# End-User Application Development for the Semantic Web

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 are 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. We discuss a collection of tools built into the Haystack platform that address many of the shortcomings of current applications, and allow composing reusable fragments of information and associated operations and views from the Semantic Web 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.

## 1 Introduction

Many information-based tasks require a person to assimilate and manipulate multiple pieces of information and the same information can be visualized and used in different ways in multiple tasks. Such tasks can range from a military commander monitoring troop movement, logistics support and weather feedback from a battlefront to decide on the next maneuver, to a doctor viewing a patient's past visit, prescription and x-ray history with respect to a diagnosis in order to determine how to proceed with a particular therapy.

Currently, we often tackle these tasks using applications developed to handle a range of closely related tasks. Each application offers users the opportunity to work with a certain pool of information by giving them a set of information views and a pool of operations that can be invoked on the information being viewed. But when an information management task does not exactly match one envisioned by the application developers, users find themselves fighting with the application, struggling to simultaneously display 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, users are forced to wade through cluttered desktops full of multiple application windows, each holding a small piece of the needed information (and lots of other distracting information).

The Semantic Web hints at a solution to some of these problems. The Resource Description Framework, its single, unified data model, is powerful enough to hold all of the information typically scattered across multiple applications. But merely unifying the data is insufficient; users must manage multiple sources of information in their tasks, and the amount that they can mentally manipulate is limited. To use the information to solve a particular task, users still need tools that will allow selecting and aggregating the information they need into a meaningful presentation (a task or information workspace) that lets them view and manipulate it as needed for their unique task. Such a capability might simply increase the efficiency of the task that would otherwise require juggling multiple applications (e.g., the doctor above). Or, it can be critical for success in a situation where multiple streams of information cannot be individually assimilated and their relationships deduced and acted upon under time pressure (e.g., the military commander). This capability becomes even more important for tasks that are long-lived or recurring such that the user must access the relevant pieces of information on multiple occasions.

In this paper, we argue that it is both desirable and possible to let end-users create their own information management applications (task workspaces) over the Semantic Web, choosing precisely which information objects they want to work with and how they want to view and manipulate those objects to complete a task. Such "end-user application development" would let them create workspaces designed specifically to solve their particular information management problems. We discuss a collection of tools built into the Haystack platform that address many of the shortcomings of applications mentioned earlier and allow composing fragments of information from the Semantic Web and operations that manipulate them into such a task workspace tailored to the user and task. We define a task workspace as a collection of information relevant to the task at hand that can be selected, presented and operated upon based on the user and task constraints, and that the user can aggregate in a lightweight manner. Our approach combines three elements:

- A **workspace designer** that lets users lay out the sets of information objects they want to work with in their application, specify which view to use to present each type of object and stipulate the relevant operations that should be readily available;
- A **view designer** that lets users specify how each type of information object in their workspace should be shown – what properties of those objects they want to see, how users should interact with them, and how they should be laid out; and
- A **channel manager** that lets users specify queries that dynamically maintain collections of related information that can be used to specify the relevant sets of information for one or more task workspaces.

Rather than specifying views, workspaces, and channels programmatically, users put them together using natural visual operations that they are already familiar with as tools for managing their desktop environments (such as clicking, dragging, dropping, and resizing). 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 them 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. Haystack 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 [23, 24, 25, 28, 29]. 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 and presents views for different types of information entities depending on the context of use.

### 1.1 Motivation

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.

Consider a software project manager who needs to manage the tasks her team does, as well as the budget and schedule for the project. She might need to manage contact information for team members, to-do list and assignment of action-items to individuals, e-mails corresponding to the project, outstanding bug reports, a budget spreadsheet and a schedule. In order to react to a customer e-mail about a software bug, she must switch from the e-mail client to the bug tracking software to enter a bug report. Then, she must switch back to the e-mail client, recall which software developer would be best suited to fixing the bug, look up his/her contact information, and send him/her a bug report number. The project manager may also need to meet with the developer, and hence negotiate a meeting time via e-mail, and update her personal calendar. After the meeting, the project manager will need to switch to the scheduling application to update the project schedule to reflect the time that will be consumed in fixing the bug. Another application switch may be needed to update the developer's outstanding tasks if the scheduling software does not support to-do lists for team members. Finally, she must switch to the spreadsheet software to update the project budget to take into account the resources consumed by the bug fix. As this example demonstrates, rigid applications restrict users to behavior patterns that force them to become passive recipients of information

rather than being able to actively mold it to impose their own world view. They are forced to bridge the gaps between applications by having to:

- manually collect information by opening various applications;
- reenter the data elsewhere (e.g., retrieving information from one application using an identifying key from another or duplicate it in order to manipulate it in another application), acting as the glue between applications because applications do not understand each other's native information format;
- mentally select and associate items of interest, and reason with them, since they cannot be easily juxtaposed; and
- dig deeply into menus and multiple dialog boxes to find just the right operation to invoke.

Although, a single application could be developed that included functionality for all project management tasks, it would still be a rigid solution. What if the project manager also wanted to use the new application's functionality, such as calendar and e-mail, outside of the project management application, e.g., to see when her dentist's appointment is, or to receive mail from her spouse? What if she now had to also track news about a competitor's products? What if she wanted to define a new "task" that involved approving and disapproving bug fix recommendations that aggregated *just* the bug information from the various project workspaces into a new workspace? All such continual user requests for customizations would be difficult to satisfy by a limited number of application developers.

Today, the World Wide Web is an indispensable information resource and the Semantic Web, with metadata annotated information, will be even more vital for completing information-based tasks in the future [1]. On the Semantic Web, information will be produced faster using automated means, with finer access and semantic granularity and shared via web services than can be handled manually. Attempting to manage this torrent using multiple applications instead will lead to a proliferation of specialized applications that fragment information arbitrarily, making information management more complex, time-consuming and error-prone. A more robust solution is needed that does not require significant developer support to easily adapt to and reuse unanticipated types of information fragments for unforeseen tasks as soon as they become available on the Semantic Web.

### 1.2 Organization

The remainder of this paper is organized as follows: first, we provide a detailed system description. Next, we discuss related work in various areas that impact our current work, followed by a discussion of the high-level approach to developing a task management infrastructure. Then, we present important design and implementation decisions, followed by a summary of the salient benefits of our tools. We conclude with an outline of future research goals.

## 2 System Walkthrough

Consider Ann, a neurologist investigating brain structures. Figure 1 shows a task workspace that she has designed to assist in the task of writing a research paper attempting to identify the correlation between seemingly unrelated symptoms and mental illness diagnoses, based on the volumes of various brain structures. (The example is based on actual data from the Internet Brain Volume Database that describes the volumes of sundry brain structures in test groups having various mental illness diagnoses, and the research papers that report such data [31].) She has designed a workspace 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 (top left). In addition, she has defined a query for the groups to which this data correspond, by having the latter query (bottom left) extract the group related to each member in the first one. Also, Ann is interested in seeing the publications that reported the relevant data (top center), the other publications by authors of the reporting publications (bottom right), as well as a list of collaborators for the paper writing task (top right). Ann chooses to view the Caudate and Group data using a small view that exposes only certain relevant

properties so that she can easily glance at multiple items and quickly identify interesting ones. The publication view shows the title of the paper, but also allows her to click on any of the authors to get additional information about him or her. The view used for people who are authors of the interesting publication reuses the publication view to show a collection of their publications, which is the only aspect she is interested in knowing about each author in her current mind-set. On the other hand, the view she uses for people who are her collaborators simply shows their name as she is in frequent contact with them and does not need additional information yet. As can be seen, frequently used operations relevant to the information in the various winlets are within easy reach. (We coin the term winlet to refer to a tile in the workspace that captures particular content and relevant operations; e.g., the Interesting Publications winlet.) Clicking on the operations implicitly uses the currently selected item in the winlet as a parameter if it matches one of the required parameters for the operation.


**Fig. 1.** Initial Paper Writing Workspace in Usage Mode

As she comes to work this morning, Ann decides that the past publication history of her collaborators would be useful to have close at hand. In addition, since the paper sub-tasks are getting to be numerous, she decides that it's time to start tracking them via a to-do list. 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 winlet in order to show the papers published by the selected collaborator (far right, below Collaborators winlet). The various layout manipulation operations (split horizontally, split vertically and delete) available on the toolbar to the right of each winlet can be used to lay out the workspace as desired. Clicking on the maximize button expands a winlet to take up all available space when configuring its details. Each winlet can be named and resized. In addition, winlets allow one to configure the content (Content Pane), change how it is viewed (Presentation Pane), and how it can be manipulated (Manipulation Pane). The Content Pane allows the user to specify which particular information object(s) are to be shown in a particular winlet. They can be specified either explicitly by identifying a particular entity, or implicitly by specifying a query (implying a collection of information objects) or the currently selected item in another winlet. The Presentation Pane allows the user

to select the particular view to be used to render the information object(s). (Each type of object can have multiple associated views, e.g., a small summary view versus a larger, more detailed one.) Finally, the user can drag operations that are frequently used and/or applicable to the information objects being shown into a collection in the Manipulation Pane.

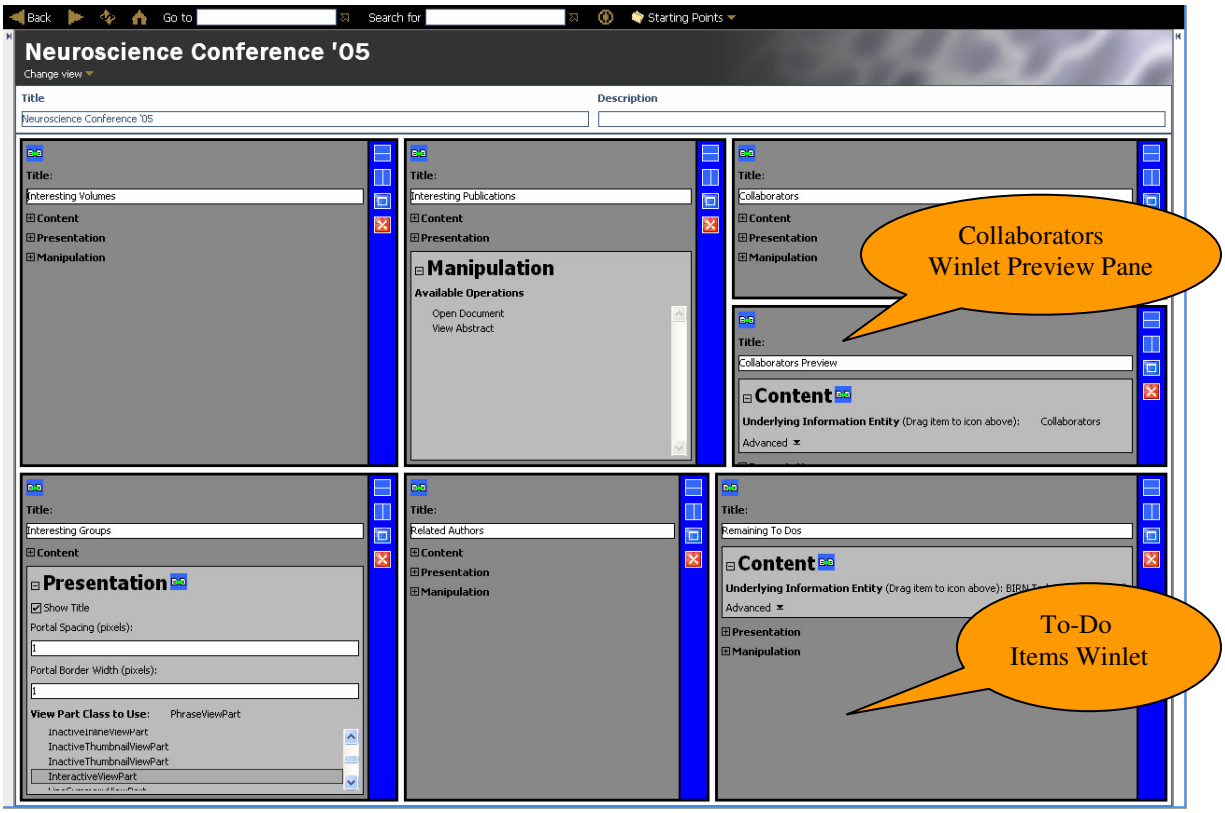


**Fig. 2.** Paper Writing Workspace in Design Mode

Ann then uses the channel manager (Figure 3) to define a new information channel (query) for to-do items for the paper, and adds a pane in the task workspace to track this collection (bottom right of Figure 2). A query can be defined by dragging and dropping a query primitive onto the channel. A query instance is instantiated and the user may then specify values for its parameters. In order to simplify the query building process and reduce usability barriers, the channel manager allows one to copy existing channels and interactive evaluation of the query by allowing the user to tweak query parameters as necessary and observe the updated results. Finally, in Figure 4, Ann changes her query for interesting volumes to look at Amygdala volumes (channel viewer docked in right pane), changes to a more detailed view for the groups reporting the data (bottom left), and docks an information channel viewer that only shows e-mails from home outside the task workspace (right pane), but within her field of view. The left pane in Figure 4 shows the final task workspace in Haystack. Note that users need not commit all information they wish to work with (even if temporarily) into the workspace if it is not relevant to the task. (The right pane in Haystack can be used as a scratchpad to temporarily place any entity within view.). The notion of channels allows selective interruption to be made possible as Ann has allowed herself to be temporarily informed about e-mails from home by docking a channel viewer tool in the right pane.

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 as it updates periodically. The reader will notice that the content of the various winlets relying on dynamic channels have changed appropriately in Figure 4. Thus, the workspace stays current with respect to the current query definitions as well as information corpus, enforcing the relationships between the various winlets and simplifying data exploration – a task that would otherwise require significant manual re-querying and correlating if the corpus or the query had changed. In addition, Ann has decided to look at

more detailed information about the new groups she will now be perusing, and hence has changed the view that should be used. She has also chosen to use the author view for her collaborators in the preview pane. Note that although no software was developed for the "paper writing application," Ann was able to create it with relative ease. Furthermore, the "application" is amenable to change as user needs change.

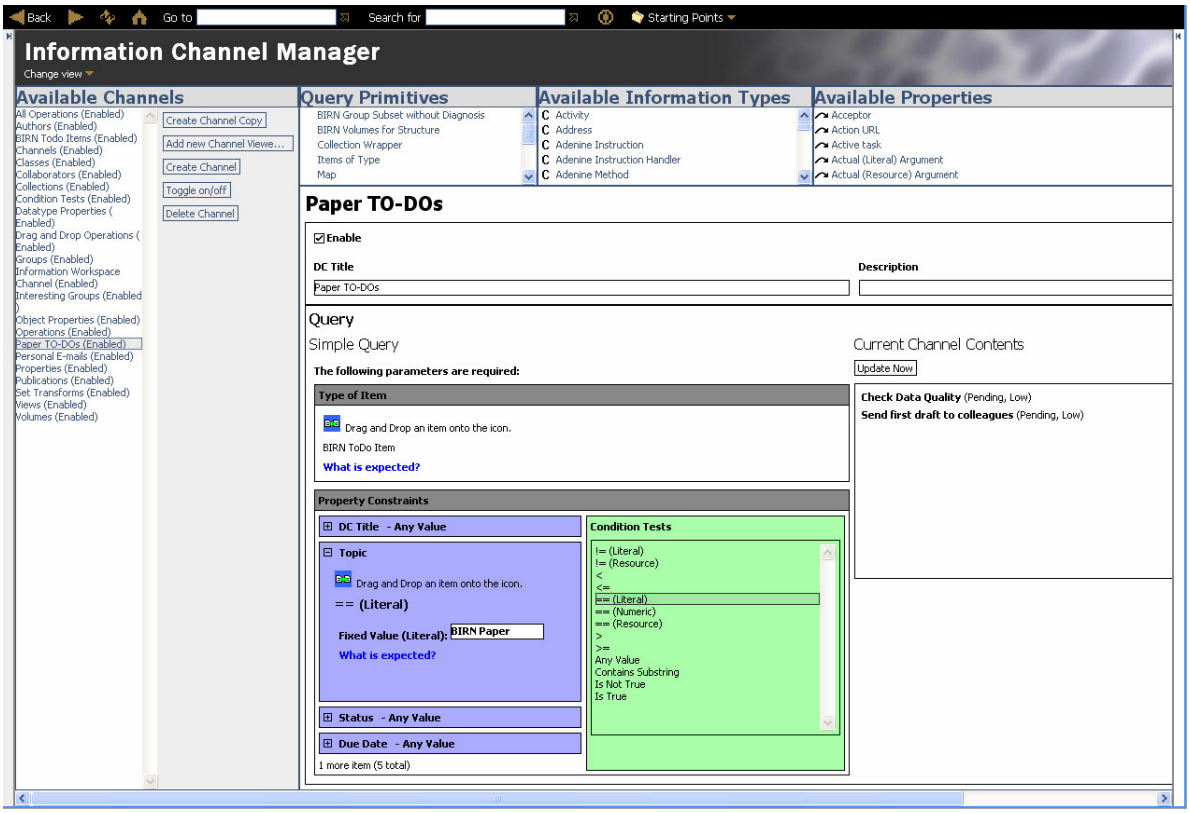


**Fig. 3.** Information Channel Manager

## 3 Related Work

In this section, we attempt to highlight some important milestones that have marked two orthogonal trends that collectively have a significant bearing on our work: the evolution of information management workspaces from collections of application windows managed by window managers to applications with task focused interfaces, and the increasing sophistication and ability of end-users to move from merely configuring their information management tools to directly specifying the content, presentation and manipulation aspects of their information management tasks.

That users would require multiple information resources to complete a task became clear with the advent of the earliest windowing systems and toolkits used to build applications that could be simultaneously active on the desktop. 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 window managers, each providing users with assorted tiling and resizing semantics [2] (see [7] for a survey). In addition to window managers that allow flexible control over UI real estate usage, much work has also been done in allowing users to allocate 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.

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. The windowing systems for the Microsoft Windows and Apple Macintosh operating systems comprise two such commercial efforts that can provide one desktop

per user by default. Some flavors of Linux extend this feature set by allowing multiple virtual desktops per user that are persisted even across system reboots. A more involved research effort, Rooms, extended the concept of a virtual desktop further by allowing users 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). Rooms could share control panels of common tools and information. However, Rooms and other window management functionality operated at a coarse information management granularity; they allowed managing windows of entire applications as opposed to just the subset of an application's information and operations that were relevant to a particular task.

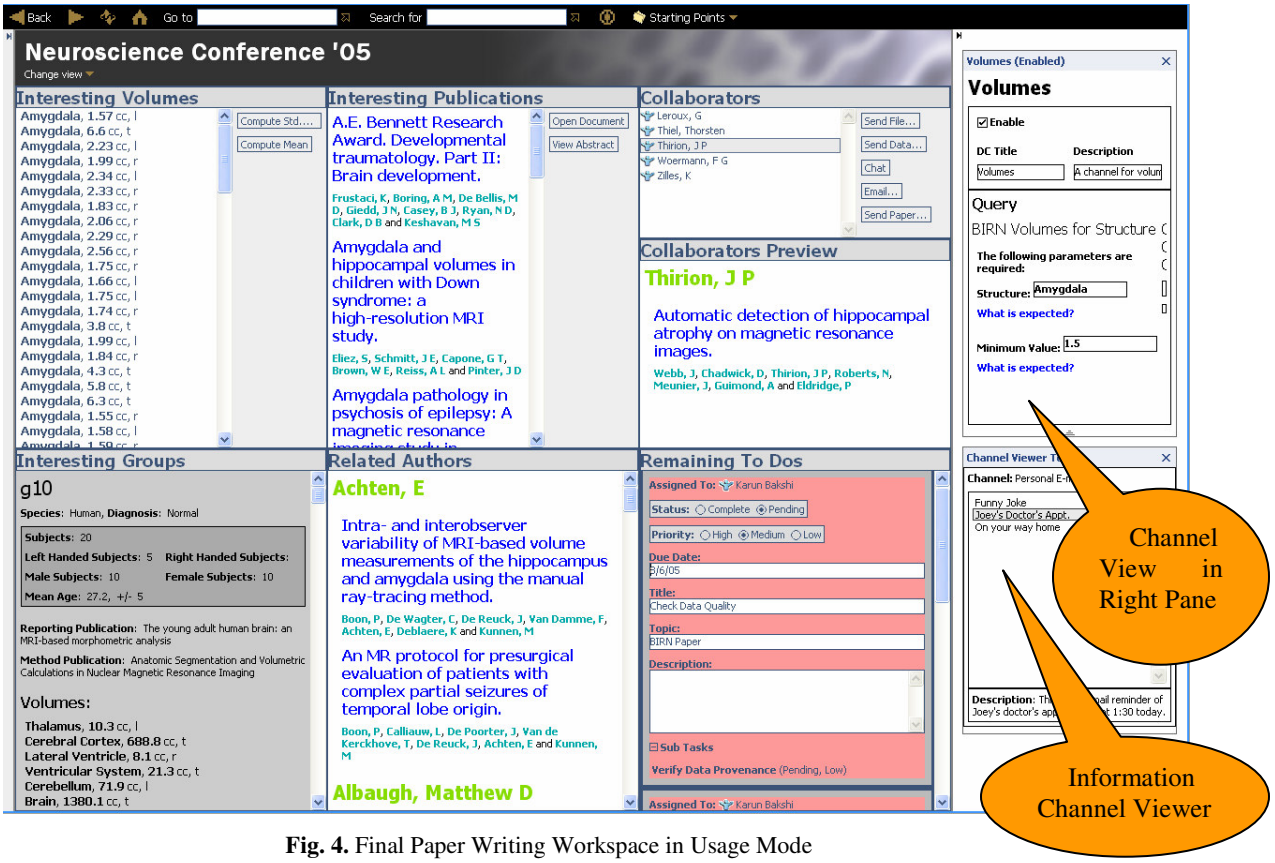


**Fig. 4.** Final Paper Writing Workspace in Usage Mode

As user tasks employing multiple applications or different parts of a single application have been elucidated, application user interfaces have progressed from merely allowing information manipulation to interfaces that are specifically designed to assist in a particular task; many window management ideas have seeped into single applications for tasks that used to require multiple applications, yielding interfaces with heterogeneous but focused subsets of information and operations juxtaposed in a single presentation and specifically tailored to a particular task. Taskmaster is a Visual Basic add-on for the Microsoft Outlook client that targets project management activities in mail clients [8]. It makes information 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 presentation or manipulation capabilities. Also, users have little queryable control over the content; the developers have already decided on the content queries whose results will populate the project interfaces. The Kubi Client is a commercial effort in the same vein [9].

Task workspaces can be created and maintained by users, in addition to developers. Web Montage employs a subtle, implicit approach to creating task interfaces; it observes users and uses machine learning

to build a personalized user web portal for web browsing [10]. However, the aggregated content is read-only, not manipulable. Similarly, the Microsoft Office suite of applications implicitly supports user tasks such as reconfiguring the user interface to only show recently used menu items by default, presumably employing the "most recently used" heuristic as a means to approximate the required task resources [11]. However, learned interfaces are error prone, and take time to adapt. Also, such interfaces do not respond well to interleaved tasks that force them to unlearn the settings from the previous task.

User sophistication has also increased over the years; applications that only allowed configuring of simple rendering preferences have ceded greater explicit control over the task interface creation process to the increasingly experienced average user. E-mail clients such as Microsoft Outlook, for example, support one aspect of task definition – content – by allowing user-defined rules that trigger based on message arrival or message sending events to maintain dynamic collections [11]. Hunter Gatherer is a recent system developed at the University of Toronto that simplifies capturing parts of an arbitrary web page (again, read-only content) into a contextualized collection [13]. Similarly, Presto, an interactive document management system, allowed defining tasks interfaces based on sets of relevant individual documents and other relevant, document collections, where each document can appear in multiple workspaces [21]. 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. Although they support greater user influence, these examples are limited by the range of task workspace aspects that can be controlled, i.e. they allow controlling just the content.

Other examples that allow control over the content, layout and manipulation capabilities exist, but are limited by the task domain. For example, content portal (e.g., *MyYahoo!* ) and news organization websites nowadays provide a rich set of primitives that can be used to create personalized webpages which allow users to filter and/or aggregate the content provided by the underlying organization(s) as well as specify its presentation for the news reading task [12]. Relevant operations such as search or e-mail are also close at hand. Microsoft Visual Studio, on the other hand, allows the layout of various fixed content panes to be controlled [30].

Generic workspace building tools for arbitrary domains also exist. With the advent of the World Wide Web and other queryable multi-media repositories that require direct user access, being able to shape the result set of information via a good query/result interface has steadily gained more attention. Delaunay$^{MM}$, a querying framework for distributed, heterogeneous, multi-media data stores allows users to specify a query and layout preferences for its results without *a priori* knowledge of the underlying schema or query language [14]. Similarly, Snap-Together Visualization (STV) allows users to query a relational database and link the result set visualizations in support of a complex information exploration task [15]. However, the STV system required users to understand database concepts and directly manipulate the MS-Access database and associated queries, making it less usable by lay people. Both Delaunay$^{MM}$ and the Snap-Together Visualization system represent simple user workspaces as users can specify content of interest, along with presentation preferences. Nevertheless, although both examples can be employed in arbitrary domains, they are constrained by a particular database instance (or small sets thereof) and its associated schema(s). That is, they cannot scale easily to capture multiple domains with heterogeneous or semi-structured data models.

Recently, the notions of task workspaces and user control over their specification have merged in the WinCuts project that allows users to aggregate only the relevant portions of source windows into a new task context [16]. Each WinCut is a live connection to the source window and can redirect user interactions to it. Thus it allows visualization and interaction with information fragments as in the original application, but it does not provide direct access to the underlying semantic entity. Nevertheless, it advocates and supports the need for users to work with subsets of information found in arbitrary applications.

With the Semantic Web in its formative stages, much attention has been paid to (ontology and content) authoring on the Semantic Web as was paid in the original World Wide Web [17, 19, 20]. Recently however, more attention is being devoted to using semantics to enhance the end-user browsing and navigational experience for specific domains [18, 22]. General UIs for Semantic Web browsing have also evolved from simple RDF graph viewers [26] to semi-structured data browsers [27]. With the content on the Semantic Web anticipated to be annotated at a much finer granularity than documents (as in Presto) and the ability to capture and relate arbitrary domains, we foresee an opportunity to merge task workspaces, user control by leveraging semantics to allow creating task workspaces that allow semantic rather than pixel-level interaction. It is this vision for information management on the Semantic Web that we hope to

realize with our system. In the process, we borrow from many of the ideas discussed above and learn from their limitations to inform our approach and implementation.

## 4 Approach

The software project manager's example starkly illuminates the gap between how users think about the task and supporting information (either because of personal preferences, task needs, etc.) and how supporting applications are designed based on *a priori* developer assumptions for a limited range of user, task and domain scenarios that rigidly define the boundaries for what can be accomplished. The design and implementation of these applications fail to take into account the fluid nature of information management tasks; the required information, visualization and operations change as different users define a particular task differently, tasks definitions change over time (e.g., to include information from other domains), varying levels of expertise require either more or less information to complete the task, and users may have different preferences for visualization and interaction. Yet, the supporting tools are static in nature, rigidly bottling up information in application containers. Information producers have most of the control over how information is packaged, presented and manipulated. The information consumer (whose productivity these decisions significantly affect) has little say in these decisions.

We posit that giving users control over three primary aspects of a task definition will grant them sufficient power to eliminate many of the shortcomings of the current information management infrastructure. Users need to be able to define and modify the content, presentation and manipulation capabilities of their task workspaces.

The information content that users require, devoid of its presentation, is perhaps the most critical ingredient of the task that the user is attempting to accomplish. Yet, users often lack the ability to easily select the subset of the information corpus to show and manipulate in a given context. On the Semantic Web, this issue will have even greater import 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; the authoring ontology should not restrict the applicability of the information.

User control over information presentation is also crucial, and a significant portion of the functionality in many applications today is devoted to allowing users to modify the presentation of the information in a manner consistent with how they want to manipulate it. We consider the problem of information presentation as consisting of two parts: specification of high level layout of multiple types of information and specification of the view used to display a particular information entity. The layout capability relates to the ability to aggregate and co-locate arbitrary information from various sources – a capability that would have allowed the software project manager in our earlier example to avoid "hopping" across applications as relevant information could have been aggregated into one interface. Users should also be able to select or create views of the underlying information that are relevant to the task at hand. Given the same information, not everyone manipulates it or visualizes it in the same way either. 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.

Finally, many decisions about operations (how the information is to be manipulated) associated with the information are made by information producers: certain operations may only be available in certain views, or have certain means of access (e.g., toolbar, menu item, etc.), force re-entry of the same parameters each time they are invoked, etc., and users must thus adapt how they perform a task based on this static aspect of applications. The user, who actually determines the frequency of the operation invocation based on his/her task, has little say in these important decisions.

In order to provide users with the type of flexibility to create fluid task workspaces that can take advantage of the rich semantic and application-independent information on the Semantic Web as soon as it becomes available, what is needed are small units of user interfaces, and application logic that they can easily combine to yield larger, more powerful task interfaces. We have developed several tools that enable the creation and stitching together of such reusable application fragments in order to provide users with greater control over the three high level aspects of tasks discussed above.

Our tools are situated in Haystack, a general information management platform for the Semantic Web that provides many of the building blocks we require at the system level. We discuss the important design and implementation details of Haystack and of our tools below.

## 5 System Design and Implementation

In this section, we discuss some of the important design decisions we made and implementation techniques we employed in building the various tools that support end user workspace creation. The primary components include a workspace designer that allows aggregating and laying out custom content and operations into a workspace, an information channel manager that makes it possible to specify queries for dynamic collections of related information, and a simple view designer that allows users to specify how to present underlying Semantic Web entities. But first, we briefly discuss the desirable ideas embodied in the Haystack platform and the building blocks it provides for our tools.

### 5.1 Haystack Platform

Haystack encompasses and makes available several key ideas and components that support creating, visualizing and manipulating Semantic Web content. At its core, Haystack employs a single data model consisting of a semantic network expressed using the Resource Description Framework (RDF), the standard for knowledge representation on the Semantic Web. It provides a language (Adenine) for simplifying expression, manipulation and querying of RDF data. Furthermore, imperative Adenine code that manipulates the data can itself be compiled into RDF data using a target, portable, runtime ontology akin to Java bytecodes, thereby rendering a majority of the Haystack system declaratively specified. Adenine serves as the lingua franca of the system, enabling communication between (and implementation of) its various components via the generic blackboard-like RDF store. Haystack can also host services that automate various tasks, e.g., categorization, summarization, extraction, learning, recommendation.

Haystack provides a user interface framework that recursively renders views of an information entity by rendering the views of other information entities that comprise it. It consists of the Slide Ontology, an extensible, HTML-like ontology for content layout and rendering that is used to create views of information entities. An information entity is accessed by browsing to it (i.e., Haystack navigates to the underlying Uniform Resource Identifier on the Semantic Web). A view for an entity is automatically selected from the pool of applicable views by Haystack based on the entity's type, and the context of use (e.g., available UI real estate). Furthermore, the UI framework supports context sensitive manipulation such as context menus and drag-and-drop operations. Finally, imperative code in Adenine can be exposed to the user and invoked by him/her via operations. Operations are parameterized Adenine methods that perform a task for the user using the specified parameters. Relying on metadata annotations on the operations themselves, Haystack employs an automated technique (UI Continuations) for instantiating instances (closures) of the Adenine method calls and collecting the relevant parameters from users. Also, operations may be curried, i.e., the user may customize an operation by specifying values for some, but not all, parameters, but not all [6]. The resulting curried operation can be saved as a new operation, and used as a template for applying the operation in new contexts, that all share the same value for the saved parameter, but require the remaining parameters to be specified during each subsequent invocation.

### 5.2 Workspace Designer

The workspace designer in Haystack allows creating, copying, deleting and configuring instances of task workspaces. A task workspace in Haystack is data. It follows a certain ontology that captures the layout of the comprising winlets as well as their associated properties. Each winlet allows the user to specify the content to show, how to present it, and which operations should be co-located (corresponding to the three aspects of task spaces we sought to give users greater control over). A Boolean flag allows disambiguating how the underlying content is to be interpreted: as itself or the entity it represents, e.g., should the underlying query parameters be shown, or its results? The view specified by the user is then applied to the appropriate interpretation. Finally, operations that the user specifies can also be curried versions of other operations.

A task workspace has two primary views: a design view that lets users edit its properties, and a usage view which interprets its properties to render a task workspace. It is rendered by recursively rendering each of the winlets using the appropriate design and usage views. Users can iteratively prototype a workspace by

switching between its design and usage views. Finally, workspaces, like channels and views can be copied to avoid having to start a new one from scratch.

Several important design decisions were made in the design of the workspace functionality. Perhaps the most important design decision that affected the construction of workspaces by users was how to allow users to allocate space to winlets. We chose to facilitate a tiled layout engine rather than allowing free floating windows. 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. 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 winlet rather than place them in the context menu in order to minimize user clicks when invoking them. Arguably, the tiling functionality could be made more sophisticated and using smaller icons for the operations would minimize the wasted space.

We chose to allow modality in workspace interaction (design vs. usage modes) primarily because the act of specifying underlying content for a winlet was most effectively done by using drag-and-drop, which required modality in order to be able to disambiguate between the act of specifying the underlying content and dragging an item onto the winlet in usage mode. Thus, we felt it would be easier for the user to have a clear separation of modes rather than be distracted by extraneous mode specific widgets which would also waste UI real estate.

### 5.3 Channel Manager

Whereas a particular entity can be specified to be shown in the workspace explicitly via a drag-and-drop operation, and Haystack supports creation of collections by the explicit specification of its members by the user, the channel manager allows an implicit collection specification mechanism by providing the user an interface to query the underlying RDF store. As in Presto, collections defined by channels are self-maintaining, organizing mechanisms 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. Thus, given a store being modified by the user, agents and the arrival of new information, the user can create a stream of information that is important to him/her independent of the source or creation method of the information. Channels can then be treated as standalone units of content that can be used to specify the underlying content of a workspace winlet.

The Channel Manager allows creating, copying, deleting and configuring channel descriptions. A channel in Haystack, like task workspaces, is also data. It follows the channel ontology, which captures whether its title, description, whether it is enabled, its query, and the current query results. The channel shown in the channel manager earlier showed its editing view in the bottom right of Figure 3.

The Channel Manager functionality can be fundamentally divided into the processing and user interface components. The processing component can be further subdivided into the channel manager agent and the set of available query primitives. Channels are computed by a channel manager agent (running as a Haystack service) that periodically updates all channels by evaluating their queries. A query is simply an Adenine function with various parameters. The query primitives available to the user always return a set of URIs of matching entities. Any new Adenine method that meets the above constraints can be exposed as a query primitive to the user and immediately interoperate with the channel manager agent using appropriate annotations, thereby allowing the set of query primitives to be extended. Dragging a query primitive to a channel results in a new instantiation of that Adenine method call: a new query primitive closure. The channel then refers to this closure as its underlying query. The view for a channel consists of sub-views of the underlying query primitive instantiation that allows the user to modify its associated parameters.

Current RDF query primitives allow using the associated metadata annotations of the underlying RDF to "carve" the information by selecting out only the relevant portions. The simple conjunctive boolean query

primitive which allows the user to select entities that meet constraints on literal valued properties is perhaps the most common query primitive. Users can specify string matching, numeric relation, or boolean typed condition tests for the literal valued properties; items matching all the conditions (boolean conjunction) satisfy the query definition. The conditions tests, like the query primitives, are also special types of Adenine methods and can be similarly extended. Another useful primitive is the Map primitive that allows mapping a property over all members of a source channel to generate a new channel of entities related to entities in the first channel via the mapped property.

Along with other RDF query primitives that enable simple and common RDF querying use cases such as selecting items of a particular type, etc., a set of channel operator query primitives (e.g., set union) are also available that 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 terminology simply by writing the query code, and, if the default view is deemed too esoteric, designing a custom view to prompt for relevant parameters.

The decisions to disallow duplicate entries, ordering preferences, and projecting properties of the returned items were made in the hopes of limiting the complexity of decisions a lay user would have to make. Also, we felt that specifying a single view that shows a particular set of properties for an underlying entity would be easier for the user than specifying how to view the tuples of property values resulting from a projection of its underlying properties.

### 5.4 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 operations (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 the semantics of the underlying information entity.

We developed a domain-independent, baseline view designer that lets users inspect any properties of any information entity in Haystack, by taking advantage of minimal information semantics. The view designer allows users to create metadata lens views that select and render values of properties (metadata) of interest from the underlying entity. The metadata lens views share the same layout engine as in the workspace functionality and also have similar design and usage views. Thus, users can segment a rectangular area into winlets, and instead of content, presentation and manipulation setting, specify a property to expose by dragging it to the appropriate winlet. Thus, each winlet represents a particular property to expose. The views created by users can be named and are then available in the workspace designer for future (re)use. They can be copied as a starting point for a new view or used as a sub-view. The only semantics the designer understands are that property values can be either literals or resources – the base assumption in RDF. However, even this simple capability becomes powerful, when combined with the power to reuse existing views in creating new views. 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. If more than one instance of the property exists for the underlying entity, all the values are shown.

## 6 Discussion

Coupled with the relative simplicity and power of expressing information in arbitrary domains using RDF and the view abstraction in Haystack, the aforementioned tools situated within Haystack make significant gains toward our goal of empowering users to uniquely conceive an information-based task on the Semantic Web and build a corresponding workspace.

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. In fact, the tools we have discussed so far were indeed implemented as workspaces; they aggregated the necessary information resources for building channels, views and workspaces. (Look closely again at the channel manager and

note that it consists of winlets showing various collections of information that are relevant to the channel building task, e.g., query primitives, information types, etc. Fig. 5 shows the same channel manager's design view.)

The workspaces that users can design feature several desirable properties. For example, the navigate operation in Haystack can be curried with a destination workspace as a parameter, and added to a workspace to allow users to navigate between related tasks, much as in the Rooms project. More generally, the default Haystack single click behavior of navigating to the underlying entity when it is rendered using a particular type of view can be used to easily switch to a different workspace or information entity by simply adding a collection of such items to any workspace, and clicking on them as necessary. 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, condition tests, layout engines, views or view designers. 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.

Haystack also provides a number of benefits which we discuss below. The RDF data model immediately enables it to model information in multiple domains as well as interoperate with the Semantic Web by default. Additionally, since all system components are written in Adenine which itself can be compiled down to RDF, the system implementation itself is simply data that can be manipulated by the user. Thus, when users are changing the data in a workspace, they are reconfiguring Haystack's programming, in effect (re)programming it. By default, Haystack employs a single blackboard style store that allows content shared across multiple workspaces to remain consistent. Pervasive support for context menus and drag-and-drop facilitate a uniform interaction modality to all task workspaces.

We feel that the notion of channels as persistent queries whose result set is always current will be an important one as users are forced to deal with more information 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 query description. Thus, each individual item need not be annotated with certain properties; the fact that an item has certain properties that allows it to be a member of a channel, also allows it to (virtually) "inherit" the properties of the channel dynamically. Modularity also allows information channels to be reused in different contexts and redirected to different portions of the UI (within, or outside a task workspace) 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. For example, the notion of viewing e-mails related to a particular topic regardless of which e-mail account it may have arrived in is a powerful one. 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. For example, a user working on a task requiring information from a set of channels need not be interrupted by newly arrived (or created) information that does not fall into any of the channels.

Having implemented the simple view designer, we recognized that more powerful views can be designed by view designers that better understand the domain specific 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, while preserving valid data. For example, semantics used to interpret information can be used to allow users to choose between styles of views, e.g., a list of genetic bases (Adenine, Guanine, Thymine, Cytosine) can be interpreted appropriately by a view designer to allow the user to specify a preference of whether just the genetic sequence, or its complement is also to be rendered. Unlike the metadata lens view designer which uses a tiled layout and simply exposes property values for literal properties, such a view designer could allow users to select domain-specific preferences

such as laying out the genetic bases simply in a horizontal line as opposed to a helix, the colors to use for the genetic bases, etc.

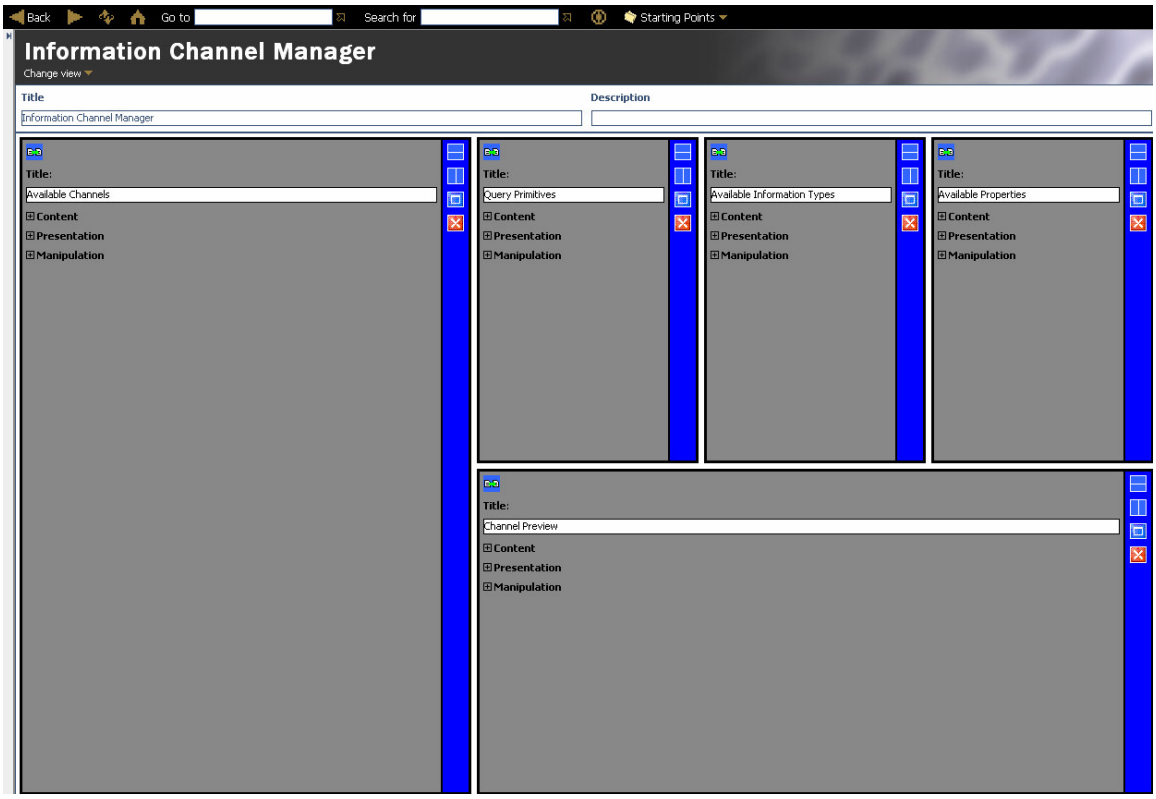


**Fig. 5.** Design View for the Information Channel Manager workspace

## 7 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 the Semantic Web immediately, without having to wait for a corresponding application to be developed. We have identified task workspaces as simultaneously an application of the Semantic Web, as well as a solution to information management problems it will complicate. In addition, we have attempted to identify three 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. Furthermore, we recognize that these aspects of workspaces can be captured in reusable fragments: channels as units of content, views as units of presentation, and operations as units of manipulation. Whereas many of their constituent ideas and implementation techniques are not new, we believe that the tools discussed above are unique in that they embody all of them simultaneously. We believe that such tools that help people use the information available on the Semantic Web will be an enabling technology critical to its widespread adoption.

Our future work will concentrate on evaluating the utility of our tools via user studies, further investigating and refining the customizing capabilities that users need in building their workspaces, as well determining the best techniques to make such capabilities available (e.g., better UIs for querying RDF, other usability improvements, etc.). In addition, we hope to investigate other enabling infrastructure technologies that we anticipate will be required to make the Semantic Web a success and support task workspaces, including ontology translation servers that can translate between competing ontologies and prevent ontological “islands,” as well as view servers that can be used to share views that users have built for manipulating various information types.

## 8 Acknowledgements

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. http://office.microsoft.com.
12. *MyYahoo!* http://my.yahoo.com.
13. Schraefel, M., and Zhu, Y. Hunter Gatherer: A Collection Making Tool for the Web. Extended Abstracts of the Conference on Human Factors in Computing Systems 2002.
14. Cruz, I., and Lucas, W. A Visual Approach to Multimedia Querying and Presentation. Proceedings of the fifth ACM International Conference on Multimedia, 1997.
15. 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.
16. 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.
17. 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.
18. 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.
19. 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. Sure, Y., Erdmann, M., Angele, J., et al. OntoEdit: Collaborative Ontology Development for the Semantic Web. Proceedings of ISWC 2002, p. 221.
21. 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.
22. 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.
23. Quan, D. and Karger, D. How to Make a Semantic Web Browser. Proceedings of WWW 2004.
24. Quan D., Karger, D. and Huynh, D. RDF Authoring Environments for End Users. Proceedings of the International Workshop on Semantic Web Foundation and Application Technologies, 2003.
25. 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.
26. http://simile.mit.edu/welkin/
27. http://simile.mit.edu/longwell/
28. Quan, D., Huynh, D. and Karger, D. Haystack: A Platform for Authoring End User Semantic Web Applications. International Semantic Web Conference, 2003.
29. Quan, D. and Karger, D. Haystack: Metadata-Enabled Information Management. UIST, 2003.
30. Microsoft Visual Studio. http://msdn.microsoft.com/vstudio/
31. The Internet Brain Volume Database. http://www.cma.mgh.harvard.edu/ibvd/