```
   String [] readPreferences(String prefFile, String language, String
query, String bindingVariable, Constraint[] constraints)
```

Reads a preferences file (in specified *RDF* language) using an (RDQL) query to pull out preferences (bound to the binding variable). Each Constraint contains a binding variable and a value which must match the binding variable. Results from the query must satisfy all Constraints, otherwise they are not included. The method returns an array of all bindings to the binding variable returned from the query where the constraints are satisfied.

However, this latter approach, while giving us a little more flexibility, is in fact still quite closely bound to the format of the preferences query, so feel that the complexity/flexibility tradeoff is not worth it.

The semquery preferences is driven by this ontology:

```
   # View of the edit form
qv:QueryView
   a rdfs:Class .

# Preferred Property for a QueryView
qv:preferredProperty
   a rdf:Property;
   rdfs:domain qv:QueryView;
   rdfs:range rdf:Property .

# Name for a QueryView
qv:viewName
   a rdf:Property;
   rdfs:domain qv:QueryView;
   rdfs:range rdfs:Literal.
```

This ontology allows us to build a preferences file of the form:

```
   @prefix semquery: <http://jena.hpl.hp.com/semquery#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix blog: <http://jena.hpl.hp.com/blog#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix rss: <http://purl.org/rss/1.0/> .

blog:myBlog
   a blog:Blog ;
   blog:hasPreference [ a semquery:QueryView ; semquery:viewName "entry" ;
semquery:preferredProperty dc:creator ] ;
   blog:hasPreference [ a semquery:QueryView ; semquery:viewName "entry" ;
semquery:preferredProperty rss:title ] ;
   blog:hasPreference [ a semquery:QueryView ; semquery:viewName "entry" ;
semquery:preferredProperty dc:subject ] .
```

semquery_servlet

The results of the query form are passed to a second component, semquery_servlet, which actually performs the query. The appropriate query will be different for items and entries and the exact form of the query may be driven by the semquery properties files. Semquery_servlet takes the user input and queries the metadata store (via *RDF*Access) to retrieve metadata for an item. The algorithm is as follows:

```
FOR each request parameters starting with <PREFIX>
   Extract FULL_NAME and <value>
   Add to propertyURI's and Value's
DONE
Lookup appropriate query based on <queryType>
Construct query using propertyURI's and Value's
```

```
FOR each result
   Construct request parameter eg <RESULT>+<URI>
```

semquery_results

The query results are used to populate some request parameters which are then forwarded to the semquery_results component (which is, in the blojsom case, a *JSP* page). Semquery_results is responsible for appropriate rendering of the results. This is done using semantic view functionality, so it can provide either record card or table view according to user (run time) preference. The user preferences can be specified using the `resultview` URL parameter:

```
Choose view type based on resultView parameter
FOR each request parameters starting with <RESULT>
   Lookup metadata for this item from RDFAccess
   Display metadata according to preferences
DONE
```

Important Changes

There are no really major changes from our original design. Like semantic view, it would be nice to allow the user to choose the order of the metadata fields. We did this in ePerson [☞EPERSON] using a 1-9 priority, with higher/lower numbers being presented first, and equal priorities displayed alphabetically. We can as a workaround display the property names alphabetically, which at least gives consistency. In addition, the concept of personalisation is easy to add to this framework, because the QueryViews have viewNames (just like BlogViews). Again, the idea of a *UI* to customise query views is a 'nice to have'.

For item based query views, it would be nice to make the results show a view of the items, and to further allow a one click path to "entries about this item". In fact, one can imagine there is much one can do to improve the user interaction with semantic query. This is one direction that the semantic portals work [☞SWADE-SEMPORTALS] will take.

A.9 Ontologies - We finish this appendix by outlining a set of ontologies used in our semantic blogging demonstrator.

Bibliographic metadata

Our first requirement was for a schema that captures, in a fairly lightweight manner, the required bibliographic data we need for our blog; elements like author, title, journal, year etc. While we need this to be adequate for our needs, we must be careful not to over-engineer this component. In particular, it must be pluggable - that is, it should be easy, in least in principle, to apply our semantic blogging demonstrator to items other than bibliographic records. Some of the extant library standards, such as FRBR [☞FRBR] and MODS [☞MODS], fall into the 'over complicated' camp. However, simpler, generic schemes, like Dublin Core [☞DC] (which has author and title fields) are insufficient. In our view, BibTeX [☞BibTeX] poses a nice, intermediate position. This has all the fields we are likely to need and yet should be fairly simple to implement. In addition, a BibTeX to *RDF* translator [☞BIBTEX2RDF] is already available, with its own ontology.

There are some limitations with BibTeX that might be addressed in future work. One of the nice things about the FRBR model is that it has a notion of works themselves and instantiations of that work (for example, the Italian translation of 'Othello', or the 2nd edition of 'Java Servlets'). While BibTeX can differentiate between such items (for example, by ISBN number, or year of publication) there is no reliable way to group such related items together (Title and author will work in some, but not all, cases). This is related to the 'name by property' extension referred to in the requirements specification [☞SEMBLOG-RQMTS]. A second limitation is that BibTeX does not constrain fields. For example, there is no imposed formatting for year or page number (and although there is a formatting rule for authors, the field itself is unconstrained). It is possible that datatypes, or even controlled vocabularies, may be a useful extension to an *RDF* encoding of the BibTeX schema. However in our current version we use BibTeX without such constraints.

We also had some discussion about how to represent bibliographic items. We cam up with this model: ***a blog item can contain a number of bibliography items***, each with a set of custom bibliographic fields. We discussed, and then rejected, two-way links (which blog entries are this bibliographic item contained in?). The `contains` property is drawn from the `semblog` ontology. To avoid semantic creep, the domain of this property is left as resource, and not overconstrained to be `BlogEntry`.

Annotations

The idea here is to provide a simple way of marking up peer annotations. We did not implement this in the current demonstrator, however these design notes might be of use in the future.

Note that there are two subtly different requirements here. The first is a way to mark up annotations about blog entries - they are known as comments in the blogging world, and can be denoted as `annotate:reference` using the RSS annotate module [☞RSS-ANNOTATE]. (there is a full list of RSS modules available online [☞RSS-MODULES]). This will work quite well as a link between the blog entry and a comment on that entry. The second is a way to mark up annotations about bibliographic items themselves. Note that this is a comment about a specific item within someone's blog entry. There may be more than one of these items. In both cases, we want the annotations themselves to be citable. Therefore they should be first class entities, complete with author, title and other citable details.

The idea of having citable annotations is quite an interesting one, and led to some discussion about how best to represent this. Our current recommendation is that peer comments on the blog entry itself, and peer comments on bibliographic items, are both handled as first class, citable entities. They are of class `Annotation`, and of class `BibItem` to reflect their dual nature. We considered having 2-way links between the comment and the item it is commenting on (for example, `dcterms:isReferencedBy` & `annotea:hasAnnotation` but this is not really necessary. In Jena, it is easy to ask for example 'which annotations `annotea:annotates` this item'. The use of two distinct properties, `rssAnnotate:reference` and `annotea:annotates` is still unresolved.

We propose a combination of the Annotea schema (which treats annotations as first class entities) and the bibliography schema described above (to enrich annotations with citable details) to address these needs. Other possibilities considered included IBIS terms and ClaiMaker, though these are more relevant to semantic linking (see below). An extension to this ontology would be a mechanism (or hook) to allow, at least in principle, partial annotations (annotations about part of a bibliographic item).

Semantic Links

This section again outlines some design notes for a possible future extension

The requirement here is to link comments with the thing they are commenting on using semantic links ("this comment disagrees with this bibliographic item", "this comment agrees with the previous comment"). The essential idea here is to replace the `annotea:annotates` property in the above diagram with a more specific, semantic, property (likes `agreesWith`). At first sight, schemes like ClaiMaker [☞CLAIMAKER] seem very appropriate to this. However, ClaiMaker models semantic links between concepts. So a paper will have a set of concepts, and it is the concepts themselves that take part in the semantic network. Such a scheme, while conceptually attractive, enforces a more complicated worldview than we want to employ here. Danny Ayers' IBIS schema [☞IBIS] is essentially a simplification of ClaiMaker so the same considerations apply. A more tractable alternative is Annotea threads [☞ANNOTEA-THREADS]. This has concepts like agreesWith, disagreesWith etc. It is simple and well thought out. It also has other useful elements which will be remarked upon in other sections of this document. The main disadvantage is that it is not a perfect fit to our domain, containing perhaps irrelevant properties like followsUp, initialPost etc. In short, our recommendation is to use Annotea threads, but as a 'pluggable' ontology. That is, to allow users, at least in principle, to replace the threads ontology with another set of semantic link terms, such that the machinery will function largely unchanged. At the ontology level, this should be straightforward (it is using this information at the application layer to present useful information to the user that is a challenge!). In order to do this, we need to have a higher level, 'vanilla' ontology to which we can map Annotea threads (or other semantic linking ontology). The ontology must be user extensible.

Another possibility is a lightweight way of linking ontologies together. So, for example, we can map one term in one users ontology to a term in another user's. Note that by ontology we are essentially talking about concept hierarchies. In addition, we conflate the idea of 'topic' and 'concept' for simplicity. We considered a number of alternatives. *RDF*S/DAML/OWL provide an equivalence relationship. This might work, but has semantic implications (for example, being a mutual subclass relationship it does mean that each concept inherits the children of the other concept). We might, in some cases, want a looser connection. XFML [☞XFML] and Topic Maps [☞TOPIC-MAPS] contain a concept mapping operation that isn't semantically constrained in the specification (but does have quite a specific social meaning). This disadvantage with these solutions is that they only allow one type of connection, with operational semantics, and the format is *XML*. We also considered WordNet [☞WORDNET], which provides a great deal of richness in mapping between terms (and concepts) but is probably too complex for our needs. We also wanted a solution that would encourage users to build up mappings themselves. Note that the ontological machinery is reasonably straightforward here. Once we have defined a hierarchy and a way of mapping between schemas it is a matter of allowing the user to build links as appropriate.

Our preferred option is the Thesaurus Interchange Format (*TIF*; [☞TIF]) proposed by Rutherford laboratories. This has a number of relationships (exact equivalence, inexact equivalence, one to many etc). Although these relationships are semantically unconstrained in the specification, they (probably) have quite specific meanings in the thesaurus community. With appropriate scope notes to guide the user, the use of these terms would give us all the power we need. From a *SWAD-E* perspective, this solution also has the benefit of increasing cross-talk between project partners and workpackages.

Topic Structure

The need here is for an ontology to structure a set of topics. Note that although this structure can be arbitrarily complicated, we expect the basic structure to be a simple hierarchy. So, primarily, we need properties like parent/child, superclass/subclass, broader/narrower etc. There are a number of possibilities here. XFML and Topic Maps provide a base structure to organise topics, but are *XML* based. RSS2.0 [☞RSS20] allows links into a categorisation scheme but again it is *XML* based. *RDF*S/DAML/OWL class hierarchies provide an *RDF* mechanism, but we feel that we would like an organisation that is specifically tailored to concept hierarchies. The subclassing inference machinery may have undesired side effects for a topic hierarchy. Certain structure, like DMOZ [☞DMOZ] and WordNet [☞WORDNET], come with both mechanisms to define links between concepts, and the concepts themselves. Since we are interested in an approach where users build up topic hierarchies themselves, we rejected solutions like these, because they require 'buy in' to a predefined hierarchy.

These considerations leave two front runners. The first is Annotea thread description language [☞ANNOTEA-THREADS], which despite its name has some elements well suited to blog categories. For example, the property `partOf` relates an `Post` to a topic. The topics can be linked together in a hierarchy using a `categoryOf` property. The advantage with this scheme is that if we are already using it for semantic linking then it fits in neatly. The disadvantage is that the `partOf` property has to come from a `Post` item. Thus, bibliographic items would have to be of type `Post`. Hence, the schema is not a perfect fit to our domain. The second possibility is the thesaurus interchange format [☞TIF]. This has properties like `BroaderConcept` and `NarrowerConcept` from which we can easily build a hierarchy. The hierarchy is expected to be built of resources of type `Concept`. This would fit neatly in with the ontology mapping work (should we decide to use *TIF* for this). Note also that a topic can appear in multiple places in the hierarchy. It also has the advantage, as mentioned before, of working with another *SWAD-E* partner. It has the disadvantage that we need a property (eg `hasConcept`) to link the bibliographic items and the concepts in the thesaurus. It is also a 'moving target' in the sense that the specification is not yet stable. On balance, we advocate the use of thesaurus interchange format. There are no special design considerations for this ontology, although we might note that the semantics of `broaderTerm`/`narrowerTerm` or `categoryOf` need to be clearly explained to the user in a scope note.

A future extension would be another topic hierarchy into which we can classify our bibliographic items. There are several predefined classification schemes available. Obvious possibilities are DMOZ, WordNet and the ACM topic hierarchy (also available in *RDF*). However, we would like to enable the user to build up his/her own scheme - to use, share and manipulate it as required. The envisioned use is that a peer group will jointly define a top level, coarse grain topic structure (probably hierarchy). As time goes on, different users will be able to add to this (simply by adding a more specialised topic; topic reorganisation, renaming or deletion is not allowed). A nice (optional) utility would be to allow the user who has added a topic to recategorise items from the immediate parent topic if required. This task is not expected to be unduly arduous for small group bibliographic data. It might be possible to start from a very high level bootstrap topic ontology, but we do not advocate the use of a detailed domain ontology. Therefore in this particular case we advocate a predominantly 'roll your own' approach. Again, if we allow the user to build their own hierarchy, then there is little subtlety to consider in the underlying ontology. We don't expect to build an ontology editing tool; for the purposes of this demonstrator we expect the ontologies to be pre-built (using any method, eg text editing, ontology editor) and then slotted in. Essentially the topics are just classes, linked using the channel hierarchy ontology.

Presentation

We need special purpose ontologies to define certain presentation components. Examples include layout of the metadata edit dialog, semantic view and semantic query. There is little option here but to create our own. Certainly we advocate using as simple an ontology as possible. We expect to specify some broad preferences, which the *UI* can then intelligently use to allocate display resources as appropriate. Details have been provided in the appropriate sections

B Easy News Topics

This appendix lays out a suggested approach to encoding Easy News Topics (ENT) in RSS1.0. Please see the official ENT Spec [☞ENT] for reference. The RSS2.0 definitions in this document are taken from there. See also the short RSS1.0 taxonomy module [☞RSS-TAXO] for an alternative RSS1.0 way to do a similar thing.

Namespace

RSS2.0: `xmlns:ent="http://www.purl.org/NET/ENT/1.0/`
RSS1.0: `xmlns:ent="http://jena.hpl.hp.com/ENT/1.0/`

Cloud

RSS2.0	RSS1.0	Description
`ent:href` (Attribute)	`<uri of resource>`	A URI which serves as the unique identifier for the cloud.
`ent:resourceRef` (Attribute)	`ent:resourceRef` (Property)	An optional URI which refers to an external resource where the topics used in the cloud are formally defined.
`ent:infoRef` (Attribute)	`ent:infoRef` (Property)	An optional URI which refers to a page of human readable information about this cloud
`ent:description` (Attribute)	`ent:description` (Property)	An optional string that can be used to describe the cloud.

Topic

RSS2.0	RSS1.0	Description
`ent:id` (Attribute)	`<uri of resource>`	Unique identifier for this topic. In RSS1.0 this must be globally unique - a URI.
`ent:classification` (Attribute)	`ent:classification` (Property)	For systems that wish to relate topics to one another a classification can be specified (questionable value in RSS1.0)
`ent:href` (Attribute)	`ent:href` (Property)	The href attribute is available for systems to indicate a human readable page related to the topic.

	`ent:inCloud`(Property)	New RSS1.0 property

The other *RDF* things we need are:

- a `Topic` and `Cloud` class.
- an `ent:hasTopic` property that is used to link an item to its topic.
- Possibly an `ent:hasTopics` (or similar) property that is used to link a cloud to its topics.

Design Choices
There are a number of design choices we have made in representing ENT as RSS1.0. These are:

- ENT attributes (in RSS2.0) are properties (in RSS1.0)
- Topics are not 'bagged up' into clouds (as in ENT RSS2.0) or bags (as in Taxonomy module)
- Items have topics not clouds! While it may be natural in RSS2.0, it makes no sense (in *RDF*) to say that an item 'has' a cloud of topics: rather that it 'has' a set of topics, each of which belongs to a cloud.
- We have introduced a `Topic` and `Cloud` class, an `ent:hasTopic` property and an `ent:inCloud` property.

Some examples are shown in the following table:

RSS2.0	RSS1.0

`<item>`
`<title>A sample item</title>`
`<description>A sample item</description>`
`<cloud href="http://www.example.org/">`
`<topic id="sample">Sample</topic>`
`</cloud>`
`</item>`
	`<item>`
	`<title>A sample item</title>`
	`<description>A sample item</description>`
	`<ent:hasTopic`
	`rdf:resource="http://www.example.org/Sample"/>`
	`</item>`
	
	`<ent:Topic rdf:about="http://www.example.org/Sample">`
	`<ent:inCloud rdf:resource="http://www.example.org/" />`
	`<dc:title>Sample</dc:title>`
	`</ent:Topic>`

`<item>`
`<title>Giants go 7-0</title>`
`<description>(snipped)</description>`
`<ent:cloud ent:href="http://matt.blogs.it/">`
`<ent:topic`
`ent:id="sf_giants"`
`ent:classification="team"`
`ent:href="http://matt.blogs.it/topics/`
`topicsS.html#sf_giants">`
`San Francisco Giants`
`</ent:topic>`
`<ent:topic`
`ent:id="bosox"`
`ent:classification="team"`
`ent:href="http://matt.blogs.it/topics/`
`topicsB.html#bosox">`
`Boston Redsox`
`</ent:topic>`
`</ent:cloud>`
`</item>`
	`<item>`
	`<title>Giants go 7-0</title>`
	`<description>(snipped)</description>`
	`<ent:hasTopic`
	`rdf:resource="http://matt.blogs.it/sf_giants"/>`
	`<ent:hasTopic`
	`rdf:resource="http://matt.blogs.it/bosox"/>`
	`</item>`
	
	`<ent:Topic rdf:about="http://matt.blogs.it/sf_giants">`
	`<ent:inCloud`
	`rdf:resource="http://matt.blogs.it/" />`
	`<ent:classification>team</ent:classification>`
	`<dc:title>San Francisco Giants</dc:title>`
	`<ent:href`
	`rdf:resource="http://matt.blogs.it/topics/topicsB.html`
	`#sf_giants"/>`
	`</ent:Topic>`
	
	`<ent:Topic rdf:about="http://matt.blogs.it/bosox">`
	`<ent:inCloud rdf:resource="http://matt.blogs.it/"/>`
	`<ent:classification>team</ent:classification>`
	`<dc:title>Boston Redsox</dc:title>`
	`<ent:href`
	`rdf:resource="http://matt.blogs.it/topics/topicsB.html`
	`#bosox"/>`
	`</ent:Topic>`

`<item>`
`<description>`
`Here is the text of the item.`
`</description>`
`<ent:cloud ent:href="http://matt.blogs.it/">`
`<ent:topic`
`ent:id="sf_giants">`
`San Franscisco Giants`
`</ent:topic>`
	`<item>`
	`<description>`
	`Here is the text of the item.`
	`</description>`
	`<ent:hasTopic`
	`rdf:resource="http://matt.blogs.it/sf_giants" />`
	`<ent:hasTopic`
	`rdf:resource=`
	`"http://www.examples.com/mlb/players2003.xtm#barry_bonds"`
	`/>`
	`<ent:hasTopic`
	`rdf:resource=`
	`"http://www.examples.com/mlb/players2003.xtm#ray_durham"`
	`/>`
	`<ent:hasTopic`
	`rdf:resource=`
	`"http://www.examples.com/mlb/players2003.xtm#felipe_alou"`
	`/>`
	`</item>`
	
	`<ent:Topic rdf:about="http://matt.blogs.it/sf_giants ">`
	`<ent:inCloud rdf:resource="http://matt.blogs.it/"/>`

`<ent:cloud`
`ent:href="http://www.examples.com/mlb/"`
`ent:resourceRef=`
`"http://www.examples.com/mlb/players2003.xtm">`
`<ent:topic`
`ent:id="barry_bonds"`
`ent:classification="player">`
`Barry Bonds`
`</ent:topic>`
`<ent:topic`
`ent:id="ray_durham"`
`ent:classification="player">`
`Ray Durham`
`</ent:topic>`
`<ent:topic`
`ent:id="felipe_alou"`
`ent:classification="manager">`
`Felipe Alou`
`</ent:topic>`
`</ent:cloud>`
`</item>`
	`<ent:classification>team</ent:classification>`
	`<dc:title>San Franscisco Giants</dc:title>`
	`</ent:Topic>`
	
	`<ent:Topic rdf:about=`
	`"http://www.examples.com/mlb/players2003.xtm#barry_bonds"`
	`>`
	`<ent:inCloud rdf:resource=`
	`"http://www.examples.com/mlb/players2003.xtm" />`
	`<ent:classification>player</ent:classification>`
	`<dc:title>Barry Bonds</dc:title>`
	`</ent:Topic>`
	
	`<ent:Topic`
	`rdf:about=`
	`"http://www.examples.com/mlb/players2003.xtm#ray_durham" >`
	`<ent:inCloud rdf:resource=`
	`"http://www.examples.com/mlb/players2003.xtm" />`
	`<ent:classification>player</ent:classification>`
	`<dc:title>Ray Durham</dc:title>`
	`</ent:Topic>`
	
	`<ent:Topic rdf:about=`
	`"http://www.examples.com/mlb/players2003.xtm#felipe_alou">`
	`<ent:inCloud rdf:resource=`
	`"http://www.examples.com/mlb/players2003.xtm" />`
	`<ent:classification>manager</ent:classification>`
	`<dc:title>Felipe Alou</dc:title>`
	`</ent:Topic>`
	
	`<ent:Cloud rdf:about="http://www.examples.com/mlb/">`
	`<ent:resourceRef rdf:resource=`
	`"http://www.examples.com/mlb/players2003.xtm" />`
	`</ent:Cloud>`

So an example entire document would be:

```
   <?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:taxo="http://purl.org/rss/1.0/modules/taxonomy/"
xmlns:ent="http://jena.hpl.hp.com/ENT/1.0/"
xmlns="http://purl.org/rss/1.0"
xmlns:dc="http://purl.org/dc/elements/1.1/">

<item>
<title>A sample item</title>
<description>A sample item</description>
<ent:hasTopic rdf:resource="http://www.example.org/Sample"/>
</item>

<ent:Topic rdf:about="http://www.example.org/Sample">
<ent:inCloud rdf:resource="http://www.example.org/" />
<dc:title>Sample</dc:title>
</ent:Topic>

</rdf:RDF>
```

C References

[ANNOTEA]
: *Annotea*: a system for creating and publishing shareable annotations of Web documents.
☞http://www.w3.org/2001/Annotea/
(latest draft Dec 2002): ☞http://www.w3.org/2002/12/AnnoteaProtocol-20021219

[ANNOTEA-THREADS]
: Reply protocol in Annotea
Document: ☞http://www.w3.org/2001/Annotea/User/Protocol.html#ReplyProtocol
Schema: ☞http://www.w3.org/2001/03/thread

[ATOM]
: The *Atom* project
☞http://www.intertwingly.net/wiki/pie/FrontPage

[ATOM-API]
: The *Atom* API
☞http://bitworking.org/rfc/draft-gregorio-07.html

[BibTeX]
: *LaTeX: A Document Preparation System* by Leslie Lamport, 1986, Addison-Wesley.
BibTeXing (☞btxdoc.tex), by Oren Patashnik, February 1988, (BibTeX distribution).
☞http://www.ecst.csuchico.edu/~jacobsd/bib/formats/bibtex.html

[BIBTEX2RDF]
: *BibTeX-2-RDF translator*, by Michel Klein
☞http://www.cs.vu.nl/~mcaklein/bib2rdf/

[BLEDIT]
Bledit, an editing plugin for Blojsom [☞BLOJSOM]
Available from the *Pop2Blog* project ☞http://sourceforge.net/projects/pop2blog
[BLOGGER-API]
The *Blogger* API
☞http://new.blogger.com/developers/api/1_docs/
[BLOGTALK]
BlogTalk, - A European Conference On Weblogs
BlogTalk 2003: ☞http://2003.blogtalk.net/
BlogTalk 2004: ☞http://www.blogtalk.net/
[BLOJSOM]
Blojsom, a java based blogging platform
☞http://blojsom.sourceforge.net/
[BROWNSAUCE]
Brownsauce, an RDF Browser
☞http://brownsauce.sourceforge.net/
[CACM]
Communications of the ACM
☞http://www.acm.org/cacm/
[CLAIMAKER]
ClaiMaker: a tool from the *ScholOnto* project
☞http://claimaker.open.ac.uk/
[CITESEER]
CiteSeer Publications Research Index (NEC Research Institute)
☞http://citeseer.nj.nec.com/cs
[DARCUS]
Bruce D'Arcus, Bibliographic blogger
☞http://netapps.muohio.edu/movabletype/darcusb/darcusb/
[DC]
The Dublin Core Metadata Initiative
☞http://dublincore.org/groups/citation/
DC Element Set, Version 1.1: ☞http://dublincore.org/documents/dces/
[DMOZ]
DMOZ - The open directory project
☞http://dmoz.org/
[ENT]
Easy News Topics - RSS2.0 Module
☞http://www.purl.org/NET/ENT/1.0/
[EPERSON]
The ePerson Snippet Manager: a Semantic Web Application
Banks, Dave; Cayzer, Steve; Dickinson, Ian; Reynolds, Dave
☞http://www.hpl.hp.com/techreports/2002/HPL-2002-328.html
[FOAF]
The *friend of a friend* project.
☞http://www.foaf-project.org/
FOAF Vocabulary Specification (RDFWeb Namespace Document 16 August 2003; Dan Brickley & Libby Miller): ☞http://xmlns.com/foaf/0.1/
[FRBR]
Functional Requirements for Bibliographic Records
A project of the OCLC [☞OCLC]
☞http://www.oclc.org/research/projects/frbr/
[GOOGLE-TOOLBAR]
The Google Toolbar
☞http://toolbar.google.com/
[IBIS]
Issue-Based Information Systems (IBIS) Terms, Danny Ayers October 2002
☞http://ideagraph.net/xmlns/ibis/
[INNOGY]
Innogy, a UK Energy Company
☞http://www.rweinnogy.com/index.asp
[JENA]
Jena, a semantic web toolkit
☞http://www.hpl.hp.com/semweb/jena.htm
[JENA-ARP]
ARP, the Jena RDF/XML Parser
☞http://www-uk.hpl.hp.com/people/jjc/arp/
[JOSEKI]
Joseki, the Jena RDF Server
☞http://www.joseki.org/
[METAWEBLOG]
The MetaWeblog API by Dave Winer
☞http://www.xmlrpc.com/metaWeblogApi
[MODS]
Metadata Object Description Schema

☞http://www.loc.gov/standards/mods/mods-overview.html

[MODS-DARCUS]
Biblio Schema
A Relax NG bibliographic schema by Bruce D'Arcus (MODS derived)
☞http://www.users.muohio.edu/darcusb/files/biblio-schema.html

[MT]
MovableType personal publishing system
☞http://www.movabletype.org/

[MT-RDF]
Selection of utilities to add RDF to MT blog entries, written by Matt Biddulph
☞conversion template, ☞n3 parser, ☞href detector

[MT-SPAM]
Comment Spam on MovableType
☞http://www.sixapart.com/log/2003/10/comment_spam.shtml

[MT-TYPEPAD]
Typepad, a hosted MovableType solution
☞http://www.typepad.com/

[N3]
N3, an RDF language for the Semantic Web
☞http://www.w3.org/DesignIssues/Notation3.html

[NETAPI]
RDF Net API. W3C Member Submission 2 October 2003
Graham Moore and Andy Seaborne
☞http://www.w3.org/Submission/2003/SUBM-rdf-netapi-20031002/

[NETWORK-INFERENCE]
Network Inference
☞http://www.networkinference.com/index.html

[NEWSCIENTIST]
New Scientist
☞http://www.newscientist.com/

[NEWSMONSTER]
NewsMonster RDF based news, weblog, and RSS aggregator
☞http://www.newsmonster.org/

[OCLC]
Online Computer Library Center
☞http://www.oclc.org/

[OCLC-DC-RDF]
Online Computer Library Center - Connexion
Features include export in Dublin Core RDF/XML:
☞http://www.oclc.org/connexion/about/features/default.htm

[ONTOWEB]
The OntoWeb Ontology
☞http://ontoweb.aifb.uni-karlsruhe.de/Ontology/

[OWL]
OWL Web Ontology Language Reference
W3C Candidate Recommendation 18 August 2003
☞http://www.w3.org/TR/owl-ref/
Other OWL resources available at ☞http://www.w3.org/2001/sw/WebOnt/

[RAVENSBOURNE]
Ravensbourne College of Design and Communication (London, UK)
☞http://www.rave.ac.uk

[RDF-LANGUAGE]
Resource Description Framework (RDF) Model and Syntax Specification
W3C Recommendation 22 February 1999
☞http://www.w3.org/TR/REC-rdf-syntax/
RDF/XML Syntax Specification (Revised)
W3C Proposed Recommendation 15 December 2003
☞http://www.w3.org/TR/rdf-syntax-grammar/
Resource Description Framework (RDF): Concepts and Abstract Syntax
W3C Working Draft 10 October 2003
☞http://www.w3.org/TR/2003/WD-rdf-concepts-20031010/
Other RDF resources available at ☞http://www.w3.org/RDF/

[RDF-MODEL]
RDF Semantics
W3C Proposed Recommendation 15 December 2003
☞http://www.w3.org/TR/rdf-mt/

[RDF-OBJECTS]
RDF Objects
HP Labs project
☞http://www.hpl.hp.com/semweb/rdfobjects.htm

[RDF-SCHEMA]
RDF Vocabulary Description Language 1.0: RDF Schema
W3C Working Draft 10 October 2003
☞http://www.w3.org/TR/rdf-schema/

[REDLAND-RAPTOR]
Raptor RDF Parser Toolkit
☞http://www.redland.opensource.ac.uk/raptor/
[REUTERS]
Reuters
☞http://www.reuters.com/
[RELAXNG]
RELAX NG, a schema language for XML
☞http://www.relaxng.org/
[RSS10]
RSS 1.0 Standard
☞http://web.resource.org/rss/1.0/
RSS1.0 Specification: ☞http://web.resource.org/rss/1.0/spec
RDF Rich Site Summary - history and resources: ☞http://www.oasis-open.org/cover/rss.html
[RSS20]
RSS 2.0 Standard
☞http://backend.userland.com/rss
[RSS-ANNOTATE]
RSS 1.0 Modules: Annotation
Niel Bornstein, Edd Dumbill, Kendall Clark, Ken MacLeod
☞http://web.resource.org/rss/1.0/modules/annotation/
[RSS-CONTENT]
RDF Site Summary 1.0 Modules: Content
Gabe Beged-Dov, Aaron Swartz, Eric van der Vlist
☞http://web.resource.org/rss/1.0/modules/content/
[RSS-MODULES]
RDF Site Summary 1.0 proposed modules
☞http://web.resource.org/rss/1.0/modules/proposed.html
[RSS-TAXO]
RSS 1.0 Modules: Taxonomy
Gabe Beged-Dov, Dan Brickley, Rael Dornfest, Ian Davis, Leigh Dodds, Jonathan Eisenzopf, David Galbraith, R.V. Guha, Ken MacLeod, Eric Miller, Aaron Swartz, Eric van der Vlist
☞http://web.resource.org/rss/1.0/modules/taxonomy/
[SEMAVIEW]
Semaview
☞http://www.semaview.com/
[SEMBLOG-DEMO]
Semantic Blogging Demonstrator
☞http://jena.hpl.hp.com:3030/blojsom-devt/blog/
[SEMBLOG-DESIGN]
Semantic Blogging Demonstrator - Design Notes
☞http://jena.hpl.hp.com/~stecay/downloads/architecture.pdf
[SEMBLOG-JAVADOC]
Semantic Blogging Demonstrator - Javadoc
☞http://jena.hpl.hp.com/~stecay/javadoc/semblog/index.html
[SEMBLOG-RQMTS]
SWAD-Europe: Semantic Blogging and Bibliographies - Requirements Specification
Steve Cayzer, Paul Shabajee
☞http://www.w3.org/2001/sw/Europe/reports/open_demonstrators/
hp-requirements-specification.html
[SEMBLOG-VISION]
Semantic Blogging Demonstrator - The Vision
☞http://jena.hpl.hp.com/~stecay/downloads/blogTalk.pdf
[SHIRKY-SYLLOGISTIC]
The Semantic Web, Syllogism, and Worldview
☞http://www.shirky.com/writings/semantic_syllogism.html
See summary of discussion at ☞http://www.poorbuthappy.com/ease/semantic/
[SKOS]
Simple Knowledge Organisation System
A deliverable of [☞SWADE] workpackage 8.1
☞http://www.w3c.rl.ac.uk/SWAD/deliverables/8.1.html
SKOS core schema ☞http://www.w3c.rl.ac.uk/2003/11/21-skos-core
SKOS mapping schema ☞http://www.w3c.rl.ac.uk/2003/11/21-skos-mapping
[SSR]
Simple Semantic Resolution - RSS 2.0 Module
☞http://ideagraph.net/xmlns/ssr/
SSR-Enabling an RSS 2.0 Module
☞http://ideagraph.net/xmlns/ssr/modules.htm
[SWAD]
Semantic Web Advanced Development ☞http://www.w3.org/2000/01/sw/
[SWADE]
Semantic Web Advanced Development - Europe ☞http://www.w3.org/2001/sw/Europe/
[SWADE-ANALYSIS]
Semantic Web Applications - Analysis and Selection HP SWADE Report 2002

☞http://www.w3.org/2001/sw/Europe/reports/open_demonstrators/hp-applications-selection.html

[SWADE-MAPPING]
Schema Technology Survey
Stephen Buswell, Dan Brickley, Brian Matthews
SWAD-Europe Deliverable 5.1
☞
http://www.w3.org/2001/sw/Europe/reports/xml_schema_tools_techniques_report/Overview.html

[SWADE-SEMPORTALS]
Semantic Portals
part of SWAD-Europe Deliverable 12.1
Requirements Specification:
☞http://www.w3.org/2001/sw/Europe/plan/workpackages/live/esw-wp-12.1.html

[SWADE-THESAURUS]
SWAD-E Thesaurus activity
☞http://www.w3c.rl.ac.uk/SWAD/thesaurus.html
SWAD-E [☞SWADE] Workpackage 8 description:
☞http://www.w3.org/2001/sw/Europe/plan/workpackages/live/esw-wp-8.html
RDF Thesauri: ☞http://www.w3c.rl.ac.uk/SWAD/rdfthes.html

[SWADE-XML]
Integration with XML Technology SWAD-E [☞SWADE] Workpackage 5
☞http://www.w3.org/2001/sw/Europe/plan/workpackages/live/esw-wp-5.html

[SWADE-USER-STUDY]
User Study on Bibliographic Management
☞
http://www.w3.org/2001/sw/Europe/reports/open_demonstrators/hp-requirements-specification.html#sec-userStudy
Part of [☞SEMBLOG-RQMTS]

[TIF]
Thesaurus Interchange Format
Paper: "Modelling Thesauri for the Semantic Web"
☞http://www.w3c.rl.ac.uk/SWAD/thesaurus/tif/deliv81/final.html
Schema: ☞http://www.w3c.rl.ac.uk/SWAD/thesaurus/tif/tif.html
Now replaced by [☞SKOS]

[TOPIC-MAPS]
Topic Maps activity
☞http://www.topicmaps.org/

[VIEWSOURCECLAN]
The View Source Clan, a term used by Atom [☞ATOM] developers
☞http://www.intertwingly.net/wiki/pie/ViewSourceClan

[WARWICK]
Warwick University, UK
☞http://www.warwick.ac.uk/

[WBLOGGAR]
W.Bloggar Blog Interface
☞http://www.wbloggar.com/

[WEINBURGER-SEMBLOG]
Joho the Blog (David Weinburger) Blog entry from BlogTalk [☞BLOGTALK]
☞http://www.hyperorg.com/blogger/mtarchive/001528.html

[WORDNET]
WORDNET - a lexical database for the English language.
☞http://www.cogsci.princeton.edu/~wn/
RDF Schema: ☞http://xmlns.com/2001/08/wordnet/

[XFML]
eXchangeable Faceted Metadata Language - core specification
☞http://www.xfml.org/spec/1.0.html