Semantic Web Applications

Karun Bakshi and David R. Karger

MIT Computer Science and Artificial Intelligence Laboratory
32 Vassar Street
Cambridge, MA 02139 USA
kbakshi@mit.edu, karger@mit.edu

Abstract

Although a lot of information has become readily accessible and necessary for daily work, the current infrastructure for managing information is ill-suited for information-oriented activities: information and functionality is scattered across applications and websites, making it difficult to aggregate and reuse just the right set of content and operations required for unique user tasks. In this paper we discuss a collection of tools built into the Haystack information management platform that address many of the shortcomings of current applications, and allow composing reusable fragments of information from the Semantic Web and the operations that manipulate them into a task workspace tailored to the user and the task. Users can change the workspace to immediately meet changing requirements to easily include, remove or reuse information in multiple tasks simultaneously. The time that a user invests for the initial setup and occasional updates to the workspace is amortized over the numerous times he or she returns to the task, and all relevant information resources are co-located and ready to use.

Introduction

Many information-based tasks require a person to assimilate and manipulate multiple pieces of information, e.g., a military commander monitoring troop movement, logistics support and weather feedback from a battlefront to decide on the next course of action, or a doctor viewing a patient's past visit, prescription and x-ray history with respect to a diagnosis in order to decide determine how to proceed with a particular therapy. We often tackle these tasks using applications. Each application offers users the opportunity to work with a certain pool of information by giving them a set of information views and a collection of operations that can be invoked on the information being viewed. But when users' information management tasks do not exactly match the ones envisioned by the application developers, they can find themselves fighting with the application, struggling to simultaneously see all the information they need, or to invoke operations that are buried deeply in the feature set of the application. Worse, users often find that the information they need is spread out over multiple applications. In such cases, they find themselves wading through cluttered desktops full of multiple application windows, mentally or manually processing information from each one and ignoring lots of other distracting information.

Today, the World Wide Web is an indispensable resource and the Semantic Web, with metadata annotated information, will be even more vital for completing information-based tasks [1]. On the Semantic Web, agents and other automated processes will produce more information faster and at a finer level of access and semantic granularity that can be shared via web services. Attempting to manage this torrent of information using multiple applications will lead to a proliferation of applications, further partitioning related information thereby exacerbating the current complex, time consuming and error prone nature of many information-based tasks. A more robust solution is needed that can easily adapt to evolving user and task needs by working equally well with multiple, unanticipated types of information fragments as they become available on the Semantic Web.

The Semantic Web hints at a solution to some of these problems. It offers a single unified data model powerful enough to hold all of the information currently scattered among multiple applications and the metadata annotations can be used to select relevant information. But merely unifying the data is insufficient. To use it to solve a particular task, users still need tools that will aggregate the information they need into a meaningful presentation that lets them view and manipulate it as is needed for their task. What are needed are small, flexible and reusable units of content and their associated user interfaces and application logic that can be arbitrarily combined to yield larger, more powerful task interfaces.

In this paper, we argue that it is both desirable and possible to let end-users create their own information management applications over the Semantic Web, choosing precisely which information objects they want to work with and how they want to view and manipulate those objects. Such "end-user application development" would let end-users create

workspaces designed specifically to accomplish their particular information management tasks. Our approach combines three elements:

- A task workspace designer that lets users specify the information objects that they want to lay out to work with in their application and the views and operations that should be applicable to those information objects. We consider the problem of information presentation as consisting of two parts: specification of high level layout of multiple types of information and of the view of a particular information entity (For example, a person in the accounting department may be interested in the spreadsheet view of some sales figures, whereas a higher level executive would prefer a chart based on this data.) Layout capability is not only important in managing a set of related content within an application, but will also be important for *a priori*, unrelated content that the user has aggregated and juxtaposed for his/her task. A view should be the unit of user interface interaction that allows exposing relevant properties and direct interaction capabilities of an object to support constraints imposed by task or user preferences.
- A view designer that functions as a supporting tool and allows users specify how each of the information objects in their workspace should be shown – what properties of those objects they want to see, and how they should be laid out; and
- A channel manager supporting tool that lets users specify content queries that dynamically maintain collections of information relevant to the task. On the Semantic Web, this issue will have be important as the authoring ontology is not intended to necessarily match all usage scenarios, and hence users must easily be able to extract the relevant portions of the information efficiently.

Rather than specifying views, workspaces, and channels programmatically, users put them together using natural visual operations such as dragging, dropping, and resizing, that they are already familiar with as tools for managing their desktop environments. The workspaces, views, and channels designed by these end-users are themselves represented using RDF in the Semantic Web, creating an opportunity for users to share the views and workspaces that they design with others, and for unsophisticated users to craft their "applications" by tweaking preexisting ones instead of creating them from scratch.

We have implemented our system as part of the Haystack information management platform. This tool provides a set of cooperating technologies and tools supporting end-user creation, visualization and manipulation of Semantic Web content, as well as application development for the Semantic Web [22, 23]. Along with a blackboard style RDF store, it hosts agents to provide automated reasoning and supports a user interface framework that provides pervasive context menus, drag and drop capability, and a view architecture that appropriately selects views for entities of different types depending on the context.

Related Work

That users require multiple information resources to complete a task became been clear with the advent of the earliest windowing systems and toolkits. In addition, a realization that users have different needs and preferences in how their available UI real estate needs to be allocated to these various resources led to the development of various window managers, each providing users with various tiling and resizing semantics [2]. Additional experience with ephemeral window managers that required repeated work to set up a task workspace led to more persistent options such as virtual desktops, which realized the value of capturing user task context in a returnable environment. A more involved research effort, Rooms, shared a similar goal of allowing a user to be able to create a separate user task workspace (a room) that specifies which tools are open, as well as the layout and presentation of their windows [5]. These rooms (which could be shared between users) allowed capturing settings implicitly (e.g., window locations or sizes) as well as explicitly (e.g. connections between rooms). However, Rooms allowed managing windows of entire application as opposed to just the subset of the information that was relevant. In addition to window managers that allowed flexible control over UI real estate usage, much work has also been done in allowing users to user their available space as efficiently as possible [3, 4]. QuickSpace for example, implements simple window management operations to allow users to quickly allocate greater space to their primary operating window while maintaining the overall layout of the desktop.

A closer study of user window management behavior (see [7] for a survey) has led to capturing the most effective techniques being used in applications that feature task-based interfaces by aggregating relevant information for the user. Taskmaster is a Visual Basic add-on for the Microsoft Outlook client that targets task management activities in mail clients [8]. It makes information centric resources easy to access "at a glance, rather than scrolling around inspecting folders" by taking advantage of the heuristic that items in the same e-mail thread generally correspond to the same task. Thus, (incoming, outgoing and draft) messages are grouped into a project context based on such message data. Although the ideas embodied by Taskmaster do increase usability of information from the perspective of content customization, the benefits are restricted to a predetermined domain and users have little control over the layout or other presentational or manipulation capabilities. Also, users have little queryable control over the content. The Kubi Client is a commercial effort in the same vein [9]. Web Montage employs a different approach to creating task interfaces; it observes users and uses machine learning to build a personalized user web portal [10]. However, the aggregated content is read-only, and not manipulable.

In parallel, user sophistication has increased over the years, as applications that only allowed configuring of simple rendering preferences have ceded greater control over the task interface creation process. E-mail clients such as Microsoft Outlook, for example, supports user-defined rules that trigger based on message arrival or message sending events to maintain dynamic collections [11]. Content portals (e.g., *MyYahoo!*) on the World Wide Web and news organization web sites nowadays provide a set of primitives that can be used to create personalized web pages that allow the user to filter and/or aggregate the content provided by the underlying organization(s) [12]. However, the portals are limited; users can select from or configure a set of prespecified layout templates or content queries rather than create them, certain content can only appear in certain panes, and there is limited linked interaction between content in different parts of the portal. Snap-Together Visualization allows users to query a relational database and link the result set visualizations in support of a complex information exploration task [13]. Although it can be employed in arbitrary domains, it is constrained by a particular database instance (or small sets thereof) and its associated schema(s). It cannot easily scale easily to capture multiple domains or semi-structured data models.

Recently, the notions of task workspaces and user control over their creation have merged in the WinCuts project that allows users to aggregate only the relevant portions of source windows into a new task context [14]. Each WinCut is a live connection to the source window and can redirect manipulations to it. Although it allows visualization and interaction with information fragments, it does not provide direct access to the underlying semantic entity. Similarly, Presto, an interactive document management system allows defining tasks interfaces based on sets of relevant individual documents and other relevant, document collections, where each document can appear in multiple workspaces [20]. The document collections themselves are dynamic, and described via a query that utilizes underlying key/value attribute pairs that facilitate organization, search and retrieval of documents in the corpus. Presto marks a subtle, but important shift in task conceptualization, where the building blocks of tasks have become content instead of applications windows.

Although, much attention has been paid to ontological and content authoring on the Semantic Web [15, 17], recently, more attention is being devoted to using semantics to enhance the end-user browsing and navigational experience for specific domains [16, 21]. With the content on the Semantic Web anticipated to be annotated at a much finer granularity than documents, we have the opportunity to merge task creation, user control and semantics and allow creating task workspaces that allow semantic rather than pixel manipulation. It is this vision that we hope to realize with our system.

System Walkthrough

Consider Ann, a neurology researcher investigating brain structures who needs to manage her research tasks. She might need to explore brain volume data, other research papers, to-do items, her calendar, as well as e-mails with collaborators. In order to write a research paper attempting to identify the correlation between seemingly unrelated symptoms and mental illness diagnoses, and the volumes of various brain structures, she might have to use and constantly switch between multiple applications, e.g., the e-mail client, the brain volume database querying view, a repository of neurology papers, etc. She is forced to

- manually collect information by opening various applications
- mentally select and associate items of interest, and reason with them, since they cannot be juxtaposed
- Reenter the data elsewhere (e.g., extract the papers that reported the data based on a key), acting as the glue between applications.

Whereas, a single application could be developed that included functionality for all her tasks, it would still be a rigid solution. What if the she also wanted to use the new application's functionality, such as calendar and e-mail, outside of the research application, e.g., to see when her dentist's appointment is? What if he/she now had to also track news about spinal cord injuries? What if the she wanted to define a new "task" that involved just looking at the various data groups?

Figure 1 shows a workspace she has designed to assist in her paper writing task. The workspace has been designed to help aggregate various information resources required to support the hypothesis and the paper writing process; she has defined a query of all Caudate instances having a volume of greater than 9 cc as potentially intriguing. In addition, she has defined a query for the groups to which this data corresponds, by having it depend on the first one. Also, Ann is interested in seeing the publications that reported the relevant data, the other publications by authors of the reporting publications, as well as a list of collaborators.

As she comes to work, she decides to look into the past publication history of her collaborators to see if they have done relevant work. In addition, since the paper sub-tasks are getting to be numerous, she decides that it's time to start tracking them. Also, since her spouse has taken their son for a doctor's appointment, she wants to be kept apprised of e-mails from home. In order to handle these changes in her task description, she switches the task workspace to design mode (Figure 2) using the "Change View" menu, and adds a preview window to link with the collaborators portal to show the papers published by the selected collaborator. The various layout manipulation operations (split horizontally, split vertically and delete) available on the toolbar on the right of each portal can be used to lay out the workspace as desired. Each portal

allows configuring the content, how it should be viewed, and how it can be manipulated. In addition, she goes to the channel manager (Figure 3) to define a new query for to-do items for the paper, and adds a pane in the task workspace to track this collection. The queries are specified by instantiating various query primitives via drag and drop. In order to simplify the query building process, the channel manager allows copying existing channels and interactive evaluation of the query. Finally, she changes her query to look at Amygdala volumes, changes to a more detailed view for the groups reporting the data, and docks an information viewer that only shows e-mails from home, outside the task workspace, but within her field of view (Figure 4).

If a query's parameters are changed (e.g., she decides to investigate the Amygdala brain structure rather than the Caudate), all dependent queries will also be recomputed, causing a ripple effect in the workspace. Thus, the workspace stays current with respect to the current query definitions as well as information corpus, enforcing the relationships between the various portals and simplifying data exploration – a task that would otherwise require significant manual re-querying and correlating if either changed. As can be seen, frequently used operations relevant to the information in the portal are within easy reach. Clicking on the operations implicitly uses the currently selected item as a parameter if it matches one the required parameters. Note, that although no paper writing application was written by anyone, the user was able to create it with relative ease. Furthermore, the "application" is amenable to change.

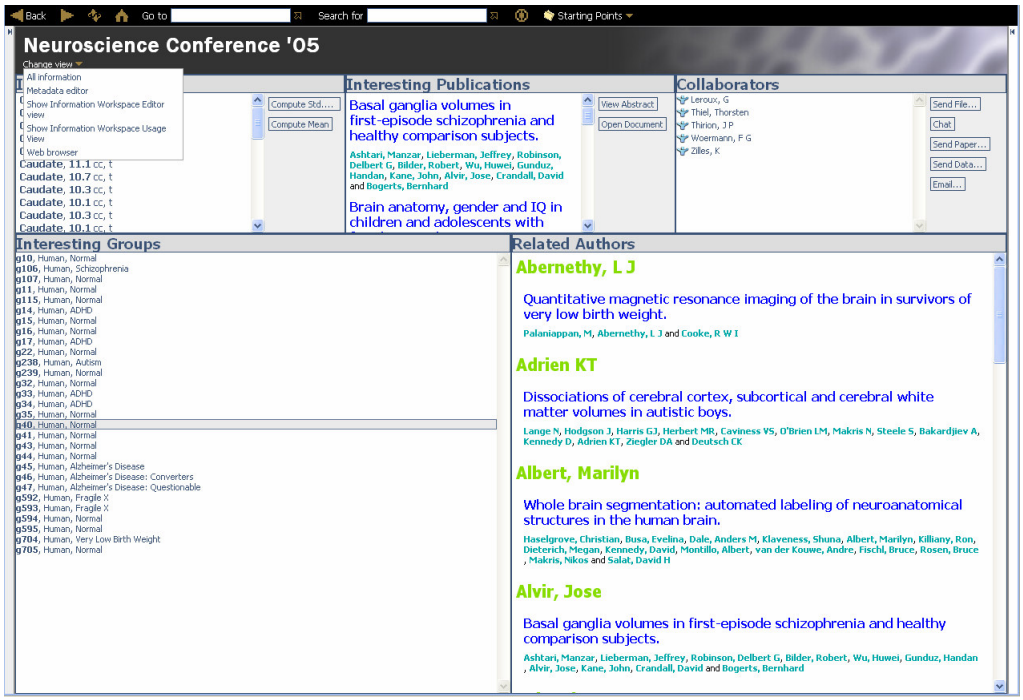

Fig. 1. Initial Paper Writing Workspace in Usage Mode

System Description

We have developed several tools in Haystack that allow them to access functionality that has heretofore been the domain of developers. In general, all components, whether the backend agent imperative code or the UI front-end, were implemented in Adenine, a language developed specifically to ease RDF expression and manipulation on the Haystack platform.

Workspace Designer

A workspace consists of a collection of portals and a layout specification for them and is rendered by recursively rendering each of the portals using the appropriate design and usage views. Several important design decisions were made in the design of the workspace functionality. We chose to allow modality in workspace interaction (design vs. usage modes) primarily because the act of specifying underlying content for a portal was most effectively done by using drag and drop, which required modality in order to be able disambiguate between the act of specifying the underlying content and dragging an item onto the portal in usage mode. Thus, we felt it would be easier for the user to have a clear separation of modes rather than wasting UI real estate on extraneous mode specific widgets.

Another important feature of the current information space design allows users to leverage a built-in capability in Haystack to specialize operations in order to customize the operations available in an information portal. In general, operations that the user may invoke, may solicit him for values of arguments in order to carry it out. Currying is a technique in Haystack that allows users to specify some or all of the arguments that he/she feels will always have the same values, and save the resulting partially specified operation closure as a new operation that, when invoked, will ask the user for any remaining argument values [6] This operation, like all other operations, can then be associated with a particular information portal, for which the previously specified arguments make sense.

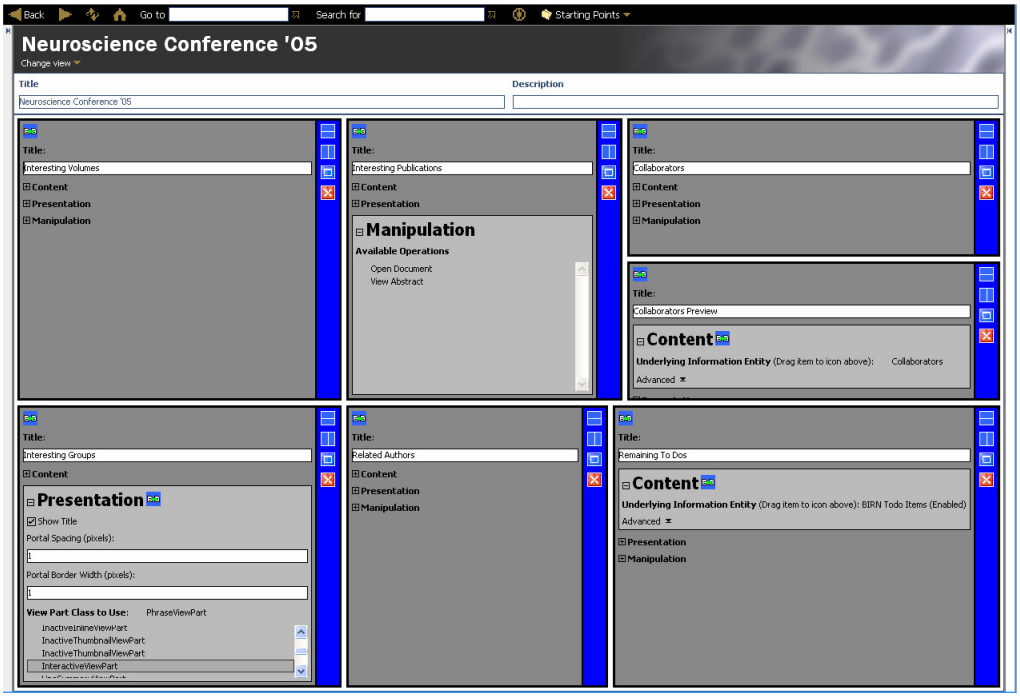

Fig. 2. Paper Writing Workspace in Design Mode

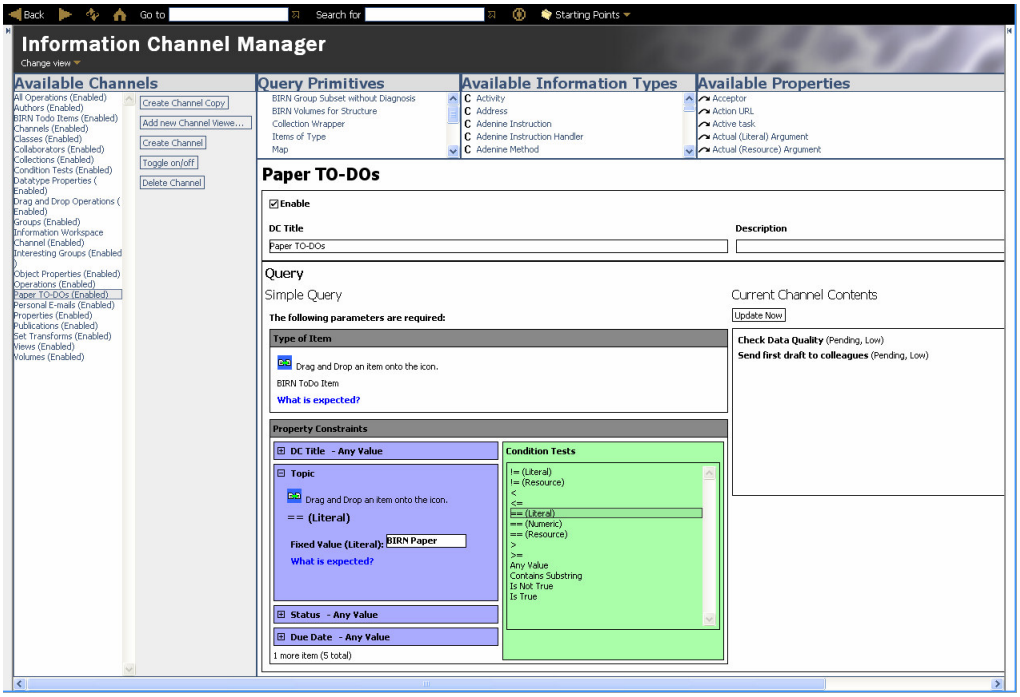

Fig. 3. Information Channel Manager

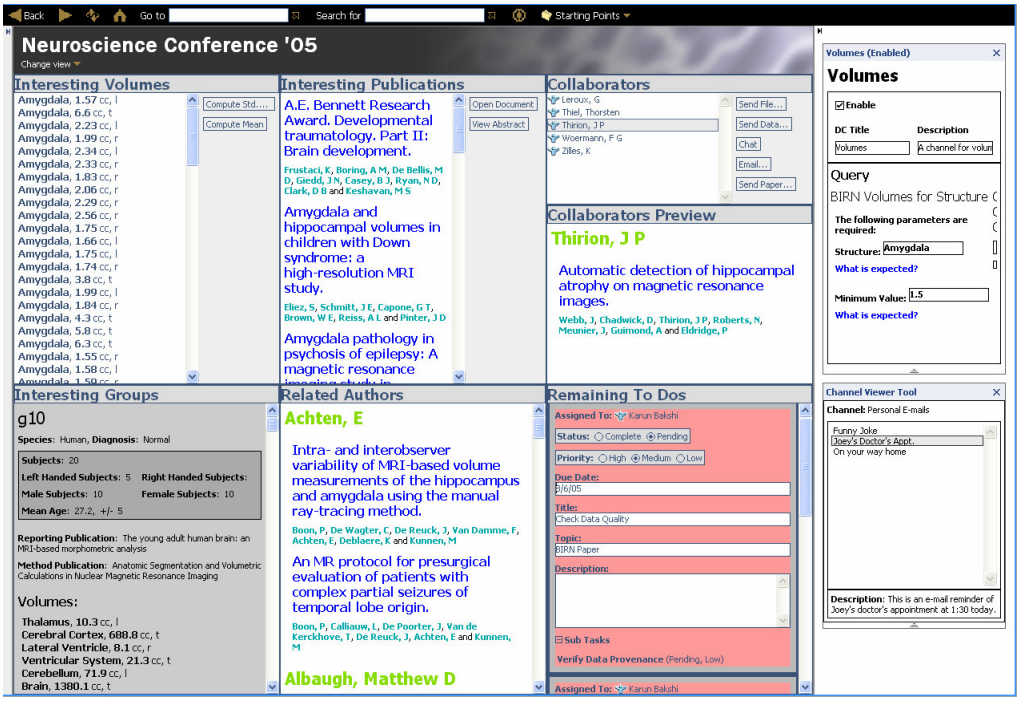

Fig. 4. Final Paper Writing Workspace in Usage Mode

Perhaps the most important design decision that affected the construction of workspaces by users was how to allow users to allocate space to portals. Given our understanding of the portal creation task from *Yahoo!*, and the ideas espoused by the QuickSpace project, it seemed that the goal of user controlled content layout was to allow users access to a canvas that is appropriately segmented and completely devoted to showing the content of interest, with the user being able to specify the location of items, and possibly their size directly or indirectly. Such an approach allowed users to allocate maximal space to the content of interest. Furthermore, it allowed the space to adapt to a local change: if a user increases the amount of space for one item, the space for other items automatically decreased without overlapping – an important feature desired by users, as demonstrated by QuickSpace. The decision to only allow the user to split cells into two rows or columns was driven by the desire to keep the operation of segmenting the usable space as simple and efficient ("one-click") as possible. Similarly, we chose to allow users to place operations in the right pane of a portal rather than use the context menu in order to minimize user clicks.

Channel Manager

The channel manager allows an implicit collection specification mechanism by providing the user an interface to query the underlying RDF store. As in Presto, channels are self-maintaining organizing mechanism for dynamic corpora of information that allow users to specify a query that maintains an up-to-date dynamic collection of items; they allow the user to impose his/her world view on an otherwise raw and changing set of information by defining a set of persistent indices of relevant information that are then maintained up-to-date by the system. Channels are computed by a channel manager agent that periodically updates all channels. The query primitives available in the channel manager are simply parameterized, imperative code fragments that always return a set of URIs of matching entities.

The query primitive set supported by the channel manager is extensible. Any new code fragment that meets the above constraints can be exposed as a query primitive to the user and immediately work with the channel manager agent. The current primitives can be thought of as belonging to three primary categories: RDF query primitives, Channel Operators, Domain Specific. RDF query primitives enable simple and common RDF querying use cases, e.g. the empty set, items of a particular type, etc. Channel operators (e.g., set union, etc.) allow combining channels or user-specified fixed collections to create new channels. Finally, developers can add new primitive types that support complex domain specific queries, or use domain specific language simply by writing the query code, and designing a view to capture relevant parameters.

View Designer

When working with information, it is critical that users be able to customize views not just in terms of cosmetic properties (e.g., color and font) or pre-determined/fixed, domain-specific properties (e.g., sorted e-mails by sender) deemed to be useful by the developer, but also by controlling which set of properties are accessible and how they can be visualized and manipulated in conformance with its semantics.

Since Haystack can represent information from arbitrary domains, it should support the addition of new view designers. As a baseline however, we developed a domain-independent, baseline view designer that lets users inspect any properties of an information entity in Haystack, regardless of its domain and taking advantage of minimal information semantics. The view designer allows them to create metadata lens views that select and view properties of interest from the underlying entity. The views created by users can be named and are then available in the workspace builder for future use, and can be copied as a starting point for a new view. The only semantics the designer understands are that property values can be either literals or resources. For literal properties, the user may specify whether or not the property is editable (i.e., an edit box is shown). Resource valued properties allow selection of an appropriate view to use to display it. However, even this simple capability becomes powerful, when combined with the power to reuse existing views in creating new views. The metadata lens views share the same layout engine as in the workspace functionality and also have design and usage views.

Discussion

The workspaces that users can design feature several desirable properties. We have argued that our tools should allow building task specific workspaces for arbitrary tasks in any domain. A good test of their power, then, would be to see how we could have used them to build workspaces for the various tasks involved in creating channels, views and workspaces. The tools we have discussed so far were indeed workspaces as they aggregated the necessary information resources for building channels, views and workspaces. Furthermore, a simple operation can be curried to allow users to navigate between the workspaces for these related tasks, much as in the Rooms project. Also, the workspaces are not static, requiring developers to provide the functionality users need; users can change the workspace to meet changing requirements to include, remove or reuse information in multiple tasks immediately. Nevertheless, the various tools support significant extensibility by developers to either provide new query primitives, layout engines, views or view designers. Additionally, the initial setup time that the user invests is amortized over the numerous times he or she returns to the task, and all resources are readily available in order to immediately be productive. Thus, users can capture and transform what used to be a process of using multiple applications, into a single workspace application. Finally, since the workspaces themselves constitute information entities captured in RDF, they can easily be serialized and shared with others, thereby further amortizing the initial time investment; for example, one person may create a workspace, which is then used by others in a group.

We feel that the notion of channels as persistent returnable queries will be an important one as users are forced to deal with more information, in less time on the Semantic Web. The modularity of channels lets users define information properties in a virtual manner ahead of time, without knowing what they will apply to. This is possible, because the nature of the channel's content is known *a priori* based on the channel's description. Thus, each individual item need not be annotated with certain properties; its membership in a channel allows it to (virtually) "inherit" the properties of the channel dynamically. Modularity also allows information channels to in different contexts and redirected to different portions of the UI (within, or outside an information space) where the corresponding subset of information is useful. Or, the information corresponding to a channel may be redirected to a different device altogether, e.g. if an employee becomes sick or seeks to work from home, the channels appropriate to the work project can be subscribed to from the home computer. Channels as an abstraction are also useful in hiding the distributed and segmented nature of information by allowing aggregation of information from multiple stores. As an indicator of the current user task focus, channels allow an information management platform a simple but useful technique to perform gate-keeping actions by minimizing users' interruption with events or information unrelated to the current task.

Having implemented the simple view designer, we recognized that more powerful views could be designed by view designers that better understand the semantics of the underlying information that the views they generate will be manipulating. We advocate giving users the power to create their own views, using appropriate view designers that interpret the underlying information using various semantics, and can expose appropriate primitives for creating corresponding views. Semantics can be leveraged in various ways by view designers to allow creation of powerful views by users, while preserving valid data. For example, a simple understanding of the properties the user wishes to view/manipulate would allow the view designer to present appropriate widgets, e.g. a checkbox for a Boolean property. Semantics used to interpret information can also be used to allow designers to correspond to styles of views, e.g., a list of genetic bases can be interpreted appropriately by a designer, and allow the user to specify a preference of whether just the sequence, or its complement is also to be rendered.

Conclusion & Future Work

In this paper, we have proposed an approach to information management that avoids the rigidity of domain specific applications and empowers the users to create flexible task workspaces that can take advantage of information on Semantic Web immediately, without having to wait for a corresponding application to be written. In addition, we have attempted to identify the primary ingredients of workspaces that given sufficient customization control over, users can employ effectively in building these workspaces to increase their productivity and minimize information overload. Whereas many of the constituent ideas and implementation techniques are not new, when applied to the Semantic Web, we believe that such tools to help people use the information will be an enabling technology critical to the Semantic Web's widespread adoption. The information workspace builder is an important contribution to this endeavor. Our future work will concentrate on further understanding and refining the customizing capabilities that users need in building their workspaces, as well how best to make them available. In addition, we hope to investigate other infrastructure technologies that we anticipate will be required to make the Semantic Web a success and support task workspaces.

Acknowledgements Future Work

This study was supported, in part, by the Biomedical Informatics Research Network (www.nbirn.net), Morphometry BIRN Testbed: 5U24RR021382.

References

1. Berners-Lee, T., Hendler, J. and Lassila, O. The Semantic Web. *Scientific American*, May 2001.
2. http://xwinman.org/
3. Kandogan, E., Schneiderman, B. Elastic windows: Evaluation of multi-window operations. CHI 1997, pp. 250-257.
4. Hutchings, D. and Stasko, J. QuickSpace: New Operations for the Desktop Metaphor. Extended Abstracts of the Conference on Human Factors in Computing Systems 2002.
5. Card, S. and Henderson, D. Rooms: The Use of Multiple Virtual Workspaces to Reduce Space Contention in a Window-Based Graphical User Interface. *ACM Transactions on Graphics* 5(3), July 1986, pp. 211-243.
6. Quan, D., Huynh, D., Karger, D. and Miller, R. User Interface Continuations. Proceedings of User Interface Software and Technology (UIST) 2003.
7. Myers, B. A. A Taxonomy of Window Manager User Interfaces. *IEEE Computer Graphics and Applications*, 8(5), September 1988, pp. 65-84.
8. Bellotti, V., Ducheneaut, N., Howard, M. and Smith, I. Taking Email to Task: The Design and Evaluation of a Task Management Centered Email Tool. Proceedings of the Conference on Human Factors in Computing Systems 2003.
9. Kubi Software. www.kubisoftware.com.
10. Anderson, C. and Horvitz, E. Web Montage: A Dynamic Personalized Start Page. Proceedings of the Eleventh International Conference on the World Wide Web, 2002.
11. Microsoft Office Online, Outlook. http://www.microsoft.com/outlook/.
12. *MyYahoo!* http://my.yahoo.com.
13. North, C. and Shneiderman, B. Snap-together visualization: A User Interface for Coordinating Visualizations via Relational Schemata. Proceedings of the Working Conference on Advanced Visual Interfaces, 2000, p.128-135.
14. Tan, D.S., Meyers, B. and Czerwinski, M. WinCuts: Manipulating Arbitrary Window Regions for More Effective Use of Screen Space. Extended Abstracts on Human Factors in Computing Systems, CHI 2004.
15. Hogue, A. and Karger, D. Wrapper Induction for End-User Semantic Content Development. Proceedings of First International Workshop on Interaction Design and the Semantic Web, ISWC 2004.
16. Rutledge, L., Houben, G. and Frasincar, F. Combining Generality and Specificity in Generating Hypermedia Interfaces for Semantically Annotated Repositories. Proceedings of First International Workshop on Interaction Design and the Semantic Web, ISWC 2004.
17. Missikoff, M., Navigli, R. and Velardi, P. The Usable Ontology: An Environment for Building and Assessing a Domain Ontology. Proceedings of ISWC 2002, p. 39.
20. Dourish, P., Edwards, W. K., LaMarca, A., Salisbury, M. Presto: An Experimental Architecture for Fluid Interactive Document Spaces. *ACM Transactions on Computer-Human Interaction* (TOCHI), v.6 n.2, June 1999, pp.133-161.
21. Sicilia, M. and Garcia, E. Interaction Design of Ontology-Based Adaptive Resource Browsers Based on Selection. Proceedings of First International Workshop on Interaction Design and the Semantic Web, ISWC 2004.
22. Quan, D. and Karger, D. How to Make a Semantic Web Browser. Proceedings of WWW 2004.
23. Huynh, D., Karger, D. and Quan, D. Haystack: A Platform for Creating, Organizing, and Visualizing Information Using RDF. The Eighteenth National Conference on Artificial Intelligence Workshop on Ontologies and the Semantic Web.