Haystack's User Experience for Interacting with Semistructured Information

David Huynh
MIT Artificial Intelligence Laboratory*
dfhuynh@ai.mit.edu

Dennis Quan
MIT Artificial Intelligence Laboratory*
dquan@media.mit.edu

David Karger
MIT Laboratory for Computer Science*
karger@theory.lcs.mit.edu

ABSTRACT

A major stumbling block in the use of information management tools such as desktops, email managers, file browsers and web browsers is the mismatch between the user's mental model of information objects and the system's implementation model. While a user may look at a name on the screen and think of a person, the system may consider it dead text. A user who wants to remember that a certain email message was sent by a distant cousin may be foiled by the fact that email management and family relationship records are the domains of two separate applications that do not talk to each other.

The goal of our Haystack system is to reduce the mismatch between the user's and the system's models of information, creating a *user-object*-oriented interface. To achieve this goal, we model the user's data uniformly and at fine levels of granularity using the Resource Description Framework (RDF). We demonstrate how this internal system model of data can be reflected out to the user interface so that the user can manipulate UI representations of the information in a uniform manner that makes sense to the user. We explain how RDF itself can be used to build a UI framework that performs such reflection systematically. Furthermore, we propose that operations on and mechanisms for organizing RDF data be modeled in RDF as well, making the whole system reflexive and generic. We envision that the user's everyday use of our system for interacting with the information that he or she cares about will generate metadata and contribute to the global pool of RDF data. The Semantic Web can readily leverage the information collected by Haystack that matters to the end users.

Categories and Subject Descriptors

H.5.2 [**Information Interfaces and Presentation**]: User Interfaces – *graphical user interface, interaction styles, prototyping, user-centered design*.

General Terms

Design, Human Factors.

Keywords

Context menu, direct manipulation, drag and drop, information management, metadata, modal, modeless, object oriented, RDF, Semantic Web, uniformity, user interface, user experience, view binding.

* 200 Technology Square
Cambridge, MA 02139 USA

WWW 2003, May 20-24, 2003, Budapest, Hungary.
ACM xxx.

1. INTRODUCTION

In the current software paradigm, several applications are needed to store, view, edit, and manage different types of information, with each application tailored toward one or a few types of information. Cooperation among different applications is limited. For instance, the user cannot specify that a favorite recipe (abridged and stored in a "recipe database" software) comes from a cookbook (indexed by a "personal library" software), which has been lent to a friend (whose contact information is managed by an "address book" software). Because each application focuses on one domain of information (e.g. recipes, books, contact info), cross-domain relationships such as those in the example are hard to express and record on the computer.

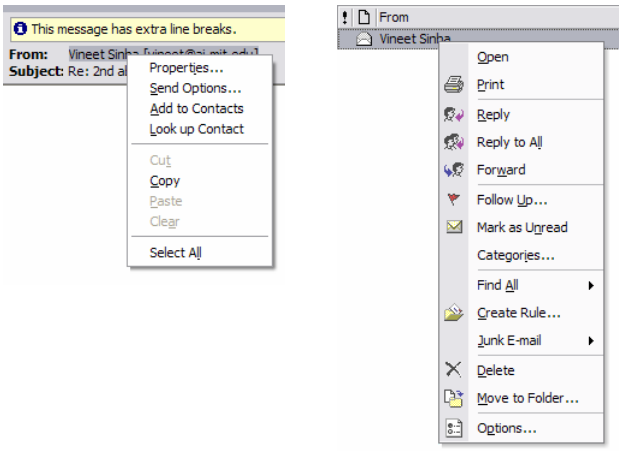
Even in dealing with only one application which specializes on a limited number of domains, certain types of information in those domain are still not expressible in the application's schemas. For instance, since Microsoft Outlook 2000 manages meetings, one would expect all things and tasks related to meetings to be in the domain of Outlook. In particular, one would expect the ability to store a map of a particular meeting location, most logically by attaching that map to the "object" representing that location. Unfortunately, meeting locations are not first-class objects in Microsoft Outlook—they are just text fields of meeting objects. Consequently, to increase the chance of finding the map where the user would need it, the user would have to paste the map into the body of every single meeting held at that location. The granularity at which the software models the information is not fine enough to express the information need of the user in this scenario: the first-class object types in Microsoft Outlook are simply too coarse.

These problems arise because the applications fail to model information according to the user's mental model. While a user may look at a name on the screen and think of a person, the system may consider it dead text. Why can one spell check only the body of an e-mail message but not its subject, or the text of a document but not its filename? Why do the context menus invoked from two different occurrences of a person's name in the UI list two different sets of operations that do not seem to correlate with the contexts in which the name is displayed (see Figure 1)? Why is drag and drop supported in some cases and not others?

1.1 RDF-Based Data Modeling

The application-centric paradigm on which most of today's software has been built does not satisfy the user's need. First, information is not modeled according to the user's mental model of his or her information: the user often expects to interact with information at various levels of granularity when the software can only handle what it considers first-class objects in its schemas. Second, the user's data is segregated by application barriers in such a way that metadata describing relationships between all of the user's

data cannot be easily added. In order to address these issues, information must be modeled independently of applications and modeled in schemas expressive enough to match the user's model of information. Haystack [7], our information management platform, has adopted the Resource Description Framework (RDF) [1], a core technology of the Semantic Web project [2], for these two purposes. RDF is capable of describing fine-grained semistructured data in highly expressive and extensible schemas: this functionality allows us to better match the user's model of information. Furthermore, by storing all of the user's data in RDF, we break the information free from application specific formats and open possibilities for synergy between different types of information.



(a) **(b)**

Figure 1. Inconsistent context menus invoked from the text strings "Vineet Sinha" in two different views of Microsoft Outlook XP: (a) supports adding the e-mail's author to Contacts while (b) does not.

Because the user's information is modeled at the appropriate levels of granularity, we can support user interface manipulations of such information in the way that the user expects. Furthermore, the single format in which data is stored, namely RDF, enables information of different types to be used together in more flexible ways that prove useful and intuitive to the user. In this paper, we describe our vision of how the user should interact with semistructured information in Haystack and explain how the use of RDF enables the realization of that vision.

1.2 Interaction Model for RDF Data

First, we adopt the web browser navigation paradigm for allowing users to navigate corpora of annotated data. Each piece of information can be viewed by browsing to it. Using RDF predicates as links, traversing a corpus involves hopping from one piece of information to another. This navigation paradigm puts the focus on information rather than on applications. This information centricity is emphasized by the ability of RDF to model information at fine levels of granularity.

RDF itself enables us to extend this navigation paradigm. Links, modeled by predicates, are labeled and annotated. Annotation on links help improve the user's navigation experience by drawing together more information relevant to each navigation step. The sources of relevant information can also be modeled in RDF, further abstracting away storage and local issues from the user.

Second, we propose a system-managed binding between the presentation of information in the user interface and the metadata concerning that information stored in the system beneath. This binding allows the semantics of the information to be exposed systematically throughout the UI and facilitates system support for direct manipulation mechanisms such as context menus and drag and drop. Direct manipulation in turn enables the integration of simple operations whose use can be composed to perform complicated tasks. This is in contrast with monolithic applications that provide fixed sets of functionalities whose use, because of application barriers, cannot be intermingled.

Third, we describes several modalities in which the user can organize his or her information. These organization mechanisms are themselves modeled in RDF and made to work on any type of information. They include classification mechanisms through the concept of collections, and annotation mechanisms through various user interfaces.

Finally, we illustrate the whole user experience brought forth by all of the above functionalities through their implementation in our Haystack system.

1.3 Related Work

Many systems have also been built for visualizing graph data. TheBrain.com[1], the latest commercial endeavor, models and presents data as unlabeled, undirected graphs. As a consequence, benefits available from the directedness in the RDF model are not present in the TheBrain.com system. Furthermore, while graph presentations might be suitable for showing related thoughts, they are insufficient to provide a coherent interaction experience for the common users. Few users perceive their information as semantic networks. Graph manipulations are at worst incomprehensible and unusable to novice users, and at best inconvenient and tedious for the expert users in the domain of information management.

There are also systems developed to augment existing information types with more metadata. For instance, the Placeless Documents system [4] adds user-specified properties to documents so that they can be manipulated more directly and managed in a more uniform fashion. However, these systems rarely address the usability issues involved in manipulating information in general.

In addition, since the birth of the Semantic Web, tools have been built for visualizing and editing RDF data and schemas. These RDF authoring tools generally come in three flavors: (1) ontology editors; (2) graph-based representation viewers; (3) schema-specific user interfaces. However, these kinds of tools often focus on collecting data that conforms to some ontology and not on addressing the HCI issues of information collection. Ontology editors such as Protégé [5] and Ont-o-mat [7] provide effective interfaces for allowing ontology modeling experts to enter information according to specific ontologies with a high degree of precision. They are, however, unsuitable for users who simply want to manage information such as e-mail and photographs.

[1] http://www.thebrain.com/.

2. NAVIGATION PARADIGM

Figure 2 shows a screenshot of Haystack's user interface. On the left side of the window is the *start pane*, and on the right is the *content area*. The start pane can contain several items, e.g., the user's instant messaging status, a list of favorite documents or links, a clipboard-like scrapbook, and a list of starting points from which the user can begin a task. In general, the start pane makes available the tools and information that the user currently needs to accomplish some task at hand. Other objects can be dragged and dropped into the start pane.

The content area acts like a web browser. It has a *navigation toolbar* at the top, with the conventional *Back*, *Forward*, *Refresh*, and *Home* buttons, and an *address box*. Immediately below the navigation toolbar is the *title bar*, which shows the title of the object being browsed to as well as a set of commands available for altering the view. In Figure 2, Haystack is depicted displaying a particular information object: the user's *Inbox*. The content area hosts a typical list view containing e-mail messages and a preview pane below in this example.

Any piece of information in Haystack can be viewed by browsing to it, much as one does in a web browser. A list of visited information objects is maintained to allow backward and forward navigation. A navigation can be triggered by clicking on a link or by typing in a Unique Resource Identifier (URI) naming an RDF resource that models an information object. In Figure 2, one can left-click on "Google" in the *Favorites* list shown on start pane to browse to http://www.google.com/. In other cases where left clicks are inconvenient, one can right-click on an item and select the *Browse to* command from the popped up context menu (e.g. the name "Jake Beal" in the selected e-mail message).

As the Web can be surfed by jumping through links, so can RDF data be browsed by following RDF predicates between different RDF resources which model different information objects. In the following subsections, we describe how the navigation paradigm of the Web can be extended with the power of RDF.

2.1 Predicated Links

Hypertext links in HTML are unlabeled. More accurately, they are labeled on a per-instance basis: in one HTML document, the link to its author is labeled "Author: John Doe" while in another document, the link to its author is labeled "Written by Joe Smith." With such inconsistencies in the markup, hypertext links are mostly useful to the human reader and cannot be processed automatically by the computer.

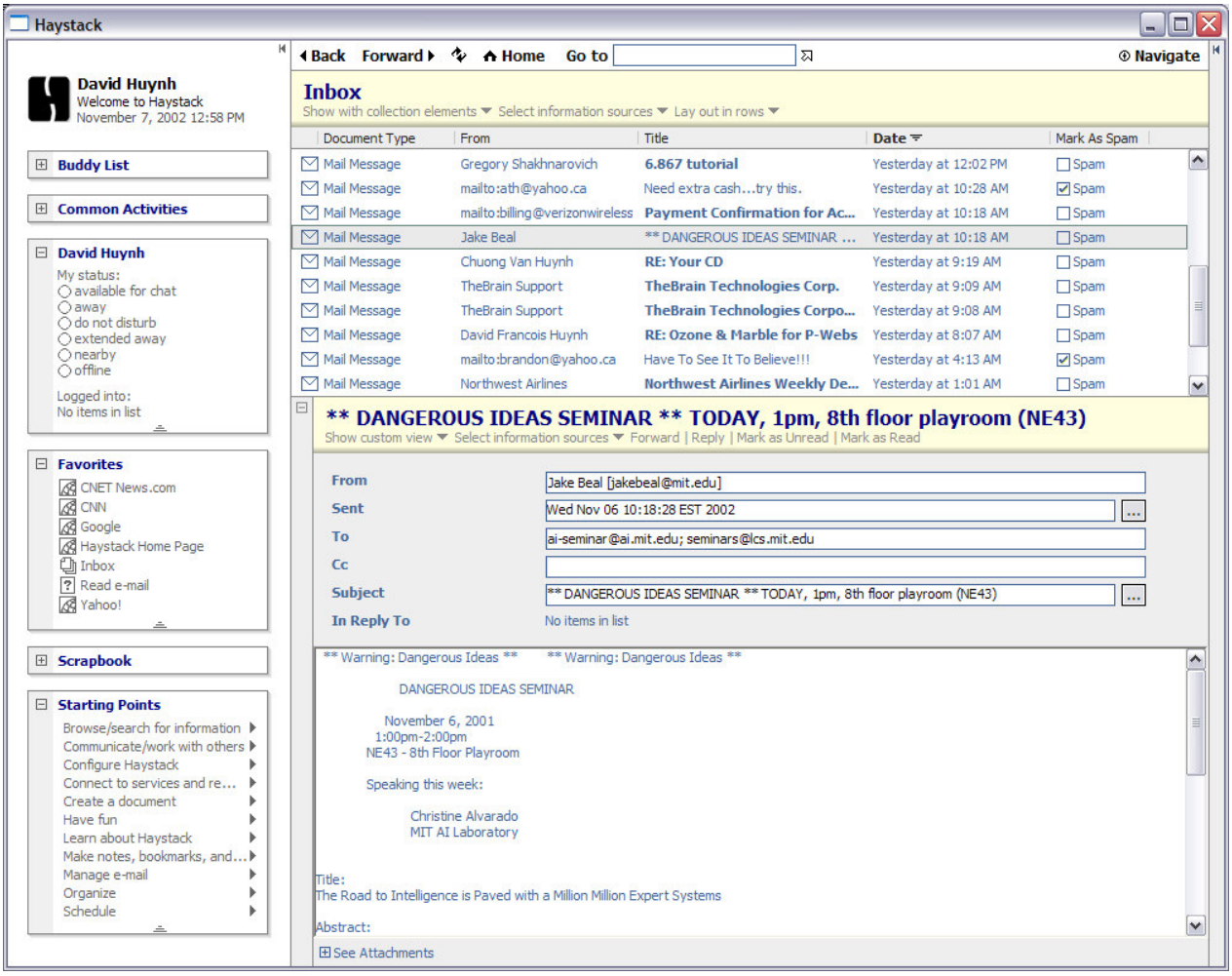


Figure 2. Haystack's user interface

On the Semantic Web, links are modeled with RDF predicates, which are resources named uniquely by URIs. These links can be systematically analyzed and processed in order to improve the user's browsing experience. For instance, when the user hovers the mouse over the link from a document to its author, Haystack can recognize the authoring relationship and pop up a tool-tip listing other documents by the same author. On the other hand, if the user hovers the mouse over the link from a meeting to the same person, who was an attendee in that meeting, Haystack will instead show a list of contributions the person made during the meeting.[2] These types of relevance suggestions can only be made because the links are labeled for machine processing—this is the primary vision of the Semantic Web [3].

2.2 Information Centricity

Because the Web can be surfed by simply clicking on links to the information one wants, the Web appears information-centric to the user. Different types of information (e.g. lecture notes, news articles, maps, business addresses, restaurant menus, etc.) can all be accessed through the same window—that of a web browser. There is an absence of applications: in most cases there is no need for the user to explicitly start up or shut down applications. This is by design: the Web contains content that is independent of operating systems and independent of software applications. To the user, the Web simply contains information. It is this information centricity that makes the Web easy to use.

By adapting the web browser navigation paradigm, Haystack benefits from this information centricity. However, in Haystack, the information centricity is even more apparent. Since RDF can model information at any level of granularity, the user can even navigate to small "miscellaneous" pieces of information. For instance, a meeting location can be viewed by itself rather than as a text field in meeting objects. Whereas information on the Web is packaged into the unit of web pages, information in Haystack can be accessed in any unit that makes sense to the user.

2.3 Storage Abstraction

The Web's navigation paradigm also provides a storage abstraction. Each resource on the Web is identified by a URL. The user only needs to know its URL in order to access it. All the details about the address of the machine on which the resource is stored and the full file path corresponding to that resource have been made transparent to the user.

Similarly, in Haystack, each information object is identified by a URI. Unlike file paths which are storage dependent and change when files are moved, URIs are unique, universal and remain unchanged. The user of Haystack only needs to know the URI of an information object to access it. The system automatically resolves the URI to the storage location of metadata about that URI.

Metadata can also be used to specify the sources of more metadata about a particular URI. This capability is more powerful than network address resolution or even HTTP redirections, as multiple sources can be specified all at once. For instance, when the user seeks information on a person, metadata on that person can be retrieved automatically, with authorization, from several sources including the person's company's databank, student records from the schools he or she attended, and certain government databases.

[2] These heuristics have not been implemented in Haystack. Other forms of assisted navigation are explored in [11].

As illustrated, RDF is powerful in its reflexivity: RDF itself can be used to help improve navigation through metadata encoded in RDF. Similarly, in the next section, we explore how RDF can be used to systematically construct UI presentations for RDF data.

3. SEMANTICS IN PRESENTATION

RDF allows us to model data in accordance with the user's mental model of information. Now that data is structured correctly at the right levels of granularity, this structure needs to be reflected in the presentation of the data so that the user can interact with the data naturally. This is accomplished by providing mappings from each information object to some on-screen representations—*views* of the object, such that the user can readily associate the views with the object.

Each information object is assigned a default view so that when an object is navigated to, its information automatically appears through this view without any explicit user action and without any seeming help from applications. To the user, information appears capable of displaying itself, and displays itself in the most sensible form based on its semantics. This works toward making the UI information-centric.

3.1 Finer-grained Views

A piece of information might not appear only when it is navigated to. It might appear as part of the presentation of another piece of information. For instance, contact objects can appear in the view of a meeting object to show meeting attendance. In this example, the contact objects are displayed in smaller views than the default views presented when those contact objects are navigated to individually (Figure 3).

Haystack makes use of very fine-grained views to present information. In the extreme, a view might consist of only a string of text, rendered to flow inside a paragraph, capable of flowing from one line to another. Smaller views can be easily composed to make bigger views. The whole user interface of Haystack is built by embedding views within one another. These views make every part of Haystack's user interface, no matter how small, seem self-rendering—no application is explicitly called upon to render them. Furthermore, every pixel on the screen can be traced back to the information objects whose views render that pixel. In other words, there is a mapping between the presentational elements in the UI and the metadata in the RDF store.

3.2 Self-updating Views

So that views remain true representations of the information they render, Haystack keeps live the mapping between UI presentation and the underlying information being rendered: when the a piece of information is changed, its UI representation is updated. The mapping becomes a binding. This binding ensures that there is no stale version of data on the screen.

Because small fine-grained views can be easily composed to make bigger views, views can be reused in many places in the user interface. As views are self-updating, true representations of underlying information, they can simply be embedded and left to manage themselves. Consequently, reuse of views is made easy for the UI designers. Through reuse, the presentation of any information object will appear the same in similar contexts where the object is presented. This makes the UI appear uniform and coherent to the user.

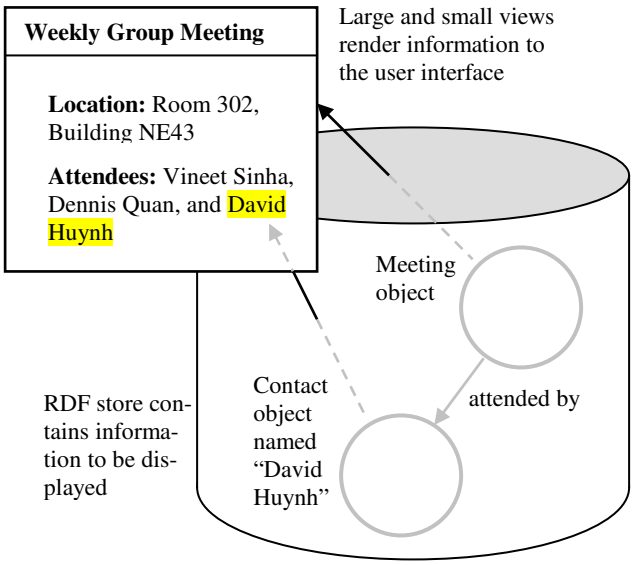


Figure 3. Composition of views

Once again we wish to emphasize the self-sufficiency of RDF. RDF is both the cause and the solution for the generic UI framework of Haystack that we present herein: Since RDF can encode more and finer-grained information types, it calls for a generic framework to present the information. At the same time, it enables the encoding of metadata that helps build such a framework.

4. SEMANTICS OF MANIPULATION

Through the use of nested views, each UI element on the screen can be systematically traced back to underlying information objects whose views render that pixel. Taking advantage of this binding, Haystack lets the user manipulate the UI elements on screen in order to interact with the underlying information objects. In essence, this is system support for direct manipulation, automated and taken to a fine level of granularity. To the user, each UI element, and even each pixel, appears to reflect the information in the RDF store.

4.1 Operations

In order to support direct manipulation, we must define the semantics of available actions that can be taken. In Haystack, information objects can be acted upon by *operations*. An operation is just another resource annotated with the types and characteristics of objects that it can act on, and the code that should be executed to carry out the action.

Because operations are themselves first-class information objects kept in the RDF store, they can be:

- queried, e.g. to find out which operations are applicable on an object;
- edited, e.g. to use newer implementations;
- annotated, e.g. to record their frequency of use;
- bookmarked by the user for regular use;
- organized (see section 5);
- sent to another user;
- searched for by the user;
- etc.

Because Haystack is built on a reflexive data model, Haystack itself is reflexive. In other words, operations can be operated on in the same way other information objects can be. As a consequence, the whole UI appears uniform and coherent, and it is easy for the user to develop a simple mental model of the system.

4.2 Direct Manipulation

A special type of operation is those operations that can invoked by direct manipulation. Direct manipulation in Haystack is currently supported through two mechanisms: drag and drop, and context menus.

4.2.1 Drag and Drop

When a UI element is dragged, Haystack determines the innermost view containing that element, and traces to the underlying object being rendered by that view. That object is the *drag source*.

During a drag operation, as the mouse pointer travels over various UI elements on the screen, Haystack determines the innermost views containing those elements, and highlights those views to provide feedback to the user (Figure 4). When a drop occurs, Haystack detects the UI element receiving the drop and determines the associated underlying object—the *drop target*. Haystack then queries for all operations of type drag and drop operation, which take two arguments, the first applicable to the drag source and the second to the drop target. If there is only one such operation, it is carried out immediately. Otherwise, the user might be prompted to select the desired operation among many; or the system can decide to choose the most likely operation, but allow the user to undo it and select the desired one.

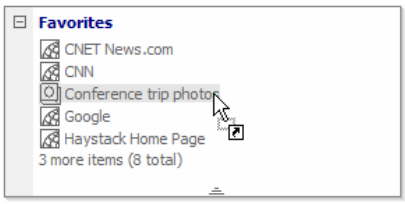


Figure 4. Highlighting of views during drag and drop

Heuristics and machine learning algorithms can be used to detect which object the user wants to drag among those rendered by the hierarchy of nested views enclosing the pixel where the drag is initiated. The drop target can also be detected similarly. The current implementation chooses the object corresponding to the innermost enclosing view as the first-order approximation.

4.2.2 Context Menus

When the user right clicks on a UI element, Haystack determines the associated underlying objects and queries for all applicable operations. Figure 5 shows a sample context menu, which lists all underlying objects whose views enclose the clicked pixel (i.e. both the picture entitled "S_0729" and the album containing that picture). With each object is a pop-out submenu listing applicable operations. Machine learning algorithms can be used to specify the order of the operations, such that the most desirable ones are at the top. The current implementation orders the operations in alphabetical order.

Because context menus are implemented automatically, they are provided everywhere in the Haystack UI. The implication of this uniform support is that the user always knows how to find an operation on any information object being displayed. There is one guaranteed way for listing the available operations.

The avid reader might note that as more operations are incorporated into the system, the context menus might become overcrowded. This is yet another instance of the information overload problem, and a common solution can be used to address all instances. Haystack provides unified mechanisms for organizing and retrieving information of all types, from e-mail messages to operations. Section 5 has a detailed description of these mechanisms.

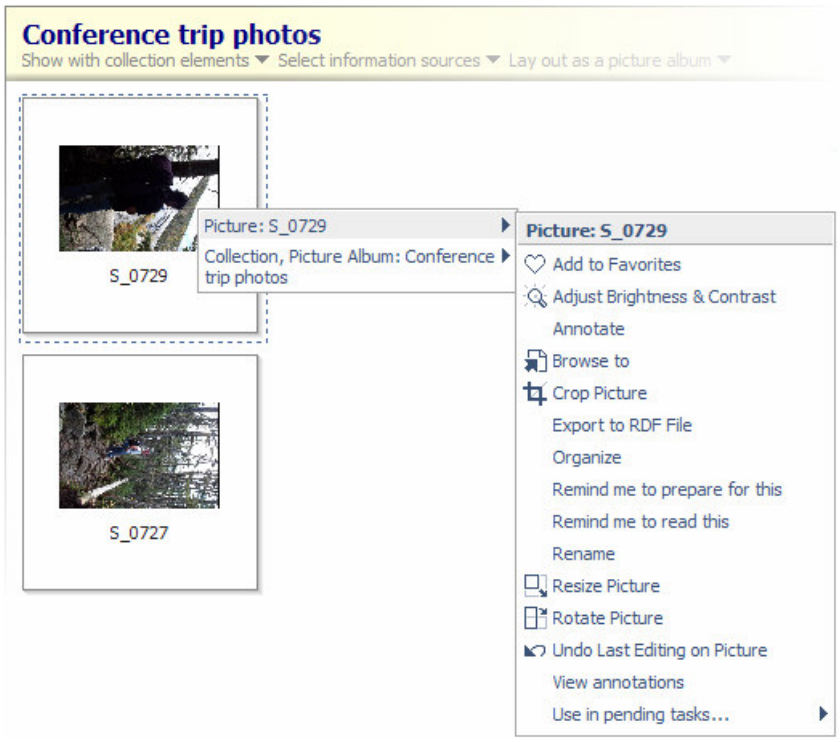


Figure 5. Sample context menu

4.3 Operation Composition

By unifying the format in which data is encoded, RDF makes functionality integration easier. Every piece of information is encoded in RDF and accessible through the same RDF store interface; information objects of the same type share the same schema. This blackboard architecture allows each piece of functionality to specialize in handling only data in a small set of specific schemas very well rather than attempting to deal with a larger domain of data less consistency and finesse. As all information objects in Haystack are made to appear capable of rendering themselves and capable of offering operations intrinsic to themselves, the concept of applications—software that acts upon data—has been changed.

In Haystack, there is no visible presence of monolithic applications. Instead, there are operations that perform small tasks. Such operations are individually simple and easy to learn, and their use can be composed arbitrarily by the user to perform complicated tasks. By "operation composition" we mean the carrying out of an ordered sequence of steps, each involving the use of one particular operation, aimed to accomplish an end goal not achievable by applying any operation alone.

4.4 Encapsulation of Unfinished Operations

Figure 6 shows a piece of UI inserted into the start pane when the *Rotate Picture* operation is selection from the context menu shown in Figure 5. This piece of UI is the view of a resource used to capture the carrying-out of the *Rotate Picture* operation on the *S_0729* picture. This resource is called a *UI continuation*, essentially the encapsulation of an unfinished operation. The UI continuation provides a user interface for completing the operation and returning to the original context where the operation was invoked (i.e. the view of the picture *S_0729* in this example).

Since a UI continuation is itself an information object, it can be manipulated just like any other object. For instance, it can be bookmarked and returned to at a later time. This is useful when the user decides to turn his or her attention to another task. Because the UI continuation is modeless, if the user keeps on the same task, he or she can still navigate to other information objects in order to find more information that helps complete the task. The UI continuation does not lock the user into the same screen until he or she finishes the task.

In contrast, existing applications do not have any mechanism for capturing the context of an unfinished task. The user has to trace through a sequence of user interface actions to return to a previously unfinished task (e.g. re-opening an application, invoking a certain menu command, and clicking on a sequence of buttons on a series of dialog boxes). Because the context of a task is often defined by a set of blocking user interfaces in the form of modal dialog boxes, the user is locked into that context, unable to make use of other parts of the same application to help completing the task. The user is forced to dismiss a stack of modal dialog boxes in order to have access to other functionalities in the application and then reconstruct the context by opening the same stack of modal dialog boxes again.

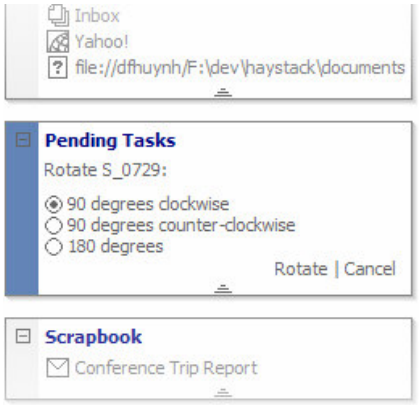


Figure 6. Sample UI continuation

5. UNIFIED ORGANIZATION

As the user's information accumulates, there is a need for organization either by the user or automatically by the system. Modeling data in RDF helps implement organization mechanisms that better match the user's concepts of organization. First, organization can be applied to all information objects, big and small. The user can organize various types of items ranging from e-mail messages, contacts, documents, bookmarks, journal entries, notes, appointments, and tasks to operations, system notifications, instant messages, editing comments on documents, frequently used bibliographic entries, favorite video clips, etc. and organize them using the same set of mechanisms.

Furthermore, objects of different types can be organized together. The Favorites list can contain more than just links to web pages: to the user, any object can be a favorite regardless of its type (Figure 7). E-mail messages and instant messages can be placed in common lists to show conversations that span over several delivery modes [10]. Appointments and tasks can appear together on the user's calendar; they can be sorted and prioritized with respect to one another. The same timeline view can be used to display company meetings and family photos. The user has to learn to use only a small set of organization mechanisms that work on several types of information. In contrast, today's software offer many application-specific organization mechanisms that overlap in functionality but differ in their user interfaces.

5.1 Organization by Classification

The most common mechanism for organization is classification: the assertion of each item's membership in one or more classes. The assertion can take the form of storage selection (e.g. placing the item into a box) or of category tagging (e.g. sticking colored labels on the item). The former gives immediately the feel that information is being divided into smaller units of organization and that the user has indeed made progress in organization such that only smaller subsets need to be dealt with at any one time. The latter allows the flexibility of classifying each item into more than one category. The latter can emulate the former by enforcing single membership. In Haystack, we model category tagging through the concept of *collections* and adapt it to support storage selection where appropriate.

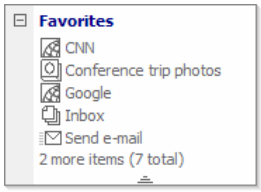


Figure 7. The Favorites list contains items of different types: web pages (*CNN* and *Google*), photo albums (*Conference trip photos*), operations (*Send e-mail*), and other collections (*Inbox*)

A collection is a mathematical set. Items can be inserted into a collection by asserting their memberships in that collection. An item can belong in several collections, much as a person can be member of several clubs. Organization can be achieved by putting items into collections.

Collections are used pervasively in Haystack for the purpose of organization. For instance, the Favorites list is a collection; the user's Inbox is a collection; the list of menu commands in a context menu is a collection. By implementing generic mechanisms for constructing and managing collections, we provide the user with a uniform way for organizing his or her information as well as interacting with the system's data. Figure 9 shows that the collection of operations in the context menu shown in Figure 5 can be displayed in the same view as e-mail messages (compare to Figure 2). The same functionality including sorting and grouping can be applied uniformly on both e-mail messages and operations. In addition, when the user manipulates the items in this view, the context menu will reflect the changes.

The generic mechanism for building collections is drag and drop. Custom operations can be provided for special collections: Figure 5 shows one such operation—the *Add to Favorites* operation.

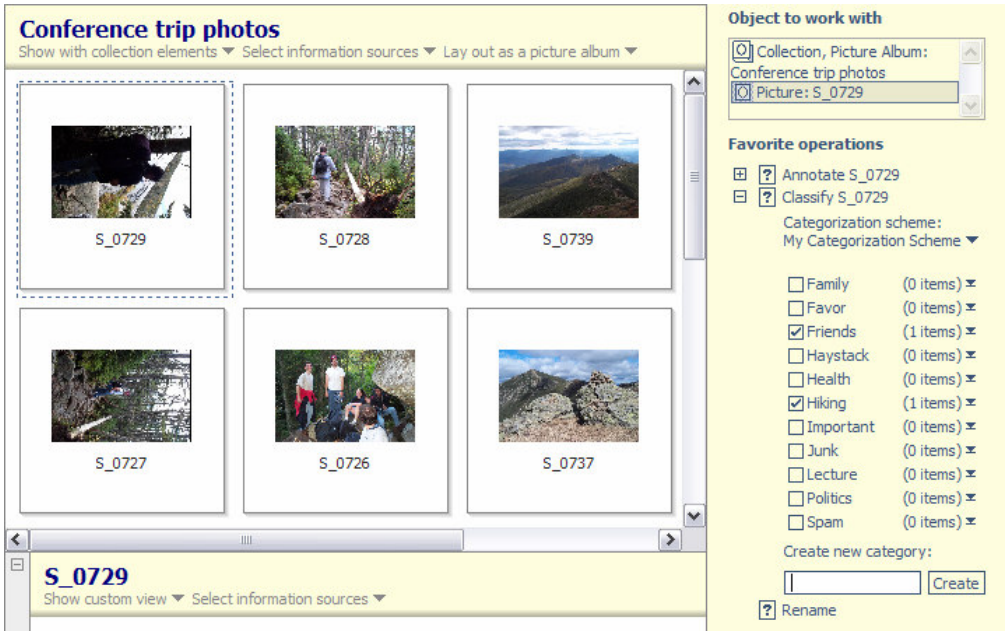


Figure 8. Categorization checkboxes

Other custom UIs are provided where appropriate. Figure 8 displays a list of checkboxes corresponding to several collections; by checking a checkbox, the user puts the selected picture into the corresponding collection. Multiple checkboxes make it easy to classify an item into more than one category. Figure 2 shows another custom piece of UI that allows the user to quickly classify an e-mail message as spam.

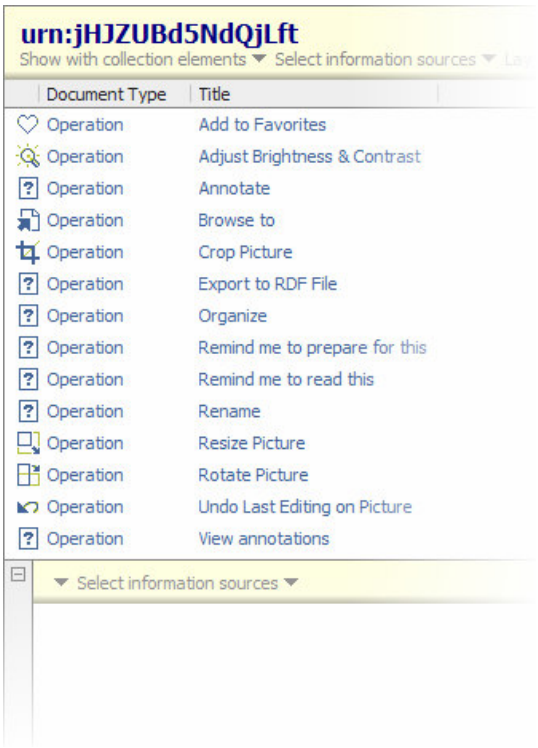


Figure 9. Collection of operations

5.2 Organization by Annotation

Another mode of organization is by annotation. The user can insert comments into the body of a long document and then use the resulting list of comments as bookmarks into the document. In other cases, annotations create associations between various information objects that are meaningful to the user. For instance, by annotating a certain book with the fact that it has been lent to a friend, a forgetful user can later determine how to locate the book.

In Haystack, annotation can be performed implicitly or explicitly. When the user drags a document onto an e-mail message, Haystack infers that the document should be attached to the message. The *attach* predicate is added between the message and the document. As a result, an annotation has been made implicitly. When the user replies to an e-mail message, the *replyTo* predicate is added between the new message and the original one.

Annotations can be added more explicitly through Haystack's relationship view, which presents items as graph nodes and allows the user to draw labeled arrows between the nodes. This UI is particularly useful when the user needs to handle information in graph or tree forms. It is easy to construct a family tree by drawing arrows from parents to children. Likewise, thoughts captured as notes can be linked together in a graph to show the logic of an argument.

Haystack also provides a metadata editor for editing raw RDF data directly. Figure 10 shows this editor listing all outgoing predicates from the *Favorites* collection object. The user can add his or her own predicates in order to extend existing schemas or to create new schemas.

5.3 Organization as Metadata Input

The acts of organization that the user takes cause metadata to be added to the system. As users go about their daily use of Haystack, organizing and manipulating information that they care about, metadata is being constantly added to the global pool of Semantic Web data.

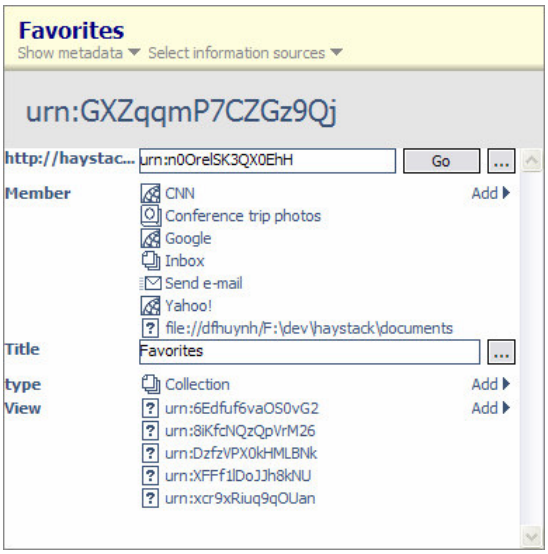


Figure 10. Metadata editor

6. USAGE SCENARIO

In this section, we illustrate the use of Haystack to perform a series of information management tasks. Consider the following scenario:

- The user is reading an e-mail sent from a colleague about a recent conference (Figure 11). The colleague asked the user to verify that the word "plethora" means "abundance." The user right clicks on the word "plethora" and finds that it offers an operation to look up its meaning in a dictionary. The operation brings the user to an online dictionary, which specifies that "plethora" is a synonym of "abundance."

- The user clicks the *Back* button to return to the e-mail message, and clicks on the *Forward* button on the title bar to forward the e-mail to her supervisor, as requested by her colleague. This action navigates to a newly created e-mail message, with the attachments of the old message automatically included.

- The user decides to classify the e-mail message being composed as its topic is still fresh in her mind. She simply checks the appropriate categories as illustrated for organizing pictures in Figure 8. In a conventional e-mail client, the user would have to open the *Drafts* folder, locate the message, drag it to the appropriate folder, and return to the composition window.

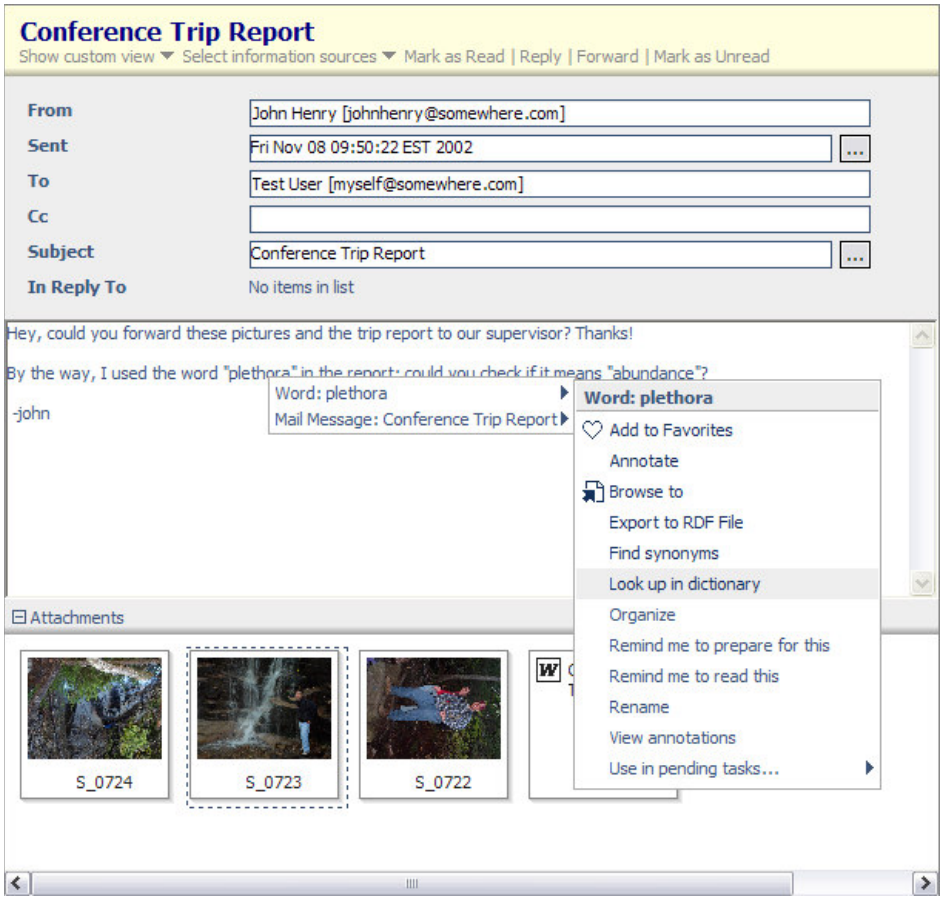


Figure 11. Looking up a word in an e-mail message

- The user finds that some of the attached pictures are not in proper orientation as evident from their thumbnails. The user right clicks on some of the thumbnails (in the *Attachments* pane of the message's view) and selects the *Rotate* operation. There is no need to save the attachments to files, open an image editing application, open the files, rotate the images, resave them, and re-attach them to the e-mail message. The images can be manipulated directly wherever they are displayed, even when displayed as thumbnails.
- The user wants to attach her own photographs to the e-mail message. She clicks on the link labeled "Conference trip photos" in her list of *Favorites* things (Figure 2). This action navigates to the photo album entitled "Conference trip photos." She then drags one of the pictures (S_0739) into the *Scrapbook* on the *Start pane* (Figure 12).
- The user then clicks *Back* to return to the e-mail message, and drags the picture from the *Scrapbook* into the *Attachments* area of the message.
- The user fills in her supervisor's e-mail address and clicks on the Send button in the title bar to send the message.

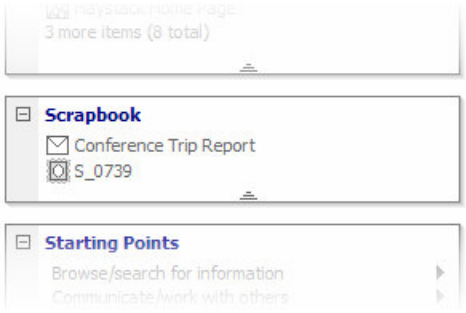


Figure 12. Picture in Scrapbook

7. ACKNOWLEDGMENTS

This work was supported by the MIT-NTT collaboration, the MIT Oxygen project, and IBM.

8. REFERENCES

[1] Resource Description Framework (RDF) Model and Syntax Specification. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/.

[2] Berners-Lee, T. Primer: *Getting into RDF & Semantic Web using N3*. http://www.w3.org/2000/10/swap/Primer.html.

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

[4] Dourish, P., Edwards, W.K., et al. "Extending Document Management Systems with User-Specific Active Properties." ACM Transactions on Information Systems, vol. 18, no. 2, April 2000, pages 140–170.

[5] Eriksson, H., Fergerson, R., Shahar, Y., and Musen, M. Automatic Generation of Ontology Editors. In *Proceedings of the 12th Banff Knowledge Acquisition Workshop, Banff, Alberta, Canada*, 1999.

[6] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns*. Boston: Addison Wesley, 1995.

[7] Handschuh, S., Staab, S., and Maedche, A. CREAM—Creating relational metadata with a component-based ontology-driven annotation framework. K-CAP '01.

[8] Huynh, D., Karger, D., and Quan, D. (2002). "Haystack: A Platform for Creating, Organizing and Visualizing Information Using RDF." Semantic Web Workshop, The Eleventh World Wide Web Conference 2002 (WWW2002). http://haystack.lcs.mit.edu/papers/sww02.pdf.

[9] Lansdale, M. "The Psychology of Personal Information Management." Applied Ergonomics, vol. 19, no. 1, 1988, pages 55–66.

[10] Quan, D., Bakshi, K., and Karger, D. "A Unified Abstraction for Messaging on the Semantic Web." Submission to The Twelve World Wide Web Conference 2003 (WWW2003).

[11] Sinha, V. and Karger, D. "Information Retrieval for Semistructured Data." Submission to The Twelve World Wide Web Conference 2003 (WWW2003).