How to Make a Semantic Web Browser

Dennis Quan
IBM T. J. Watson Research Center
1 Rogers Street
Cambridge, MA 02142 USA
dennisq@us.ibm.com

David R. Karger
MIT CSAIL
200 Technology Square
Cambridge, MA 02139 USA
karger@theory.lcs.mit.edu

ABSTRACT

Two important architectural choices underlie the success of the Web: numerous, independently operated servers speak a common protocol, and a single type of client—the Web browser—provides point-and-click access to the content and services on these decentralized servers. However, because HTML marries content and presentation into a single representation, end users are often stuck with inappropriate choices made by the Web site designer of how to work with and view the content. RDF metadata on the Semantic Web does not have this limitation: users can gain direct access to the underlying information and control how it is presented for themselves. This principle forms the basis for our Semantic Web browser—an end user application that automatically locates metadata and assembles point-and-click interfaces from a combination of relevant information, ontological specifications, and presentation knowledge, all described in RDF and retrieved dynamically from the Semantic Web. With such a tool, naïve users can begin to discover, explore, and utilize Semantic Web data and services. Because data and services are accessed directly through a standalone client and not through a central point of access (e.g., a portal), new content and services can be consumed as soon as they become available. In this way we take advantage of an important sociological force that encourages the production of new Semantic Web content by remaining faithful to the decentralized nature of the Web.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques – *programmer workbench, user interfaces.*

General Terms

Human Factors, Design

Keywords

Semantic Web, RDF, user interface, Web Services, bioinformatics

1. INTRODUCTION

1.1 Motivation

The World Wide Web revolutionized the Internet by providing a number of mutually reinforcing capabilities. HTTP offered a simple standard by which information could be fetched from any Web server. HTML provided a uniform syntax in which publishers could present information that would be rendered in human-readable form in a Web browser. And URLs gave a way for any Web page to refer to any other Web page, regardless of its location. Taken together, these capabilities meant that a lay user could seamlessly browse the entire space of Web information, viewing information without concern for location and using a simple point and click interface to navigate from any Web page to others that it referenced.

Though substantial, the powerful information access capability engendered by the Web has its limitations. Through its use of HTML and HTTP servers, the Web demands the production of content already formatted for presentation in a particular human-readable fashion. Implicit is the idea that a publisher will be able to figure out the right way to present its information to end users. It should be clear, however, that the information *consumer* will often have the best sense of what is important in the fetched information and how best to make use of it.[1] Every Web browser offers its user some limited ability to override presentation characteristics such as the font and font size. Stronger evidence of the need for clients to control the presentation of information can be seen in the development of HTTP content negotiation standards, in which the client describes its capabilities and hopes that the server will deliver something that can be presented reasonably [33], as well as the Web Accessibility Initiative's attempt to develop guidelines for crafting presentations of Web pages so that they can be used by people with disabilities [34]. Finally, efforts such as NewsIsFree show that means as extreme as screen scraping are employed in order to enable Web site content to be viewed in alternate ways (in this case, as RSS news feeds in news tickers, news alert tools, etc.) [36].

The Semantic Web offers a particularly extreme example of differently-abled clients: nonhumans. In the Semantic Web vision, autonomous agents will be able to pull information from the Web and manipulate it on behalf of their users. HTML is clearly a terrible data presentation language for such applications—its visual markup hides the semantic content that agents actually care about. This problem has motivated the development of RDF, a language for describing semantic information in a machine-readable form *without* the distraction of presentation markup.

We argue that beyond its support for automation, the Semantic Web lets us dramatically improve the way people *directly* access information. The Semantic Web gives us the opportunity to separate content—the proper purview of the publisher serving the information—from presentation—an issue in which the end user or their local application, aware of the uses to which they are putting the information, should have substantial say. The fact that information is semantically marked up, instead of just being formatted for display, makes it possible for end user applications to make intelligent decisions on how to present the information based on the purposes for which it is being used. One application may show more details of an information object, another less. One may present visually rich attributes of

[1] Arguably, this is an application of the classic end-to-end argument [32] that lower levels of an application should not be making choices best left to the top elements that understand the overall goals of the system.

WWW 2004, May 17–22, 2004, New York, New York, USA.
ACM 1-58113-844-X/04/0005.

an object, another focus on textual or audible content. An application accessing a city description for the purposes of travel planning can present entirely different information from one being used to prepare a history report or evaluate a business opportunity.

1.2 Approach

In this paper we describe Haystack, an application that can be used to browse arbitrary Semantic Web information in much the same fashion as a Web browser can be used to navigate the Web. Haystack aggregates RDF from multiple arbitrary locations and presents it to the user in a human-readable fashion, with point and click semantics that let the user navigate from one piece of Semantic Web data to other, related pieces. Haystack's presentation of the information is controlled by presentation "recommendations," much like cascading stylesheets, that are themselves described in RDF; such recommendations can be issued by the content server but can also be made by context-specific "applications" that understand how the information can best be presented to meet a particular need or complete a particular task, or even by a third party offering helpful viewing advice. Finally, to support information gathering and structuring, Haystack offers a rich model of collections. Users can build collections of links to Semantic Web objects with much the same ease as they presently have creating bookmarks; at the same time, Haystack's rich presentation layer can display those collections as thumbnails, Web pages, taxonomies, or various other views that can help the user exploit the collections once created.

Overall, our approach is based on the principle that content can be broken down into its key elements, i.e., data, presentation recommendations, and functionality, and that these elements can be individually published and consumed by users. Individual data, presentation styles, and pieces of functionality are describable in RDF and can be incrementally pieced together by Haystack at the client end to form custom user interfaces and functionality for particular individuals and applications.

We believe that our approach to *Semantic Web browsing* offers several key benefits that motivate its adoption for a variety of end user applications. For example, separate pieces of information about a single resource that used to require navigation through several different Web sites can be merged together onto one screen, and this merging can occur without specialized portal sites or coordination between Web sites or databases. Furthermore, services applicable to some piece of information need not be packaged into the Web page containing that information, nor must information be copied and pasted across Web sites to access services; instead, semantic matching of resources to services that can consume them can be done by the client and exposed in the form of menus.

The separation of content from presentation means lowers the bar to publishing, since individuals can now produce "unformatted" semantic information, relying on end user clients to figure out good ways to present it. Conversely, users can publish new ways of looking at existing information without modifying the original information source.

The existence of a good Semantic Web browser may also speed the proliferation of the Semantic Web. Much has been made of the potential for the Semantic Web to support powerful information interchange and automation, but to date the amount of accessible Semantic Web information remains rather small. Arguably, the rapid, organic growth of the Web was due in large part to the ubiquity of the Web browser—a universal client that provides immediate access to new content as soon as it comes online. This ability for new Web sites to be instantly available to users is a key sociological driving force that should not be underestimated: it encourages numerous individuals to produce content, in the knowledge that there will be easy access to it. An effective Semantic Web browser that allows users similarly instant access to newly published Semantic Web information may likewise motivate individuals to publish Semantic Web content.

The Haystack system, which embodies the ideas described in this paper, is an Open Source RDF-based information management environment [1] [7]. Haystack is written in Java and is built into the Eclipse platform [6], providing a stable basis for extension writers (another important feature of the Web browser). A screenshot of Haystack appears in Figure 1. To demonstrate the practicality of our approach, we have applied Haystack towards an important target domain for the Semantic Web—bioinformatics, an area rich with various kinds of resources (protein structure data, genetic sequences, published papers, etc.), metadata connecting them, and services that operate on these objects [38]. Just as anyone can browse and publish to the Web without understanding how Web servers work, making the Semantic Web accessible to scientists who are experts in their domains but not necessarily on the supporting technology is a crucial first step to expanding the reach of the Semantic Web. Furthermore, bioinformatics is rich with cross-linked information, making it well-able to take advantage of the cross-linking representation that is central to the Semantic Web. In this way it is similar to many other domains, including personal information management, other research areas, and even business process integration. Finally, bioinformatics provides us with real, large data sources that are expressed in RDF via the Life Science Identifier (LSID) standard [5].

1.3 Outline of the Paper

The paper starts off with a summary of important related work. Then we characterize our approach in a bottom-up fashion. First we describe the critical elements of a resource naming scheme: universality and the ability to retrieve metadata. Next we discuss our strategy for managing connections to disparate RDF repositories and various approaches for federating metadata that take into account the diversity of store implementations and network inefficiencies. We describe Haystack's support for LSID and HTTP URIs via a caching metadata store and show why universal retrieval is important from the perspectives of both automation and usability. Afterwards, we introduce Haystack's user interface paradigm and the visualization ontologies that support it. We talk about collections—a powerful RDF-based mechanism for grouping related items together, reflecting the frequent use of Web pages to aggregate related items and building on the notion of customization introduced by bookmark managers in the Web browser. Furthermore, we show how one might apply Haystack's user interface techniques to service discovery and invocation. We tie up our discussion with an example of Haystack being used to browse several bioinformatics databases and build up a collection of interesting items. Finally, we discuss practical issues of publication to the Semantic Web: trust, inference, and dealing with ontological mismatch.

1.4 Related Work

An important distinction between the approach presented in this paper and other Semantic Web metadata visualization approaches can be found in examining who maintains control over what metadata is shown to the user and how this metadata is presented. For example, with a semantic portal (e.g., SEAL [2] or Semantic Search [8]), it is the administrator who aggregates semantically-classified information in a centralized location for dissemination to users. Because these portals often use Web servers to distribute their in-

formation, server side HTML templates are typically employed to convert metadata into a human-readable presentation. The semantic portal approach has the advantage of maintainability, since all of the presentation logic and choice of data sources are configured in one central location. Furthermore, Semantic Web information can be consumed by users of the existing Web; end users gain access to important metadata without needing to be aware that RDF is involved.

Unfortunately, the dynamic, *ad hoc* nature of the Web—anyone being able to author a piece of information that is immediately available to everyone—is thus buried within ostensibly monolithic aggregations under centralized control. In particular, if someone wishes to publish information reflecting a new schema, no portal will be able to present it unless and until the portal administrator modifies his or her display system. This model also detracts from the free contribution of content because it forces a content producer to either set up or locate a specialized semantic portal to host the new content.

On the other hand, if users are allowed to directly access information sources, then we could return content production to its powerful decentralized behavior. Users would gain immediate access to the growing range of available RDF-based data and services. Enabling users to feel the network effects of the Semantic Web's expansion is crucial to gaining greater acceptance of the Semantic Web and its potential to enable agent-based automation.

Although the effective separation of content from presentation might conceivably be achieved using a properly designed server that fetches content and presentation from outside when needed, the decision to eliminate the centralized administration of presentation knocks out the main argument for a server model. Instead, we have chosen to implement our Semantic Web browser as an application running on the user's local machine because it is simpler to implement, is more scalable, and provides a higher-fidelity user interface than an equivalent server-based setup.

Other systems exist for visualizing RDF metadata that take the form of end user applications. These systems commonly employ automatic form generation techniques seen in desktop database applications; a good example is Protégé [30], an ontology editor. Such systems are capable of taking a schema or ontology definition and presenting specialized, key-value pair-based forms to the user that allow instances of classes in the ontology to be created, modified, and queried. This is one approach used by our user interface framework, which also incorporates human-computer interface-inspired points of flexibility, such as the ability to group property fields into lenses and to allow for multiple styles of presentation (views) for instance resources.

Other applications take another approach to visualization that is inspired by the notion of the Semantic Web being an extension of the existing Web. Systems such as Magpie augment standard Web browsers with the ability to act on resources described in Web pages and to find resources semantically related to a Web page [31]. Tools such as Annotea allow users to embed and read RDF-encoded textual annotations in Web pages from a Web browser [9]. The primary difference between these approaches and the one described in this paper is that we are providing support for visualizing RDF metadata in its own right, not just the metadata connecting or embedded in Web pages. In other words, Haystack is a *Semantic Web* browser, not just a Semantic *Web browser*.

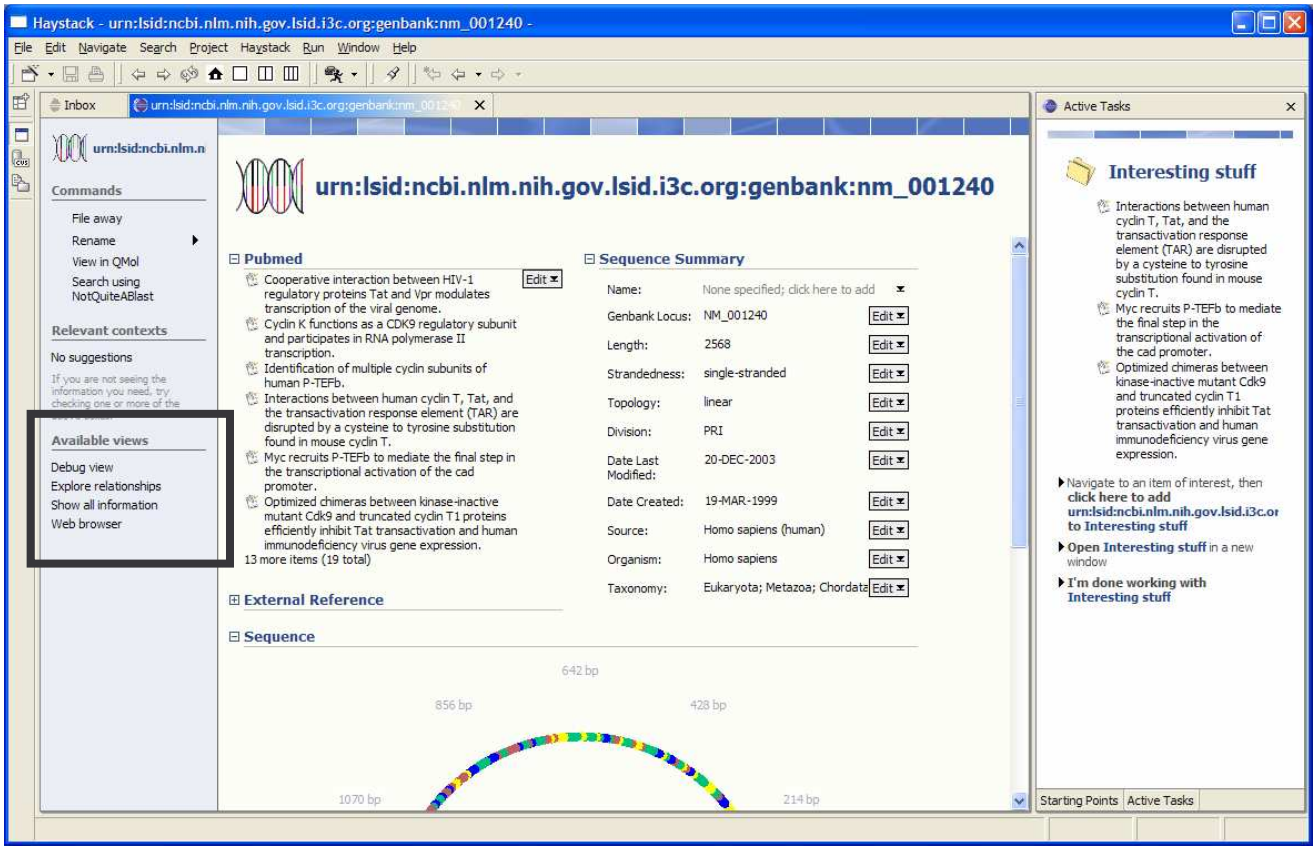


Figure 1. Haystack displaying mRNA sequence data named by LSID from multiple sources (side: an *ad hoc* collection of related resources collected by the user; thick box indicates view selector control).

2. RESOURCE NAMING SCHEMES

A necessary piece of infrastructure for the Semantic Web is an appropriate shared naming scheme. There seems to be a general consensus around the ultimate arrival of some common URI scheme, with some resolution layer that lets one fetch all or some information associated with a given URI. But this consensus has yet to evolve into a universal standard. In the meantime, various groups have attempted to jump-start universal naming by defining naming schemes such as handles [26]. In this paper we give particular attention to Life Science Identifiers (LSIDs), a naming scheme being pushed by life sciences informatics communities [5]. A large number of biological resources, such as proteins, published papers, and genetic sequences, are already named by LSIDs. Like URLs, LSIDs are URIs that contain a field that can usually be resolved in DNS to locate a server that can be contacted to resolve (provide information about) a given LSID. LSID servers can be accessed using a SOAP-based Web Services protocol to retrieve octets and/or RDF-encoded metadata.

3. ACCESSING RDF SOURCES

Above URIs is the next important layer of the Semantic Web layer cake—distributed pieces of RDF metadata. As with any distributed body of information, such as a distributed file system, much information of interest is stored remotely, and one needs efficient means for tapping into the network. To achieve the goal of allowing users to type in any URI and be immediately able to see relevant information, a simplistic approach would be to assume that all URIs have a server field and to have the browser contact that server to obtain the chunk of all RDF statements related to the resource named by the URI. This simplistic approach is actually the one adopted by LSIDs and the subset of HTTP URIs that is actually hosted by Web servers that can return RDF metadata today; the implicit assumption is that these URIs have an authoritative server that can always be contacted as a useful starting point.

The per-URI RDF download approach is not always practical. Sometimes, all that is needed is a focused answer to a complex query, so it would be more efficient for this query to occur where the data is located. Also, unlike the Web, the Semantic Web allows parties other than the authoritative server to provide statements about resources, and these metadata residing on separate servers should be accessible. For these reasons, many RDF repositories on the Semantic Web, such as Annotea [9], TAP [8], and Joseki servers [16], can resolve queries over arbitrary resources written to fairly expressive query languages. On the other hand, many RDF repositories are simply RDF files sitting on an HTTP or FTP server. In this case, it may be desirable to download the entire file and to host it in a local database capable of answering queries in order to minimize network traffic.

For Haystack's implementation, we have chosen to model the various kinds of RDF sources mentioned above with an *RDF store abstraction* [29]. Specifically, Haystack's notion of an RDF store allows specific forms of queries and optionally allows RDF statements to be added and removed. Furthermore, Haystack's RDF store abstraction supports an event mechanism: components can be notified when statements matching specific patterns are added to or removed from the store. Haystack includes a default RDF store implementation built on top of a database and an instantiation of this implementation that we refer to as the "root" RDF store, which is used to locally store metadata created by the user (we discuss uses for this store throughout this paper). To support RDF sources such as LSID, Annotea, TAP, and Joseki, one can imagine implementing a virtual read-only RDF store adapter that accepts queries and translates them into the appropriate protocol understood by a given RDF query service. In this fashion, new RDF sources can be made available to Haystack by providing plug-ins.

For network efficiency reasons, we have chosen instead to wrap many of these RDF sources as *caching RDF stores*—ordinary local RDF stores that cache data from another source. In the case of a read-only RDF file on a Web server, the process is straightforward: a blank RDF store is instantiated and the RDF file is imported into it. In the case of a query service, the wrapping process works as follows. When requests for information are made against the store (e.g., by the user interface), they are resolved using the data already in the store (which may produce empty result sets if there are unresolved URIs); meanwhile, in a background thread, unresolved URIs mentioned in the query are resolved against the query service, and the new metadata is added to the store. As new information is incorporated into the store, events are fired, notifying services, user interface components, and other listeners of the fact that updated information satisfying their queries is available. While caching and an event mechanism are not necessary to create a browser, they do provide a useful way to implement asynchronous downloading and rendering of information to the screen.

For example, built into Haystack is a caching RDF store that handles LSID URIs. When a user requests to see information on an LSID-named resource, he or she initially sees a blank page, but as the background thread resolves the LSID, the RDF describing the resource incrementally pops onto the screen (much like images background-loading in a Web browser). In this fashion, users can enter URIs (in this case LSIDs, but metadata-backed HTTP URIs are similarly supportable) they find in papers from the literature, e-mails from colleagues, or maybe one day even advertisements and billboards, and the system takes care of locating the necessary server and retrieving the necessary metadata.

Furthermore, Haystack supports metadata coming from multiple RDF sources at once with the notion of a federated RDF store: a store that exposes an RDF store interface but distributes queries across multiple RDF stores. There is much work in the literature on federating RDF sources together [21] (especially for use in portals [2]), but these efforts are mainly focused towards environments in which a database administrator or other expert is present to set up connections to data sources and to define ontological mappings. We focus on the problem of providing end users with semi-automatic database federation while still giving a fine level of control over which repositories are being used for those who need it, since many users want to know where information is coming from in order for them to form valid judgments on its trustworthiness and usefulness. (The issues of trust and ontological mapping are discussed further in Section 8.) Part of the job of federating data rests with the user interface, which is discussed in Section 4.

Initially, Haystack's root RDF store and the LSID caching store are federated together. For those users who need more control, the Information Sources pane shows users the list of "mounted" RDF sources, i.e., the RDF sources that have been federated into Haystack. Users can easily mount additional RDF stores that speak Annotea's RDF store protocol [9] or RDF files that exist on the Web; support is being added to enable other protocols, such as Joseki [16]. For users who wish to explore developing for the Semantic Web, we feel being able to mount a set of known RDF sources in order to begin browsing and "playing" with the combined data set is an intuitive way to get started.

4. PRESENTING RDF TO USERS

Haystack's user interface has the challenge of providing a sensible presentation to the user given the metadata available to it. As a basic paradigm, we have chosen to center the presentation around one resource at a time, just like the Web browser model. In fact, Haystack's user interface works more or less like a Web browser: users can type URIs into the Go to box or click on "hyperlinks" shown on the screen to navigate to resources. We also provide back, forward, and refresh buttons on the toolbar. Of course, unlike a Web page, a Semantic Web resource has no predefined presentation specification; instead, the metadata describing that resource must be leveraged to generate an intelligent presentation.

Ontologies and schemas are specifications for use by software in determining how to best process metadata written to these specifications. These specifications are incorporated into Haystack when an unknown RDF class is referenced by the resource being viewed (e.g., by an **rdf:type** statement) and the URI of the class resolves to a piece of RDF containing this specification, as described in the previous section. However, typical schemas offer little information regarding the best means of presentation. For example, a calendar can easily be characterized by a DAML+OIL ontology, but a generic display (usually a key-value pair editor or a directed graph display) is unlikely to be intuitive to those unfamiliar with the abstract notions of RDF. We argue that to support appropriate presentation of Semantic Web information to end users, it will be necessary to define an ontology for describing presentation knowledge, such as which are the important properties of a class. We will refer to this ontology as VOWL: the View Ontology Web Language. Haystack uses VOWL in addition to the RDFS and DAML+OIL schema description languages in an effort to bridge the gap between a user's display needs and the underlying data model.

In this section we discuss what presentation guidance Haystack can derive from standard ontological specifications and what else must be provided by VOWL to produce a better presentation for a given resource. This additional presentation knowledge is itself describable in RDF and can be added to the RDF-encoded ontological specifications returned when the class URI is resolved. Rather than giving RDF/XML examples, we have chosen to use high level explanations and diagrams in this section to explain the important concepts in Haystack's user interface; specifics can be found at the project Web site [37].

4.1 Views

There is no single way a generic Semantic Web resource must appear in a browser as there is with an HTML page; in fact, it is often useful to look at a resource in many different ways. As a result, we have abstracted the notion of multiple presentation styles into the concept of *views*. For any resource, multiple views exist; we focus on one particular view in this section, which shows metadata as a segmented form. Other views are discussed later in this paper. Users are given control over which view of a resource they want by means of the *view selector*, seen in Figure 1. More details on Haystack's view mechanism can be found in previous papers on Haystack [1] [3].

4.2 Titles

Perhaps the most basic view of an object is a simple reference to it on the screen by a human-readable name. The ultimate fallback name for any object is its URI, but URIs tend not to be meaningful or memorable for humans. Instead, if **dc:title** or **rdfs:label** properties are provided, Haystack will use one of the values of these properties, giving higher priority to **dc:title**. At the schema level, one can explain how to title all resources of a given class by annotating the class with a **vowl:titleSourcePredicate** property, telling Haystack which property contains a literal suitable for use as the resource's title. For example, in Haystack, the title source predicate for the type **mail:Message** is **mail:subject**.

4.3 The All Information View

The All Information view is the default view shown for resources with non-HTTP URIs if Haystack has no further knowledge of how to display the resource. (As a heuristic, resources named by HTTP URIs are, in the absence of an RDF type specification, shown in a Web browser view.) The All Information view renders together the *lenses* that apply to the types of the resource being displayed. A lens is defined to be a list of properties that make sense being shown together. The reason for defining lenses is that there could potentially be an infinite number of predicate/object pairs characterizing a resource; lenses help filter the information being presented to the user. Lenses are shown as panels that display some fragment of information about a resource. When a new applicable lens that can display further information fragments is defined, the All Information view will automatically incorporate it. An example of a customized version of the All Information view is shown in Figure 2; it contains three lenses specific to flights and two that are broadly applicable.

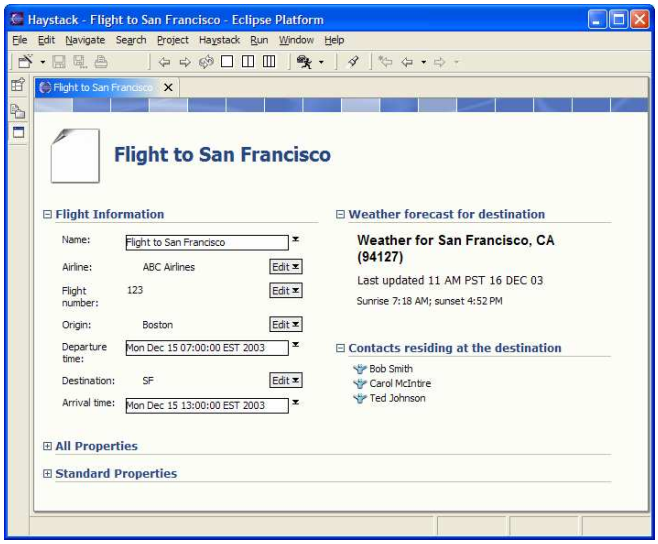


Figure 2. A flight itinerary shown in All Information view.

The three specialized lenses, Flight Information, Contacts residing at the destination, and Weather forecast for destination, are specialized lenses for the Flight type. The Flight Information lens is straightforward to define in that it is simply a list of RDF properties. In contrast, the remaining two flight lenses make use of *virtual properties*—connections between one resource and another that are not materialized in the RDF representation for efficiency reasons. To motivate the need for virtual properties, it is important to bear in mind that a significant amount of Semantic Web metadata originates from relational databases whose schemas were designed with the objectives of consistency and optimized access in mind. On the other hand, a user may also care about properties that are derivable from or are redundant with properties already given by the schema or ontology, such as Contacts residing at the destination. For this reason, Haystack allows virtual properties to be defined and used in the specification of lenses. It is, of course, possible to implement virtual properties using an inference engine; in Haystack, an ex-

tremely limited form of forward-chaining inference is employed to support virtual properties.

4.3.1 Defining Lenses

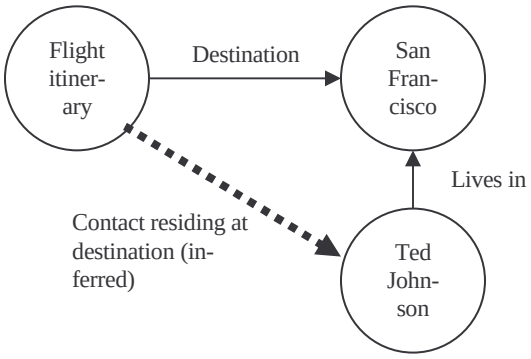
A lens is defined by specifying a (DAML+OIL) list of properties. When Haystack renders a lens to the screen, it makes use of information that likely already exists in the ontology:

- The human-readable names of the properties (e.g., **travel:origin**) are specified by **rdfs:label** properties in the schema, as described in Section 4.2.
- Referring to the screenshot in Figure 2, one notices that some fields have an Edit button and some do not. Properties that are defined to have type **daml:UniqueProperty** are assumed to be single-valued, and Haystack will not display the Edit button in those cases. (Uses for the Edit button are discussed later in Section 6.1.)
- Haystack also uses knowledge of whether the properties being displayed have the **daml:DatatypeProperty** or **daml:ObjectProperty** types to determine whether to show a text field or a list of resource hyperlinks (displayed using their titles according to the rules given in Section 4.2), respectively.

For Haystack to know which lenses are available for a given class, the **vowl:hasLens** property of the class is used. When the All Information view is displayed for a resource, it queries all RDF sources for the RDF types of the resource and accumulates all of the lenses that correspond to those types. Two special lenses are implicitly associated with all types: the All Properties lens shows every single property and value associated with the resource, while the Standard Properties lens shows a bundle of properties from the Dublin Core ontology, such as title and author/creator, which seems to be useful for a wide variety of types.

4.3.2 Defining Virtual Properties

Contacts residing at the destination and Weather forecast for destination are examples of lenses involving only one property, here a virtual property. Virtual properties can be defined in a number of ways, but perhaps the simplest is in terms of an RDF graph matching pattern. For example, the Contacts residing at the destination virtual property is defined by the thick dotted line as follows.



The Weather forecast for destination property can be thought of as having a literal value that is defined in terms of the result of a Web Services invocation, which involves scripting. Advanced support for defining virtual properties and rendering formatted text is described in the Haystack documentation on our Web site and in other papers [1] [7] [37].

5. COLLECTIONS

There is a diverse spectrum of different kinds of pages on the Web. Even so, one notices that a large fraction of Web page real estate is devoted simply to the task of listing links to other Web pages. Good examples include search results, product listings, taxonomical classifications (e.g., the Open Directory Project [19]), individuals' publication lists, and RSS news feeds. We see this as a social phenomenon not specific to HTML and anticipate that the Semantic Web will be similarly populated with purposefully-gathered *collections* of related resources. In fact, some of the above examples from the existing Web today are already part of the Semantic Web, as the Open Directory and RSS 1.0 [22] Weblog feeds are both described in RDF.

As evidenced by the appearance of search result, bookmark, and history panes in Internet Explorer and Mozilla, collections deserve baseline support built into the browser. But how should they be displayed? One finds many examples of a user wishing to see the same underlying collection in different ways (e.g., a product listing sorted by price or rendered as a page of thumbnail images, etc.). At present, such multiple-view functionality is offered at the server end or by Web-site-specific JavaScript. Individual users' Web pages, often containing collections, do not even have the domain-specific multi-view support offered by more sophisticated Web sites: one will likely not be able to view them as hierarchies, in multiple columns, or sorted by date. We can correct this problem on the Semantic Web by using the RDF description of a collection to generate multiple different views from within the browser.

5.1 Ontological Specifications for Collections

From an ontological standpoint, a collection—a resource that represents a set of resources—is conceptually simple: one relationship, often called "member-of", ties a collection to its members. However, there are many variations on this pattern. For example, one can also choose to model collections with added structure, as DAML+OIL and OWL do with Lisp-style lists. One also has many choices on the kinds of restrictions placed on membership in a given collection class. One simple baseline case is to have no restrictions placed on the kinds of resources that could belong to a collection (in DAML+OIL, we would say that the **rdfs:range** of the membership predicate is **daml:Thing**). The resulting arrangement—the heterogeneous collection—turns out to be extremely powerful, as we discuss later. Haystack predefines a heterogeneous collection class called **hs:Collection** and an unconstrained membership predicate **hs:member**. Naturally, more specialized, homogeneous collections are useful for other applications, and these are also supported by the system.

5.2 Browsing Collections

Haystack offers many collection visualizations, allowing the user to choose one depending on the particular task at hand. The default view simply shows a list of the resources in the collection. The calendar view shows the resources on a daily calendar view, using date/time information encoded in the **dc:date** predicate. (In Haystack, a calendar is not a specific type; rather it is a collection that happens to contain resources with date/time information.) The photo album view shows the members of the collection as thumbnails, or as tiles with the name of the resource inside if a thumbnail is unavailable (e.g., for resources that are not images). Finally, the Explore relationships between collection members view shows property relationships among the resources that belong to the collection as a labeled directed graph; the arcs shown are the ones described by some appropriate lens. For example, Figure 4 shows a collection of

people in terms of the Human resources lens, a lens that includes the "manages" property.

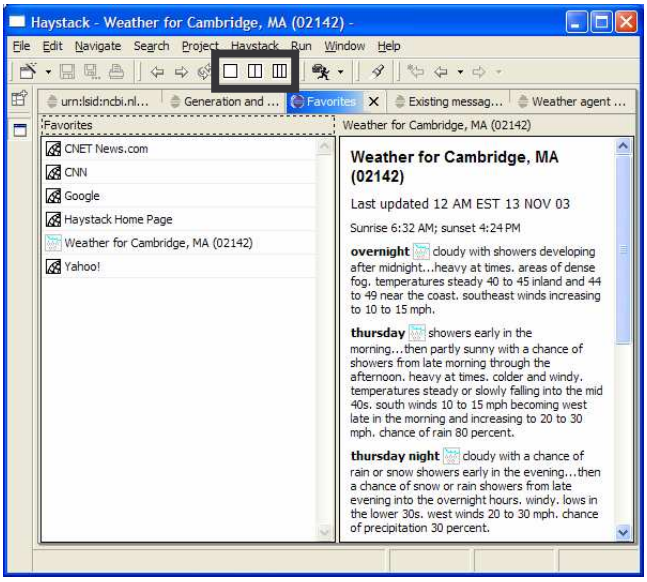


Figure 3. Browsing a collection in double pane mode (thick box indicates preview pane controls).

Collections are so pervasive that they appear in many slightly different representations—sets, ordered lists, directories, etc. In a big picture sense these are subclasses of the collection class. We want a way to ignore the superficial differences in representation and focus on their fundamental "collectionness", letting users manipulate them as collections. Haystack provides the developer with tools for explaining how a given class can be treated as a collection; this allows our collection views to be applied to things like DAML+OIL lists and file directories.

Additionally, the browser can be placed into one of three modes to facilitate browsing through the members of a collection. These modes are enabled by means of three buttons on the toolbar, indicated in Figure 3. The single pane mode is the default and mimics the behavior of the Web browser: when users click on hyperlinks, the pane navigates to the new page. The double pane and triple pane modes allow a user to keep a collection on screen while viewing members of the collection at the same time. (Higher numbers of panes can be set up through a dialog box.)

Figure 3 shows an example of browsing a collection (the user's favorites collection) in double pane mode. When the user clicks on the "Weather for Cambridge, MA (02142)" link on the left, instead of the entire window navigating to the weather forecast, the second pane on the left is used to display it. Similarly, in the triple pane mode, hyperlinks activated in the second pane appear in the third pane, and so on. Incidentally, it is worth noting that Haystack's multi-pane functionality has been extremely useful for browsing collections, but any resource can be browsed with this mechanism. For example, if a user clicks on a link inside of the All Information view of a resource when in double pane mode, the target link's resource will appear in the second pane.

5.3 Collections as a Bookmarks Facility

In addition to downloading Web pages and rendering HTML to the screen, Web browsers also play an important but often overlooked role in helping users personalize portions of the Web most important to them [10]. All of the major Web browsers include bookmark facilities (of varying degrees of quality) that let users group related pages into labeled, usually hierarchical containers called folders. Some browsers let users export these bookmark hierarchies into new HTML pages; some, such as Lynx, store users' bookmarks in HTML files to begin with. It is not uncommon to find homepages whose main purpose is to hold nicely formatted bookmarks. The important point here is that the generic notion of making collections of Web resources is fundamental both to the philosophy of having a Web and to a user's own ability to keep track of portions of the Web of personal interest.

Users can create their own local heterogeneous collection resources by using the Create collection operation. These collections can be used to group resources together based on some commonality such as project, task, RDF type, similarity, or perhaps even a trait not easily articulated but highly intuitive to the user [12]. Furthermore, users can drag and drop items they encounter while browsing into collections; the metadata associated with the collection is stored in Haystack's root RDF store.

There is an additional important usability benefit to modeling collections (including bookmark listings) in RDF. In such a model, there are no infrastructural reasons for prohibiting membership of a resource in more than one collection at once. In some folder systems, such as the Windows file system or many e-mail and bookmark managers, it is difficult to manage bookmarks that are placed into multiple collections at once. But this is primarily a problem with the view rather than the underlying data model. User studies have shown that there is significant value in being able to file resources into multiple collections at once rather than having to ambiguously choose between collections [12]. Thus, in Haystack we provide views supporting easy multiple categorization that can be applied when appropriate [11], as well as other more standard views that emphasize the hierarchical/taxonomic aspects of collections. Each is useful at different times, and each can be applied to the same collection.

6. SEMANTIC WEB SERVICES

Another large part of the Web consists of form-driven services that let users submit requests to Web servers. Like most Web pages, these service forms are largely meant for human consumption. Additionally, services (e.g., purchase an item) are usually served off the same Web site as the data meant to be consumed by that service (e.g., items being sold on an e-commerce Web site). Using a service from one Web site on data from another involves a lot of manual effort, such as copying and pasting part numbers, unless the two Web sites have a back-office arrangement in place.

With the growth of Web Services, many of these HTML form-driven services are being exposed in a machine-accessible fashion [27]. As with content, we believe that services being described in a machine-readable fashion can play a role in improving the user experience. At present, standard Web Services descriptions, such as those in WSDL format [17], only provide relatively low-level details on how to connect to services. For example, a service that purchases an item may be described as accepting two string parameters—the expectation is that a developer, writing tools that access the service, will read documentation describing the semantics of how the service should be invoked. Some researchers have begun to consider the possibility of agents accessing Web Services autonomously. These services will need a more semantic description of the service parameters, e.g., that the two parameters above are a product ID and a credit card number. Services that are characterized in terms of what they do and what kinds of resources they operate on, rather than in

terms of the low level data types involved, have been dubbed Semantic Web Services [23].

As with content, we believe that services described in a machine-readable, presentation-free fashion can also play a role in improving direct human interaction with information. When services are marked up with semantics, we can build interfaces that help individuals locate the appropriate services to invoke for a given task, that help users fill in the necessary arguments to the services, and that support naïve-user customization of the services for the users' own purposes.

There are only a few drafts of standards available for describing Semantic Web Services, such as DAML-S [24] and OWL-S [25], none of which have been adopted formally by any standards body at the present time. As a result, Semantic Web Services are even scarcer than the Semantic Web metadata and resources they are designed to consume. Nevertheless, we wish to characterize how services would be incorporated into a Semantic Web browser, given the importance of the analogous HTML form-based services to the existing Web.

Although not many Semantic Web Services exist yet "in the wild", all of the actions that a user can take in Haystack, such as menu commands, are actually implemented as a kind of single-method "mini Web Service" called an *operation*. Operations are pieces of functionality that take parameters and return sets of values, and they are used in Haystack to implement commands in the user interface, such as "e-mail a link to the current resource". Like Semantic Web Services, operations have parameter specifications constrained by RDF types rather than XML Schema types. For example, the "e-mail link" operation might be configured to accept one parameter of any type (the resource to send) and another that must have type **hs:Identity** (e.g., a person or an e-mail agent).

6.1 Invoking Operations

One benefit to semantically marking up operations is that we can use the markup to help users invoke them properly. When a user starts an operation, Haystack checks to see if the resource currently in view unambiguously satisfies any of the operation's parameter types. If there are unresolved parameters or the resource type checks against multiple parameters, Haystack displays a form prompting the user for parameters. This form is actually a special view of an *operation closure* resource configured to display a lens constructed from the parameters to the operation. An operation closure is a resource that represents an operation invocation and has, as properties, the parameters to that operation. The RDF that is used to describe the current state of the operation closure is stored in Haystack's root RDF store, as is metadata for user-created collections. Users can fill in the form by dragging and dropping items from other views or by using the Edit button, which exposes an appropriate interface for collecting the kind of resource or literal needed for that parameter. When the user is done, he or she can click the OK button to invoke the operation.

There are many benefits to representing operation invocations as resources, which are described in another paper [4]. One particular feature we highlight here is our ability to reuse the existing browsing infrastructure to expose forms for parameter collection. When a service is described semantically, a form for accepting parameters from the user can be constructed automatically by Haystack. Note that the appearance of the form can be customized, as discussed in a related paper [1].

6.2 Finding Operations to Invoke

On a Web page describing some item, it is not uncommon to find links to services that can be used on that item. For example, a product might contain a "buy" button that activates a service and initiates a commercial transaction. Similarly, we want to show users what Semantic Web Services are available given the current resource being displayed.

With semantically marked up operations, we can automate the same process in Haystack. Haystack exploits annotations that declare which operations can be invoked on which data items. To show the available services, Haystack exposes the Commands pane, seen in the top left of Figure 1. (In actuality, the Commands pane is simply a lens that has been docked to the left hand side.) Here, the Search using NotQuiteABlast service (in this case, an operation) is shown because it has a parameter that type-checks against the mRNA resource being viewed. Additionally, users may right-click on objects on the screen (e.g., titles of resources shown in lenses) and see context menus listing the operations and services appropriate to those objects. In this sense, the Commands pane is simply a permanently open context menu bound to the current object.

The set of services relevant to a given item is, of course, a collection, to which one can bring to bear all of Haystack's collection browsing capabilities. Menus are a convenient lightweight collection display that are effective when a user basically knows what they want to do. A user in need of more guidance, however, may choose to browse to a full-screen view of the collection of available operations, where they might find descriptions of what each operation does and what parameters it takes. The "Starting Points" display, the first view encountered when a user starts up the system, is essentially such a view.

6.3 Customization

Another important benefit of our first class representation of operations is that it lets users customize their operations. For example, since menus are simply collections, users can employ collection management tools to add operations and otherwise rearrange or create menus as they see fit.

A deeper type of customization is the derivation of new operations from existing ones at runtime. Users are able to save an in-progress operation closure and turn it into a new operation by selecting the option from the user interface continuation's context menu that instructs the system to bind the state of the current operation together with the already specified parameters [4]. One benefit of this technique is its ability to allow users to create specialized operations suited for users' own purposes. For example, if a specific product ordering service is frequently invoked with one or more fixed parameters (e.g., account number, shipping method, etc.), then the user can easily create a custom operation using this technique that has these parameters pre-specified.

6.4 Accessing Remote Web Services

As Semantic Web Services standards become finalized, we anticipate incorporating support for making Semantic Web Services act as remote operations. As with other remote invocation systems, problems such as marshalling resources and literals into specific data formats and dealing with object identity must be dealt with. One benefit to Semantic Web Services, when used in conjunction with LSIDs or resolvable HTTP URIs, is that resources can simply be passed by reference, since the contents of the resource can be resolved separately by the receiving end, simplifying marshalling. In the end, the idea is to use the operation support in Haystack to give

users the ability to invoke Semantic Web Services directly on the resources they are browsing within the same environment without the need for specialized clients.

When we consider remote services instead of those already installed in Haystack, we must also solve the problem of finding those services. The fragment of RDF that is retrieved from the HTTP or LSID server when the resource is originally resolved can include references to relevant services. However, the authoritative HTTP or LSID server is not likely to return information about services that are not hosted by the party controlling that server. One could imagine there being RDF sources that act as Semantic Web Services directories at some point; a user could query such a service (or arrange for their browsers to do so automatically). On a side note, such RDF sources could themselves be Semantic Web Services [20]; mounting such a source could be as simple as typing in its well-known URI, letting the system download the service's metadata, and invoking the "mount" operation on it.

If the user were instead interested in locating the service before specifying the data to act on, then a client such as Haystack could be used more directly to browse the space of services. Services are of course themselves resources, and their URIs could potentially be typed in to browse to service descriptions. Another possibility is to have a collection named by a resolvable URI that contains a set of useful services. As the user browses forward to services of interest, their descriptions would in turn be downloaded into the system.

7. PUTTING IT TOGETHER

To see how the various elements of our Semantic Web browser work together to enable users to browse the Semantic Web, let us return to the screenshot in Figure 1. The screen shown is a result of the user typing in an LSID URN, perhaps obtained from a paper he or she was reading, into the Go to box. As a result of being told to navigate to the resource named by that LSID, Haystack performed an RDF query to determine the types specified for the resource. Initially, the system knew nothing about the resource, so the query returned an empty result set, prompting Haystack to show the Standard Properties and All Properties lenses. The All Information view also registered for events to find out when new information about the resource's RDF types entered the system.

Meanwhile, in a background thread, the LSID cache store picked up on the query containing an LSID, found that the LSID had not been resolved, extracted the authority name from the LSID, and looked it up using DNS. The LSID cache store then contacted the server using the resolved hostname via the LSID Web Services protocol and requested the RDF metadata associated with the LSID. The returned metadata, which included a specification of the resource's RDF type, were added into the cache store, causing an event to fire to the All Information view.

The view reacted to this event by finding the known lenses for the newly discovered types of the resource. Four lenses, Sequence Summary, Sequence, External Reference, and Pubmed, were discovered and rendered to the screen. These lenses are built into Haystack, though ultimately they might be retrieved by an appropriate search of the Semantic Web, so long as the RDF types are named by resolvable URIs and the metadata returned upon resolution of RDF class resources contain links to lenses (which could also be named by resolvable URIs).

Each lens in turn contained a list of properties to be queried, so the lenses queried for the values of these properties of the resource. In the case of the Sequence Summary and Sequence lenses, these were literal values, which were immediately shown. In the case of the External Reference and Pubmed lenses (Pubmed is the name of a literature database, and this lens shows a list of referenced publications), resources named by LSID were found. (The External Reference lens is collapsed in the screenshot.) The user interface queried for the **dc:title** and **rdfs:label** properties (there is no **vowl:titleSourcePredicate** for mRNA sequences), but found nothing, so it displayed the raw URIs for these resources. Again, the user interface registered to be notified when information on the **dc:title**

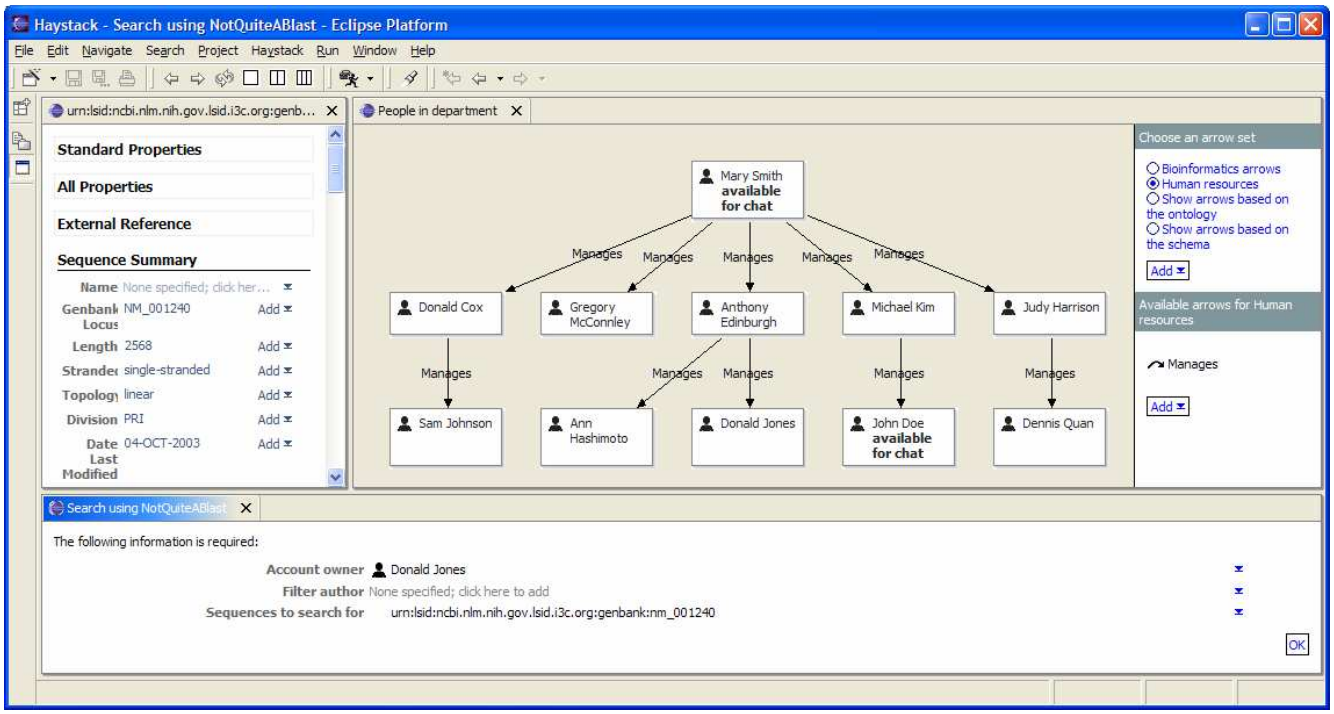


Figure 4. Form for collecting information for Semantic Web Services invocation (bottom).

or **rdfs:label** properties of these resources was added. Also, the LSID cache store's background thread noticed queries being made on unknown LSIDs—potentially with different authority names from the original LSID—and resolved them. Events were fired as the **dc:title** property was found for each of the resources, and the user interface updated the display with the human-readable titles. In this way, the display incorporated many aspects of the resource of interest, some of which depended on information that came from various other stores.

The user, noticing the rather large list of referenced publications, chose to collect a subset for future research. He or she invoked the Create collection operation from the side pane's context menu, creating a new collection named "Interesting Stuff". The user then dragged some interesting publications into this new collection. As is the case with any other collection on the Semantic Web, this personal collection can be viewed in a number of different ways, including as a relationship graph to see, for example, if the papers had cited each other. After creating the collection the user could invoke the File Away operation that places the just-created object into their bookmarks (or another) collection for later use.

At this point, the user could elect to do one of a number of different activities. One possibility is to invoke a Semantic Web Service on this mRNA sequence, such as BLAST, a search engine commonly used in the bioinformatics community to locate similar sequences [35]. Such a service would be visible from the left hand pane. The user could click on the BLAST command to invoke the service (we have simulated this with a mocked up Semantic Web Service called NotQuiteABlast). If other parameters are needed, the user would be prompted for them with a form such as that shown in Figure 4. A user could, of course, browse to other resources to find useful information and use drag and drop to directly fill in parameters, as is shown in the example. After invoking the service, a collection of similar sequences looking like the one the user created him or herself on the right side of the screen would appear. This is in contrast to how services such as BLAST are made available from the Web today; in general, either an identifier or a DNA sequence string would need to be copied and pasted manually into the form. Furthermore, the results returned would likely not be persistent unless the user copied and pasted the results Web page into a file.

8. DISCUSSION AND FUTURE WORK

The Semantic Web browser we have outlined in this paper lays the foundation for the construction of clients that can produce hyperlinked presentations of resources from RDF metadata originating from multiple sources. There are a number of additional issues we have yet to resolve but that deserve further discussion, including possible strategies for enabling users to contribute to the Semantic Web, inference, trust, and ontological mismatch. We give some comments on these problems below.

Tim Berners-Lee's original vision was of a Web in which users were not only able to easily browse interconnected Web pages but also able to contribute new content to the Web. Admittedly, some Web browsers support HTML composition and expose functionality for uploading pages to Web servers, but few users take advantage of these features. In looking towards the Semantic Web, we feel there is a renewed opportunity to create new opportunities for user contribution. One factor making it easier to publish to the Semantic Web than the original Web is the fact that the minimum unit of publication is simply a fact, such as one providing a person's name, as opposed to an HTML document, which may require spellchecking, proofreading, formatting or layout. The harder problem is finding ways for users to input RDF metadata. Haystack includes support for editing as well as browsing the metadata of resources, as was hinted at in the earlier sections on collections and Semantic Web Services. This topic is covered in more detail in other papers [1] [3].

In terms of publishing the metadata itself, Haystack provides support for sending RDF fragments between users [28] and for exporting RDF to files, but not yet for posting RDF fragments to some shared site automatically. One possibility is to have Haystack host an HTTP or LSID server and to serve out metadata to others, but this approach may not work for mobile users. However, the more difficult problem is one of privacy: even if we could easily publish the metadata produced by every individual to the Semantic Web, we must still find ways of describing which metadata is public and which should not be readily disseminated.

On a related note, one relatively esoteric community where one observes a group of Web users both contributing and consuming content is the Web design community. Web sites devoted to the issues of Web site design often include galleries of stylesheets, HTML templates, widgets written in JavaScript, and images. Because of the way in which user interface presentation style can be separated from metadata content on the Semantic Web, one can imagine sites containing descriptions of views, lenses, and other reusable elements becoming more mainstream, since the average user would be able to take advantage of these components on his or her own.

On the issue of inference, Haystack does not itself include support for basic RDFS, DAML+OIL or OWL inference, but it is entirely feasible to attach an inferential RDF store to Haystack, such as the Jena RDF store [18]. Inference is one possible means for resolving ontological mismatch—the problem that occurs when two data sources that are to be used together are based on ontologies that use different names for similar or identical concepts. Ontological mappings [14] could be downloaded from servers in much the same fashion as views and lenses. However, another approach to addressing ontological mismatch is to simply provide lenses and other user interface elements that make items from different ontologies at least appear the same to the user. This is the approach we discussed earlier to dealing with multiple forms of collections.

With respect to trust, we have taken the simplistic approach adopted by Web browsers, which is in essence to trust all information that enters Haystack. But here an important difference arises between the current and Semantic Webs. With a real Web page, a user is better able to judge the trustworthiness of content because there is often a significant body of text, so the facts embedded in the page can be judged with respect to the context of the whole page [13]. Also, the entire Web page tends to come from a single source server whose trustworthiness can be considered. On the Semantic Web, the fact that an RDF presentation may incorporate facts from multiple sources creates more difficulty. As a first cut, the user's choice of which RDF sources to "mount" determines a subset of trusted RDF. At a finer grain, we are considering the idea of a belief layer placed between the underlying RDF sources and the user interface that would allow only those RDF statements signed by a party trusted by the user to pass through to the user interface.

9. ACKNOWLEDGMENTS

We would like to thank Mark Ackerman and Joseph Latone for their feedback on this paper. This work was supported by the MIT-NTT collaboration and the MIT Oxygen project.

10. PREFIXES USED IN THIS PAPER

dc: http://purl.org/dc/elements/1.1/

rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#

rdfs: http://www.w3.org/2000/01/rdf-schema#

daml: http://www.daml.org/2001/03/daml+oil#

mail: http://haystack.lcs.mit.edu/schemata/mail#

travel: http://haystack.lcs.mit.edu/schemata/travel#

vowl: http://haystack.lcs.mit.edu/schemata/vowl#

hs: http://haystack.lcs.mit.edu/schemata/haystack#

11. REFERENCES

[1] Quan, D., Huynh, D., and Karger, D. Haystack: A Platform for Authoring End User Semantic Web Applications. Proceedings of ISWC 2003.

[2] Stojanovic, N., Maedche, A., Staab, S., Studer, R., Sure, Y. SEAL: a framework for developing SEmantic PortALs. Proceedings of the International Conference on Knowledge Capture October 2001.

[3] Quan, D., Karger, D., and Huynh, D. RDF Authoring Environments for End Users. Proceedings of Semantic Web Foundations and Application Technologies 2003.

[4] Quan, D., Huynh, D., Karger, D., and Miller, R. User Interface Continuations. Proceedings of UIST 2003.

[5] Werner, P., Liefield, T., Gilman, B., Bacon, S., and Apgar, J. URN Namespace for Life Science Identifiers. http://www.i3c.org/workgroups/technical_architecture/resources/lsid/docs/LSIDSyntax9-20-02.htm.

[6] The Eclipse Project. http://www.eclipse.org/.

[7] Haystack project home page. http://haystack.lcs.mit.edu/.

[8] Guha, R., McCool, R., and Miller, E. Semantic Search. Proceedings of WWW 2003.

[9] Kahan, J. and Koivunen, M. Annotea: an open RDF infrastructure for shared web annotations. Proceedings of WWW10.

[10] Abrams, D., Baecker, R., and Chignell, M. Information Archiving with Bookmarks: Personal Web Space Construction and Organization. Proceedings of CHI 1998.

[11] Quan, D., Bakshi, K., Huynh, D., and Karger, D. User Interfaces for Supporting Multiple Categorization. Proceedings of INTERACT 2003.

[12] Lansdale, M. The Psychology of Personal Information Management. *Applied Ergonomics* 19 (1), 1988, pp. 55–66.

[13] Lin, J., Quan, D., Sinha, V., Bakshi, K., Huynh, D., Katz, B., and Karger, D. What makes a good answer? The role of context in question answering systems. Proceedings of INTERACT 2003.

[14] Dertouzos, M. The Unfinished Revolution. New York, NY: HarperCollins, 2001.

[15] Berners-Lee, T., Hendler, J., and Lassila, O. The Semantic Web. *Scientific American*, May 2001.

[16] Joseki. http://www.joseki.org/.

[17] Web Services Description Language 1.1. http://www.w3.org/TR/wsdl.

[18] Jena Semantic Web Toolkit. http://www.hpl.hp.com/semweb/jena.htm.

[19] Open Directory Project. http://www.dmoz.org/.

[20] Richards, D. and Sabou, M. Semantic Markup for Semantic Web Tools: A DAML-S Description of an RDF-Store. Proceedings of ISCW 2003.

[21] Halevy, A., Ives, Z., Mork, P., and Tatarinov, I. Scaling up the semantic web: Piazza: data management infrastructure for semantic web applications. Proceedings of WWW 2003.

[22] RDF Site Summary 1.0. http://web.resource.org/rss/1.0/.

[23] Trastour, D., Bartolini, C., and Preist, C. Semantic web support for the business-to-business e-commerce lifecycle. Proceedings of WWW 2002.

[24] DAML Services. http://www.daml.org/services/.

[25] OWL-S. http://www.daml.org/services/owl-s/1.0/.

[26] Handle System. http://www.handle.net/.

[27] W3C Web Services Activity home page. http://www.w3.org/2002/ws/.

[28] Quan, D., Bakshi, K., and Karger, D. A Unified Abstraction for Messaging on the Semantic Web. Proceedings of WWW 2003.

[29] Beckett, D. The design and implementation of the redland RDF application framework. Proceedings of WWW 2001.

[30] Noy, N., Sintek, M., Decker, S., Crubezy, M., Ferguson, R., and Musen, M. Creating Semantic Web Contents with Protege-2000. *IEEE Intelligent Systems* 16 (2), 2001, pp. 60-71.

[31] Dzbor, M., Domingue, J., and Motta, E. Magpie: towards a semantic web browser. Proceedings of ISWC 2003.

[32] Saltzer, J., Reed, D., and Clark, D. End-to-end arguments in system design. *ACM Transactions on Computer Systems* 2 (4), November 1984, pp. 277-288.

[33] RFC 2616. Hypertext Transfer Protocol -- HTTP/1.1. http://www.w3.org/Protocols/rfc2616/rfc2616.html.

[34] W3C Web Accessibility Initiative. http://www.w3.org/WAI/.

[35] Altschul, S., Gish, W., Miller, W., Myers, E. and Lipman, D. Basic local alignment search tool. *Journal of Molecular Biology* 215, pp. 403-410.

[36] NewsIsFree. http://www.newsisfree.com/.

[37] Haystack developer documentation site. http://haystack.lcs.mit.edu/developers/.

[38] Wilkinson, M. and Links, M. BioMOBY: an open-source biological web services proposal. *Briefings in Bioinformatics* 3 (4), 2002, pp. 331-341.