A User Interface Framework for Supporting Information Management Tasks in Haystack

by

David François Huynh

Submitted to the Department of Electrical Engineering and Computer Science

in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2003

Author .....
Department of Electrical Engineering and Computer Science
May 23, 2003

Certified by .....
David R. Karger
Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by .....
Arthur Smith
Chairman, Departmental Committee on Graduate Students

A User Interface Framework for Supporting Information Management Tasks in Haystack

by
David François Huynh

Submitted to the Department of Electrical Engineering and Computer Science
on May 23, 2003, in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

The means for satisfying basic information management needs in today's operating environments come from the amalgamation of support implemented by individual, disparate applications. Since each application is more than often developed independently of the others, the union of their support still leaves wide gaps in which users' information management needs are not provided for.

This thesis advocates that the support for basic information management tasks be provided by the infrastructure of any environment, so to expose a uniform experience for managing information. Five different aspects of information management needs are identified to serve as the focus for our initial investigation: expressing, perceiving, locating, manipulating, and organizing information.

From these five aspects, principles for building information management environments are suggested. A centralized, expressive data modeling framework is advocated as being crucial to achieve coherency in the user interaction experience. View-based presentation schemes are proposed for flexibly projecting internal data to the UI. And pervasive support for object-oriented UI manipulations is called for to make the interaction experience scalable.

A user interaction experience for our Haystack information management platform is then designed based on those principles. Finally, the design rationales and implementation details of Haystack's user interface framework are discussed.

Thesis Supervisor: David R. Karger
Title: Professor of Computer Science and Engineering

Acknowledgements

I would like to thank all members of the Haystack group whose helpful comments, ideas, and discussions helped shape many aspects of my research.

I thank my advisor David R. Karger for his insights and guidance, and for many hours of tireless discussions on the topics relevant to this thesis.

I would like to thank Nippon Telegraph and Telephone Corporation and IBM for their financial support.

I would like to thank all of my friends for their encouragement and support.

Last but absolutely not least, I thank my parents for all their sacrifices and encouragements, and for their belief in me.

Table of Contents

CHAPTER 1 Introduction 13

1.1 The Information Management Tasks 14

1.2 Expressing Information 15

1.2.1 Incomplete Schemas 15

1.2.2 Lack of Synergy Between Schemas 17

1.2.3 Lack of Generic Schemas 17

1.2.4 Lacking Naming Schemes 18

1.3 Locating Information 18

1.3.1 Indirect Access 19

1.3.2 No Single Starting Point 19

1.3.3 Limited Query Support For Non-First-Class Information 20

1.3.4 No Query Support For Cross-Domain Information 20

1.4 Perceiving Information 20

1.4.1 Different Renderings 21

1.4.2 Missing Information 23

1.5 Organizing Information 23

1.5.1 Segregation by Type 23

1.5.2 Different Mechanisms for Organization 24

1.5.3 Mixing Concepts of Access Location and Classification 25

1.5.4 Organization Not Sharable 25

1.6 Manipulating Information 25

1.6.1 Inter-application Inconsistencies 26

1.6.2 Intra-application Inconsistencies 28

1.7 Problem Statement 29

1.8 Structure of Thesis 30

CHAPTER 2 Background Work 31

2.1 Intra-application Solutions Through Automation 31

2.1.1 User Interface Management Systems (UIMSes) 31

2.1.2 Model-Based Interface Designers *32*
2.2 Intra-application Solutions Through Design 33
2.3 Inter-application Solutions Through Integration 34
2.4 Inter-application Solutions Through Re-architecture 34
2.4.1 The Lisp Machine *34*
2.4.2 Object Brokerages *35*
2.4.3 The Web *36*
2.4.4 The Semantic Web *39*
2.4.5 The Views System *40*

CHAPTER 3 Approach 43

3.1 Principles 43
3.1.1 Refactor support for information management tasks *43*
3.1.2 To achieve coherency, use a centralized, expressive data modeling framework. *44*
3.1.3 Present information in the forms fitting the nature of that information *45*
3.1.4 Support object-oriented UI manipulations pervasively *46*
3.2 Haystack's User Interaction Experience 47
3.2.1 Expressing Information *47*
3.2.2 Perceiving Information *48*
3.2.3 Locating Information *49*
3.2.4 Manipulating Information *50*
3.2.5 Organizing Information *51*
3.3 Contributions 52

CHAPTER 4 The Haystack Platform 55

4.1 Architecture Overview 55
4.1.1 RDF Stores and RDF Containers *56*
4.1.2 Services and Agents *60*
4.1.3 User Interface Framework *60*
4.2 Basic Data Models 61
4.2.1 Collections *61*
4.2.2 Lists *62*
4.3 Adenine 64

CHAPTER 5 User Interface Framework Design Rationales 67

5.1 Information Transformations 67
5.2 View Abstraction 69
5.2.1 An Analogy 70
5.2.2 View Request Specifications 71
5.2.3 Persisting View Requests 72
5.3 Rendering Abstraction 73
5.3.1 UI Elements 73
5.3.2 Persisting Rendering Requests 74
5.3.3 Composing Rendering Requests 75
5.3.4 Mixing View Requests with Rendering Requests 75
5.4 Layout Abstraction 76
5.5 Data Computation Abstraction 77
5.6 The Transformation Pipeline 79
5.7 Transformation Trees 80
5.8 Environments 82
5.9 Optimizations 83

CHAPTER 6 User Interface Framework Implementation 85

6.1 Component Architecture 85
6.1.1 Registration of Evaluators 86
6.1.2 Evaluator Java Interface 89
6.1.3 Context 89
6.2 UI Elements 91
6.2.1 Event Passing 91
6.2.2 Layout Negotiation 91
6.3 View Producers 94
6.4 Data Producers 97
6.5 Supporting Direct Manipulations 101

CHAPTER 7 Usage Scenario and User Study 103

7.1 Points to Illustrate 103
7.2 Scenario 105
7.3 User Study Design 116
7.3.1 Design Rationales 116
7.3.2 Procedure 121

7.4 User Study Results and Discussion 123
7.4.1 General Observations 123
7.4.2 Numerical Data 124
7.4.3 Written Responses 132
7.4.4 Conclusion 133
CHAPTER 8 Future Work 135
8.1 Component Architecture 135
8.2 Views 136
8.3 UI Elements 137
8.4 Data Computations 138
Appendix A – User Study Instructions 139
Appendix B – User Study Survey 141
Appendix C – Survey Written Responses 145
References 153

List of Figures

Figure 1. Specifying the artist of an audio track in Winamp 3.0 16
Figure 2. Specifying the author of a document in Microsoft Word 16
Figure 3. Specifying the department of a contact in Netscape 7.0 Mail & Newsgroups 16
Figure 4. Locating the master list of categories in Microsoft Outlook 2002 19
Figure 5. Viewing file attachments while composing an e-mail message in Netscape 7.0 Mail & Newsgroups 21
Figure 6. Viewing file attachments while composing an e-mail message in Microsoft Outlook 2002 22
Figure 7. Finding the total size of several files in Windows XP Explorer 22
Figure 8. Properties displayed by Windows Media Player 8 23
Figure 9. Java files are displayed with warning flags in IBM's Eclipse 2.0 23
Figure 10. Organizing an e-mail message in Netscape 7.0 Mail & Newsgroups using context menu. 24
Figure 11. Context menu of an image file in Jasc Paint Shop Pro 7.0 26
Figure 12. Context menu of an image file in Microsoft Windows XP Explorer 27
Figure 13. Context menu of a file in Microsoft Windows XP Explorer 27
Figure 14. Context menu of an attached file in Microsoft Outlook 2002 28
Figure 15. The Properties dialog in Microsoft Word 2002 29
Figure 16. Example of an RDF statement about an RDF predicate 47
Figure 17. Haystack's user interface 50
Figure 18. A sample context menu 51
Figure 19. The Organize tool 52
Figure 20. Architecture overview 56
Figure 21. A sample implicit collection 62
Figure 22. A sample explicit collection 62
Figure 23. A sample DAML list 63
Figure 24. A sample non-empty mutable list 63
Figure 25. A sample empty mutable list 64
Figure 26. An initial illustration of the transformation pipeline 70
Figure 27. A second illustration of the transformation pipeline with rendering requests and UI elements included 74
Figure 28. The transformation pipeline 80

Figure 29. A transformation tree 81
Figure 30. Association between a rendering request and a UI element 88
Figure 31. Hierarchy of nested UI element instantiations inherent in a transformation tree 92
Figure 32. Resolving a super class view request to a view class 95
Figure 33. Generating a specific class view request from a view class 96
Figure 34. Scenario: reading Alisa's e-mail message 107
Figure 35. Scenario: Creating a shortcut to the map to Celine's home 108
Figure 36. Scenario: invoke the Rotate command on a picture 108
Figure 37. Scenario: choosing direction to rotate picture 109
Figure 38. Scenario: composing an e-mail to Ben 109
Figure 39. Scenario: attaching Celine's portrait to e-mail message to Ben 110
Figure 40. Scenario: resizing a picture 111
Figure 41. Scenario: classifying a song into a project 112
Figure 42. Scenario: classifying Alisa's e-mail message into more than one category 113
Figure 43. Scenario: Seeing more information on Celine's portrait (actual full scale photograph omitted to observe copyright) 114
Figure 44. Scenario: viewing information of the photographer of Celine's portrait 115
Figure 45. Scenario: browsing more photos of Celine 116
Figure 46. Comparison of subjects' performances on Haystack and Windows 127
Figure 47. Breakdowns of subject pool based on whether particular tasks were skipped 129
Figure 48. Subjects' responses to how often they need to perform particular types of task 131
Figure 49. Subjects' answers to whether Haystack or Windows was easier, or both were equally easy, for particular tasks indicated in six survey questions 132

List of Tables

Table 1. Shortcomings of existing information management environments 30
Table 2. Comparisons between several inter-application solutions by re-architecture 42
Table 3. Specific types of prescription and evaluator 85
Table 4. Timings of subjects' performances on two environments 125
Table 5. Subjects' performances with respect to first environment used 126
Table 6. Recordings of whether subjects skipped particular tasks 128
Table 7. Subjects' responses to multiple-choice survey questions 130

CHAPTER 1

Introduction

On the surface, the personal computer appears very flexible, providing utility in a multitude of domains: every computer user is engaged in at least some combination of Web surfing, e-mail and instant messaging, contact information management, document authoring, and multimedia interaction. By acquiring more software packages, a user can even manage recipes, peruse encyclopedias, publish professional-looking brochures, play games, design home interiors, compose music, etc. all on the same machine. When taken together as *information management environments*, these software applications and the operating systems on which they run seem versatile and powerful.

At the same time, it is common knowledge that the general-purposed personal computer is difficult to use. In this introduction, I discuss the various shortcomings that the existing information management environments exhibit with regards to their information management functionalities. In particular, I will show where they are lacking in providing capabilities for performing information management tasks, including expressing, perceiving, locating, manipulating, and organizing information.

The description of these shortcomings serves as the basis for forming my problem statement in section 1.7, which is an endeavor to determine the set of basic functionalities that the infrastructure of any information management environment should support to address these shortcomings in a manner consistent throughout that environment.

The end of this chapter lays out the structure of this entire thesis.

1.1 The Information Management Tasks

The sources of difficulties in using computer software are numerous, but they can be summed up in a nebulous observation that software either does not support some tasks that users want to perform, or performs certain tasks not in accordance with users' expectations. In considering the collective software on a personal computer as an information management environment, we recognize that the fundamental tasks in that environment are the tasks of managing information. These tasks include perceiving, expressing, manipulating, locating, and organizing information.

I do not intend to classify every information management task into one of these five types exclusively. These different types are meant to be different aspects of a task on which we can focus. A task that involves manipulating information almost certainly involves expressing information; however, I may want to emphasize on only one of the two aspects to highlight certain points during a certain part of my discussion.

In fact, these five aspects can also be used to provide different perspectives on any task, not just the tasks readily recognized as involving information management. This is because complex, high-level tasks often break down into smaller tasks that inevitably involve managing information. The high-level tasks, hence, can be inspected from the five perspectives. For instance, the authoring of a scientific paper involves locating and digesting relevant works, organizing the topics in the paper, expressing associations between the relevant works with the topics, etc. Composing a marketing brochure involves collecting marketing facts, organizing them into a flowing presentation, selecting graphics, manipulating the graphics, etc. Holding a meeting requires locating the meeting participants, locating a free room, locating, digesting, and organizing materials needed for the meeting. From this observation that every task involves some of the five aspects of information management to some degrees, I argue that fundamental mechanisms for managing information should be provided in a consistent fashion throughout an environment before any task can be supported.

Of course, the manner in which a specific aspect of information management occurs in a particular domain may be unique within that domain. Seeking stanzas in an audio track is quite different from finding a file in a directory. Reading a city map is certainly different from reading DNA sequences. Drawing the architecture of a building is not the same as editing a text document. And organizing criminal investigation evidences might be much more involved than filing e-mail messages.

However, there are certain similarities in the information management tasks in different domains that warrant an environment-wide unified support for managing information, on top of which there might be domain-specific support. The task of locating all red gears in a mechanical drawing is analogous to the task of finding all MP3 files in the Adult Contemporary genre in a file directory. Both tasks involve narrowing the universes of objects by type (i.e. gear and MP3) and then by a property (i.e. color = red and genre = Adult Contemporary).[1] Likewise, both tasks of reading a city map and perusing an orchestra's score may benefit from a highlighting capability: the map layer encoding a particular feature (e.g. restaurants, parking lots, shopping malls) can be highlighted, as can the score of a particular instrument (e.g. clarinet, piano, guitar).

It is these common, environment-wide information management functionalities that I am interested in identifying and providing. But first, we must examine how these functionalities are lacking in existing information management environments. The next five sections describe several deficiencies in those environments. Some of these shortcomings, or similar ones, have been articulated elsewhere; some are my own observations. They have been herein classified into the five aspects of information management for the purpose of my discussion.

1.2 Expressing Information

Although there is a plethora of information domains supported by various software applications, expressing information within these domains or across them remains difficult in all existing environments.

1.2.1 Incomplete Schemas

The data schemas of certain applications appear incomplete. For instance, no audio software application allows users to manage contact information for audio artists. Only the names of artists can be inputted and they remain as text fields of audio tracks (Figure 1). Similarly, documents' authors, if supported by a file system, are just text properties of documents (Figure 2); and contacts' departments are no more than text properties of contacts (Figure 3). Because those are just text properties, users cannot specify more information about them, such as recording the work address of an audio artist, the phone number of a document's author, or the manager of a contact's department. The schemas in question seem rigid and restrictive.

[1] While type might be just another property in the view of system designers, I believe that users tend not to think of type as a property, because usual properties such as color can be changed (by painting over) whereas type remains immutable and fundamental to each object.

Figure 1. Specifying the artist of an audio track in Winamp 3.0

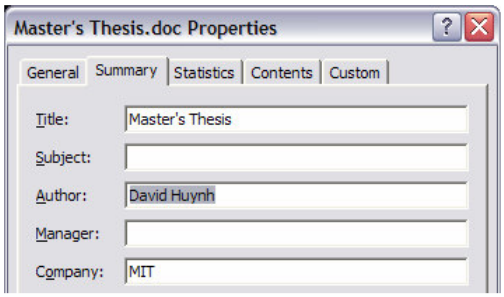

Figure 2. Specifying the author of a document in Microsoft Word

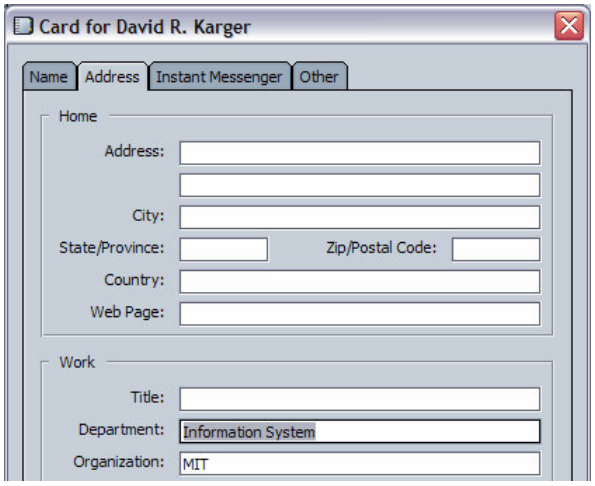

Figure 3. Specifying the department of a contact in Netscape 7.0 Mail & Newsgroups

The incompleteness of schemas is understandable: if audio artists were modeled as first-class objects in an audio software application, what else should be first-class? The artist's company, that company's president, the president's home address, the home address' map, the map's author, etc.? The chain of linked properties can go on indefinitely and the best solution for the designer of the audio software application is to cut it short before it crosses outside the domain of audio resource management.

While the designer of the audio software application would not take on the burden of modeling audio artists as first-class objects, there are other software applications that already consider persons, artists included, first-class: e-mail clients. Likewise, there exist some applications that model as first-class objects the various other types of information in the previous example: companies, addresses, and maps. Nevertheless, these applications cannot cooperate in such a way that their schemas can be merged: an audio track in an audio software application cannot be assigned an artist whose contact information is managed by an e-mail client.

As a result of the inability to merge schemas, users perceive inconsistencies in the functionalities offered on "person" objects in different applications. In one application, contact information can be recorded for persons; in another application, only names can be specified.

1.2.2 Lack of Synergy Between Schemas

In the example from the previous section, the relationship between an audio track and its artist is a close and common one. Most users would expect to be able to record the artist of an audio track. There are other relationships that are less common: an audio track's recording company, the top ten list on which the track has ranked, the radio channel that has aired the track, the user's acquaintances of whom the track is a favorite, the user's daughter's first reproduction of the track, etc. These are very specific pieces of information that might be meaningful to only some users. They capture a synergy between schemas that is rarely, if ever, allowed for in existing information management environments because they are too specific and too diverse to be planned for, and because they do not seem to fall into the responsibilities of any application involved.

1.2.3 Lack of Generic Schemas

Not only are too specific relationships neglected, too generic relationships are not supported either. There is no unified means for expressing that a piece of information, regardless of which application manages it, is "important" so that it should be kept easily accessible and guarded against accidental deletion. Likewise, it is not possible to group two or more pieces of information together to indicate their relatedness without recording their exact relationships, which may remain elusive even to the user or which may be too cumbersome and not useful to specify in details. (File directories can only group objects at the granularity of files. Instant messages, for example, are "smaller" than files and cannot be organized by a file hierarchy.)

1.2.4 Lacking Naming Schemes

Expressing information about a particular object requires referencing it. The naming schemes of existing environments are lacking in a major way: the name of each object (its reference) is most often the same as its access location. For instance, there is no other way to refer to a document except by its file path, i.e. how one accesses the document. If the document is moved to a different location, its access location is obviously changed, but so is its name. Even though the document still retains the same identity in the user's perspective, its name is now different. For one thing, this behavior is counterintuitive because it is not reflected in the physical world: a person still retains his or her identity and name even if he or she moves to a different address; whether a book is on a table or on a bookshelf, it is still the same book and is still called by the same name. If the book were to be called by different names in different locations—as it would be if it were stored on the computer, then it would be almost fruitless to express any information about that book inasmuch as such information would remain valid only until the book is moved.

The existing naming schemes also fall short in addressing minute objects, such as a shape in a CorelDRAW! vector drawing. In fact, there is simply no scheme for naming such a shape. As a consequence, there is no way to express information about it, except through very fixed UI mechanisms of CorelDRAW!. (Note that the shape is a first-class object in CorelDRAW!, but it still cannot be referred to elsewhere in the whole environment.)

This problem of inaddressable objects extends to our greater networked world. Microsoft Outlook 2002 refers to the current user's Inbox by the URL `outlook:Inbox` and the Sent Items folder of an opened Personal Folder Store file named "Archived Folders" as `outlook:\\Archive Folders\Sent Items`. Clearly, this naming scheme does not take into account the user's identity nor the machine on which Outlook is running. As a consequence, the naming scheme cannot address, say, the Inbox of another user on another machine. Any information expressed on the URL `outlook:Inbox` remains ambiguous as it can apply not to one unique entity but to several.

1.3 Locating Information

Not only are users restricted in expressing new information, they are also confined to awkward ways of locating existing information.

1.3.1 Indirect Access

Often, even if a piece of information can be located, it can only be located in a long winded way. For example, in Microsoft Outlook 2002, in order to arrive at the master list of categories, one must open the *Categories* dialog for any e-mail message through its context menu, and then click on the *Master Category List* … button (Figure 4). There is no shorter way for locating this list, and there is no way to create a shorter way. Unlike resources on the Web, dialog boxes cannot be bookmarked for quick access at later times. Users are constrained to the static UI routes for locating information and cannot rewire these routes to suite their information locating needs.

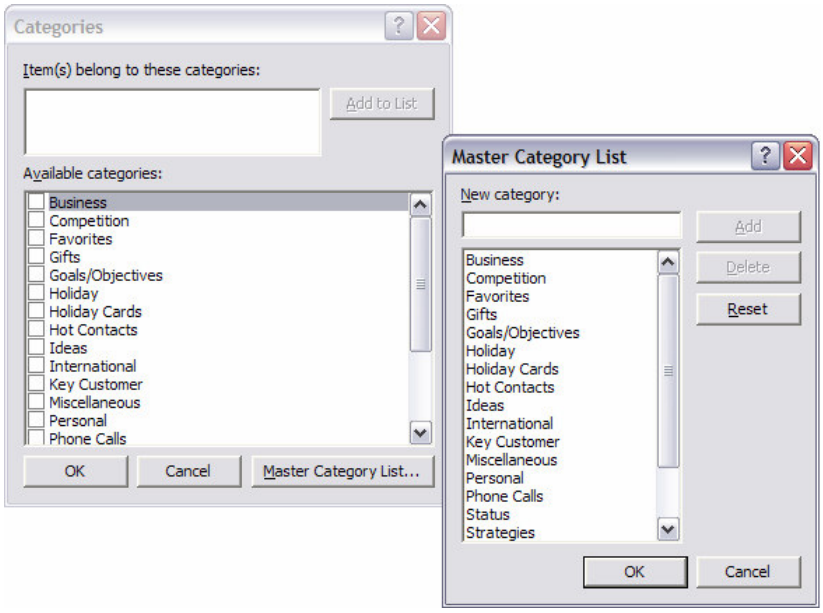

Figure 4. Locating the master list of categories in Microsoft Outlook 2002

1.3.2 No Single Starting Point

The process of locating a piece of information primarily consists of a sequence of browsing and querying steps. The choice of either browsing or querying should be left to the user as logically possible rather than restricted due to implementation limitations. However, in current environments, users are often required to browse first because there is no a single starting point for specifying a query: in order to find a text document, start by opening the operating system's file browser; in order to find an e-mail message, start the e-mail client. If a user wants to search for all e-mail messages *and* text documents containing a particular string, he or she must perform that query twice, once in the file browser and another time in the e-mail client. If the environment had a mechanism for specifying a query in which the type of the sought items can be specified with a disjunction, the user would have been able to achieve the desired results with just one query.

At the same time, some types of information can be searched starting at multiple points. For instance, a user can attempt to locate a particular MP3 through the file browser or through the audio software application, presuming all MP3 files have been imported into the audio software. However, these two starting points offer two different sets of search capabilities: only the file browser supports queries by modification date, and only the audio software application allows searching by the number of times an audio track has been played. The user may begin at one starting point and fail to find the functionalities offered only by the other starting point.

1.3.3 Limited Query Support For Non-First-Class Information

The inexpressiveness of data models discussed in section 1.2 gives rise to a particular deficiency in locating information: users cannot query for a collection of pieces of information that are not first-class objects. For instance, in most if not all address book applications, users can record the company of a person only as a text property. Given a contact, a user can locate the contact's company by opening some particular view of the contact. However, given several contacts, there is no mechanism for arriving at the collection of companies for whom those contacts work because each company cannot be located (or viewed) independently of the associated contact. Hence, in order to send such a list of companies to another user, one would have to manually copy out the names of the companies from the views of the contacts and eliminate duplicates as necessary.

1.3.4 No Query Support For Cross-Domain Information

The query support in existing environments is also lacking when it comes to cross-domain information. (This is partially because certain cross-domain information is not even expressible in current environments, as discussed in section 1.2.2.) For instance, it is not possible to query for the addresses of audio artists whose works are in one's collection. Even when all the cross-domain information involved is managed by the same application, querying is not supported thoroughly: it is not possible to specify a single query in Microsoft Outlook 2002 to find all e-mail messages sent recently by all participants of a given meeting. Half of the information involved is stored in the meeting module of Outlook, and the other half in the e-mail module. Querying across these two modules is not allowed.

1.4 Perceiving Information

When a piece of information can be located in more than one way, it may be displayed differently depending on how it has been located.

1.4.1 Different Renderings

Consider the order in which a name is displayed in Microsoft Outlook 2002. In the Contacts folder and on Contact forms, users can specify how names are to be displayed: first name first or last name first. However, in E-mail folders and on E-mail forms, such setting is entirely ignored: names are displayed the ways they have been encoded within e-mail address specifications. Consequently, there is no way to sort the messages in an E-mail folder by the last names of senders or by the first names of senders, as some senders specify their first names first, and some last names first.

For a second example, consider how file attachments are displayed in e-mail clients. When a file is attached to an e-mail message in Netscape 7.0 Mail & Newsgroups, it is displayed as an icon labeled with the file name (Figure 5). There is no mechanism for changing the way it is displayed. This can be inconvenient: suppose one would like to verify that one has attached the correct picture, it would be helpful to present the attached file with a thumbnail image because the file name (such as "DCP_0691.JPG" in Figure 5) might not immediately remind the user of the file's content. Very few e-mail clients, if none at all, can display picture attachments as thumbnails even though most, if not all, modern operating system file browsers can do so with ease. This shortcoming is clearly not a lacking of any particular e-mail client, but an epidemic flaw common among several programs.

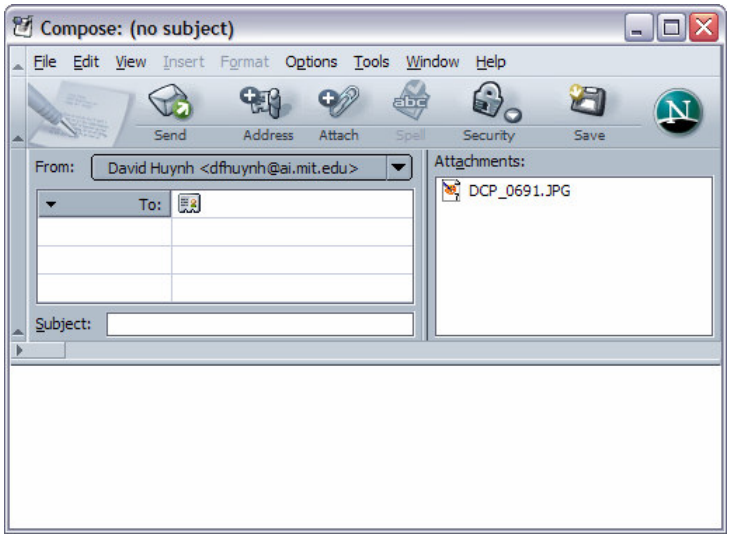

Figure 5. Viewing file attachments while composing an e-mail message in Netscape 7.0 Mail & Newsgroups

Microsoft Outlook 2002 does offer an improvement over Netscape 7.0 Mail & Newsgroups by displaying each attached file's size (Figure 6). However, it does not display the total size of all attachments, which is more useful than the sizes of individual files. If these files were selected together in the file browser of an operating system such as Microsoft Windows XP, their total size

would have been easily known to the user (i.e. "670KB" shown in the window's status bar in Figure 7). Even though file sizes are known to Outlook, it does not present such information in a form useful to users.

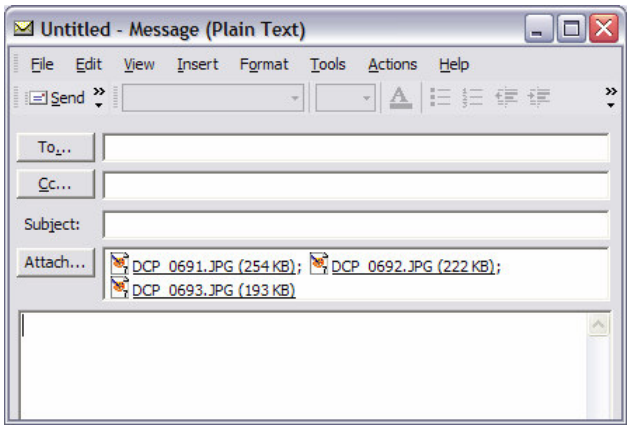

Figure 6. Viewing file attachments while composing an e-mail message in Microsoft Outlook 2002

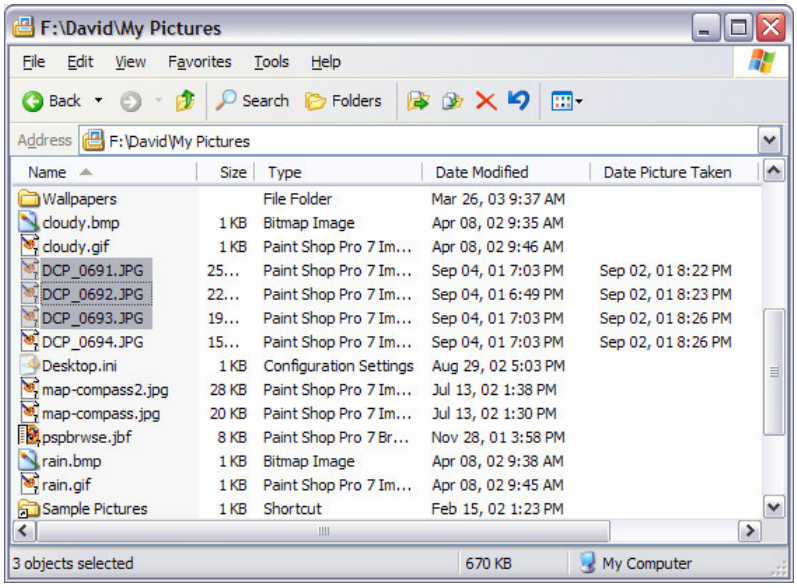

Figure 7. Finding the total size of several files in Windows XP Explorer

In addition to file sizes, other file properties are also readily available through file systems. However, they are not always used in presenting files. In Windows Media Player 8, each MP3 file is shown with several properties (Figure 8), but this set of properties lacks several intrinsic file properties such as author, date accessed, category, comment, etc. offered by NTFS. Even though Windows Media Player 8 can easily retrieve these properties and display them, it has decided not to.

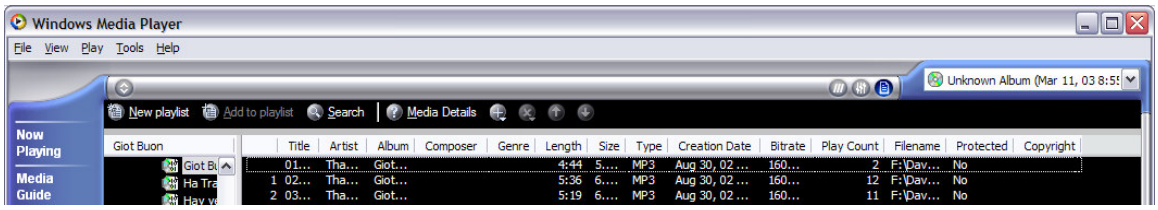

Figure 8. Properties displayed by Windows Media Player 8

1.4.2 Missing Information

Some details may not be available to a piece of code that renders a piece of information. For instance, while Windows Media Player 8 knows the number of time an audio track is played (its Play Count property in Figure 8), the operating system's file browser does not and hence cannot show play counts. Similarly, Java files are shown with warning flags in IBM's Eclipse 2.0 (Figure 9) but not anywhere else.

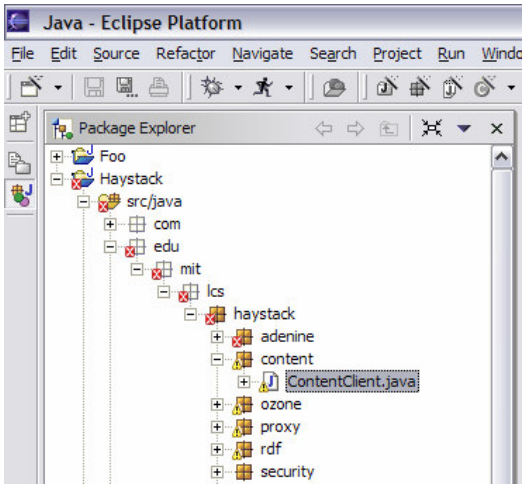

Figure 9. Java files are displayed with warning flags in IBM's Eclipse 2.0

1.5 Organizing Information

The various inconsistencies in the mechanisms for locating and perceiving information that we have discussed so far are also seen in the means for organizing information.

1.5.1 Segregation by Type

Pieces of information of different types often cannot be organized together: there is no way to file e-mail messages and instant messages in one common hierarchy of folders when the mode of delivery (instant or relayed) does not matter to the user. (In fact, few instant messaging applications, if any,

offer a means for organizing instant messages.) Similarly, it is not easy to organize e-mail messages and text documents together as they should be when, for instance, they are related by projects.

1.5.2 Different Mechanisms for Organization

The most common UI mechanism for organization is drag and drop. There are several alternatives to drag and drop. Netscape 7.0 Mail & Newsgroups allows the user to invoke the context menu of an e-mail message and select the destination folder. Most file browsers implement a different mechanism: files can be selected and then copied or cut and pasted into the destination folder through the operating system's clipboard. Netscape 7.0 Mail & Newsgroups does not support cut/copy-and-paste operations on e-mail messages and file browsers do not support specifying destination folders through context menus.

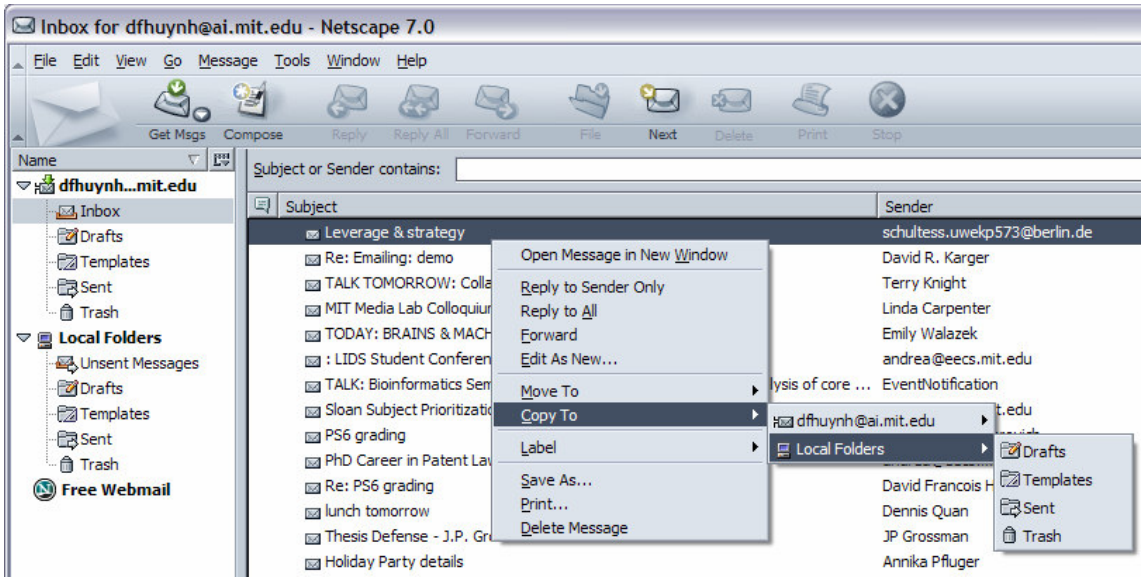

Figure 10. Organizing an e-mail message in Netscape 7.0 Mail & Newsgroups using context menu

If the user needs to explore the contents of various folders before filing information into one of them, then the cut/copy-and-paste mechanism works best since the selection of items to be filed is safely stored in the clipboard. If the user knows for certain which destination folder to file into and does not wish to switch away from the current folder, then the context menu mechanism is the choice. However, the user is not offered both mechanisms in both Netscape 7.0 Mail & Newsgroups and file browsers. As a consequence, organization may be more convenient in one application than in another even though the user's needs are the same.

1.5.3 Mixing Concepts of Access Location and Classification

A common method for digesting a corpus of information, i.e. for organizing its content in one's mind, is to classify items in that corpus. However, the mechanism for classification is often the same as that for access: in order to file away a document, one must move it into a folder. In doing so, one has changed the access location of the document. Similarly, filing an e-mail message requires removing it from the Inbox. Such an action may not be desirable as the user might want to keep the message in the Inbox to serve as a reminder for an unfinished task. Because the concepts of access location and classification are muddled together, organization through classification cannot take place in this example until the user no longer needs the message as a reminder.

In addition, because classification is done by fixing access location, and each object can be accessed from only one location, one cannot classify a piece of information in more than one way. Often, an item belongs to more than one category in a classification scheme. For instance, an e-mail can be both work-related and fun-related if its content is mixed. One can make copies but one will need to keep the copies synchronized.

Furthermore, only one classification scheme can be applied at a time. It is a common practice to file e-mail messages by sender, but such a scheme breaks up conversations participated by many people. Ideally, users should be able to classify messages by conversation topic and sender simultaneously.

1.5.4 Organization Not Sharable

Also because classification is done by fixing access location, classification schemes cannot be separated from the information that is classified. For instance, one cannot easily migrate a folder hierarchy from one machine to another so that the same hierarchy can be used to organize files on the latter machine. Such migration would involve moving the hierarchy with its content and deleting the contents after the move. The content can be expensive to move, or it should not be copied to the second machine in the first place because of privacy issues. As a result, it is not easy to share one's classification scheme with someone else.

1.6 Manipulating Information

Functionalities offered for manipulating information within each information management environment are often inconsistent across different contexts.

1.6.1 Inter-application Inconsistencies

While both Jasc Paint Shop Pro 7.0 (Figure 11) and Microsoft Windows XP Explorer (Figure 12) can display image thumbnails, only the latter allows rotating images through their context menus. This discrepancy is bizarre since Jasc Paint Shop Pro 7.0 is an image editor and is capable of rotating images itself while the file browser is only expected to display files in a generic manner.

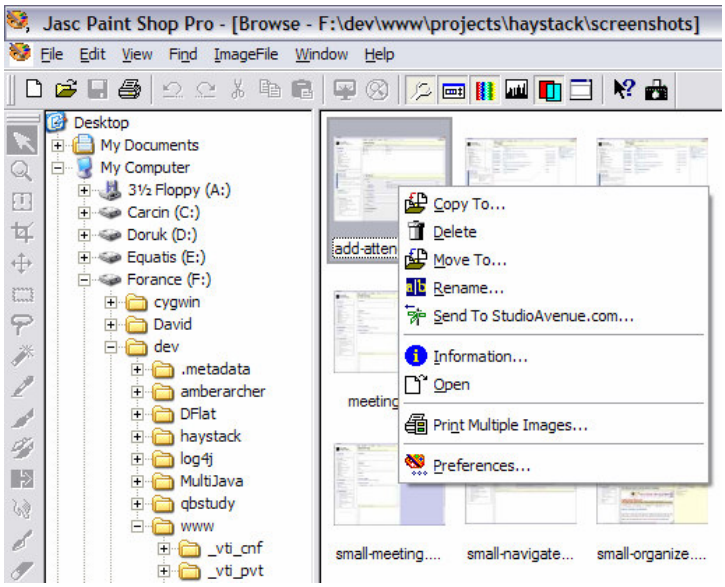

Figure 11. Context menu of an image file in Jasc Paint Shop Pro 7.0

Consider another case of cross-application inconsistency: while one can scan an EXE file for viruses in Microsoft Windows XP Explorer through the file's context menu (Figure 13), one cannot do so to the same file if it is shown as an attachment in an e-mail message in Microsoft Outlook 2002 (Figure 14). Scanning a file for viruses before sending it away makes a lot of sense. Likewise, it would be convenient if Outlook allowed zipping and unzipping an attached file from its context menu.

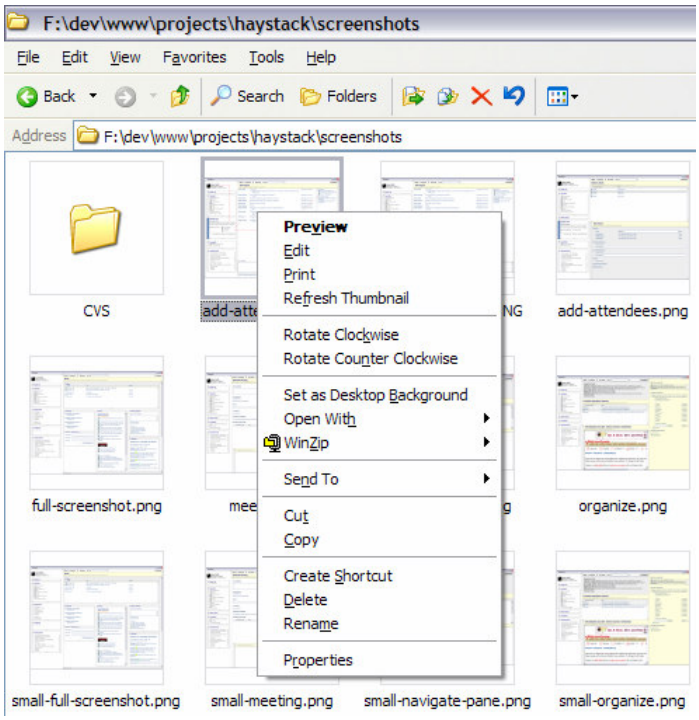

Figure 12. Context menu of an image file in Microsoft Windows XP Explorer

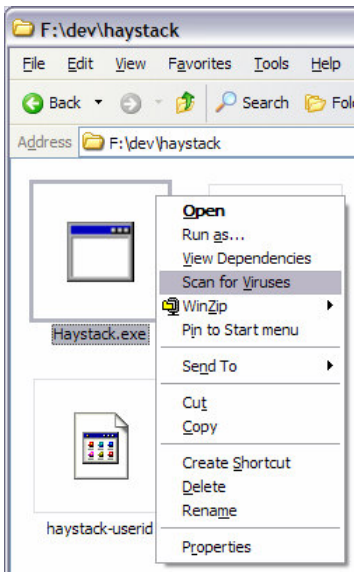

Figure 13. Context menu of a file in Microsoft Windows XP Explorer

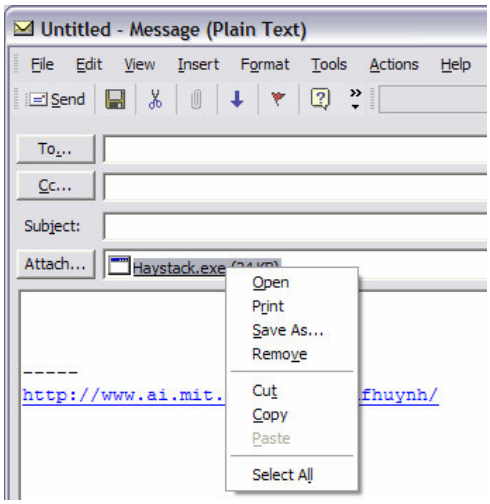

Figure 14. Context menu of an attached file in Microsoft Outlook 2002

Consider more complex examples of inconsistencies. One can select several numeric cells in any spreadsheet program and find their average but one cannot easily determine the average of several files' sizes in a file browser. One can find the statistical *mode* of several cells in a spreadsheet program, but one cannot let the computer determine one's most favorite artist within a collection of audio tracks in an audio software application.

1.6.2 Intra-application Inconsistencies

The previous section illustrates inconsistencies between different applications. There also exist inconsistencies within individual applications. While Microsoft Word 2002 may offer the spellchecking functionality on document bodies, it does not do so for other textual properties of documents. In the Properties dialog shown in Figure 15, there is no means for spellchecking the comment.

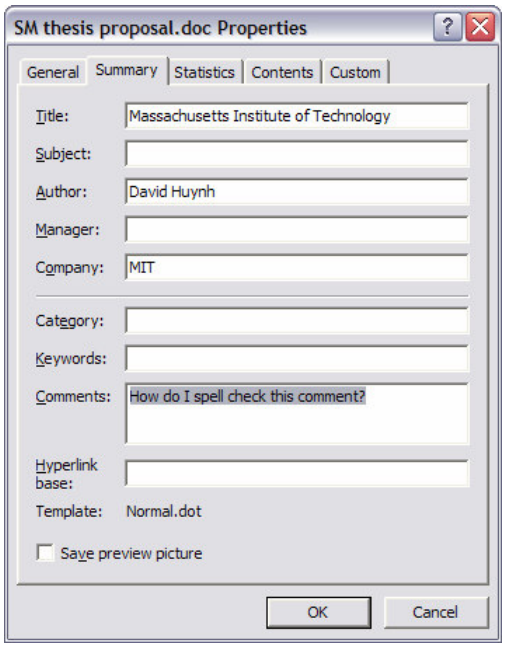

Figure 15. The Properties dialog in Microsoft Word 2002

1.7 Problem Statement

The preceding five sections have discussed how existing environments are lacking when it comes to allowing users to manage information (summarized in Table 1). I have also argued that, since the tasks of managing information form the basis of any user's experience in using the personal computer, these environments must address the deficiencies in their support for managing information before their user experiences can improve. In fact, I believe that the tasks of managing information must be supported by the *infrastructures* of these environments rather than solely by the individual applications that run on them. In this thesis, I explore a number of solutions to some of the aforementioned deficiencies through the principles and construction of the user interface framework of the Haystack information management platform. Specifically, I am seeking to determine the set of basic information management functionalities that the infrastructure of any information management environment should support to ensure a coherent experience for interacting with information. Complex task support can be built on top of this functionality set to provide domain-specific capabilities.

Table 1. Shortcomings of existing information management environments

Expressing information	Schemas incomplete, stop short with fields rather than first-class objects
	Lack of synergy between schemas: relationships spanning different schemas not expressible
	Lack of generic schemas: no default schemas for specifying generic properties such as importance on every object in system
	Naming schemes tied to access locations, minute objects not addressable, naming schemes not unique across network
Locating information	Access to certain information long winded and no means to create shortcut to visited information
	No single starting point for querying, or multiple starting points that lead to unexpectedly different results
	Limited query support for non-first-class information
	No query support for cross-domain information
Percei ing information	Renderings arbitrarily different in different contexts
	Renderings in certain contexts incomplete due to inaccessible data
Organizing information	Data organized by system type; this segregation might not be natural
	Mechanisms for organization arbitrarily different in different contexts
	Mixing concepts of access location and classification; multiple categorization not possible
	Organization (classification schemes) not sharable
Manipulating information	Inter-application inconsistencies
	Intra-application inconsistencies

1.8 Structure of Thesis

This thesis consists of seven chapters in total. This first chapter introduces the problems that I am trying to solve. Chapter 2 explores various past attempts to solve these problems. Chapter 3 dives into my own approach. Chapter 4 discusses the overall Haystack platform. Chapter 5 concentrates on the design rationales behind the user interface framework of Haystack while Chapter 6 describes its implementation details. Chapter 7 illustrates a usage scenario and reports the effectiveness of my approach through a user study. Chapter 8 suggests directions for future work.

CHAPTER 2

Background Work

This chapter examines various full and partial solutions that have been implemented and proposed to support the tasks of managing information in easy and intuitive ways in existing commercial or research information management environments.

2.1 Intra-application Solutions Through Automation

Even though we have discussed so far about environments—operating systems and applications combined, it is illuminating to note that several solutions have been offered for making user interfaces coherent and, in doing so, easing the tasks of managing information within individual applications.

2.1.1 User Interface Management Systems (UIMSes)

The field of user interface modeling has been one of the main focuses of user interface researchers, as evident by the number of user interface management systems ever constructed, to list a few:

- Common Lisp Interface Manager (CLIM) [1];
- Lockheed User Interface System [30];
- Cooperative User Interface (COUSIN) [22];
- University of Alberta UIMS [45];
- Alpha UIMS [27];
- Sassafras [23];

- University of Toronto UIMS [14];
- Penguims [24];
- George Washington UIMS [44]; and
- VUIMS [35].

Roughly speaking, user interface management systems are design-time tools and run-time infrastructures designed to "abstract the details of input and output devices, providing standard or automatically generated implementations of interfaces, and generally allowing interfaces to be specified at a higher level of abstraction." [33]

UIMSes work on one user interface, one application at a time. As the modus operandi of application development goes, the internal work of the application has already been determined by the time a UIMS is called upon to design and build the user interface. As a consequence, the UIMS can only keep the UI coherent within that application by typically projecting the inner working of the application systematically to the user interface. The UI may behave coherently, but the functionalities offered for managing information depend much on the application's inner working. If a piece of information is modeled by the application as a text field, it cannot be manipulated as a first-class object in the UI.

2.1.2 Model-Based Interface Designers

Model-based interface designers (e.g. MOBI-D [38]) improve on UIMSes by facilitating the specification of full interface models, which include user-task models, domain models, user models, presentation models, and dialog models [38]. (UIMSes allow the specification of only the last two types of model.) By specifying these models in a higher level of abstraction and relying on tools to generate concrete data models and user interface elements, we can achieve more standard data models and user interfaces. Standards mean compatibility and consistency.

However, as the relevant models of each application are specified independently from the models of other applications, the resulting data models and user interfaces of different applications still are not guaranteed to be consistent with one another. For example, the designer of an audio software application may believe that his or her users never need to perform any complex task on artists in the domain of audio resource management. As a result, he or she specifies a domain model in which artists are represented by text properties of audio tracks, and a user-task model that omits sophisticate manipulations of artist information. On the other hand, the designer of an e-mail client may specify a more sophisticate domain model with regard to person information in which persons

are represented as first-class objects. Each of these two applications can be coherent within itself, but together they do not provide a consistent experience.

The seeming negligence of UIMSes and model-based interface designers to ensure environment-wide UI and data model coherence is caused by the lack of support for data and UI sharing in the environments in which these systems work. In the example above, since the designer of the audio software cannot easily reuse the data and functionalities developed by the designer for the e-mail client, he or she must resolve to keep the model for artists simple inside his or her own application, lest his or her enthusiastic modeling work extends to include every domain possible.

2.2 Intra-application Solutions Through Design

There are also applications not built with UIMSes and model-based interface designers, but rather designed from ground up with a focus on information management, albeit within very specific domains. For examples,

1. SHriMP Views [46] offers a zoomable UI for exploring hierarchically structured information corpora such as Java code bases;

2. Presto [16] provides a unified and rich facility for managing documents with digital contents through the use of attributes (unfortunately, since not every piece of information in every domain is a document with digital content as I will discuss in section 2.4.3, Presto does not have such a wide applicability);

3. IBM's Lotus Notes [5] is a well known application for managing workflow and communications within organizations; and

4. Canon's Cat [15] [41] offered a well-designed keyboard interface for managing text documents in 1987.

These applications all enjoy the a priori knowledge of which domains of information they deal with. This is a tremendous advantage because the space of all usage scenarios becomes much smaller with the restriction of domain. I mention these applications here for the sake of completeness in this discussion of information management, but they are not examples of systems that I aim to build. I am investigating the necessary infrastructural support for information management tasks in entire

environments that work in the foreseeable and unforeseeable domains, not applications such as these that are designed specifically for very limited domains.

2.3 Inter-application Solutions Through Integration

In order to provide coherence experiences in information management, several attempts at integration have been implemented. The most readily recalled examples of integration are application suites, including Microsoft Office, OpenOffice, Sun's StarOffice, and OEone [8]. These suites' emphasis on entire environments is to be noted. However, integration often only takes on the route of cosmetic unification that makes various different applications look the same and behave similarly. For instance, they have menu commands with common names; their toolbars contain similar items; their help systems are structured similarly; etc. While certainly improving usability, such an integration approach does not yield a more coherent information management experience. Sometimes, only cosmetic integration is possible because the internal workings of the applications have already been set in stone and their individual paradigms for managing information cannot be changed without a major overhaul. More often than not, the existing applications already have fixed user models that might not be compatible, and without overhauling these models, integrating the applications brings out their discrepancies and introduces more confusion than uniformity.

2.4 Inter-application Solutions Through Re-architecture

There have been numerous efforts to build entirely new architectures that facilitate and encourage more expressive data models and more coherent user interfaces. Some even focus on selective aspects of information management.

2.4.1 The Lisp Machine

In the 1970s, the Lisp machine [32] aimed to provide a whole, coherent operating environment for AI researchers by implementing an operating system, based on the Common Lisp Object System (CLOS) [19], that exposed a common object-oriented data model to all applications. CLOS included facilities for managing Lisp objects (e.g. allocation, garbage collection) and was accelerated by specialized hardware. Instead of communicating with one another through files, pipes, and shared memory as in Unix, applications and the OS exchanged Lisp object references and called dynamically-bound methods on them.

The Lisp machine's approach of redesigning the entire operating environment was refreshing in its days, and is still a desirable direction that I would encourage today. Such an environment-wide redesign could bring uniformity to the user experience in interacting with information. However, the Lisp machine's focus was more on powerful programming abstraction for AI research than on information management. The Lisp object-oriented data model that it employed did not expose data and data schemas in an easily query-able form. For instance, it was not possible to efficiently locate (e.g. by querying for) all objects of a certain type in the system; or to retrieve a listing of different types of object in the system so that the user could start browsing the corpus by type; or to determine which other objects were associated with a particular object. The last example requires first a programmatic inspection of all schemas that had some properties whose ranges were the type(s) of the given object, and second a query for objects in those schemas whose some of those properties were actually the given object. All three examples call for (a) some query support over an object-oriented data model, which is difficult to implement, and (b) explicit schema specification. While the CLOS data model was capable of encoding rich data schemas, such support was not designed-in as part of the infrastructure. In other words, programmers were not encouraged nor assisted to expose their applications' data and schemas. Data sharing was on a voluntary basis.

Because the Lisp machine put its emphasis on operating environments and applications, it implicitly assumed that data would not exist without these applications and environments. In fact, Lisp objects could only be uniquely identified within the Lisp machine on which they had been instantiated. The naming scheme in CLOS did not guarantee uniqueness across networks, and a Lisp object on one machine could not be referenced from another machine (unless location-dependent information were added). Furthermore, an object's schema would remain hidden in the absence of the application that implemented schema-related methods on that object. There was no facility for browsing, querying, or managing heterogeneous information (objects of different types) through common user interfaces (e.g. organize instant messages and e-mail in the same hierarchy). Each type of objects can only be manipulated within its indigenous application.

2.4.2 Object Brokerages

Like the Lisp machine, object brokerage systems also take an object-oriented approach to inter-application communications. But instead of imposing a single language-specific environment, they allow independent applications to communicate through programmatic binary interfaces for the purpose of sharing data and schemas and invoking one another's functionalities; each application can be implemented in a language of its designer's own choosing. Examples of these systems include Microsoft's Component Object Model (COM) [7] and the Object Management Group (OMG)'s

Common Object Request Broker Architecture (CORBA) [3], both originating from the Andrew environment [34].

These systems provide the concept of monikers for naming objects, which can be unique across networks. Given a moniker, one can call on infrastructural facilities to retrieve an interface pointer through which one can interact with the object named by that moniker. This is restrictive because for each moniker, only one interface pointer can be retrieved. That is, only one piece of code can provide the named object's properties and functionalities. That piece of code implements a fixed set of schemas and additional schemas cannot be applied to the object without modifying that code.

In addition, these systems focus mainly on capability discovery (i.e. determination of a component's functionalities by querying its supported interfaces) rather than data modeling. As a result, there is little support for data querying. In particular, there is no means to query for properties over several objects, e.g. to find which objects have a given title.

These object brokerage systems are also designed to handle user interface components. In fact, Microsoft's COM is an extension to a previous architecture called Object Linking & Embedding (OLE) [36] that allows content components to be nested in rich documents. However, nesting usually occurs at one level depth only; and even then, the user experience is less than smooth as the merging of the functionalities of the nested component with the functionalities of the containing document into one main menu often yields an unintuitive menu.

Overall, these object brokerage systems are difficult to program in because a lot of code needs to be written to implement and expose binary interfaces properly. Hopefully, with the advance of more development tools that automate certain aspects of interface implementation, such as generating interface method stubs, component development will become easier.

2.4.3 The Web

The World Wide Web is the largest effort so far that focuses on users' information needs. It provides uniform information access by allowing information of various types (e.g. airline itineraries, maps, movie reviews, news) to be accessed through the same user interface of the web browser. In most cases, no application needs to be explicitly opened and closed to view the content of a URL. That is, users need not be aware of computational mechanisms that are called upon to serve their information needs. Users simply give references to the information they want, and the information is automatically returned in the most suitable forms.

In addition to evoking mnemonic URLs, users can also call on documents by clicking on labeled hyperlinks. In this associative browsing paradigm, hyperlinks act as strings on which users can "pull" to bring back more and more information. In contrast, conventional application-centric UIs require the user to start different applications, open various dialog boxes, and navigate various UI paths to get to certain pieces of information; and this navigation process is heterogeneous in the sense that there is no single type of navigation action like the hyperlink clicks. From another perspective, the Web does not require the user to be reactive to the UI. The user can indicate which browser window the desired information should arrive in and most websites honor that request. In conventional UI, information may come in unexpectedly opened dialog boxes and the user often has to react to them by shifting his or her attention to those dialog boxes. From either perspectives, the Web navigation paradigm is conceptually simpler to users.

Hyperlink-based navigation had been around before the coming of the Web. However, pre-Web hypertext systems existed only in small scales. Only when the Web was adopted in such a large scale did we start to reap the benefits of the hyperlink paradigm as applied on information in vast quantity and of numerous types. It is this grand scale of the Web that sets it apart from all of its hypertext predecessors [20], that shows a navigation paradigm versatile enough to replace a diversity of customized user interfaces.

Furthermore, any piece of information obtainable on the Web can be bookmarked so that it can be returned to at a later time with as few as one click. This is because all Web resources can be named. In contrast, few pieces of information in conventional UIs can be bookmarked: if a piece of information is gotten by opening a stack of dialog boxes, returning to it would involve reopening the same stack of dialog boxes in the same order—there is no shortcut possible.

Overall, the Web satisfies users' information needs in many ways: information can be accessed in an easy and uniform manner without knowledge of binary representation and application association.

Read-Only Mode

In the context of information management, the Web suffers from a major user interface shortcoming: in the default case, the web browser displays documents in a read-only mode. While this functionality is sufficient for browsing published documents, it is restrictive in information management where users need to modify and author data.

Restrictions by Focus on Content

We also find the Web's data model lacking because the Web's primary focus is on content and presentation. In particular,

1. there is no standard strategy for dealing with content-less information (e.g. what URL to use to refer to an object that has no retrievable digital content);

2. there is no standard way to encode a generic property since properties are not considered content (e.g. there is no Web standard to record a document's author);

3. relationships, expressed as hyperlinks, can only be embedded inside content—without content, there is no place to specify the relationships;

4. there is no standard way to encode types of relationship—hyperlinks only specify generic relationships or associations; and

5. any query model that works over Web relationships, i.e. hyperlinks, must always deal with content.

We will illustrate these deficiencies in details through a single example in which a user wants to assert that a particular picture has been taken by a particular photographer.

First, while it might be easy to decide which URL to use for the picture in our example, it is unclear how to refer to the photographer. He or she can have several homepages, one of which might be dedicated as the official homepage whose URL can be used to identify him or her; or one might resort to one of his or her e-mail addresses. The difficulty faced in choosing the photographer's URL lies in the fact that unlike a document, a person is not an object that has digital content. A document is its content and consequently, the URL used to retrieve the content can naturally be used to refer to the document.

Second, even if one were to dedicate a URL to refer to the photographer, one cannot record his or her information (e.g. age, gender) without creating a Web document. And the Web does not specify nor enforce how such a document should be composed—how to specify the property names and values in HTML. As a consequence, Web documents created by different authors to encode similar properties will have different formats. After all, HTML is a presentational language and its presentational aspects will distract the authors from the encoding of the properties. The resulting different formats hinder attempts to query over several properties-containing Web documents. For

instance, in order to programmatically find out the employers of some photographers, one must write code to:

- parse the HTML codes of those Web documents whose URLs are taken to refer to the photographers; and
- use natural language algorithms to guess which parts of the HTML codes refer to the employer attribute (e.g. it could be "employer", "Company", "Organization:", or any imaginable string).

Such a piece of code is highly complex, error-prone, and no more than a hack to counter this misuse of HTML for the purpose of encoding properties: HTML is primarily designed to be used as a presentational language. While SGML can be called upon to allow properties to be encoded properly, SGML has been criticized as being too complex to be a practical widely-used language [37].

Third, the relationship between the picture and the photographer can only be encoded as a hyperlink. However, it is unclear which Web document should contain this hyperlink. If one were to refer to the photographer with the URL of one of his or her homepages, then one would logically embed a hyperlink to the picture in that homepage. As with properties of individual objects, it remains unclear how to syntactically encode relationships between related objects (e.g. where in which documents, what labels to use for the hyperlinks). Due to the lack of a standard encoding scheme, querying support for inter-object relationships must depend on natural language processing.

Overall, the Web favors objects at the level of granularity of documents with well-defined digital content and relationships that have no specific semantics. As a consequence, objects of other levels of granularity and relationships with specific semantics cannot be easily encoded. In addition, the Web's emphasis on content presentation interferes with the pure encoding of information and how information can be queried and analyzed by machines.

2.4.4 The Semantic Web

The Semantic Web [13] aims to solve the various shortcomings in the Web's data model. Through the use of Unique Resource Identifiers (URI)—a superset of URLs, the Semantic Web proposes a unified naming scheme capable of referring to objects at any level of granularity, with or without digital content, and independent of their physical storage locations and binary representations. Focus is put into the encoding and publication of data schemas through the Resource Description Framework (RDF) [9]. RDF allows semantics of relationships to be specified, and relationships to be encoded independent of content presentation. As a benefit of the unified naming scheme, several schemas can be applied on the same object given its URI, and different aspects of the same object

can be managed by different software systems. For instance, an "e-mail" schema and a "photography" schema can both be applied on a single *person* object to manage his or her e-mail communications and photographic creations. Overall, the Semantic Web makes it easy to expose and share data and data schemas.

Furthermore, RDF data models, essentially directed graphs, are much simpler than the Web's hypertext model in which data is expressed as human-readable textual contents in HTML. Because of this simplicity, RDF data models can be easily stored in optimized formats different from the formats in which they are originally expressed, e.g. RDF/XML [9] and N3 [12]. Queries can be supported more efficiently and more simply over these optimized formats than over the Web's data models.

Most efforts related to the Semantic Web project have been spent on data modeling while much less attention is paid to user interface issues. Many graph-based viewers (e.g. IsaViz [6] and BrownSauce [1]) have been built to display and edit RDF data, but those are unsuitable for everyday interaction with information. For instance, few users, even the data modeling experts, would want to browse their calendar events as graphs. There are also ontology editors (e.g. Protégé [17] and Ont-o-mat [21]) that let users model data schemas but they are not designed for interacting with everyday information. There are also user interfaces for authoring schema-specific data, but they are too specific and do not cooperate to create a coherent information management environment. For instance, Annotea [26] only deals with web document annotations; there is little indication that its user interaction experience would scale for any other purpose.

Much exploration is needed to create an environment for interacting with Semantic Web information. For one thing, the freedom to share data and data schemas will lead to more information exchange and ultimately result in more information with which users have to deal. This increase in information might introduce more UI inconsistencies. The ideal user interaction experience must scale gracefully as more and more data types are incorporated into a user's environment.

2.4.5 The Views System

The Views system was a research computing environment that investigated user interface issues for the purpose of addressing UI inconsistencies [31]. It did so by separating data modeling and storage and low-leveled UI presentation logistics from applications, letting applications retain only high-leveled logic of how to manage and present their data. Data as well as UI presentations could be shared easily across applications, as both were managed by the framework. Invariants were used to

maintain consistency between UI presentations and their corresponding data: if the data changed, the UI would be refreshed; if the UI was manipulated, the data would be updated.

The Views system seems to have a few drawbacks. Its data model seems restrictive in that once the type of an object has been determined, it cannot changed. As a consequence, more schemas cannot be applied dynamically on an object. Furthermore, there is no evidence that queries can be performed across various objects.

Overall, the Views system appears very promising. Unfortunately, it seemed to be at an early stage of exploration in 1992 and there has not been more news of its progress since.

Table 2 shows a summative comparison between the various inter-application solutions by re-architecture discussed in this section.

Table 2. Comparisons between several inter-application solutions by re-architecture

	Lisp Machine	Object Brokerages	The Web	Semantic Web	Views System
Model schemas explicitly	No. Schemas are implicit in object-oriented model.	No. Schemas are implicit in programmatic interfaces.	No.	**Yes.**	No.
Object identifiers unique across networks	No. Identifiers unique within local machine only.	**Yes.** Monikers are made unique with access locations.	**Yes.**	**Yes.**	N/A. No information.
Identifiers independent of access location	N/A. Objects can only be accessed on local machine.	No. Monikers are tagged with access locations.	No. URLs are locators.	**Yes.**	N/A. No information
Can model objects without digital content	**Yes.**	**Yes.**	No.	**Yes.**	**Yes.**
Relationships can be specified independently from contents	**Yes.**	**Yes.**	No.	**Yes.**	**Yes.**
Specific relationships can be encoded	**Yes.**	**Yes.**	No.	**Yes.**	**Yes.**
Adequate query support	No. Object-oriented model hinders querying support.	No. Programmatic interfaces hinder querying support.	No. Query support must deal with contents.	**Yes.** Simple data models facilitate good query support.	No. Object-oriented model hinders querying support.
Schemas can be added to existing objects	No.	No.	N/A.	**Yes.**	No.
Interaction with information independent of binary representation	**Yes.** View-based UI framework automatically presents objects in best views.	N/A.	**Yes.**	N/A. Not yet investigated.	**Yes.** View-based UI framework automatically presents objects in best views.
Focus on user interaction	**Yes.**	No.	**Yes.**	No. Not yet investigated.	**Yes.**

CHAPTER 3

Approach

This chapter starts by discussing the principles that I advocate for constructing any information management environment, continues with the application of those principles to create Haystack's user interaction experience, and ends with an overview of the user interface framework that supports such an experience.

3.1 Principles

In this section, I explore a number of lessons learned from previous works as discussed in chapter 2, presented in the form of design principles.

3.1.1 Refactor support for information management tasks

The evolution of modern operating systems follows an application-centric paradigm. Different software vendors focus on different applications that fill different usage niches. The domains of these applications are often narrow: for instance, there are some applications whose sole purpose is to create animated GIFs. Other applications have wider domains, e.g. spreadsheets and CAD tools. Even though the domains of these applications seem disparate, they all include some common information management tasks as I have discussed in Chapter 1.

This recognition of such common functionalities leads me to propose a *refactoring* of these environments to move the support for common information management tasks from individual applications into the environments' infrastructures. This idea of refactoring comes from software

(re)engineering techniques for object-oriented systems. In [42], Roberts notes that reusable components in these systems often start off with wrong designs—they are either more general or more specific than should be. As the systems evolve, common concrete use cases arise and they point out how the reusable components should be factored in the first place. The systems are then refactored to optimize for those concrete use cases.

Refactoring an object-oriented system leads to a design that is more comprehensible, maintainable, and extensible. These advantages are transferable to my proposal of refactoring information management support. The refactored environment will be more comprehensible to both users and programmers, more maintainable whenever it needs fixing, and more extensible as support for new information management tasks is required. Users of the refactored environment will be shown a unified, coherent experience for managing information, and programmers will be able to leverage common, infrastructural support for information management tasks, avoiding duplicate work and duplicate bugs.

The biggest challenge to this refactoring is to predict how various foreseeable domains of information needs will fit into the environment. This can be a daunting task, as these various domains can be very different, ranging from music composition to management of chemistry experimental results to collection of criminal investigation evidences. However, it is not necessary to "get it right" on the first try of refactoring. Evolutionary improvements following the refactoring can be made to encompass more and more domains. In fact, such improvements will be made more easily because the environment will be more properly structured after the refactoring.

3.1.2 To achieve coherency, use a centralized, expressive data modeling framework

As discussed in Chapter 1, many deficiencies of existing information management environments result from incompatible data models that are distributed throughout individual applications. It is difficult to express information that spans across several of such data models. In addition, one cannot efficiently perform sophisticated queries that draw together information from different data models. Overall, distributed data models in a single environment often lead to fragmented data. For that reason, I propose the use of a centralized data modeling framework as the first step to the refactoring of any information management environment.

The user interface and interaction experience of an environment is only as good as its data model. Without an expressive data model, the environment will fail to satisfy some users' information needs:

users will not be able to express certain forms of information, perform certain sophisticated queries, create shortcuts for quickly accessing certain pieces of information, etc. An expressive data modeling framework should have the following capabilities:

1. It must have a unified naming scheme that is location and representation independent and that guarantees uniqueness across networks;

2. It should be able to deal with objects at any level of granularity, with as well as without digital contents;

3. It should allow several schemas to be applied on any single object; and

4. It should be able to encode relationships between objects in different schemas.

After the adoption of an expressive data modeling framework, it still remains to be seen whether developers of the system will create schemas that are compatible with one another. For instance, it is possible that one schema does not make use of another schema but duplicates only a subset of the second schema's functionalities. This is a difficult challenge. There have been some researches on how to encourage schema reuse [11] [29]. As well, one can draw on results from research on code component reuse [18] [28].

3.1.3 Present information in the forms fitting the nature of that information

When dealing with flexible data models as those of the Semantic Web, it is tempting for developers to create uniform views of data in the forms of graphs and trees. While these forms are capable of showing many configurations of data, they are not suitable for human beings to use to deal with domain-specific information. For instance, a calendar presented as a graph is not usable. The nature of information and the current context need to be taken into account.

Of course, it remains a challenge to choose the best form of presentation to show a particular piece of information. This is where infrastructural support for sharing UI work comes into play. For each piece of information, there is supposedly an expert who knows best how to present it. Other developers should be able to rely on the infrastructure of the environment to find the UI work of that expert and reuse it whenever that piece of information needs to be shown. This idea is already present in any view-based UI architecture. I am simply advocating its use on a larger scale compared to prior schemes. Just as the Web took the idea of hypertext to a qualitatively different level by applying it on an tremendously larger scale [20], I believe that applying the view-based UI

construction scheme for reusing UI work on a larger scale will yield a qualitatively different UI experience.

A second challenge is ensuring that the overall user interface and interaction experience scales well as more and more data and data schemas are added to the environment. For instance, presenting information as large icons only does not scale as the number of icons grows.

3.1.4 Support object-oriented UI manipulations pervasively

Scalability issues also concern the ways users manipulate information. Existing environments that merge operations applicable on several objects into a single menu are often awkward and confusing to use: users may not know where to look for all operations applicable on a particular object, and which object a particular operation will be applied on. I propose that environments should support object-oriented UI manipulations pervasively in the following ways:

1. There should be a uniform way to browse all operations applicable on a particular object;

2. Operations should be refactored by their logical applicability on object types rather than being offered in fixed sets of predetermined usage scenarios as often seen in application-based environments; operations that have universal applicability, such as organization, should be supported by the environment itself; and

3. Operations that lend well to direct manipulation mechanisms (such as drag-and-drop and mouse gestures) should be supported through those mechanisms because it is most natural for users to invoke them in those ways.

All of these three points have parallels in the Object-Oriented Programming paradigm: First, operations applicable on a particular object are listed in that object's class definitions rather than dispersed through various functions. Second, with proper OOP modeling, all operations logically applicable to an object are available through its immediate class or by inheritance. Certain universally useful operations are wired into the root class (e.g. the java.lang.Object class) so that all objects inherit them. Finally, operators such as plus and minus that lend well to the semantics of a particular class are overridden to provide convenient access to some of the class' functionality. We believe that just as OOP has made software development more understandable and manageable for programmers, an object-oriented UI would make information management easier for users [43].

3.2 Haystack's User Interaction Experience

In this section, I present the user interaction experience of the Haystack information management platform, designed with the abovementioned principles in mind.

3.2.1 Expressing Information

The Haystack platform adopts the Semantic Web's Resource Description Framework (RDF) as its data modeling framework. As mentioned in section 2.4.4, RDF specifies a unified naming scheme in which every object is referred to by a Unique Resource Identifier (URI). URIs are supposed to be unique across networks. They form a superset of Uniform Resource Locators (URLs) and can address objects at any level of granularity, with or without digital contents. Objects modeled in RDF are referred to as *RDF resources*, or just *resources*.

RDF data takes the form of triples called *RDF statements*, or just *statements*. Each triple contains a subject, a predicate, and an object. The subject and predicate must be resources; the object can be a resource or a *RDF literal*. Literals are strings. The predicate is like an arrow pointing from the subject to the object. A corpus of RDF data is essentially a directed graph with labeled edges.

Unlike conventional directed graphs, in a corpus of RDF data, one can find arrows pointing from and to other arrows. In other words, one can make statements about predicates, as illustrated in Figure 16: the `is-Father-Of` predicate is declared to be a `Familial-Relationship` and also a `One-To-Many-Relationship`. This capability allows RDF to easily model schemas.

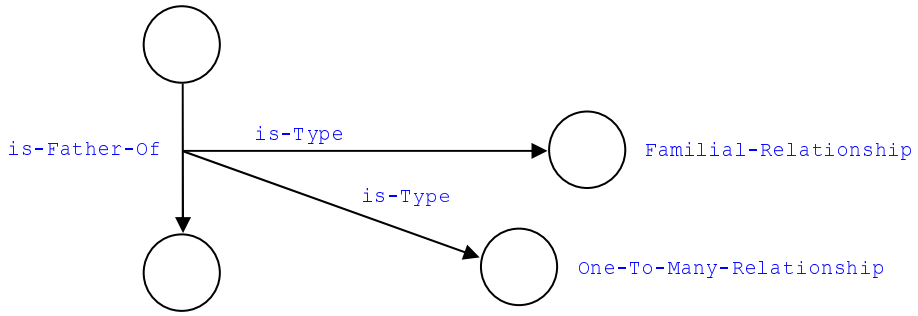

Figure 16. Example of an RDF statement about an RDF predicate

Overall, RDF allows data to be encoded in a semistructured form, fitting several schemas simultaneously but none necessarily rigorously. This capability is advantageous in situations of incomplete information, which occur frequently in the real world.

Haystack adopts RDF for three reasons:

1. It is a powerful data modeling framework that fits our needs as laid out in section 3.1.2;

2. It is new, which implies that its designers should have more history to learn from before designing it (that it is new is also a concern since its faults may not have been identified); and

3. It is a standard, which means that our data models will be immediately compatible with other data models, especially those on the Semantic Web, and that someone beside us will work on improving it.

Armed with a powerful data modeling framework, we can rest assured that users will be able to express a lot of what they want to express. The user interface mechanisms for expressing information are another issue to address. Information can be expressed through casual manipulations of data as will be discussed in sections 3.2.4 and 3.2.5, or through preliminary generic editing mechanisms [40].

3.2.2 Perceiving Information

Each piece of information—each information object—in Haystack is presented like a web page. The mapping from the information object to its presentation follows a view-based scheme: metadata in the system associates the characteristics of that object (e.g. its type) and the context in which it is shown to a particular view.

Views are onscreen presentations—pixels that together convey information. Each object can be presented by more than one view, each suitable in some contexts. Views allow us to present information objects in formats appropriate for them instead of defaulting to a graph- or tree-based presentation.

Given an information object to present, there are two aspects to consider: what information about that object to show, and how to show such information. The *what* aspect involves selecting attributes of that object and selecting other objects relevant to that object. The *how* aspect involves laying out and rendering the object's attributes and determining how to show the relevant objects. Both aspects cooperate to produce a view, which is composed of UI elements through which the user can interact with the object.

As the view of one object may specify that other objects be presented, that original view may trigger the resolutions and instantiations of views of the other objects. The original view embeds the other views. For instance, the view of an e-mail message embeds the views of the message's attachments,

each of which embeds the view of the attachment's author. Views can be nested arbitrarily deeply to bring together all information relevant to the object that the user originally wants to perceive.

Note that the view of the original object does not specify exactly which views to use for the relevant objects. It only specifies the characteristics of the views to be nested and the UI framework will resolve the desired views based on those characteristics.

Presentations in Haystack resemble web pages in the sense that they can have sophisticated document-like layouts, containing text, images, tables, and hyperlinks. Each presentation needs not be rectangular in shape; it can be a piece of text that can wrap across several lines. This flexibility allows presentations, and consequently views, to be nested arbitrarily deeply without looking rigidly rectangular. View nesting schemes such as Microsoft's Object Linking and Embedding quickly break down because their views assume large rectangular forms that do not fit elegantly together.

3.2.3 Locating Information

Since Haystack adopts RDF as its data modeling framework, information objects in Haystack are all addressed by URIs. Like the web browser, Haystack can take a URI and "browse" to it by finding an appropriate view for the object named by that URI and displaying the view. Alternatively, just as on the Web, an object can be browsed to by clicking on a hyperlink representing it from within the presentation of another object. This navigation paradigm—a simple extension of the Web's—keeps information in the foreground: users can reach at information without explicitly managing applications. Figure 17 shows the home page of the user being browsed to.

Of course, users might not remember URIs or have all needed hyperlinks conveniently presented at all times. Haystack still provides search interfaces for finding information objects. However, unlike in existing environments, all things in Haystack, big and small, can be modeled as first-class objects. As a consequence, each object can be viewed independently and can be arrived at through whichever route most easily recalled by the user rather than through a fixed UI path. Contrast this with current environments: in order to find the address of a company in Microsoft Outlook 2002, one must indirectly find a contact who works for that company—the user has to make that cognitive leap. The address of that company is shown as a field in the presentation of the contact.

The fact that each piece of information can be modeled as a first-class object has another important advantage: no matter how an object is located the first time, it can be bookmarked so that subsequent accesses to it can be done much more quickly. In contrast, one cannot see the address of

a company without first locating some particular contacts who work for that company in Microsoft Outlook 2002.

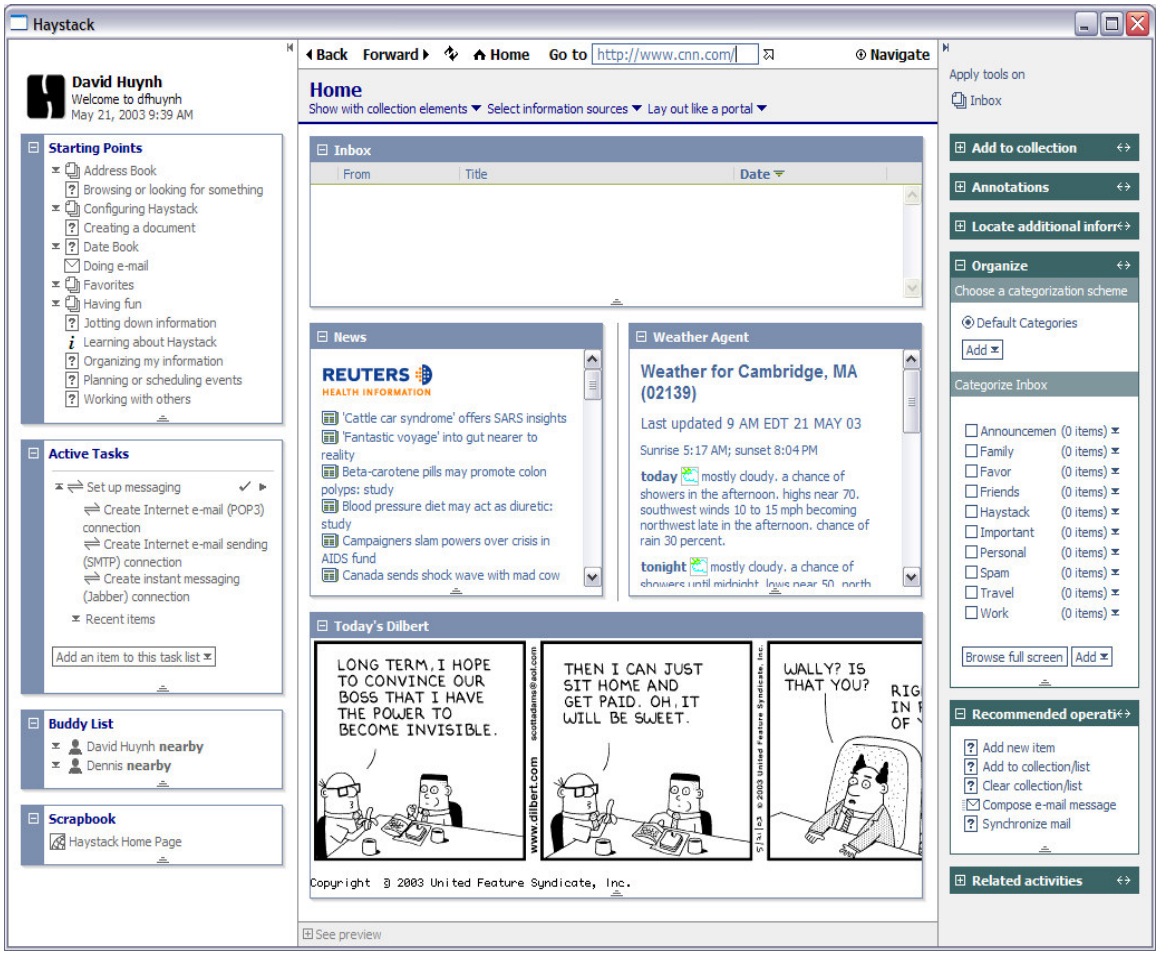

Figure 17. Haystack's user interface

3.2.4 Manipulating Information

Just as we use metadata to provide associations between objects and views, we also use metadata to associate objects and operations. Given an object, we can systematically find each operation that takes a parameter matching the type of the object. The result is a set of operations logically applicable to that object.

When a presentation of an object is right-clicked, we list all operations applicable to it in a context menu (Figure 18). This gives users a guaranteed way to find out which operations can be done on an object. This approach associates objects and their operations very closely and avoids the need to merge lists of operations of different objects into a single menu. If there are too many operations to fit into a context menu, we show only a few and provide a link that browses to the full collection of

operations wherefrom users can call upon more sophisticated tools to narrow down to the desired operation. That is, in Haystack, finding a desired operation can be accomplished much like any other information seeking task.

We also provide drag and drop support as a convenient means to perform selective operations which take two parameters, one as the dragged object and the other as the drop target. When a drag and drop action is performed, we can systematically find an operation that takes two parameters, one matching the nature of the dragged object and the other of the drop target.[2] More context needs to be taken into account to guess at the desired operation. Otherwise, we will have to ask the user to select one operation among several.

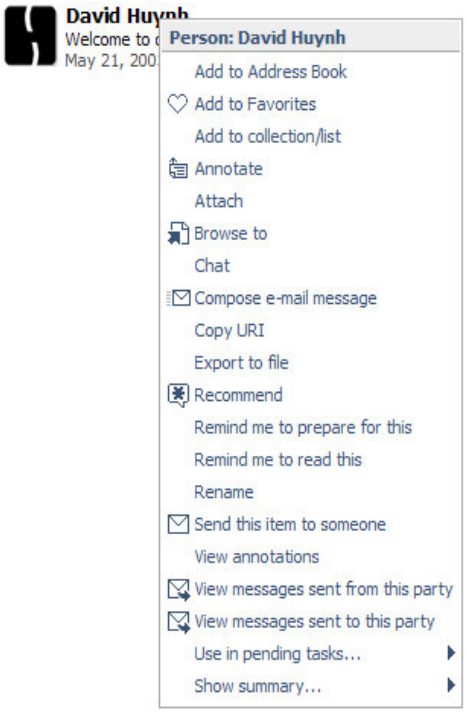

Figure 18. A sample context menu

3.2.5 Organizing Information

Haystack explores some generic infrastructural support for organization through the concept of *collections*. A collection in Haystack is just an RDF resource that is associated with its members through the predicate `hs:member`. There is no restriction on a collection's members: they can be of any type and any nature. In other words, objects of different types can belong to a common collection. Furthermore, each object can belong to several collections all at once. This flexibility

[2] In case we can find more than one such operation, we pick one of the operations non-deterministically.

allows Haystack to satisfy users' diverse needs of organizing information. Figure 19 shows the Organize tool is being used to classify an e-mail message titled "Weekend Party" into three different collections simultaneously.

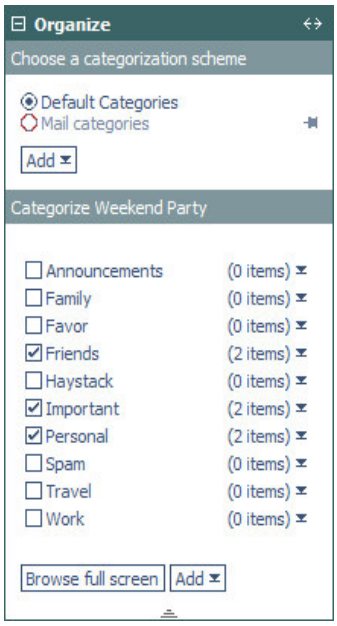

Figure 19. The Organize tool

3.3 Contributions

The contributions that I make through this thesis are spelled out here explicitly:

First, I believe that my call for environment-wide, infrastructure-deep support of mechanisms for satisfying information management needs is novel. Past approaches that actually focus on information management mostly have been concentrating on individual, selective domains. They assume that the foundations offered by contemporary operating systems are sufficient to serve as common platforms on which domain-specific applications can be built and can cooperate to yield coherent experiences. On the other hand, I advocate that some functionalities satisfying certain information management needs can and should be refactored out from individual applications into the environment's infrastructure.

Second, I have identified five different aspects of information management on which to focus. This simple characterization of information management needs might not be perfect, but it serves as an initial guide for any investigation on infrastructure support of information management mechanisms.

Third, I have provided an initial roadmap with four principles as listed in section 3.1. The individual principles may not be novel, but the entire plan is.

Fourth, I have designed the user interaction experience for Haystack and thus illustrated the application of those principles.

Fifth, I have constructed the UI framework of Haystack and in doing so, demonstrated practical approaches to building a UI framework on top of an RDF data model.

CHAPTER 4

The Haystack Platform

In this chapter, I give an overview of the architecture of the Haystack platform. I then introduce basic data models used in Haystack, which will be seen again in subsequent chapters. I also briefly touch upon the script-like language named Adenine used in Haystack for manipulating RDF data—Adenine will be used to illustrate many examples in later chapters. Finally, I recount the common development process observed thus far from the Haystack team's work in building Haystack's components.

4.1 Architecture Overview

The Haystack platform adopts a blackboard architecture in that its various components communicate mainly through a single RDF store (Figure 20). Most of Haystack's core is written in Java; other parts are in C++, N3 [12], and a Haystack-native language called Adenine (section 4.2).

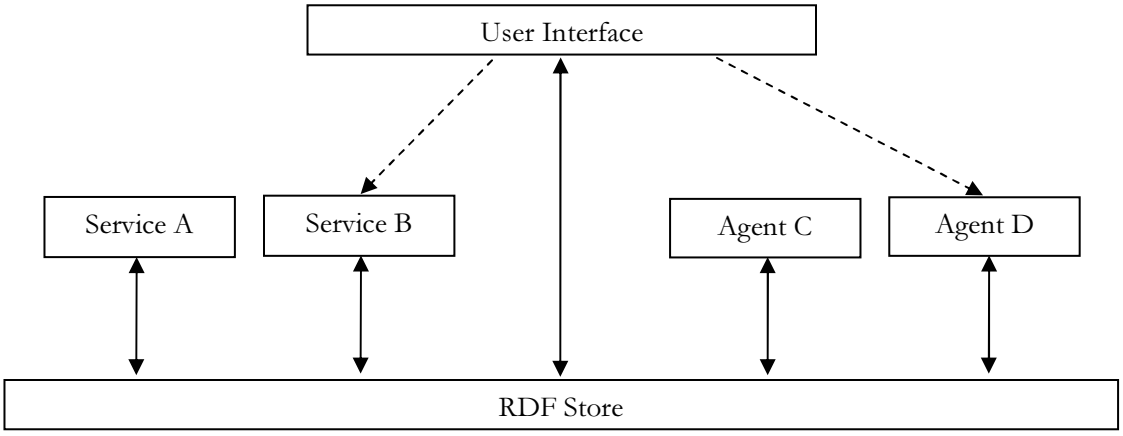

Figure 20. Architecture overview

The following subsections will briefly discuss the RDF store, the services and agents, and the user interface.

4.1.1 RDF Stores and RDF Containers

In this store is RDF data in the form of triples, i.e. RDF statements. Data can be inserted into the store or retrieved from it through the `edu.mit.lcs.haystack.server.rdfstore.IRDFStore` Java interface:

```
public interface IRDFStore extends ISessionBasedService {
    public boolean contains(
        String     ticket,
        Statement  s
    ) throws ServiceException, RemoteException;

    public RDFNode extract(
        String     ticket,
        Resource   subject,
        Resource   predicate,
        RDFNode    object
    ) throws ServiceException, RemoteException;

    public Set query(
        String         ticket,
        Statement[]    query,
        Resource[]     variables,
        Resource[]     existential
    ) throws ServiceException, RemoteException;

    public RDFNode[] queryExtract(
        String         ticket,
        Statement[]    query,
        Resource[]     variables,
        Resource[]     existential
    ) throws ServiceException, RemoteException;

    public int querySize(
        String         ticket,
        Statement[]    query,
        Resource[]     variables,
        Resource[]     existential
```

```
    ) throws ServiceException, RemoteException;

    public Set queryMulti(
        String         ticket,
        Statement[]    query,
        Resource[]     variables,
        Resource[]     existential,
        RDFNode [][]   hints
    ) throws ServiceException, RemoteException;

    public void add(
        String         ticket,
        IRDFContainer  c
    ) throws ServiceException, RemoteException;

    public void remove(
        String     ticket,
        Statement  s,
        Resource   existentials[]
    ) throws ServiceException, RemoteException;

    public void replace(
        String     ticket,
        Resource   subject,
        Resource   predicate,
        RDFNode    object,
        RDFNode    newValue
    ) throws ServiceException, RemoteException;

    public Resource[] getAuthors(
        String     ticket,
        Resource   id
    ) throws ServiceException, RemoteException;

    public Statement getStatement(
        String     ticket,
        Resource   id
    ) throws ServiceException, RemoteException;

    public Resource[] getAuthoredStatementIDs(
        String     ticket,
        Resource   author
    ) throws ServiceException, RemoteException;

    public void addRDFListener(
        String     ticket,
        Resource   rdfListener,
        Resource   subject,
        Resource   predicate,
        RDFNode    object,
        Resource   cookie
    ) throws ServiceException, RemoteException;

    public void removeRDFListener(
        String     ticket,
        Resource   cookie
    ) throws ServiceException, RemoteException;
}
```

The first 6 methods allow RDF data to be queried from an RDF store. The next 3 methods allow changes to be made to the content of an RDF store. The next 3 methods deal with authorship and reification information. The last 2 methods allow registration for notifications of particular changes to the data in an RDF store. Note that all methods require a ticket which identifies the client. For

more information on the semantics of this Java interface, consult [25]; the next few paragraphs highlight only a few important points about this interface that will be useful for later discussions.

RDF resources are encoded in the class `edu.mit.lcs.haystack.rdf.Resource`, and RDF literals in the class `edu.mit.lcs.haystack.rdf.Literal`. Both classes derive from the abstract class `edu.mit.lcs.haystack.rdf.RDFNode`, which is used in places where both literals and resources are permissible. Each `RDFNode` object stores a single string, which is interpreted as a URI or a literal by the derived class `Resource` or `Literal`, respectively.

RDF statements are encoded in the class `edu.mit.lcs.haystack.rdf.Statement`. This class contains three members: `m_subject`, `m_predicate`, and `m_object`. The first two are of type `Resource` while the last one is an `RDFNode`.

Registration for notifications of changes take the form of triples: subject, predicate, and object. Any part of the triple can be specified as `null` to signify a wildcard. Any added or removed statement that matches the given pattern causes an RDF event to be fired on the service named by the `rdfListener` argument given in the `IRDFStore.addListener()` method. The specific use of RDF events will be explained in the next chapter.

There can be different implementations of the main RDF store; the most efficient one currently in use has been written in C++ and wrapped by a Java class using JNI. The interface `IRDFStore` also allows Haystack to interact with many RDF stores at the same time in the same manner.

The `ISessionBasedService` interface that `IRDFStore` extends supports the ability to log in and log out of an RDF store, which is a session-based service:

```
public interface ISessionBasedService {
    public String login(
        Identity id
    ) throws ServiceException, RemoteException;

    public void logout(
        String ticket
    ) throws ServiceException, RemoteException;

    public String getClientClassName() throws ServiceException, RemoteException;
}
```

Logging in to an RDF store using a particular identity produces a ticket string by which the store will recognize that identity in subsequent method calls. To abstract away this ticket string once a store has been logged in, we use the interface `edu.mit.lcs.haystack.rdf.IRDFContainer`:

```
public interface IRDFContainer {
    public boolean contains(Statement s) throws RDFException;
    public RDFNode extract(
        Resource   subject,
        Resource   predicate,
        RDFNode    object
    ) throws RDFException;
    public Set query(
        Statement[]    query,
        Resource[]     variables,
        Resource[]     existentials
    ) throws RDFException;
    public RDFNode[] queryExtract(
        Statement[]    query,
        Resource[]     variables,
        Resource[]     existentials
    ) throws RDFException;
    public Set query(
        Statement  s,
        Resource[] existentials
    ) throws RDFException;
    public int querySize(
        Statement[]    query,
        Resource[]     variables,
        Resource[]     existentials
    ) throws RDFException;
    public Set queryMulti(
        Statement[] query,
        Resource[] variables,
        Resource[] existentials,
        RDFNode [][] hints
    ) throws RDFException;
    public Set queryMulti(
        Statement      s,
        Resource[]     existentials,
        RDFNode [][]   hints
    ) throws RDFException;

    public void add(Statement s) throws RDFException;
    public void add(IRDFContainer c) throws RDFException;
    public void remove(
        Statement  pattern,
        Resource[] existentials
    ) throws RDFException;
    public void replace(
        Resource   subject,
        Resource   predicate,
        RDFNode    object,
        RDFNode    newValue
    ) throws RDFException;

    public Resource        getStatementID(Statement s) throws RDFException;
    public Resource[]      getAuthors(Statement s) throws RDFException;
    public Resource[]      getAuthors(Resource id) throws RDFException;
    public Statement       getStatement(Resource id) throws RDFException;
    public Resource[]      getAuthoredStatementIDs(Resource author) throws RDFException;
    public Statement[]     getAuthoredStatements(Resource author) throws RDFException;

    public int         size() throws RDFException;
    public Iterator    iterator() throws RDFException;
    public boolean     supportsEnumeration();
    public boolean     supportsAuthoring();
}
```

Each object implementing the `IRDFContainer` interface is supposed to have logged into a given RDF store with a given identity during its construction. It thereby obtains a ticket string, which it stores away and uses whenever it needs to interact with the store. For that reason, the various data

accessing and modifying methods of `IRDFContainer` do not take ticket strings as the corresponding methods in `IRDFStore`.

`IRDFContainer` objects may also support extra functionalities. For instance, an `IRDFContainer` object can automatically record time and date of each statement added to an RDF store. For the exact semantics of `IRDFContainer`, consult [25].

4.1.2 Services and Agents

Services and agents are synonymous in Haystack, although the former implies the provision of short-lived responses to unrelated functionality requests while the latter is associated with persistent algorithms that perform long-termed tasks. Every service or agent implements the `edu.mit.lcs.haystack.server.service.IService` interface:

```
public interface IService extends IPersistent {
    public void init(
        String         basePath,      // Haystack's data directory
        ServiceManager manager,       // the service manager of Haystack
        Resource       res            // resource representing the service
    ) throws ServiceException;

    public void cleanup() throws ServiceException;

    public void shutdown() throws ServiceException;
}
```

Each service is started with a call to its `init()` method. This method takes several arguments including an `edu.mit.lcs.haystack.service.service.ServiceManager` object. This object supports several methods, including `getRootRDFContainer()`, which returns the root RDF container with which the service can store and recall data. The `ServiceManager` object provides the same RDF container to all services and agents, allowing them to use that container as a blackboard through which to communicate with one another.

The method `IService.cleanup()` is called to let a service prepare for termination, and the method `IService.shutdown()` is called just before a service is terminated and dissociated from the service manager.

4.1.3 User Interface Framework

The user interface of Haystack, named Ozone, consists of a framework on top of which domain-specific UI components are built. The framework provides a basic set of widgets with which sophisticated UI components can be composed. The framework is extensible so that more widgets can be installed and used just like the native widgets. Domain-specific UI components can also reuse

one another so to achieve consistency through the UI and to lessen the amount of work for UI designers. The next chapter will describe this UI framework in details.

Ozone can be considered a special agent in that it interacts with the RDF store just as services and agents. In fact, one can suppose that the user him- or herself is the agent of change in this case, and that the user interface of Haystack is simply the, aptly named, interface through which the user effects the desired changes. Ozone does differ from agents and services in that while agents and services can usually be started and stopped during a single Haystack usage session, Ozone remains running throughout the session. In fact, it is the running of Ozone—the user interface—that defines the duration of the session. As a consequence, Ozone does not need to implement `IService`, whose purpose is for starting and stopping services.

In addition to effecting changes in the RDF store, Ozone also watches for changes to particular information in the store in order to maintain faithful presentations of such information on the screen. Such changes may be caused by other agents or by Ozone itself. One part of the UI might cause a change that is reflected in another part of the UI.

As shown in Figure 20, Ozone can also interact with other services and agents on behalf of the user. For instance, Ozone can start a multimedia playing service and then display its progress by the second.

4.2 Basic Data Models

As with any platform, Haystack uses a set of basic data models which are used repeatedly through the system because they represent data patterns common in all domains. As an information management environment, Haystack in particular needs a means to aggregate information objects into collections and lists for several purposes including organization.

4.2.1 Collections

A collection in Haystack is a mathematical set (no order, no duplicates). There are two ways to model collections. When all members of a collection bear a common and explicitly specified relationship with another object, the collection can be modeled by a particular predicate connecting that other object with the collection members. Figure 21 shows a collection of attachments modeled using the `mail:attach` predicate on a single e-mail message. (`mail:` is an XML-style prefix that expands to

http://haystack.lcs.mit.edu/schemata/mail#.) Attachments A and B together make up the collection in question.

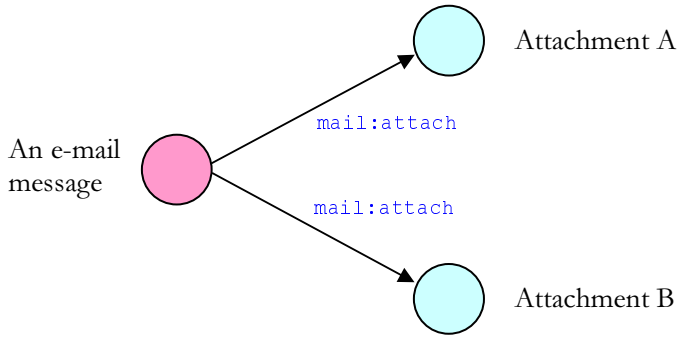

Figure 21. A sample implicit collection

When the members of a collection bear no common relationship in particular with any other object, we need to create a new resource to explicitly model the collection, and then use the generic predicate hs:member to connect the members with that resource. The resource representing that collection is asserted to be of the DAML class hs:Collection.

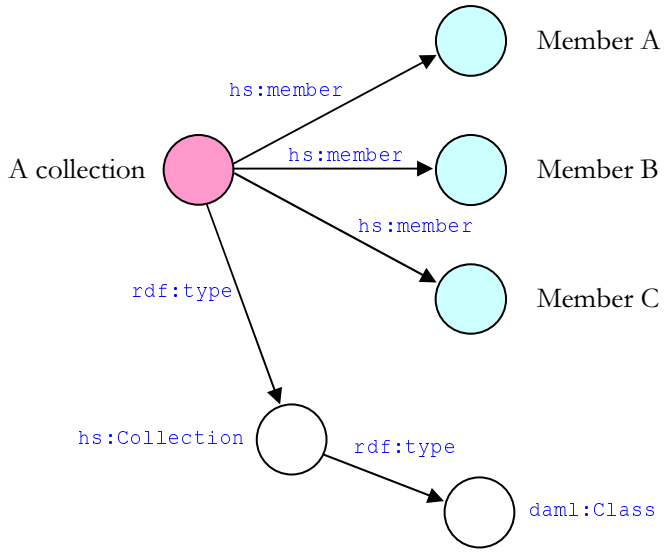

Figure 22. A sample explicit collection

4.2.2 Lists

Haystack makes use of the DAML list model for encoding immutable lists. In this model, there is only one list that is empty; it is referred to by the URI daml:nil. Non-empty DAML lists have their own URIs and are modeled in the first-rest manner with two predicates daml:first and daml:rest.

Figure 23 illustrates a sample DAML list. Note that the suffix of a DAML list starting at any valid index is also a DAML list.

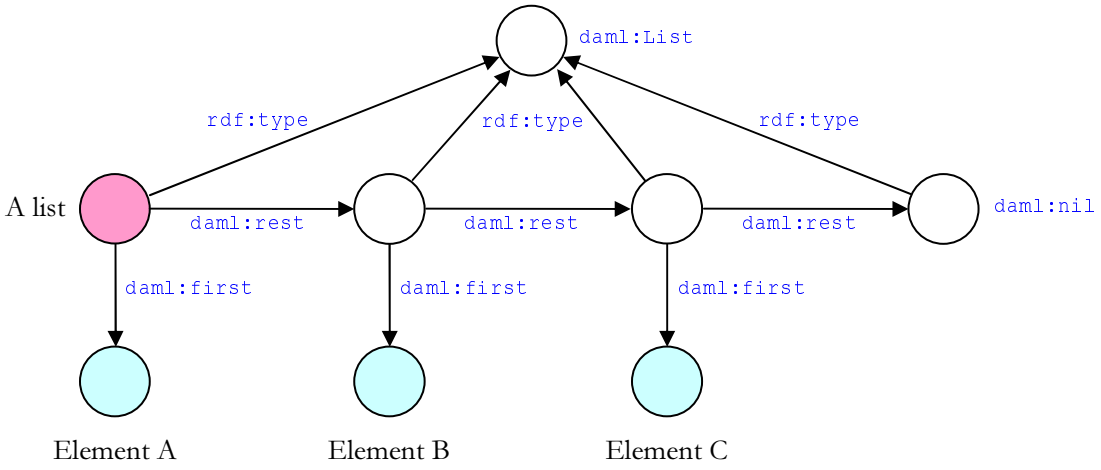

Figure 23. A sample DAML list

DAML lists are immutable because a non-empty DAML list cannot become empty without changing its identity into the resource named by the URI `daml:nil`. Haystack models a mutable list by wrapping a DAML list with another resource of type `hs:List`. Figure 24 shows a sample non-empty mutable list. Figure 25 shows the same list with all elements removed. Note that the mutable list retains its identity, i.e. its URI.

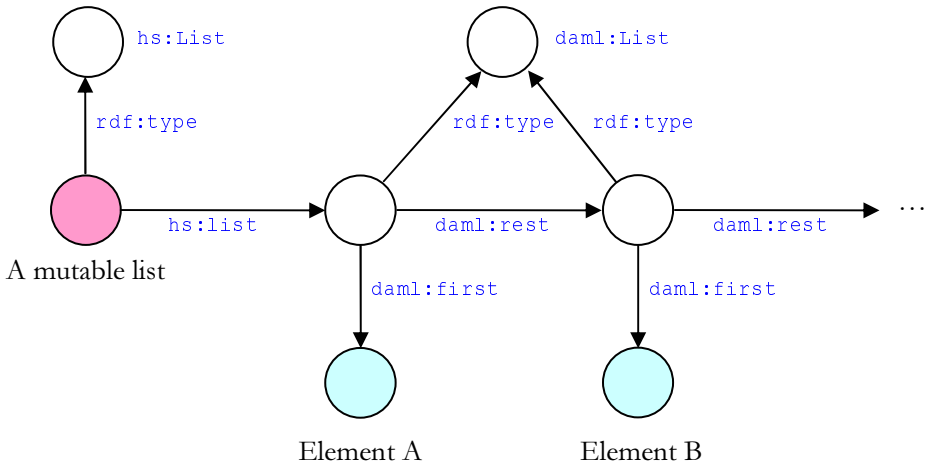

Figure 24. A sample non-empty mutable list

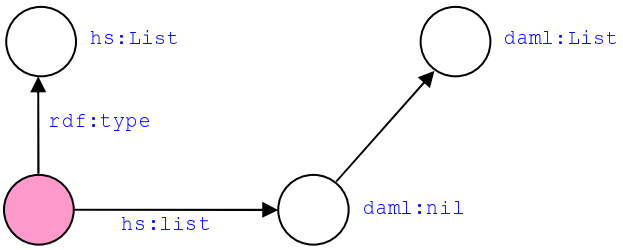

Figure 25. A sample empty mutable list

4.3 Adenine

In order to ease the manipulation of RDF data, Haystack supports a script-like language called Adenine [39]. Instead of having to write long expressions such as the following because the syntax of Java is designed to be generic:

```
RDFNode title = anRDFContainer.extract(
    new Resource("urn:haystack:favorites"),
    new Resource("http://purl.org/dc/elements/1.1/title"),
    null
);

Set titlesOfMembers = anRDFContainer.query(
    new Statement[] {
        new Statement(
            new Resource("urn:haystack:favorites"),
            new Resource("http://haystack.lcs.mit.edu/schemata/haystack#member"),
            new Resource("urn:haystack:wildcard:m")
        ),
        new Statement(
            new Resource("urn:haystack:wildcard:m"),
            new Resource("http://purl.org/dc/elements/1.1/title"),
            new Resource("urn:haystack:wildcard:t")
        )
    },
    new Resource[] { new Resource("urn:haystack:wildcard:t") },
    new Resource[] {
        new Resource("urn:haystack:wildcard:m"),
        new Resource("urn:haystack:wildcard:t")
    }
);
```

one can write much more compact expressions in Adenine to achieve the same effect:

```
@prefix dc: <http://purl.org/dc/elements/1.1/>
@prefix hs: <http://haystack.lcs.mit.edu/schemata/haystack#>

= title (extract <urn:haystack:favorites> dc:title ?x)
= titlesOfMembers (query { <urn:haystack:favorites> hs:member ?m ?m dc:title ?t } @(?t))
```

As URIs, RDF literals, and RDF statement arrays are native syntactic elements of Adenine, a lot of syntactic burdens in Java, particularly `new` operators, are unnecessary. URIs can also be shortened through the use of XML-style prefix notation (e.g. `dc:title` instead of

<http://purl.org/dc/elements/1.1/title>). Manipulations of RDF data can be expressed more succinctly in Adenine—this advantage allows for faster and easier development of Haystack.

Fragments of Adenine code can be wrapped in Adenine methods for later invocation. The following is a sample Adenine method that duplicates any given collection:

```
method <urn:utilities:duplicateCollection> collection
    = members (query { collection hs:member ?x } @( ?x ))
    = newCollection ${ rdf:type hs:Collection }
    for m in members
        add { newCollection hs:member m[0] }
    return newCollection
```

Each Adenine method is given a URI just like any other information object in Haystack. In order word, Adenine methods are first-class, can be referred to and linked to other information in Haystack. For instance, if <urn:anAppointment> is the URI of an appointment, one could specify how to notify the user of the appointment when its time is approaching as follows:

```
add { <urn:anAppointment> <urn:appointment:howToNotifyUser> <urn:utilities:beep> }

method <urn:utilities:beep>
    ...
```

For the exact syntax and semantics of Adenine, consult [39]. Code samples from this point onward will be shown in either Java or Adenine.

CHAPTER 5

User Interface Framework Design Rationales

This chapter focuses on the design rationales with which the Haystack UI framework has been implemented. It starts out with a discussion on the general idea of transforming information for the purpose of presentation, and then dives into four specific ideas of abstraction useful for formulating presentations. The overall process of presenting information is then discussed with these abstractions incorporated. Next, the concept of *environment* is described with respect to how it is useful for providing context to the process of presenting information. Finally, optimization issues are considered.

5.1 Information Transformations

When an information object is destined to be presented to the user, much processing needs to be carried out before pixels get painted onto the screen or sounds get played through the audio speakers. This processing involves making decisions of two types:

1. Selecting some information about that object to present (i.e. *what* to show); and

2. Formatting the selected information before rendering it to the screen or playing it on the speakers (i.e. *how* to show).

For instance, in order to present a meeting given little screen estate, we first decide to show only its title and its participants (rather than its other details including its location, agenda, prerequisite reading materials, etc.); this is deciding *what* to present. We then decide to show the title as a string on one line followed by the participants each on a line; this is deciding *how* to show the selected information.

If the title of the meeting is already modeled as a literal in RDF, there is little processing needed to render it, save converting the RDF literal into a string and calling the operating system's graphical APIs to paint the text string onto the screen. One can say that the task of presenting the title is more *syntactic* and less *semantic* in nature compared to the task of presenting the meeting because the former is closer to the actual pixel rendering while the latter is further removed from it.

Similarly, the task of presenting the participants each on a line is more semantic in nature than the task of presenting the title because for each participant we still need to make more decisions: what information about the participant to present and how to show it. For example, we can decide to present his or her full name as well as his or her office. The full name is to be shown as a string, and the office is also to be presented as a string on the same line (as opposed to, say, a highlighted location on a 2D map). The task of presenting the office as a string also demands more decisions: we need to decide to show only the office's number, and to show it as a string. As an alternative, we might have decided to show in addition the building in which the office is located.

All of these decisions together make up a process that transforms information originally in RDF to information encoded in pixels on screen (or sounds through speakers). From the perspective of the computer, information originally rich in semantics loses its semantics and gets "flattened out" as it propagates along this process, until it is modeled only in pixels or sounds. However, as more and more transformations are made, the information conveys more and more semantics to the user: the final set of pixels showing the original object communicates meanings more readily to the user than the original RDF model which is made up of opaque URIs linked together in a complex graph.

Note that the information involved in these transformations is not necessarily connected to the original object to present. System-wide font and color settings, for instance, may be stored in RDF but are not directly associated with the object to present. Nevertheless, such settings are used in some of the transformations leading to the final text rendering.

Neither does that information need to come only from the RDF data in the system. The size of the Haystack windows at runtime, for instance, influences how all UI elements are laid out. The position

of the mouse pointer determines whether a particular UI element (e.g. a hyperlink) should be highlighted.

This process of transformations goes as far as the writes into video memory (at which time the information to present is just RGB values and is very removed from the original object). However, since the UI framework does not deal directly with video memory, let us consider that the process terminates at the calls to the Java methods wrapping native operating system's graphical APIs.

5.2 View Abstraction

The entire process of presenting a piece of information can involve many decisions, as illustrated in the example of presenting a meeting in section 5.1. To present that meeting, decisions need to be made not only about the meeting object itself, but also about its participants and their offices. The UI designer who is responsible for writing code to present the meeting might not be knowledgeable about how to present the participants and their offices. As a consequence, he or she might not want to make decisions about these items, nor is he or she the best person to make those decisions. Furthermore, these items are also presented elsewhere rather than just inside meetings; the decisions needed to be made to present them must also be made again in different situations.

The solution to these two problems is to delegate the decisions of presenting a particular object to whomever is most knowledgeable about it, and to reuse his or her decisions whenever that object needs to be presented. This solution is implemented through the concept of *views*. A view of something is an onscreen (or on speaker) representation of that thing, in pixels and perceivable visually (or in sounds and perceivable acoustically).

When the UI designer of the meeting in our running example wants to specify how to present one of the meeting's participants, the designer specifies that a request be issued for a view of that participant. The request is aptly called a *view request*. This view request describes the characteristics of the desired view of that participant. A *view producer* is located who can understand that view request. The view producer analyzes the request and produces some renderings of the original object or some more view requests for the same object and/or other relevant objects. These subsequent view requests cause more view producers to be spawned and yet more renderings or view requests to be produced, until all the last view producers spawned produce only renderings and no more view requests.

By issuing a view request, the UI designer has delegated the decisions necessary for showing the participant to both the UI framework and other UI designers. The UI framework is responsible for dynamically (at runtime) locate a view producer capable of understanding the given view request. The located view producer captures the decisions made by another UI designer for presenting the participant. This second UI designer might him- or herself delegate certain decisions, such as those needed for presenting the participant's office, to yet another UI designer by issuing another view request.

Figure 26 shows a first illustration of the transformation pipeline that turns view requests into pixels on the screen, i.e. views. This pipeline is simple: a view request is translated by a view producer into more view requests and/or into renderings on the screen. View producers are computations and colored gray. View requests and views are information and colored white.

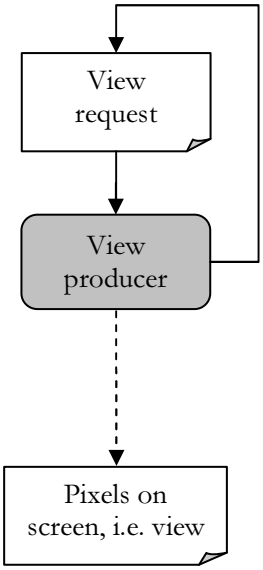

Figure 26. An initial illustration of the transformation pipeline

5.2.1 An Analogy

Let us take an analogy from the physical world. A screenwriter would like a scene of a historical building in a movie. In his script, he indicates that the historical building should be shot such that it appears both grand and ancient. The director of the movie interprets the script of the screenwriter and tells the cinematographer roughly which location and angle to shoot from. The cinematographer decides which camera lenses to use and how to set up the track, and gives the cameraman detailed instructions for the shot. The cameraman handles the camera according to the cinematographer, improvising where needed drawing from his past experience.

This analogy illustrates a chain of commands along which each decisions is being made by whoever is most capable of making that decision. The information communicated between one person and the next is a view request, while each person is a view producer responsible for elaborating instructions into more detailed, lower-level instructions.

The screenwriter only knows how the footage should look like; he neither knows what angles to shoot it from, nor how to operate the camera. However, if he describes well how he envisions the end result, and given an experienced director, a competent cinematographer, an adept cameraman, and all necessary equipments, the screenwriter will get his desired footage with just a well-written script.

5.2.2 View Request Specifications

It then begs the question of what goes into a script, or more relevantly, a view request. There are different types of view specification. The designer making a particular view request may be concerned with the screen estate occupied by the view and may consequently specify one of the following requests:

- "Give me a one-page summary of object X";
- "Give me a view of object X that fits in an area measured 50 pixels wide by 30 pixels high";
- "Give me a very short summary of object X in the form of piece of text that can fit into a text paragraph."

The designer may otherwise be concerned with the content of the view:

- "Give me a view of object X that shows how it is related to other objects";
- "Give me just an icon and a title for object X, or something enough for the user to identify the view of X";
- "Express object X as a noun phrase";
- "Present object X with some text and do not use any graphics."

Or else, the designer may care about the functionalities offered by the view:

- "Show picture Y such that the user can crop it";
- "Show document Z such that the user can edit it";
- "Let the user edit the relationships between object X and other objects";
- "Make the view behave like a hyperlink when clicked on."

The designer may be concerned with several aspects all at once and issue a combined request, e.g. "give me a view of object X that fits into an area of 50 x 30 pixels, with an icon and a title, showing its author and last edited date, and make it resizable in case the user wants to see it in a larger area".

Note that the specifications in a view request are not the only things that will affect the final view. For example, even if a view request limits the view to an area of 50 x 30 pixels, the view as perceived by the user will look different for different default system font sizes. In our movie analogy, weather conditions and equipment failures beyond the control and awareness of the screenwriter will still affect the final footage.

Furthermore, not all view requests can be met. If a view request specifies to focus on a piece of information which the user has no permission to see, the end result is a message to that effect (e.g. "Sorry, you have no permission to see X") rather than a view of that piece of information. As another example, if the focus of a view request is on a remote piece of information that cannot be downloaded immediately, the view request will not result in the rendering of the view of that piece of information immediately; rather, a progress message (e.g. "Retrieving information… 35%") is displayed in place until the information is downloaded.[3]

5.2.3 Persisting View Requests

View requests are made by UI designers to present information to users. Should users be dissatisfied with the end results, i.e. the views, they should be able to change the views by changing some of the things that generate those views. In certain situations, users want their changes to be temporary and in other situations, permanent. Modifications intended to be permanent must be persisted. The best place in Haystack to persist these modifications is the RDF store. View requests should, therefore, be described and persisted in the RDF store.

[3] The word "request" in "view request" acknowledges the fact that some view requests might not be met. Calling a view request by other names such as "a view host" asserts that a view will certainly be created, which may prove contrary to the reality.

RDF also allows us to describe the many different types of specification in view requests. We reap all the benefits of RDF: the schemas of these specifications are easily extensible and the view requests can be mingled with the rest of the data corpus.

5.3 Rendering Abstraction

As mentioned so far, view producers issue more view requests and/or render to the screen. This rendering task is complex, consisting of creating system fonts, colors, and brushes, calculating kerning and spacing, handling zoom and occlusion, optimizing text wrapping, painting text, applying font decorations (e.g. underlining), painting bitmaps, painting geometries, anti-aliasing, bitblt-ing, etc. Nevertheless, a lot of work has already been done by the operating system. For instance, text can be painted just by specifying a string, a font, and a color; there is no need to paint the curves that make up the characters, to fill in the characters, or to anti-alias them. The operating system has already provided an abstraction for graphical output. Since view producers work at the level of information, rather than at the level of system font objects, screen coordinates, etc., this abstraction proves too crude, too fine-grained, for view producers.

So that view producers can be coded more easily, UI designers need to be able to effect rendering actions at a higher level of abstraction than that of system font objects, screen coordinates, etc. We have seen such an abstraction in presentational mark-up languages like HTML. Authors of HTML documents can specify high-level rendering requests (e.g. "paint this paragraph of text at 200% the default font size") and delegate a lot of low-level rendering logistics to the web browser (e.g. calculating the absolute font size, wrapping the given paragraph of text, and painting it).

Following the solution of presentational mark-up languages, Haystack provides a set of types of high-level rendering request that UI designers authoring view producers can make use of to specify high-level rendering goals, rather than low-level rendering logistics. These rendering request types resemble the set of HTML elements. They include, for instance, Text span rendering requests, Paragraph rendering requests, Image rendering requests, etc.

5.3.1 UI Elements

Of course, the low-level rendering logistics still have to be handled by some pieces of code—the handlers that execute these high-level rendering request. However, such logistics need to be encoded only once and they can then be reused by many UI designers. The UI designers need not be aware of

how those logistics are handled, just as HTML authors never need to know how web browsers are implemented.

The handlers for high-level rendering requests are called *rendering producers*, or in a more familiar term, *UI elements*. They translate rendering requests, which specify high-level rendering goals, to low-level rendering logistics that result in pixels painted on the screen. Figure 27 updates the transformation pipelines with rendering requests and UI elements.

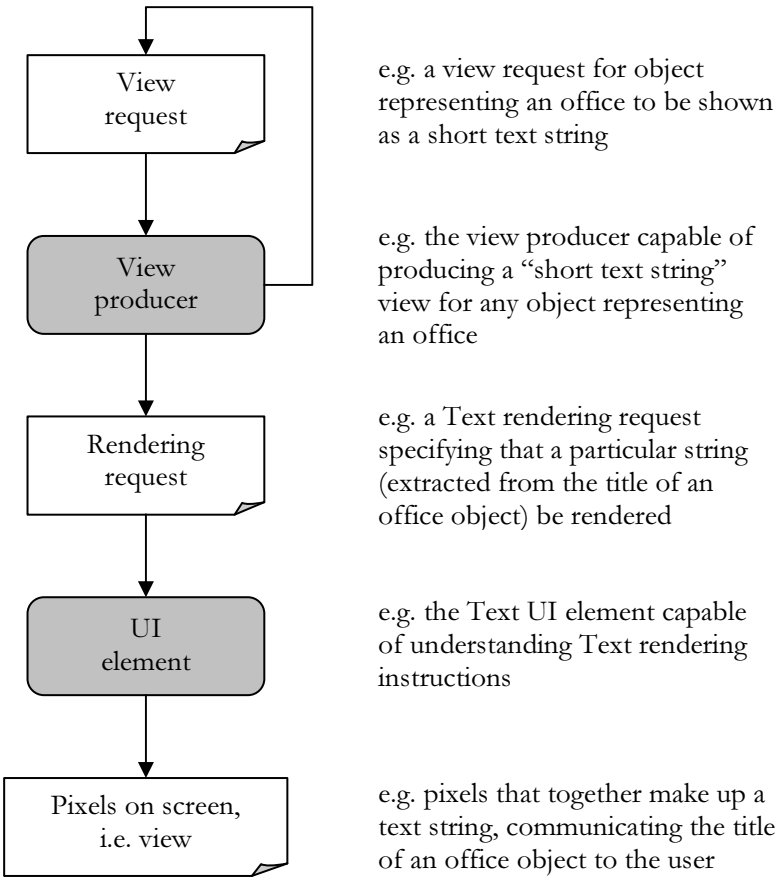

Figure 27. A second illustration of the transformation pipeline with rendering requests and UI elements included

5.3.2 Persisting Rendering Requests

Just like view requests, rendering requests are also persisted so that their modifications by users can remain permanent. Modifications to rendering requests are syntactic in nature, e.g. "bold this text".

5.3.3 Composing Rendering Requests

Haystack's rendering requests are much like HTML elements, and the best way to compose them to achieve complex renderings is to nest them within one another, just as HTML elements are nested. Here is an example of three rendering requests expressed in Adenine's syntax: [4]

```
${  rdf:type              slide:ParagraphRequest ;
    slide:children @(
        ${  rdf:type      slide:ImageRequest ;
            slide:source  <http://haystack.lcs.mit.edu/sampleData/flowers/mayflower.gif>
        }
        ${  rdf:type      slide:TextRequest ;
            slide:text    ": Mayflower is the state flower of Massachusetts"
        }
    )
}
```

The three requests are the three anonymous resources declared to be of type `slide:ParagraphRequest`, `slide:ImageRequest`, and `slide:TextRequest`. One can consider the RDF statements declaring their types to be their "opcodes". The rest of the information are the parameters of the requests. For example, the `slide:ImageRequest` request in this example is given the URI of the image to be displayed, and the `slide:TextRequest` request is given the text to render. The `slide:ParagraphRequest` request is given a list of child rendering requests; its job is to coordinate these child requests to render their data such that the output flows like text in a paragraph.

The HTML code segment equivalent to the three Haystack rendering requests above is as follows:

```
<P>
    <IMG src="http://haystack.lcs.mit.edu/sampleData/flowers/mayflower.gif">
    <SPAN>: Mayflower is the state flower of Massachusetts</SPAN>
</P>
```

Of course, in HTML, `<SPAN>` elements are often neglected. Note, however, the common nesting structure present in both the HTML code and the Adenine code. This structure dictates how the various UI elements (or HTML elements) coordinate their outputs on the screen.

5.3.4 Mixing View Requests with Rendering Requests

Since view producers generate both rendering requests and more view requests, it must be possible to mix view requests with rendering requests so that view producers can specify where a view (resulting from a particular view request) will be positioned relative to other parts of the screen. Here is an example of the rendering requests and the view request generated by the view producer for a participant in our running example:

[4] Note that this is not a piece of Adenine code. This is data added to the RDF store and expressed here in Adenine's syntax for convenience.

```
${  rdf:type                          slide:ParagraphRequest ;
    slide:children @(
        ${  rdf:type                  slide:TextRequest ;
            slide:text                participantName
        }
        ${  rdf:type                  ozone:ViewRequest ;
            ozone:focusOn             participantOffice ;

            # some specifications of how the view should behave
        }
    )
}
```

These requests and the view request together specify that the participant's view should take on the form of a paragraph (i.e. it never starts in the middle of another paragraph, but must stand by itself). The name of the participant (shown here as `participantName`) is painted first in that paragraph, followed by a view of the participant's office (`participantOffice`). The view request can be considered a *macro* rendering request, in that it ultimately results in rendering, but must be "expanded" first at runtime.

5.4 Layout Abstraction

Back to our running example of presenting a meeting: the view producer of the meeting can put together a rendering of the meeting's title and views of the meeting's participants (assuming there are two participants) by generating the following rendering requests and view requests:

```
${  rdf:type                          slide:ParagraphRequest ;
    slide:children @(
        ${  rdf:type                  slide:TextRequest ;
            slide:text                meetingTitle
        }
        ${  rdf:type                  slide:ParagraphRequest ;
            slide:children @(
                ${  rdf:type          ozone:ViewRequest ;
                    ozone:focusObject  participant1 ;
                    # some specifications of how the view should behave
                }
                ${  rdf:type          ozone:ViewRequest ;
                    ozone:focusObject  participant2 ;
                    # some specifications of how the view should behave
                }
            )
        }
    )
}
```

Most likely, the view producer does not discriminate among the two participants. Consequently, the two view requests should be identical except for the objects that they focus on. If another participant is added to the meeting, the view producer must generate another view request looking exactly the same as the existing two view requests, save its focus object.

This pattern of work being duplicated to present several objects together is seen over and over again, and it calls for an abstraction. What the view producer of the meeting really aims to specify is how each participant should be shown and how the views of all the participants should be laid out together. To put it another way, the view producer of the meeting needs to issue a view request for all participants of the meeting collectively together. This view request includes specifications for how the views of individual participants should look and behave, as well as specifications for how all those views should look and behave together. Here is a revision of the preceding RDF data segment:

```
${  rdf:type                          slide:ParagraphRequest ;
    slide:children @(
        ${  rdf:type                  slide:TextRequest ;
            slide:text                meetingTitle
        }
        ${  rdf:type                  ozone:ViewRequest ;
            ozone:focusObjects        @( participant1 participant2 ) ;

            # some specifications for individual views of participants

            # some specifications for the overall collective view,
            # such as it should behave like a paragraph
        }
    )
}
```

The view producer is now concerned only with updating the list of participants that hangs off the `ozone:focusObjects` predicate on the view request. There is no need to generate more view requests for newly added participants.

The specifications for the overall collective view are referred to as *layout constraints*, since how the collective view looks like depends much on how the individual views are laid out with respect to one another. For example, they can be listed vertically one after another as rows in a list, or made to flow like text in a paragraph with comma separators in between, or clustered into piles of icons, or laid out as tiles on a rectangular grid, etc.

5.5 Data Computation Abstraction

The concept of views abstracts away all the decisions that go into presenting a piece of information, or several pieces of information collectively. However, in order to issue a view request, a UI designer must know which piece of information, or pieces of information, identified by a URI or several URIs, to focus on. In certain situations, the UI designer does not know how exactly to retrieve the information to present.

For example, a UI designer is tasked with specifying how to present an instant message and he or she decides to include the title of the message in the produced view. The instant message is of type "Instant Message," which is a subtype of the type "Message." The subject of a message is often considered its title, and the subject is annotated on the message using the predicate `message:subject`. The message might also have another title specified using the standard Dublin Core predicate `dc:title`. Furthermore, the message may have several `dc:title` and `message:subject` predicates hanging off it. Determining the text supposedly conveying the title of a message can be computationally complicated and may require more intimate knowledge of the schema of the type "Message" than the UI designer in this example possesses. Furthermore, that same computation is needed by any other UI designer tasked with specifying how to present objects of subtypes of "Message."

It may also be that all instant messages are presented using the same view producers designated for presenting all messages. However, the problem is now reversed. The title of an instant message may be specified using another predicate, other than `dc:title` and `message:subject`. As a result, the view producers designed to present messages in general will not present instant messages properly, as they do not know about this other predicate.

These examples point out that data retrieving computations should be abstracted away, so that the decisions necessary for retrieving a property from an object, or retrieving any piece of information or several pieces of information in general, can be delegated appropriately to whoever knows best.

The following RDF data illustrates how a *data request* can be made to "order" the title of an instant message:

```
${  rdf:type           slide:TextRequest ;
    slide:textDataRequest ${
        rdf:type       data:DataRequest ;
        rdf:type       data:TitleDataRequest ;
        ozone:focusOn  instantMessage
    }
}
```

Just as view producers translate view requests ultimately into views, a *data producer* will be located to translate the data request in this code sample into data understandable by the Text rendering request.

Data requests are useful not only to encapsulate complex computations, but also to package reusable computations. For example, sorting capabilities can be supported through some types of data request. UI designers only need to specify data requests of those types given the desired sorting orders, rather than implementing sorting algorithms themselves. Another reusable computation is the

translation of a system date (e.g. "Thu Nov 19, 2002 21:39:06 EST") to a more human-readable string (e.g. "Tomorrow at 9pm").

In essence, whereas rendering requests abstract away low-level logistics of painting to the screen, data requests abstract away low-level logics of extracting data from the RDF model. By abstracting away the two ends of the transformation pipeline, we allow UI designers to concentrate on the high-level algorithms that transform data from one end, the RDF model, to the other, the screen.

Data requests and data producers are not only useful for constructing views. They can be used in any situation where the computation of data needs to be abstracted away and changes to the data are to be watched and responded to. For example, an agent responsible for watching memberships in a collection may benefit from abstracting away how memberships are being asserted in that collection. Another agent interested in analyzing titles of objects may be made oblivious to how each object's title is encoded through the use of data requests and data producers.

5.6 The Transformation Pipeline

This section recaptures all of the abstractions I have discussed previously into one transformation pipeline illustrated in Figure 28. This figure shows the dataflow between various parts of the pipeline; white blocks are data objects and gray blocks are code components. The flow is recursive but we start analyzing it at the view request. A view producer translates a view request into zero or more view requests and zero or more rendering requests. A view request might contain a data request, which is translated by a data producer to yield data that specifies which piece(s) of information to focus on. A UI element translates a rendering request into pixels on the screen. A rendering request might also contain a data request, which is translated by a data producer to yield more parameters for the UI element.

Note that view producers are allowed to paint directly to the screen if they so choose. That, however, is not a common mode of operation, since the UI elements should have already abstracted away most common rendering needs.

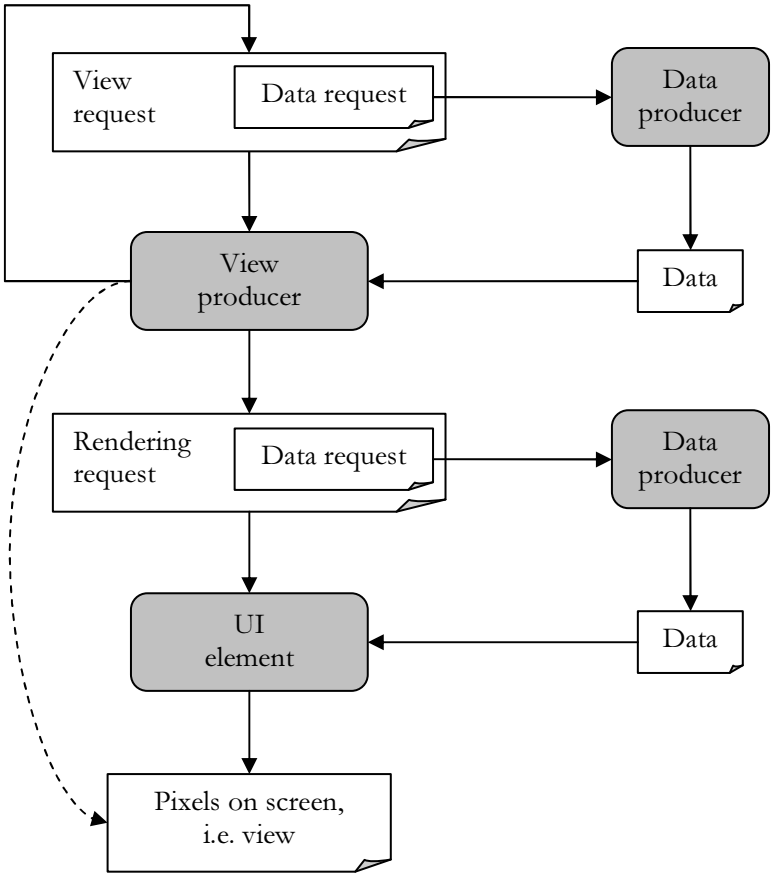

Figure 28. The transformation pipeline

5.7 Transformation Trees

Of course, the transformation pipeline, when actually run, is not recursive as in Figure 28 but rather unfolds into a *transformation tree* as different instances of each type of components in Figure 28 are distinguished from one another. Figure 29 shows an example of such a tree, neglecting data requests and data producers. The transformation tree in this figure ends *before* all the dashed arrows; its leaves are the four UI elements. After the arrows, the screen renderings (pixel blobs) of the UI elements are shown in a spatial nesting hierarchy.

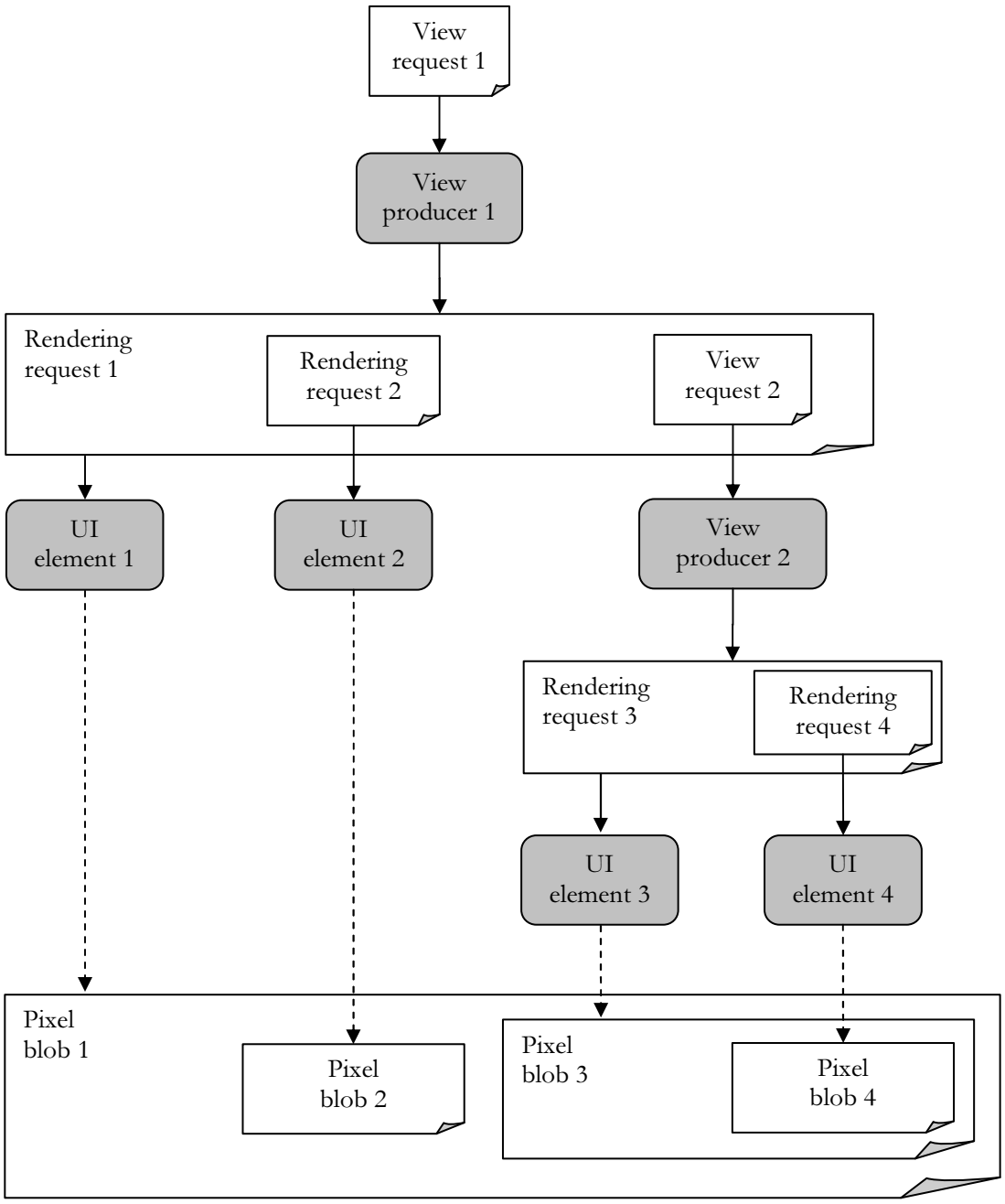

Figure 29. A transformation tree

Each transformation tree captures all decisions made to present some piece(s) of information as requested by the view request at the root of the tree. When the user manipulates some part of the screen, i.e. a pixel blob, there is exactly one path from that pixel blob back to the root of the transformation tree. This path traces through a sequence of decisions, which tell us which pieces of information contribute to the rendering of the manipulated blob of pixels. For example, if pixel blob #4 in Figure 29 is manipulated, we can trace back to the root of the tree and find that two view

requests (#1 and #2) have been issued to render the pixel blob #4. Using this technique, we infer that the user wants to interact with whichever pieces of information those two view requests focus on.

This ability to trace back from pixels to information allows us to systematically expose semantics of the information through the pixels. For example, when the user right-clicks on the pixel blob #4 in Figure 29, we can enumerate all operations that are applicable on the pieces of information focused on by the view requests #1 and #2, and show such operations in a context menu. Drag-and-drop can also be supported systematically in this way.

The overall result is that views become proxies through which the user can interact with the information being presented. This is simply direct manipulation, but it can now be supported systematically and uniformly by the UI framework rather than by individual UI designers.

5.8 Environments

So far we have been concerned with producing information for users to see. However, what users see depends on what the various components in transformation trees can "see". All information that can be "seen" by a particular component—the information that it has access to—is referred to as its *environment*.

Environments depend on system settings such as the current user's security privileges. For instance, a view producer tasked to present the employee record of a division manager to an office assistant might not "see" any property of the manager that is considered personal (e.g. marital status) or confidential (e.g. salary). It is not sufficient to restrict the assistant from seeing the values of such properties, but also from seeing whether such properties have any assigned values at all. That is, all RDF statements encoding those properties must be absent in the environment of that view producer.

The environment of a component can also depend on where that component is located digitally. A view request might be shipped from one machine to another because the view producer capable of translating that request resides on the second machine. That view producer "sees" (some of) the information stored on the second machine and perhaps some information on the first machine. A view producer residing on a machine not connected to any network may have a different environment when the machine gets connected.

Furthermore, the environment of a component can also depend on the context in which it is used. A view producer tasked with presenting a person inside the view of a meeting should be told such fact so that it can incorporate functionalities in the resulting view to suit the context of use (e.g. add an "uninvite" button). A Text rendering request should be told about inherited rendering settings such as font size, color, etc.

The reader might argue, "what isn't context?" Indeed, all things influencing an environment, including system settings, digital location, network connectivity, etc., are all context. Each component uses its environment—what it is allowed to see—to attempt to understand its context and act accordingly. The component's comprehension of its context may be incomplete or misled, as there might be external factors (e.g. security enforcers) conspiring to affect its environment without projecting themselves in the environment.

5.9 Optimizations

Having specified the various abstractions in the entire pipeline through which information is transformed from internal representations to external views, I now turn to the issue of optimizations, which also involves design rationales that influence the actual implementation of the UI framework. Optimizations are intended to speed up the application of transformations and their reapplications in response to changes. That is, when a piece of information is to be presented, its view must arrive on screen quickly and remain responsive to the user's manipulations as well as faithful to the data in the RDF store.

Fortunately, the transformation pipeline has already been divided into components by our various abstractions. These components can be optimized individually depending on their natures. For example, UI elements deal more with device I/O than with the RDF store: they should be optimized in Java so that they remain responsive to manipulations by the user. View producers, on the other hand, work almost solely with RDF: they should be written in Adenine for the convenience of UI designers.

Our decision to persist view requests, rendering requests, and data requests in the RDF store for the purpose of retaining users' preferences is also an optimization: such persistence is a form of caching that avoids the need to regenerate when a view request is issued more than once.

In other cases, not persisting data in the RDF store is an optimization. For example, data producers allow data to be channeled directly into UI elements and view producers rather than being serialized

first into the RDF store. As a consequent, the store contains less data and functions more quickly. Of course, if the computation done by a data producer is complicated and time consuming, the data producer has a choice of caching the result in the store.

These optimizations cause some code to be in Java and some code to be in Adenine, and some data to be in RDF and some data to be members of live Java objects. This dichotomy calls for a hybrid component architecture which is the topic of the next chapter.

CHAPTER 6

User Interface Framework Implementation

Given all of the design rationales discussed in the previous chapter, I now delve into the implementation details of the Haystack UI framework. First, the component architecture of the framework is described. Then the implementations of the three abstractions for view, rendering, and data computations are explained. Finally, the uniform support for direct manipulation in Haystack is discussed.

6.1 Component Architecture

The transformation process employed to turn information in internal representations to external views involves several code components that perform the actual transformations. The code components are called evaluators while the information being transformed are called prescriptions. Table 3 lists all the types of prescription and evaluator encountered thus far.

Table 3. Specific types of prescription and evaluator

Prescription	**Evaluator**
View request	View producer
Data request	Data producer
Rendering request	Rendering producer / UI element

Evaluators need to be located dynamically based on the nature of the prescriptions they are to

transform, and then loaded for invocation, executed, and eventually disposed. The management of these evaluators calls for a component architecture.

There are three requirements for the component architecture:

1. It must be extensible in that new evaluators can be easily added;
2. It must be able to deal with both Java evaluators and Adenine evaluators;
3. It must be able to keep certain evaluators alive after they have performed the needed transformations, so that reapplications of those transformations can be carried out quickly should there be any change affecting them.

To meet the first requirement, evaluators are registered in the RDF store. That is, their descriptions (e.g. how to invoke them, what types of prescription they operate on) are described in RDF. To incorporate new evaluators, one adds their metadata into the RDF store and makes sure that their programmatic realizations (i.e. Java class or Adenine method) are also in the system.

The second requirement calls for a bridge between Java and Adenine. We implement this bridge through a generic Java class that can be parameterized with an Adenine method to call. This choice ensures that all evaluators are fundamentally implemented in Java, so that they can all be managed in the same manner.

The third requirement demands evaluators to be incarnated as live Java objects, as opposed to Java methods, so that they can remain alive until explicitly disposed of. As a consequence, there needs to be a Java interface for initializing and disposing evaluators.

6.1.1 Registration of Evaluators

So that the UI framework can systematically look up an evaluator given a prescription, the mappings between evaluators and prescriptions must be described in a standard way. This is done by using a common predicate to assert for each evaluator the types of prescription that it can understand. We use the predicate `ozone:handlesPrescriptionType` for this purpose. Here are examples of the metadata used to describe various evaluators:

```
add {
    :textUIElement
        rdf:type                      ozone:Evaluator ;
        rdf:type                      ozone:UIElement ;
        ozone:handlesPrescriptionType slide:TextRequest ;
        hs:implementation ${
```

```
            rdf:type       hs:JavaClass ;
            hs:className   "edu.mit.lcs.haystack.ozone.parts.slide.TextElement"
        }

    :viewProducer
        rdf:type                      ozone:Evaluator ;
        rdf:type                      ozone:ViewProducer ;
        ozone:handlesPrescriptionType ozone:ViewRequest ;
        hs:implementation ${
            rdf:type       hs:JavaClass ;
            hs:className   "edu.mit.lcs.haystack.ozone.ViewProducer"
        }

    :webpageLineSummaryViewProducer
        rdf:type                      ozone:Evaluator ;
        rdf:type                      ozone:ViewProducer ;
        ozone:handlesPrescriptionType web:WebpageLineSummaryViewRequest ;
        ozone:adenineMethod           :produceWebpageLineSummaryView
        hs:implementation ${
            rdf:type       hs:JavaClass ;
            hs:className   "edu.mit.lcs.haystack.ozone.AdenineEncodedViewProducer"
        }

    :multiItemListViewProducer
        rdf:type                      ozone:Evaluator ;
        rdf:type                      ozone:ViewProducer ;
        ozone:handlesPrescriptionType layout:MultiItemListViewRequest ;
        hs:implementation ${
            rdf:type       hs:JavaClass ;
            hs:className   "edu.mit.lcs.haystack.ozone.layout.MultiItemListViewProducer"
        }

    :resourcePropertyDataProducer
        rdf:type                      ozone:Evaluator ;
        rdf:type                      data:DataProducer ;
        ozone:handlesPrescriptionType data:ResourcePropertyDataRequest ;
        hs:implementation ${
            rdf:type       hs:JavaClass ;
            hs:className   "edu.mit.lcs.haystack.ozone.data.ResourcePropertyDataProducer"
        }
}
```

Note that each evaluator is:

1. referred to by a URI,
2. declared to be of type `ozone:Evaluator` and of a specific type,
3. indicated to handle prescriptions of at least one type using the `ozone:handlesPrescriptionType` predicate, and
4. linked to an implementation Java class.

Prescriptions themselves are described in metadata as mentioned before. Here are a few examples:

```
add {
    :aRenderingRequest
        rdf:type              ozone:Prescription ;
        rdf:type              ozone:RenderingRequest ;
        rdf:type              slide:TextRequest ;
```

```
        slide:text              "Text to render" ;
        slide:fontBold          "true"

    :aViewRequest
        rdf:type                ozone:Prescription ;
        rdf:type                ozone:ViewRequest ;
        ozone:focusObject       <http://www.google.com/> ;
        ozone:superViewClass    ozone:LineSummaryView

    :aSpecificClassViewRequest
        rdf:type                ozone:Prescription ;
        rdf:type                ozone:ViewRequest ;
        rdf:type                web:WebpageLineSummaryViewRequest ;
        ozone:focusObject       <http://www.google.com/>

    :aMultiItemListViewRequest
        rdf:type                ozone:Prescription ;
        rdf:type                ozone:ViewRequest ;
        rdf:type                layout:MultiItemListViewRequest ;
        ozone:focusObjects      @( <http://www.google.com/> <http://www.google.com/> )

    :aDataRequest
        rdf:type                ozone:Prescription ;
        rdf:type                data:DataRequest ;
        rdf:type                data:ResourcePropertyDataRequest ;
        data:subject            <http://www.google.com/> ;
        data:predicate          dc:author
```

To clarify these examples, I have added all super types for each of the prescriptions. Figure 30 illustrates the relationships between a rendering request and a UI element in a graph. Relationships between prescriptions of other types and their corresponding evaluators are similar.

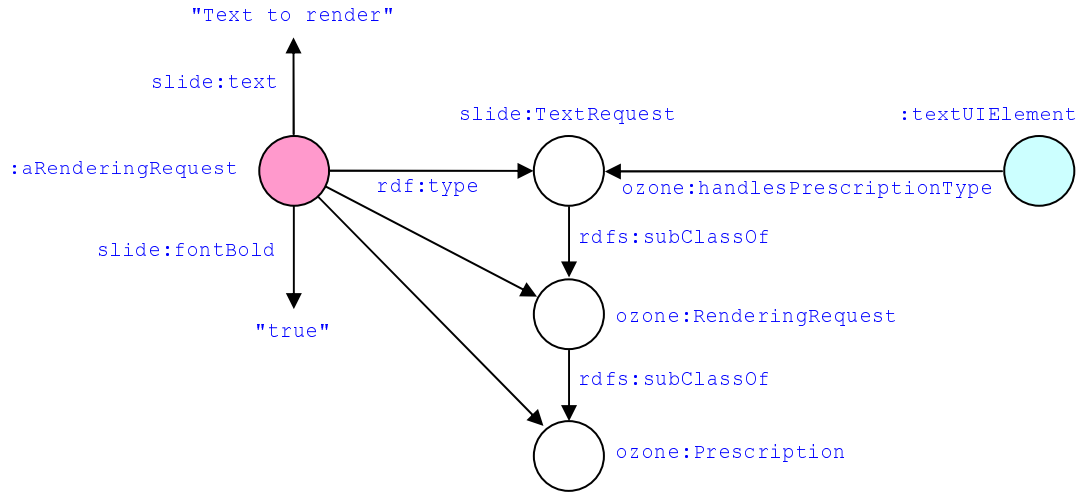

Figure 30. Association between a rendering request and a UI element

Given a prescription, such as `:aRenderingRequest` in Figure 30, the UI framework can locate the appropriate evaluator, i.e. `:textUIElement` in this case, by following two arrows: a forward predicate `rdf:type` and a backward predicate `ozone:handlesPrescriptionType`. We can express this computation in Adenine as a query to the RDF store as follows:

```
= tuple (queryExtract {
    :aRenderingRequest    rdf:type                          ?type
    ?evaluator                ozone:handlesPrescriptionType        ?type
} @( ?type ?evaluator ))
= evaluator tuple[1]
```

The `queryExtract` function in this code fragment resolves `?type` to `slide:TextRequest` and `?evaluator` to `:textUIElement`. In the case where there is more than one evaluator capable of understanding a prescription, the UI framework picks one of them non-deterministically—this results from the non-deterministic behavior of `queryExtract`. In the future, we can add versioning information to allow more accurate resolution from prescription to evaluator.

6.1.2 Evaluator Java Interface

So that all evaluators of different types can be instantiated and disposed in the same manner by the UI framework, they are required to implement a common Java interface—the `edu.mit.lcs.haystack.ozone.IEvaluator` interface:

```
public interface IEvaluator {
    public void initialize(
        IRDFContainer  source,
        Context        context
    );

    public void dispose();
}
```

After the Java class implementing an evaluator has been identified and instantiated to create an evaluator instantiation, that evaluator instantiation is then initialized with a call to its `IEvaluator.initialize()` method. This call provides the evaluator instantiation with an RDF container through which it can retrieve data from the system's information corpus, as well as a `edu.mit.lcs.haystack.ozone.Context` object that captures the context in which the evaluator instantiation has been instantiated and is to be used. The instantiation remains alive and is expected to apply and reapply its transformation until it is explicitly disposed with a call to its `dispose()` method.

6.1.3 Context

As discussed in section 5.8, contexts can be captured in environments, which are RDF models each containing information to which an evaluator has access. The evaluator tries to understand its context from that information.

There are two ways to implement environments properly:

1. create a new RDF store populate it with just the accessible information;

2. applying a filter on top of an RDF store or another environment to add accessible information and remove inaccessible information.

The first method is expensive: certain information will be duplicated in several environments. The cost of constructing and destructing RDF stores is not negligible. The second method avoids duplicating information but requires sophisticated logic for dispatching queries through multiple layers of filters; this could result in speed impediment.

Let us note that in the physical world, *what* a person sees depends on *where* the person stands. Likewise, we can control what an evaluator sees by telling it "where to stand" in the RDF store, i.e. which node(s) in the RDF store to start analyzing from. This method does not limit what an evaluator *can* see, just *how* it sees the data in the store.

The `Context` object in the method `IEvaluator.initialize()` models the context of an evaluator exactly by specifying points (i.e. RDF nodes) in the RDF store where the evaluator can start in its search for information describing its context. These points are values of *variables* in the `Context` object.

Variables in a `Context` object can have values that are URIs. These URI values effectively point into the RDF store as I have just discussed. Values can also be Java objects that encapsulate information for some reasons not serialized into RDF. For example, system font handles and color handles can be specified in the `Context` object of a UI element; these handles cannot be serialized into RDF. Other pieces of information, such as temporary results of calculations needed for wrapping text in paragraphs, can be serialized but are better kept in live Java objects for frequent access.

`Context` objects also mimic RDF store filters by chaining themselves together in a tree-like fashion. Variables of a node in the tree are inherited by its descendent nodes.

Currently, `Context` objects are mostly used for passing initialization parameters to evaluators. These parameters are actually the starting points at which an evaluator can start analyzing its context. More specifically, each `Context` object contains the variables named "evaluator" and "prescription" that identify respectively the evaluator and the prescription that the evaluator instantiation is to evaluate. `Context` objects are also used to store inheritable settings such as fonts, colors, text alignment settings, etc.

6.2 UI Elements

Every UI element implements the `edu.mit.lcs.haystack.ozone.IUIElement` interface, a sub-interface of `IEvaluator`:

```
public interface IUIElement extends IEvaluator {
    public boolean handleEvent(Resource eventType, Object event);
    public IGUIHandler getGUIHandler(Class cls);
}
```

6.2.1 Event Passing

The `IUIElement` interface provides a mechanism for passing events to UI elements through the method `handleEvent()`. This method takes a parameter called eventType that stores the type of event being passed. Event types are RDF classes named by URIs (e.g. `ozone:event.onMouseDown`). This naming scheme is beneficial in two ways:

1. Annotations can be made on event types and persisted in the RDF store (e.g. a UI element can be forbidden to handle events of a particular type due to some security reason); and

2. Events themselves can be serialized into RDF and asserted to be of some event types, which are already RDF classes with URIs; such serialization allows events to be logged; event logging policies can even be just annotations on the event types.

6.2.2 Layout Negotiation

The `IUIElement` interface also provides a mechanism for each UI element instantiation to negotiate its layout with its immediate parent UI element instantiation. Figure 31 shows the hierarchy of nested UI element instantiations inherent in the transformation tree in Figure 29. The nesting of rendering requests and view requests dictate how UI element instantiations are nested together, and how they negotiate with one another to lay out their outputs. The UI element 1 instantiation is the topmost and it is the immediate parent of UI element 2 instantiation and UI element 3 instantiation. UI element 1 instantiation will negotiate with both of the other two UI element instantiations to coordinate their outputs. UI element 3 instantiation in turn negotiates with UI element 4 instantiation. This process of layout negotiations is a depth-first traversal.

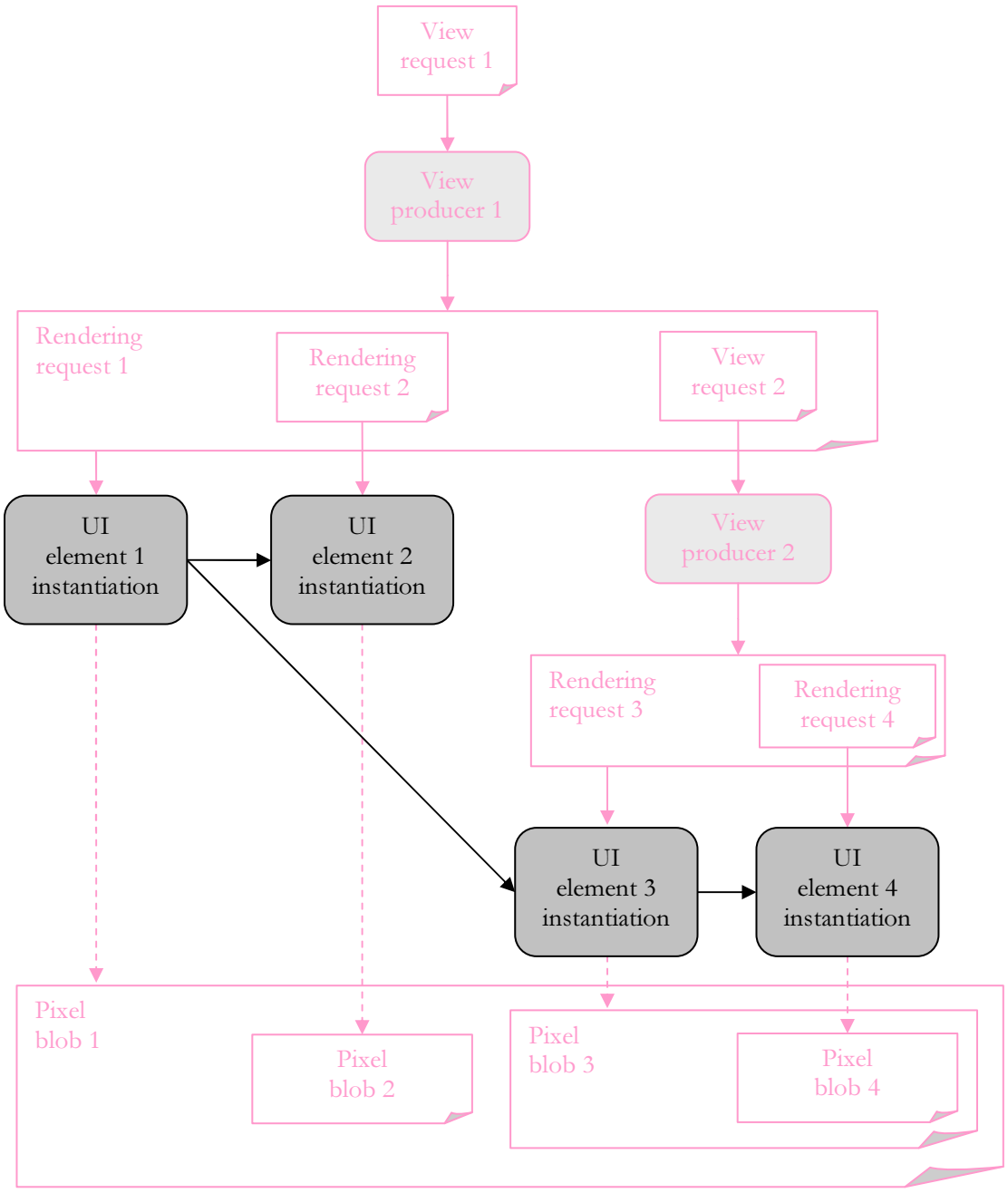

Figure 31. Hierarchy of nested UI element instantiations inherent in a transformation tree

A parent UI element instantiation (or "element" for short in this discussion) starts a layout negotiation with a child element by querying the child element for its rendering mode. There are two modes: block and inline. An element can render in block mode if it knows how to render within a rectangular area. An element can render in inline mode if it can wrap its information content like text across several lines.

The rendering capability of an element is encapsulated in a Java object of type `edu.mit.lcs.haystack.ozone.IGUIHandler` that the element can provide. There are two sub-interfaces of `IGUIHandler` that specialize the rendering capability of an element into the two rendering modes. They are `IBlockGUIHandler` and `IInlineGUIHandler`.

By specifying either interface in a call to a child element's `IUIElement.getGUIHandler()`, a parent element can retrieve the rendering capability in a particular mode of that child element. The result may be null if the child element cannot render in that mode. The parent element can also pass in a null to the method `getGUIHandler()` to retrieve the rendering capability in the default mode of the child element.

The interfaces `IGUIHandler`, `IBlockGUIHandler`, and `IInlineGUIHandler` are declared as follows:

```
public interface IGUIHandler {
    public void setVisible(boolean visible);
}

public interface IBlockGUIHandler extends IGUIHandler {
    public int getHintedDimensions();
    public int getTextAlign();

    public BlockScreenspace calculateSize(int hintedWidth, int hintedHeight);
    public BlockScreenspace getFixedSize();

    public void setBounds(Rectangle r);
    public void draw(GC gc);
}

public interface IInlineGUIHandler extends IGUIHandler {
    public void calculateTextFlow(ITextFlowCounter tfc);
    public void draw(GC gc, List textSpans);
}
```

The interface `IGUIHandler` consists of only one method, `setVisible()`, that allows an element to be notified whether it has been made visible or invisible.

The interface `IBlockGUIHandler` contains several methods that return certain characteristics of an element. The method `getHintedDimensions()` tells whether the element should be hinted with a width so that it can calculate its height, or a height so that it can calculate its width, or both a width and a height. For example, a paragraph element should be hinted with a width so that it knows how to wrap its text. (The semantics of the hints are not clearly defined in the current implementation.)

The method `getTextAlign()` returns an offset in pixels indicating how the element should be aligned with respect to the current text base line should the element be positioned within a paragraph of text. The method `calculateSize()` asks the element to calculate its size given either a hinted width, a hinted height, or both. If the element wants to be hinted neither width nor height, then the method

`getFixedSize()` is called to get the element's desired size. An image element, for example, always assumes a fixed size. The method `setBounds()` is used to notify the element of the rectangular screen area that its parent element has decided to give it after the layout negotiation. The method `draw()` tells the element to paint itself.

The interface `IInlineGUIHandler` contains only two methods. The method `calculateTextFlow()` takes in an object of type `ITextFlowCounter` [25] that encapsulates the current text wrapping settings of the parent element. The child element uses this `ITextFlowCounter` object to determine how to break up its information content into "text spans". After the parent element has negotiated with all of its child elements, it adjusts these text spans to take into account text base line alignments, and vertical and horizontal alignments. It then calls on each child element's method `IInlineGUIHandler.draw()` giving back the child element's text spans.

6.3 View Producers

Although section 5.2.2 lists a diversity of view request specifications, currently the UI framework can only handle specifications for view classes. Rather than modeling views' characteristics explicitly, we classify views into classes, and specify in view requests which classes to be used. This decision compromises the expressiveness of describing view requests for a simpler initial implementation. When we understand how views' characteristics can be modeled, we will incorporate them into the framework.

There is currently one generic view producer that can translate a view request demanding a super view class into a view request demanding the most specific view class suitable for presenting the object to focus on.

Figure 32 and Figure 33 illustrate an example of this translation. Given a view request named `:aViewRequest` of type `ozone:ViewRequest`, the UI framework looks for the appropriate view producer, which is `ozone:viewProducer`. This view producer inspects the view request, which demands a view of type `ozone:LineSummaryView` for the object named `:anOffice`. A view of that type is supposedly rendered as a line with summary information. The view producer then inspects the type hierarchy of `:anOffice` and finds that the most specific type that has a registered view class which is a subclass of `ozone:LineSummaryView` is `:Location`. The view class registered for `:Location`, `:LocationLineSummaryView`, overrides the perspective class `ozone:LineSummaryView` registered by default for all things.

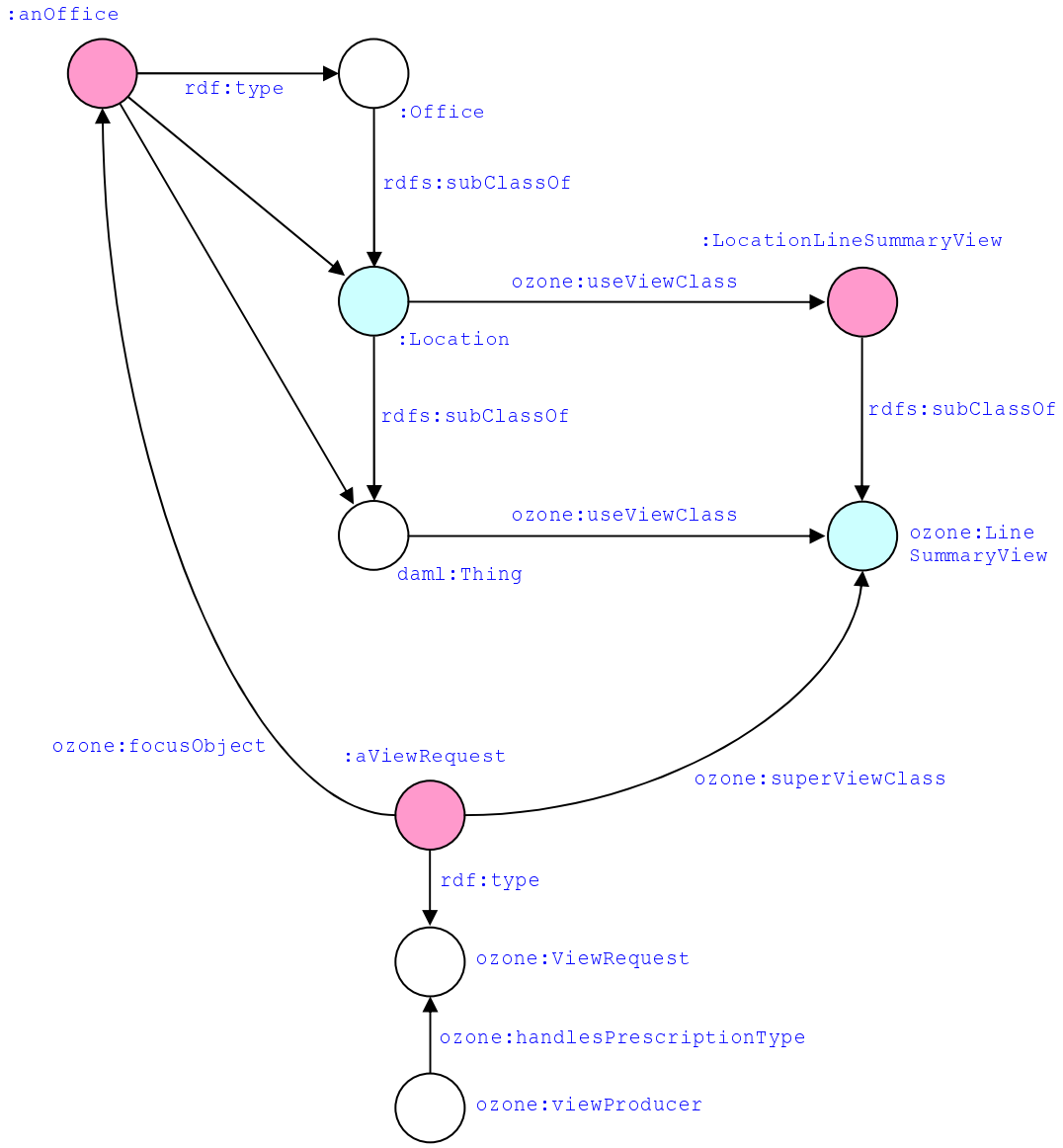

Figure 32. Resolving a super class view request to a view class

Note that the UI designer who specifies the view request does not need to know what type `:anOffice` is, nor what class of view is suitable for showing it in. She only needs to decide that the view should be a line summary and the rest is handled by the UI framework. For this reason the translation from a super view class to a specific view class is very important.

In order for a view of type `:LocationLineSummaryView` focusing on `:anOffice` to be presented, the `ozone:viewProducer` issues a specific class view request for it. Specifically, as shown in Figure 33, the view producer generates a view request of type `:LocationLineSummaryViewRequest` focusing on `:anOffice`. This is because views of type `:LocationLineSummaryView` are produced by the view

producer :LocationLineSummaryViewProducer, which understands view requests of type :LocationLineSummaryViewRequest.

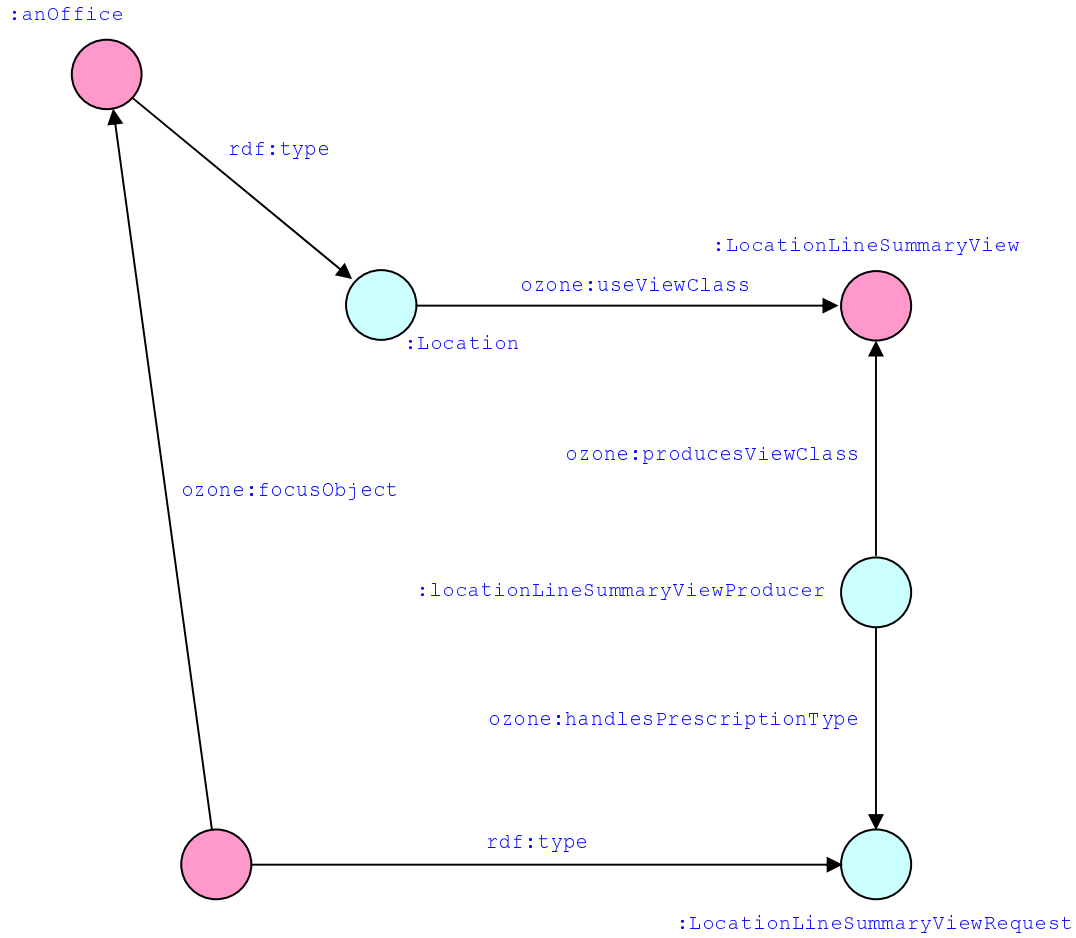

Figure 33. Generating a specific class view request from a view class

The newly generated view request can then be translated by the view producer :locationLineSummaryViewProducer into rendering requests or more view requests. Since :locationLineSummaryViewProducer deals mostly with RDF, it is convenient to code it in Adenine. Here is a sample code for this view producer:

```
add {
    :locationLineSummaryViewProducer
        rdf:type                            ozone:Evaluator ;
        rdf:type                            ozone:ViewProducer ;
        ozone:handlesPrescriptionType       :LocationLineSummaryViewRequest ;
        ozone:adenineMethod                 :produceLocationLineSummaryView
        hs:implementation ${
            rdf:type        hs:JavaClass ;
            hs:className    "edu.mit.lcs.haystack.ozone.AdenineEncodedViewProducer"
        }
```

```
}

method :produceLocationLineSummaryView viewRequest
    = location (extract viewRequest ozone:focusObject ?x)

    return ${
        rdf:type           ozone:Prescription ;
        rdf:type           ozone:RenderingRequest ;
        rdf:type           slide:TextRequest ;
        slide:textDataRequest ${
            rdf:type       ozone:Prescription ;
            rdf:type       data:DataRequest ;
            rdf:type       data:LiteralPropertyDataRequest ;
            data:subject   location ;
            data:predicate dc:title
        }
    }
```

When the UI framework locates `:locationLineSummaryViewProducer` and finds its implementation Java class `AdenineEncodedViewProducer`, the framework instantiates an object of type `AdenineEncodedViewProducer` and initializes it with a `Context` object in which the property `ozone:prescription` is bound to the view request's URI, and the property `ozone:evaluator` is bound to `:locationLineSummaryViewProducer`. The `AdenineEncodedViewProducer` object retrieves the attribute `ozone:adenineMethod` from `:locationLineSummaryViewProducer` and invokes the returned method, `:produceLocationLineSummaryView`, passing in the view request's URI. The Adenine method gets the focus object's URI, i.e. the location's URI, from the view request and formulates a rendering request and a data request. The data request specifies how to compute the title of the location and the rendering request specifies how to paint the title.

Note that there are two different view requests in this example, one specifying a super class of the view desired and one specifying a specific class of the view desired. Both are persisted in the RDF store because both might incorporate changes by the user. For instance, the user might demand that the office named by `:anOffice` be shown not as a line summary but as an icon; this change should be saved in the first view request. At the same time, the user could also customize the line summary view of `:anOffice` by inserting into the second view request specifications that the `:locationLineSummaryViewProducer` understands.

6.4 Data Producers

Data producers implement the `edu.mit.lcs.haystack.ozone.data.IDataProducer` Java interface, another interface derived from `IEvaluator`:

```
public interface IDataProducer extends IEvaluator {
    void registerConsumer(IDataConsumer dataConsumer);
    void unregisterConsumer(IDataConsumer dataConsumer);
```

```
    Object getData(
        Resource   dataType,
        Object     specifications
    ) throws DataNotAvailableException;

    Resource getStatus();

    void requestChange(
        Resource   changeType,
        Object     change
    ) throws UnsupportedOperationException, DataMismatchException;

    boolean supportsChange(Resource changeType);
}
```

This interface works together with the `edu.mit.lcs.haystack.ozone.data.IDataConsumer` interface to feed up-to-date data to the client of the data producer:

```
public interface IDataConsumer {
    void onDataChanged(
        Resource   changeType,
        Object     change
    ) throws IllegalArgumentException;

    void onStatusChanged(Resource status);
}
```

When a rendering request includes a data request, the UI element instantiation that handles the rendering request requests the UI framework to look up the data producer capable of understanding the data request, instantiates the data producer, and returns a Java object implementing the `IDataProducer` interface. The UI element instantiation then registers an `IDataConsumer` object with the returned `IDataProducer` object, so that the data producer instantiation (i.e. the `IDataProducer` object) can dynamically route data to the UI element instantiation.

When data is available or when data has been changed, the `IDataProducer` object calls the method `onDataChanged()` on the `IDataConsumer` object, specifying the type of change in the `changeType` argument (e.g. `data:stringChange`, `data:listAddition`, `data:setRemoval`) together with the actual change, the delta, whose type depends on the type of change. The `IDataProducer` object can also notify the `IDataConsumer` object of its status (e.g. 30% of data has been retrieved, connection to information server being established) but this capability has not been explored yet.

The client of a data producer instantiation can also actively use the data producer instantiation by calling its methods `getData()`, `getStatus()`, `requestChange()`, and `supportsChange()`. The method `getData()` retrieves a specific piece of data offered by the data producer instantiation. The argument `dataType` specifies the nature of the data to be retrieved (e.g. `data:listItem`, `data:string`, `data:setItems`) while the argument `specifications` gives more details on the piece of data desired (e.g. the index of a list item). The method `getStatus()` retrieves the status of the data producer

instantiation, which is named by a URI. Again, status information has not been used but one can foresee that a view request can be made for the returned resource in order to show the status of the data producer instantiation to the user.

The method `supportsChange()` asks the data producer instantiation whether it can support the type of change named by the argument `changeType` (e.g. `data:stringChange`, `data:listAddition`). The method `requestChange()` requests the data producer instantiation to make a change of the type given in the argument `changeType` and with the details given in the argument `change`. The request may or may not be met, and may be met synchronously or asynchronously. If the request is met, the client will receive a call to its `IDataConsumer` object's method `onDataChange()`.

Note that data requests can not only be embedded in rendering requests but also within one another. This is similar to nesting programmatic expressions within one another to perform sophisticated calculations. The following code fragment shows several data requests nested to specify how to retrieve all siblings of a given person:

```
= dataRequestForAllSiblings ${
    rdf:type       ozone:Prescription ;                                  #1
    rdf:type       data:DataRequest ;
    rdf:type       data:SetDifferenceDataRecipe ;
    data:minuendDataRecipe ${
        rdf:type       ozone:Prescription ;                              #2
        rdf:type       data:DataRequest ;
        rdf:type       data:SetMergeDataRecipe ;
        data:setsDataRecipe ${
            rdf:type       ozone:Prescription ;                          #3
            rdf:type       data:DataRequest ;
            rdf:type       data:MapDataRecipe ;
            data:setDataRecipe ${
                rdf:type       ozone:Prescription ;                      #4
                rdf:type       data:DataRequest ;
                rdf:type       data:PropertySetDataRecipe ;
                data:predicate :parentOf ;
                data:object    aPerson
            } ;
            data:mapDataRecipe ${
                rdf:type       ozone:Prescription ;                      #5
                rdf:type       data:DataRequest ;
                rdf:type       data:PropertySetDataRecipe ;
                data:predicate :parentOf
            } ;
            data:mapProperty   data:subject
        }
    } ;
    data:subtrahend @( aPerson )
}
```

To understand these data requests, we analyze them from inside out. To simplify my explanation, I will call the data producer instantiations created to handle these data requests `dpi1`, `dpi2`, …, `dpi5`. First, `dpi4` retrieves all parents of `aPerson`; this is equivalent to executing the following Adenine code:

```
= parentsTuples (query { ?parent :parentOf aPerson })
= parents (Set)
for tuple in parentsTuples
    parents.add tuple[0]
```

dpi4 returns this set of URIs of parents of aPerson to dpi3. For each URI in the set, dpi3 instantiates a data producer to handle the data request #5. dpi3 sets the environment of each of those data producer instantiations such that the data producer instantiation "sees" the data request it is given as having the data:subject property equal to the URI of a parent. The data producer instantiation then proceeds to retrieve all children of that parent. In effect, with the help of other data producers, dpi3 performs the following computation, as expressed in Adenine:

```
# We are mapping the computation in the body of the following for loop
# on each element of the parents set and put all the results into another
# set called mappedSet.

= mappedSet (Set)
for p in parents
    # This is done by each data producer instantiation that handles the data request #5.
    = childrenTuples (query { p :parentOf ?child })
    = children (Set)
    for tuple in childrenTuples
        children.add tuple[0]

    # Include the result in each case in the final result set
    mappedSet.add children
```

dpi3 returns a set of set to dpi2, which merges the sets together to form a union set:

```
= unionedSet (Set)
for s in mappedSet
    unionedSet.addAll s
```

dpi2 then returns this union set to dpi1, which removes from it aPerson:

```
= minuend unionedSet

= subtrahend (Set)
subtrahend.add aPerson

= siblings (Set)
siblings.addAll minuend
siblings.removeAll subtrahend
```

Effectively, the five data requests together encode the same computation as the previous four code fragments combined, except that the data producer instantiations handling those data requests can remain alive to maintain the computation up-to-date as long as their client requires, while the four code fragments execute once and terminate.

6.5 Supporting Direct Manipulations

Section 5.7 points out that the systematic manner in which the Haystack UI framework pieces together the UI through the view abstraction allows systematic and uniform support of direct manipulation mechanisms such as context menus and drag and drop. Indeed, since each evaluator instantiation in a transformation tree is initialized with a `Context` object, and all `Context` objects are chained together in a tree reflecting the embedding hierarchy of UI elements, we can systematically analyze the `Context` object tree to determine which pieces of information being focused on and hence, which pieces the user most likely wants to interact with.

The effects of direct manipulation actions are simply delayed computations that are invoked when appropriate. These computations are parameterized by the objects on which they act. They are exactly like Adenine methods, are modeled as Adenine methods and referred to as *operations*. In order to determine when an operation is appropriate, i.e. applicable, we add metadata describing its applicability. Currently, the applicability of an operation is determined by the types and values of its actual arguments. The types of acceptable arguments are annotated on the formal parameters of the operation. (The acceptable values can be enumerated by Adenine methods, which are not discussed herein.)

The following code segment shows an operation for clearing collections:

```
add { :collectionTarget
    rdf:type           op:Parameter ;
    rdfs:range         hs:Collection ;
}

method :clearCollection :collectionTarget = collections ;
rdf:type       op:Operation ;
dc:title       "Clear collection" ;
dc:description "Remove all collection members"

    for c in collections
        remove c hs:member ?x
```

The named parameter `:collectionTarget` in this example has the range of all things of type `hs:Collection`. In other words, the operation `:clearCollection` can act on any collection and only collections. Named actual arguments to operations are always passed as sets. As a result, in the method `:clearCollection`, the `remove` statement works on the elements of the `collections` actual argument.

Whenever a UI element used to render a view of a collection is right-clicked, the operation `:clearCollection` will be automatically listed in the resulting context menu.

The effects of drag and drop are modeled as operations that take two parameters each, one marked to be of type `dnd:DragParameter` and the other `dnd:DropParameter`. Here is an example:

```
add { :personTarget
    rdf:type           op:Parameter ;
    rdf:type           dnd:DragParameter ;
    rdfs:range         hs:Person
}

add { :meetingTarget
    rdf:type           op:Parameter ;
    rdf:type           dnd:DropParameter ;
    rdfs:range         meeting:Meeting
}

method :inviteToMeeting :meetingTarget = meetings :personTarget = persons ;
rdf:type       op:Operation ;
rdf:type       dnd:DNDOperation ;
dc:title       "Invite person to meeting" ;
dc:description "Invite person to meeting"

    for m in meetings
        for p in persons
            meeting:invite m p
```

The user can invoke this operation by dragging any view representing a person onto any view representing a meeting. The result is that the person is invited to that meeting.

In the future, we can investigate on how to describe the applicability of an operation based on more than just the types and natures of its actual arguments. For example, we can specify which security permissions the user must have to execute an operation.

CHAPTER 7

Usage Scenario and User Study

In this chapter, I describe a usage scenario that illustrates how the support for information management tasks by the infrastructure of an environment leads to a coherent experience for interacting with information. That scenario is then used in a user study which compares users' performance in achieving the same set of goals in two different environments: Haystack versus Microsoft Windows together with relevant applications. This is a preliminary user study purposed to provide some quantitative indication of the Haystack UI's performance as well as to elicit qualitative user feedbacks.

7.1 Points to Illustrate

It is valuable to see the extent to which the UI framework that has been constructed resolves some of the shortcomings seen in existing information management environments as listed in

Table 1 on page 30. In particular, I would like to illustrate the following points through a usage scenario:

1. The associative browsing paradigm (using hyperlinks) together with the capability to model pieces of information large and small as first-class objects yield a conceptually simpler but more versatile mechanism for locating information as compared to conventional UI routes made up of widgets such as menus and dialog boxes. While the associative browsing paradigm has been widely used for the Web, it has not been explored in depth as an alternative for browsing everyday information such as e-mail messages and digital picture albums. I wish to see how this paradigm fares against the window-dialog box paradigm.

2. Because many things are modeled as first-class objects, users can create shortcuts to quickly revisit them. This is another advantage of Haystack as compared to existing environments with regards to the support for locating information.

3. Serving as central points wherefrom operations applicable on objects can be browsed, context menus allow users to easily associate operations and the objects on which they act, and make the capabilities offered by the system transparent and readily available to users. In existing environments, wherever many objects are involved, it is not clear which operation is offered for which object, and how to exhaustively list all operations available for a particular object.

4. Drag-and-drop is a natural mechanism for invoking certain operations and its pervasive support adds a fluidity in the manipulations of information. In existing environments, drag-and-drop is not supported uniformly and where not supported, is often expected and missed.

5. Haystack's support for information organization at the system level has added flexibility and expressiveness in the ways users organize their data. In particular, things of different system types can be organized together in ways meaningful to users, and classification can be performed without relocating data.

I have purposely neglected to test the aspects of expressing and perceiving information because the Haystack platform is not complete to the extent that allows these aspects to be tested. For instance, with regard to perceiving information, there is currently no UI mechanism for the user to

dynamically change the way an object is being presented (e.g. switching the specified super view class).

7.2 Scenario

The usage scenario used to illustrate the above points is as follows:

A hypothetical user, let us named her Jane, is working on a group project with her friend Alisa. This project involves collecting information about famous Canadian singers and writing an essay about them. Alisa has managed to get some contact information for Celine Dion, a famous Canadian singer, who is incidentally one of Jane's favorites. Alisa has sent Jane an e-mail message with a contact information card attached.

1. From Alisa's e-mail message (Figure 34), Jane clicks on the attachment labeled "Celine Dion" to view the contact information of Celine Dion. The contact information card includes a portrait of Celine and a map to her home. (This step illustrates point #1 in the previous section.)

2. Since Jane intends to conduct a face-to-face interview with Celine tomorrow, she wants a quick way to recall the map to Celine's home whenever she needs to drive over to Celine's home. Jane drags the map from the contact information card over to the *Favorites* collection on the left pane (Figure 35). Note that the *Favorites* collection can contain items of any type—it is a heterogeneous collection. (Points #2, 4.)

3. Jane realizes that Celine's portrait in her contact information card is in a wrong orientation. She right-clicks on the portrait and chooses the operation *Rotate picture* (Figure 36). She selects the correct rotation direction when asked on the left pane and clicks on the button *Rotate* (Figure 37). The portrait is now properly oriented. (Point #3.)

4. Jane remembers that another friend of hers, Ben, is also a fan of Celine Dion. She wants to send him her photograph. She clicks on the operation *Compose e-mail message* in the *Favorites* collection and enters his name in the *To* line (Figure 38). She then clicks *Back* to return to the contact information card, and drags Celine's photo onto the new e-mail message which is displayed as an item in the *Scrapbook* collection on the left pane (Figure 39). (Newly created items in Haystack are automatically added to the *Scrapbook* collection, which is another

heterogeneous collection.) Jane then clicks *Forward* to return to the new e-mail message. (Points #1, 4.)

5. Jane just remembers that Ben is on dialup: he would prefer pictures to be sent in small sizes. She then right-clicks on the attached photo in the new e-mail message and choose the operation *Resize picture* (Figure 40). She enters "50%" as the new size of the photo and clicks *Resize*. (Point #3.)

6. Since Jane is confident in getting more information about Celine Dion, she decides to include Celine in the final essay. That means she needs to collect all information she has about Celine into her project, which is just a category called "Canadian Singers Project" that she has created. She clicks on the item called "Song Collection" in the *Favorites* collection on the left pane to browse to her collection of songs. She then selects the one Celine Dion's song that she has and checks the checkbox labeled "Canadian Singers Project" on the right pane in the *Organize* tool (Figure 41). Celine's song has been classified into the project. (Point #5.)

7. The e-mail message from Alisa is also relevant to the project. Jane browses back to it and classifies it into the same category. (Point #5.)

8. Jane also wants to put Celine's photo into the category. She clicks on the attached contact information card in Alisa's email to browse to Celine's contact information, which includes her portrait. Jane then drags the portrait onto the category "Canadian Singers Project". (Points #4, 5.)

9. Jane returns to Alisa's message and realizes that Alisa also asks for a plan for the weekend as well as gives some advice about the user's upcoming interview for a summer internship. Jane then checks the two checkboxes labeled "Fun" and "Internship" in the *Organize* tool on the right pane to classify e-mail message accordingly (Figure 42). (Point #5.)

10. Jane wants to know who has taken the photograph of Celine Dion, in case the photographer has taken more photographs of her. Jane clicks on the attached contact information card again, and then on the portrait. When Haystack has browsed to the portrait, Jane sees that the portrait has been taken by John Doe (Figure 43). (Point #1.)

11. Jane clicks on "John Doe" to browse to his information (Figure 44). There, she clicks on "John Doe's photographic collection" to see all of his photographic work (Figure 45). (Point #1.)

12. After browsing more photos of Celine, Jane decides to review the map to her home to see how far she will have to drive. She clicks on the item labeled "Map to Celine's home" that she has previously put into the *Favorites* collection on the left pane. (Point #2.)

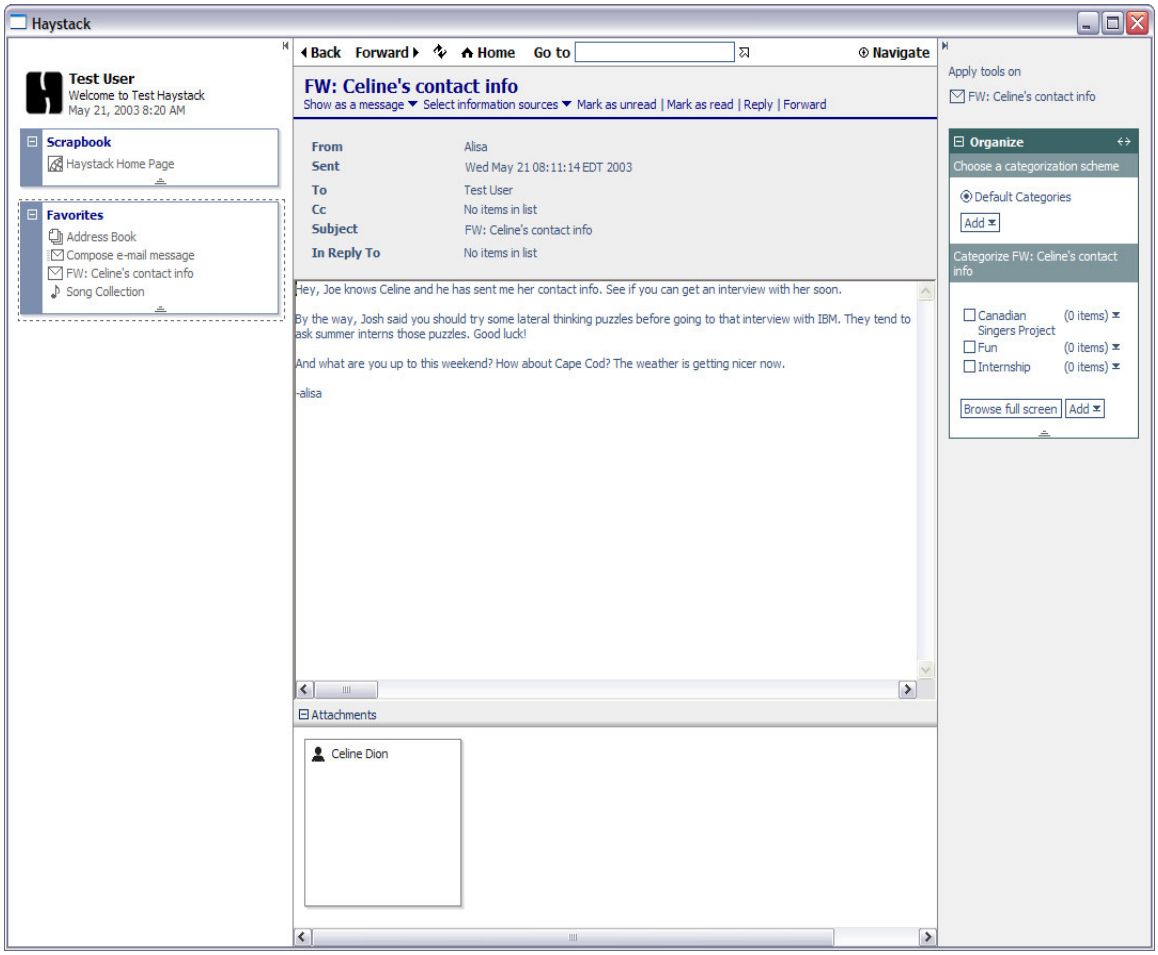

Figure 34. Scenario: reading Alisa's e-mail message

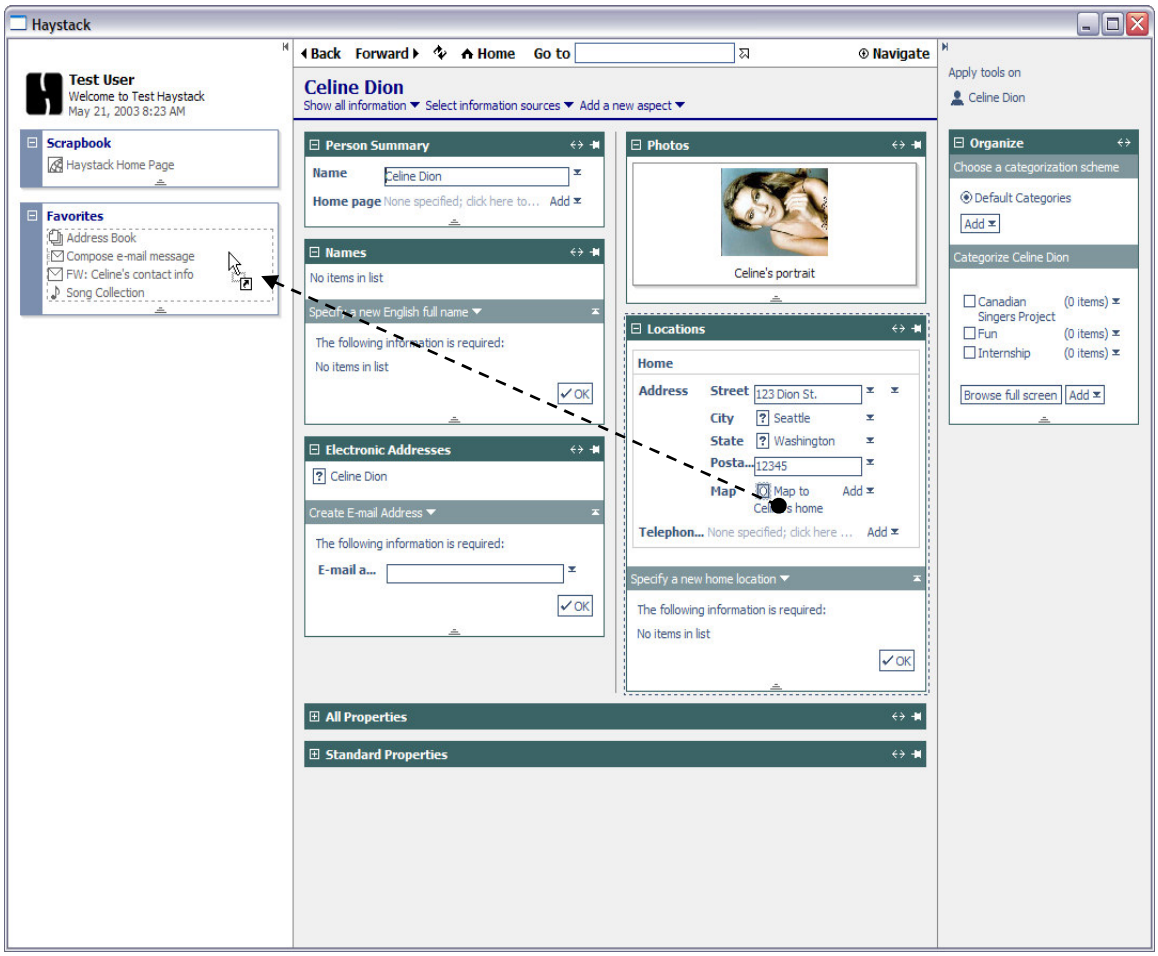

Figure 35. Scenario: Creating a shortcut to the map to Celine's home

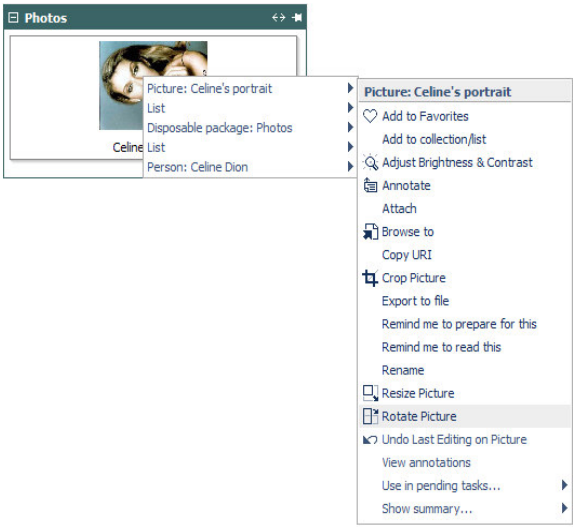

Figure 36. Scenario: invoke the Rotate command on a picture

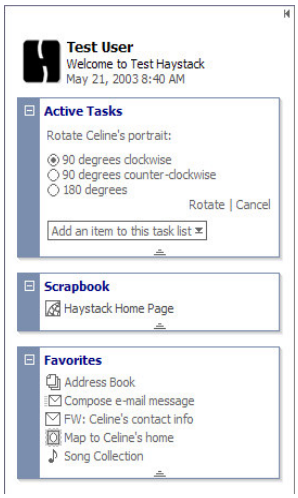

Figure 37. Scenario: choosing direction to rotate picture

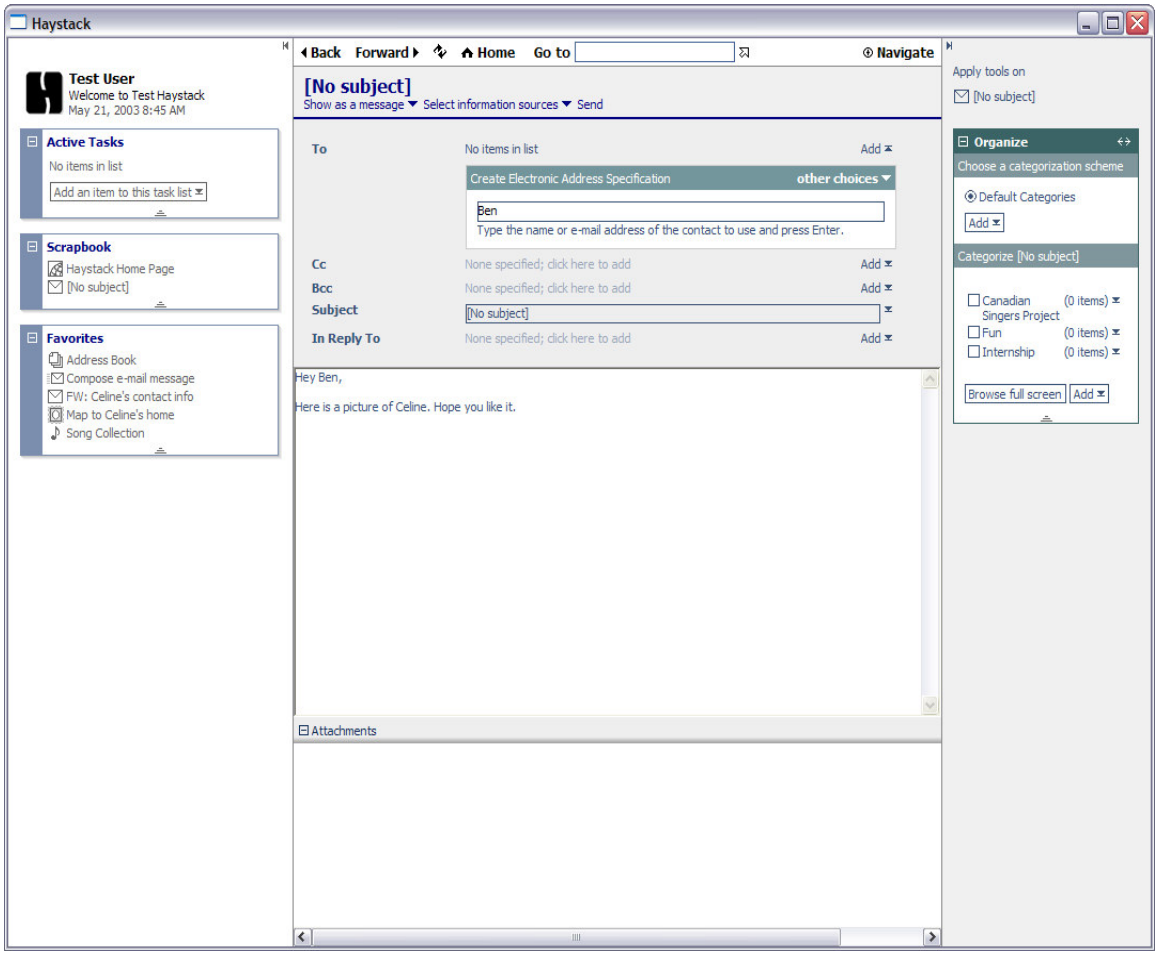

Figure 38. Scenario: composing an e-mail to Ben

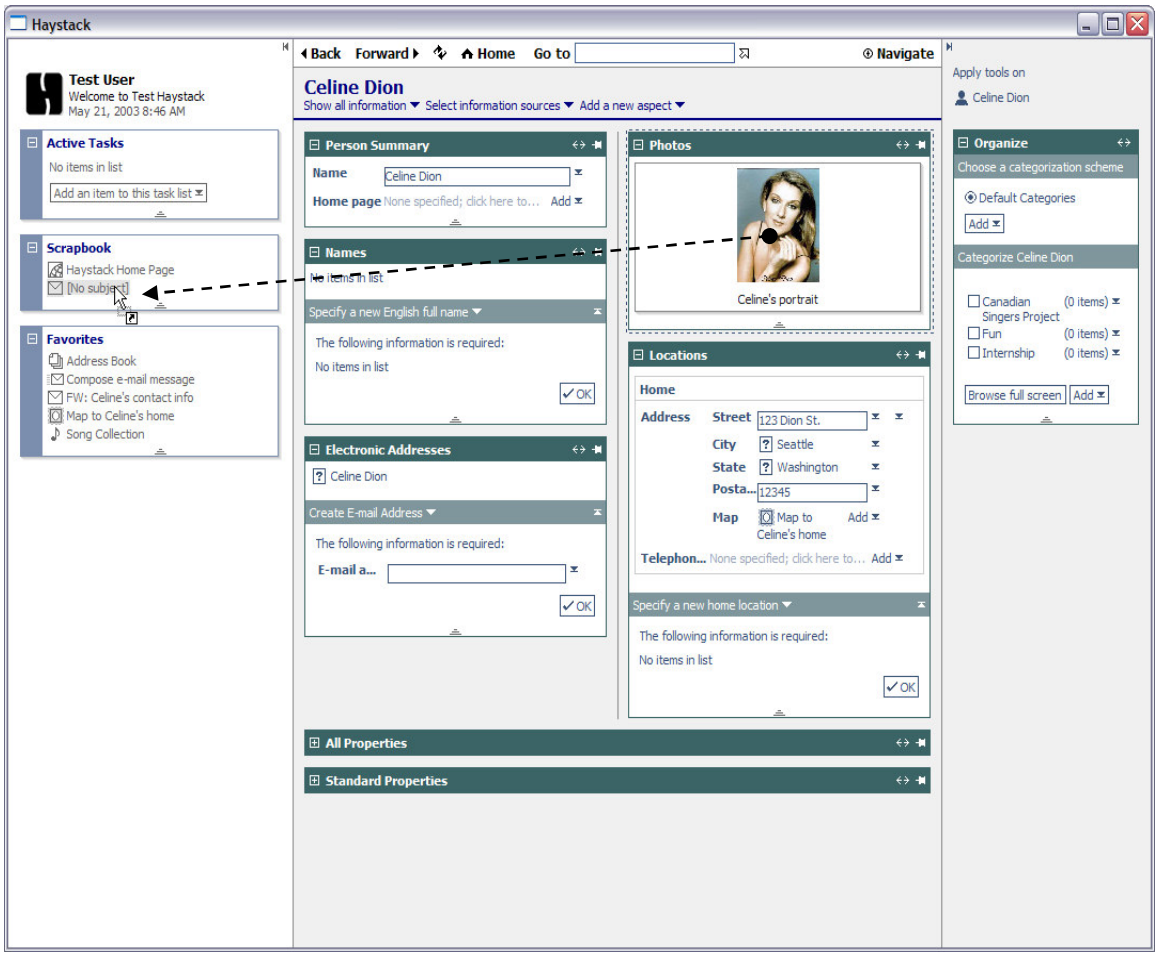

Figure 39. Scenario: attaching Celine's portrait to e-mail message to Ben

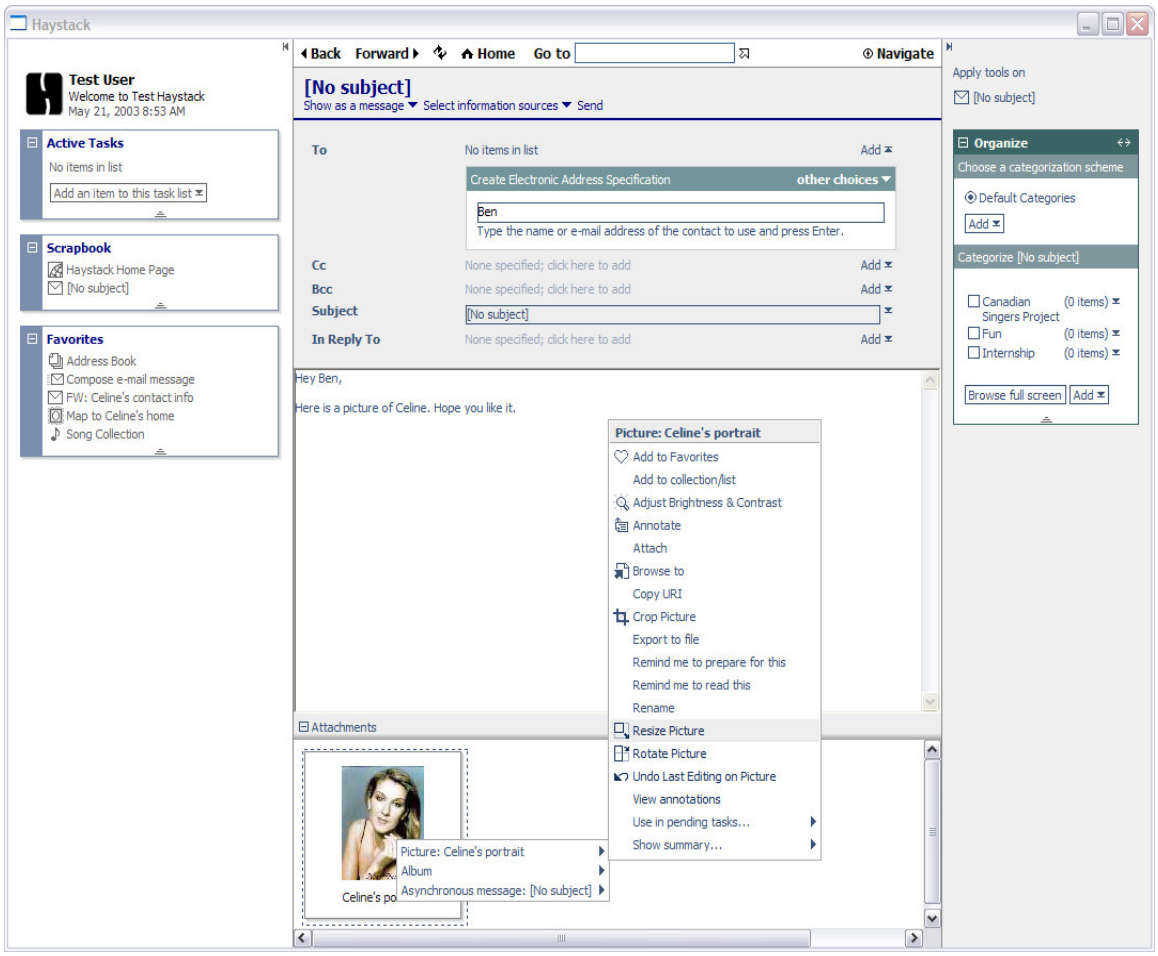

Figure 40. Scenario: resizing a picture

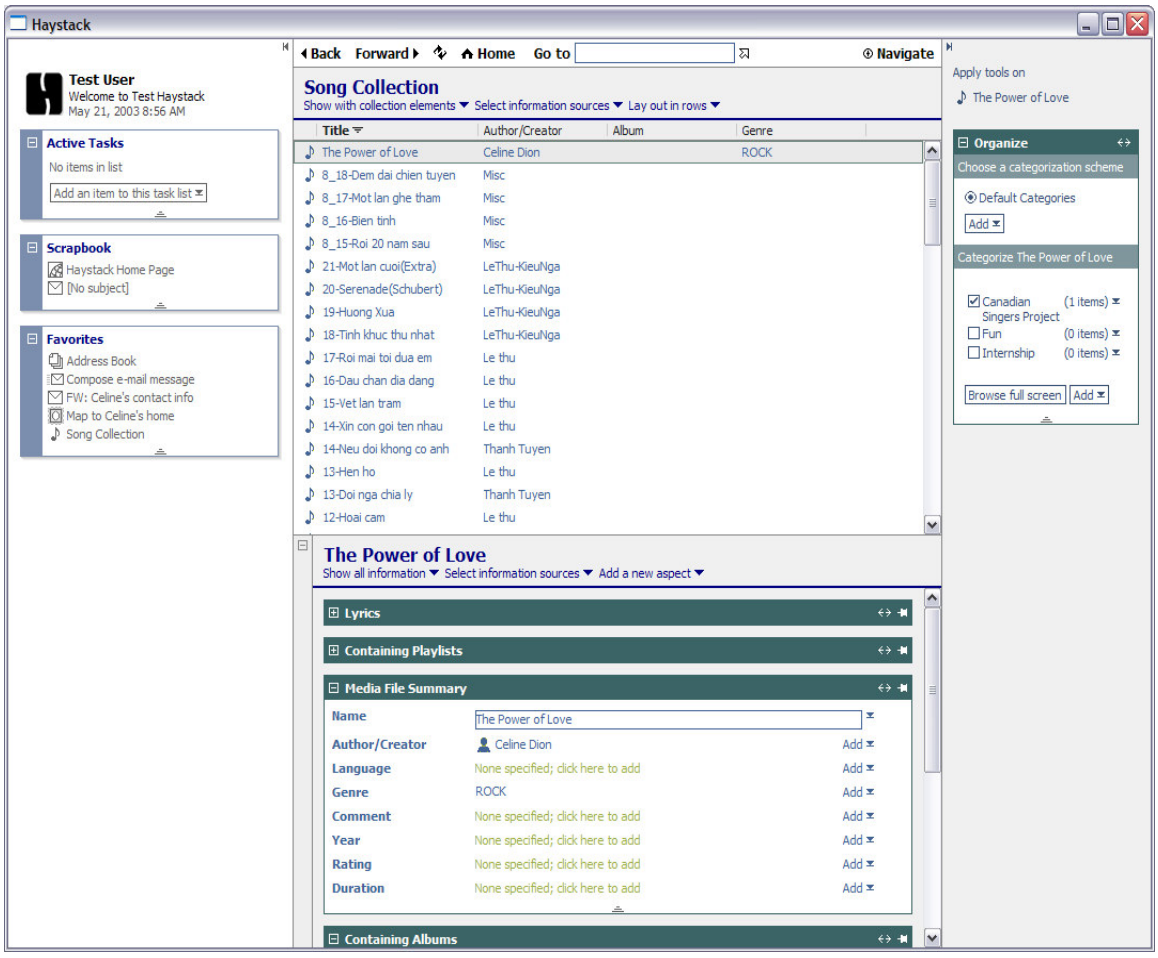

Figure 41. Scenario: classifying a song into a project

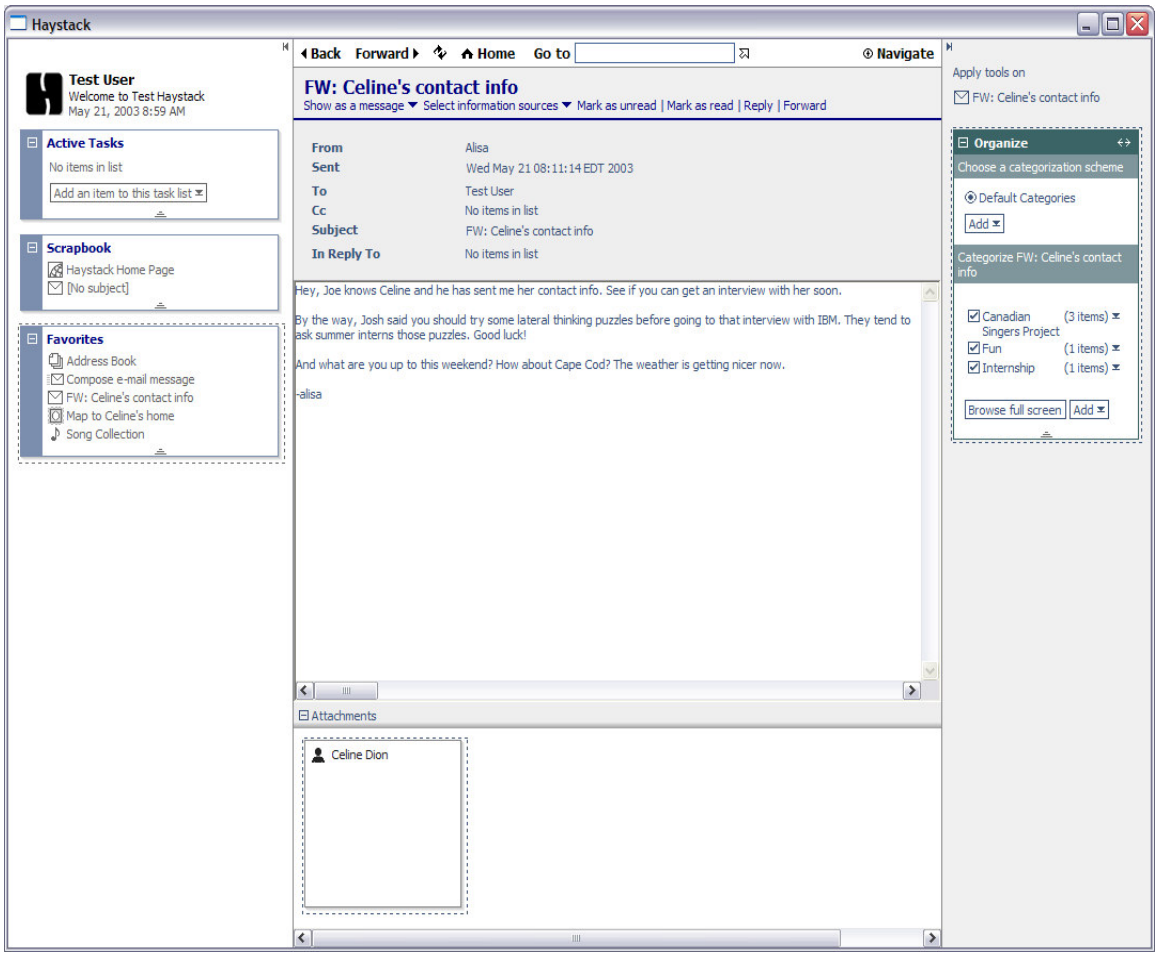

Figure 42. Scenario: classifying Alisa's e-mail message into more than one category

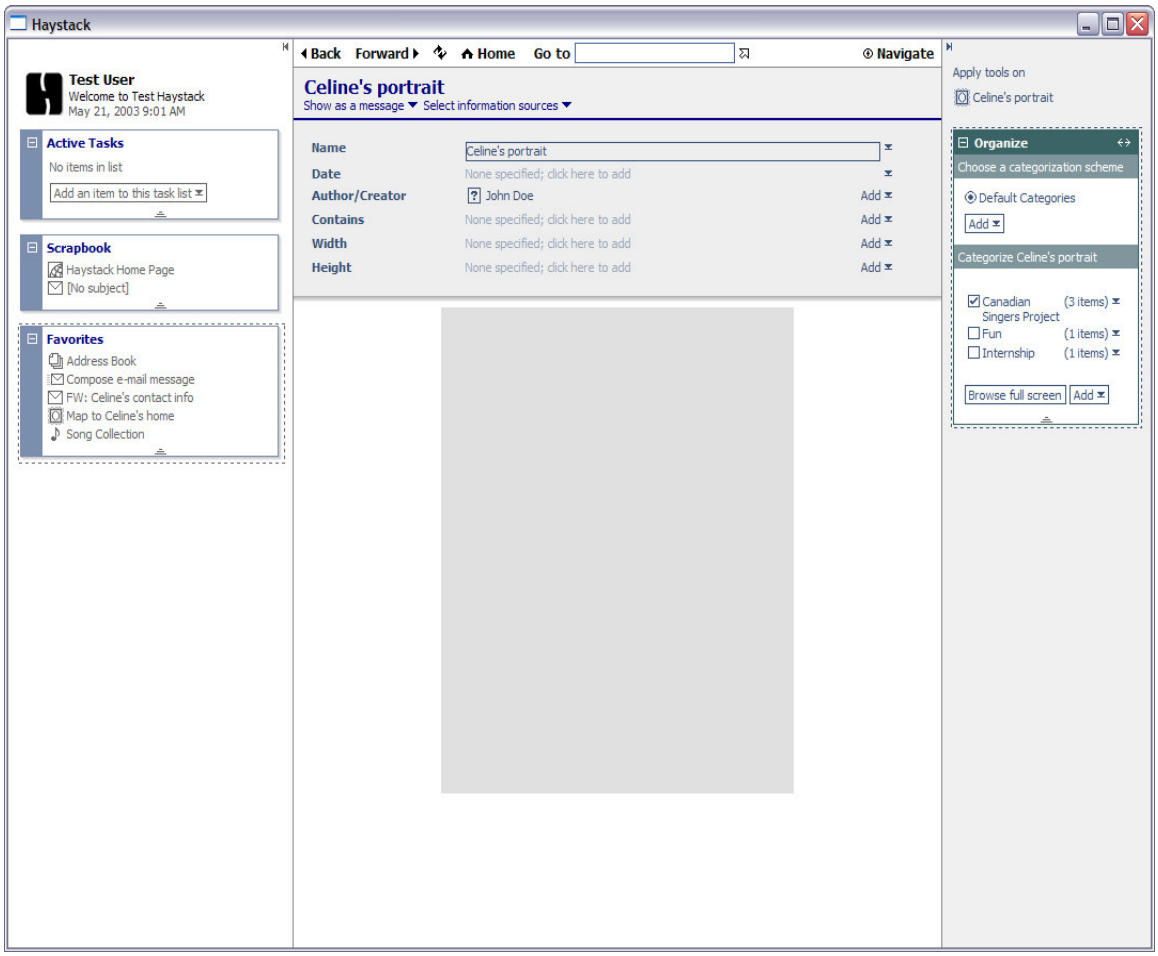

Figure 43. Scenario: Seeing more information on Celine's portrait (actual full scale photograph omitted to observe copyright)

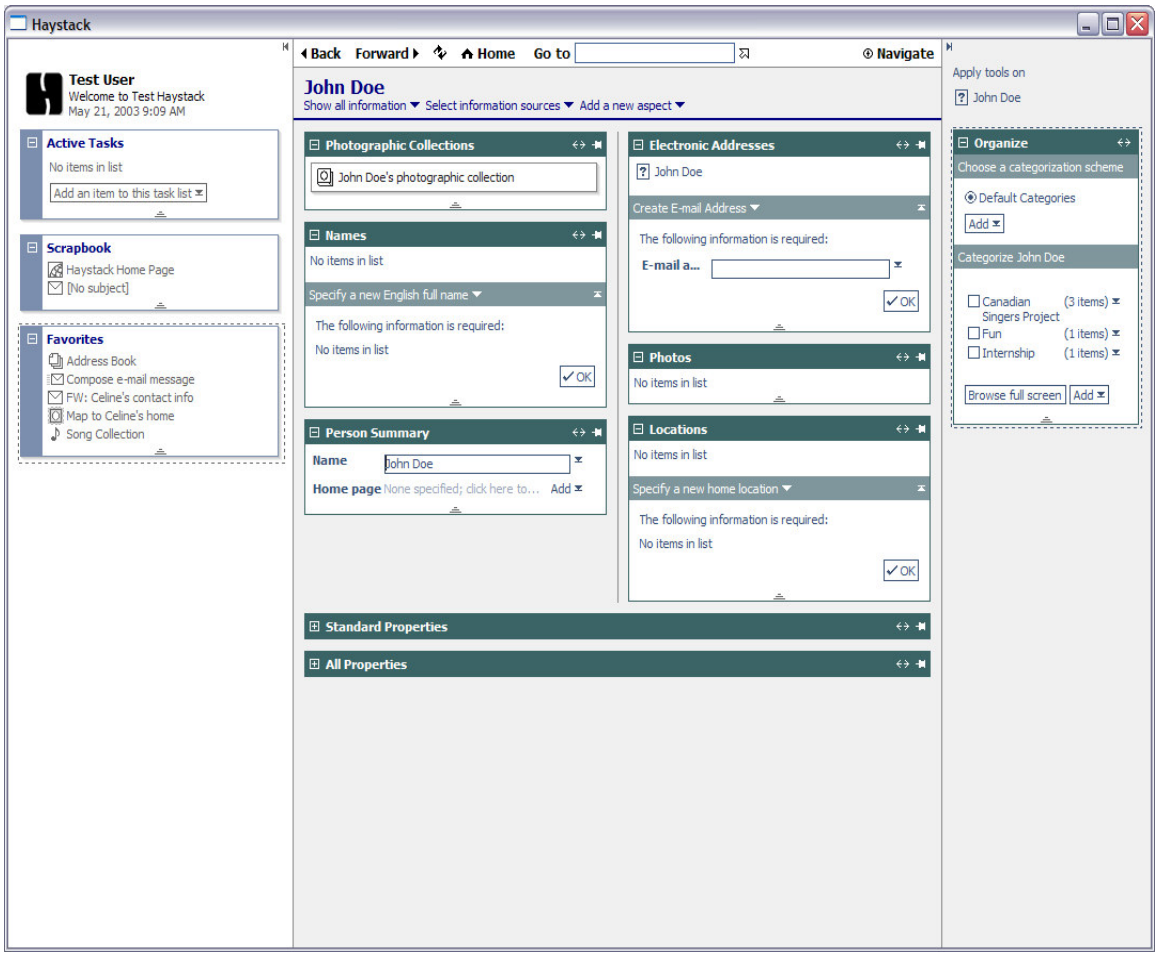

Figure 44. Scenario: viewing information of the photographer of Celine's portrait

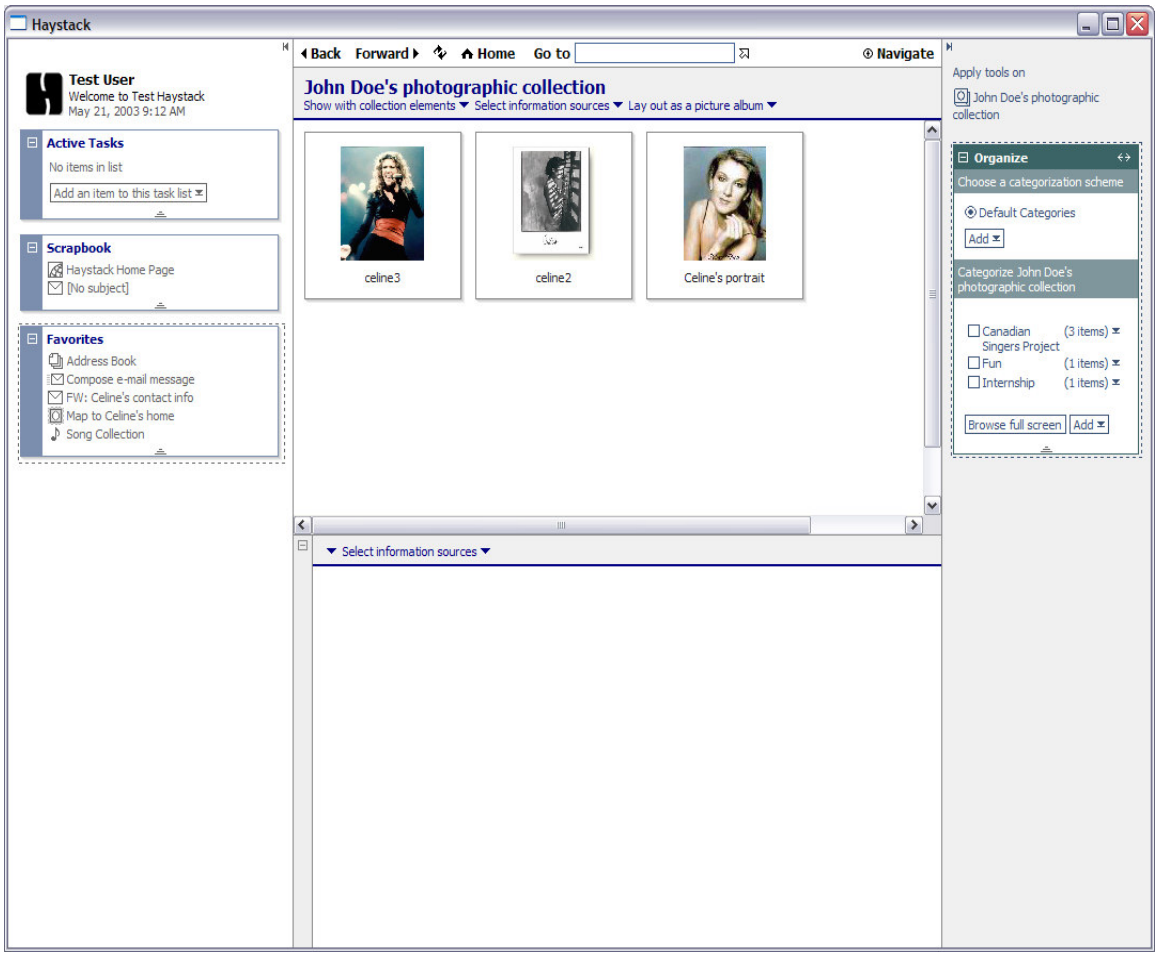

Figure 45. Scenario: browsing more photos of Celine

7.3 User Study Design

This user study sought a preliminary comparison between Haystack and an existing environment with regards to their support for performing basic information management tasks. My hypothesis was that Haystack, with infrastructural, built-in support for information management, outperforms the other environment whose means for satisfying basic information management needs come from the amalgamation of support implemented by individual components. Specifically, given the same set of tasks, subjects would be able to finish them faster and more completely in Haystack than in the other environment.

7.3.1 Design Rationales

This section discusses several challenges inherent in the nature of this study and the decisions made to overcome or sidestep them.

Choosing Tasks

The tasks in this study cannot be specified at a low level, e.g. click here, click there. Rather, they must be high-level goals so to demonstrate how each environment demands users to map high-level goals to low-level actions. It is mostly the differences in such mappings that make one environment cognitively easier to use than another. Note that even if for a common goal two environments may offer two shortest sequences of low-level actions with the same length, one environment may still be easier to use than the other as the shortest sequence of actions offered by each environment may not be the sequence of actions most users take—the high-level goal may not map most logically to that sequence.

We need to choose high-level goals that are logical and familiar to test subjects so that they are easy to understand and remember. The goals must be something that at least some test subjects would want to perform outside this study, so that it can be inferred that Haystack is useful in practical situations.

I chose to adopt the tasks in the scenario in section 7.2 for this study. I believe the high-level goals in that scenario are practical and they form a story that is logical and hence, easy to remember.

One can suppose that the scenario chosen has been optimized to Haystack's advantage. It may be true that Haystack has an advantage over the other environment with respect to this particular scenario, but that was exactly what I wanted to prove because I believe that this scenario consists of practical information management tasks that users would want to perform in other circumstances. To be sure, I decided to ask test subjects how often they needed to perform those tasks outside the user study.

Comparing Performance

Test subjects cannot be compared against one another as they differ in physical reaction times, cognitive abilities, and experiences with computers and software. Rather, each test subject should be compared with him- or herself by his or her performances on the two environments. This requirement dictates that each user must perform the same sequence of tasks on the two environments.

Since each user has to perform the same sequence of tasks twice, there is a learning effect in favor of the second time. To lessen this learning effect, I chose to ask each user to read through the sequence of tasks before starting the two phases, so that his or her knowledge of the goals to be achieved would be almost the same in both phases. In addition, I decided to let half the number of users work

in Haystack first, and the other half work in the other environment first. Each environment would be biased against half of the times.

Choosing The Competing Environment

Given an environment and a user base, there are three possibilities: the environment is known to no one, to some users, or to all users. If the environment is known to only some users, then the users' performances in that environment are not comparable. If the environment is known to no one, then they must be introduced to that environment during the study. Since I have a deeper knowledge of our environment, Haystack, than of any other environment, it would be considered a disadvantage to the competing environment if I were to introduce the users to both environments. In order to be fair to the competing environment, its designers should be the ones who introduce the users to it. This clearly is not a practical choice. Consequently, I must choose an environment that is known to all users who participate in this user study.

It is not possible to understand how each user has been introduced to this competing environment, or whether that introduction was proper. We can only ensure that each user does have some operational knowledge of that environment, which has been acquired one way or another over some period of time. Let us assume that that operational knowledge puts the competing environment on the same playing field as Haystack, which the user will be introduced to "properly" by me in five minutes. In fact, with only a five-minute introduction and no operational knowledge, Haystack seems almost at a clear disadvantage. If Haystack outperforms the competing environment, then either it is easy to use or easy to teach—both points are merits.

Since the environment competing with Haystack must be known to all test users, it should be widely known and used so that many people can participate. That environment must also be relatively new so that it has new technologies and hence, a better chance of outperforming Haystack. It must not be too new lest few users are sufficiently familiar with it. It must also have all the necessary tools capable of supporting the tasks in the scenario.

I selected Microsoft Windows XP as the operating system of the competing environment given these criteria. The Windows family of operating systems are widely known and used, and Windows XP is the latest released version. Many applications available run on Windows.

Microsoft Outlook 2002 was selected for accessing and managing e-mail messages and contact information. Microsoft Outlook 2002 is a widely known and used e-mail client, and it allows contact

information to contain pictures. This capability is useful for including with Celine Dion's contact information her portrait and the map to her home.

Microsoft Windows Media Player 8.0 was chosen for managing the songs involved in the scenario. Windows Media Player comes free with Windows XP. Contemporary competing media players such as Nullsoft WinAmp 3.0 do not offer any advantage over Windows Media Player.

Photos in the scenario would be edited with Jasc Paint Shop Pro 7.0. This software has an easy to understand interface for rotating and resizing images. It allows creator metadata information to be added to pictures, a feature crucial for annotating Celine Dion's portrait with its photographer's information. Although Paint Shop Pro is not well known, I have chosen it for its simplicity and for its metadata feature. Other photo editors such as Adobe Photoshop would be an overkill for our purpose.

Note that it is possible to offer users more than one choice of application for each particular functionality (e.g. making both Windows Media Player and WinAmp available) in order to increase the chance of the users being familiar with some applications for each task. I have chosen not to do so because certain functionalities (e.g. associating pictures with contact information) are only supported in one of the competing applications available. Furthermore, it is not possible to offer all applications that all users are familiar with. Offering several applications for each particular task may even be confusing to users: if a user is unfamiliar with all of the applications offered for a task, he or she will need to try to use many applications instead of just one.

Choosing Users

Even though I chose Windows XP as the operating systems, I did not require test users to be familiar with Windows XP in particular. I only required that they had proficient knowledge of some version of Microsoft Windows. This was a compromise of familiarity for a larger possible subject base, justifiable because Windows XP is not a great leap forward since Windows 95 if we compare their basic user interface mechanisms. There have been a lot of edge smoothing, additions of convenient features, and stability improvements, but the skills for using Windows 95 are still transferable to Windows XP.

Potential subjects were not asked for their familiarity with Microsoft Outlook 2002, Windows Media Player 8.0, or Jasc Paint Shop Pro 7.0. This was another compromise for a larger possible subject base, justifiable because, while experiences across different operating systems such as Mac OS X and Windows may differ widely, experiences among applications designed for similar tasks on the same

operating system should not. That is, if a user has used Netscape 7.0 Mail & Newsgroups, he or she should be able to use the basic functionalities of Microsoft Outlook 2002 without any trouble. After all, a key design criterion for any modern operating system is the transferability of skills across different applications.

Ads were posted in the Infinite Corridor of the Massachusetts Institute of Technology to recruit test subjects. These ads could be seen by people at MIT or any passer-bys. Subjects with any experience with any version of Microsoft Window who responded to the ads were accepted indiscriminately.

The MIT community is certainly more computing literate than average. In another study, such as one attempt to determine how easy *to learn* Haystack is compared to another environment, we might have chosen to advertise elsewhere. For this particular study, advertising to the MIT community was the best available option that also ensures sufficient user experience.

It must be noted that each test subject brings with him or her certain prejudices from past experience with software. For example, a user might expect Haystack to work in a certain way given his or her exposure to some piece of software that works in that way; the user's cognitive interaction with Haystack is consequently influenced by this expectation. It is practically impossible to compare a user's performances in two different environments without such contamination.

Configuring The Environments

The usability of a piece of software is like an evening dress: just as the smallest stain in an infelicitous spot spoils the whole dress, the smallest quirk in a frequently used feature can ruin the whole experience of using the software.

Haystack is an unreleased research platform with many components developed by many contributors. Not all components are mature. Not all known bugs have been fixed. Haystack is by no means as quality assured as commercial software applications. We had to therefore disable features not relevant to this study.

On the other hand, we could not disable certain irrelevant features in the competing environment since we did not have access to its code base. Furthermore, we might not be in the position to judge which features were irrelevant in the competing environment as we are not its designers. As a consequence, I chose to not disable any feature. However, to help limit the choices, I decided to tell the subjects about features that I believed were relevant. Of course, it is possible that I might

inadvertently have suggested the wrong relevant features and hence, lead the subjects down the wrong paths.

Surveying Users' Opinions

The Haystack UI has been built on certain principles. This user study can help us find out whether these principles are apparent to users by asking test subjects to articulate their experience with Haystack as compared to the competing environment. If the principles are simple, logical, and consistent, test subjects should be able to identify them.

The exit survey in this study was also used to gather some information about the subjects' levels of experience with the competing environment. A subject's level of experience is roughly measured by the number of years he or she has been using some version of Windows.

Considering all challenges listed above, this study is a best-effort attempt to find indications that Haystack has advantages over other environments with respect to information management support. This study does not aim to *prove* such advantages, but simply to hint at them. This study is also a means for finding out users' reactions to Haystack that we have not foreseen.

7.3.2 Procedure

This user study consisted of one session for each test subject. At the beginning of a session, the subject was asked to read thoroughly the requests listed in Appendix A. The subject could ask questions at this stage, but no question involving specific details relevant for perform any user study task was answered.

The subject was then given an introduction to both Haystack and Windows and relevant applications. If the subject would be using Haystack first, then Haystack would be introduced second, and vice versa. This was to avoid switching back and forth between the two environments.

Introduction to Haystack

The user was shown the UI of Haystack and told that Haystack acted like a web browser in that it could browse to web pages. The user was then shown that clicking on a link (e.g. the link to Alisa's e-mail message in the Favorites collection on the left pane) would cause Haystack to browse to where that link points.

The user was then told that Haystack differed from web browsers in two aspects. First, anything on the screen could be right-clicked to retrieve a list of commands applicable on what that thing represented; the text "Test User" on the upper-left corner of the screen was right-clicked to demonstrate. Second, drag-and-drop was supported; the text "Google" was dragged and dropped into the Favorites collection on the left pane.

The user was told that the Favorites collection contained all links he or she would need for the study.

The user was told that the Scrapbook collection contained newly created items. The text "Compose e-mail message" in the Favorites collection was click to create a new e-mail message as a demonstration of this point. The created message was pointed to in the Scrapbook and then removed to avoid interference with the study later.

The user was then pointed to the Organize tool on the right pane and told that it contained several categories into which the item being viewed can be classified. The user was shown how to classify the newly created e-mail message into the "Fun" category by checking the corresponding checkbox, and how to remove the message from that category by un-checking the checkbox. The user was also shown how the title of the message displayed on the title bar could be dragged into the "Internship" category.

Introduction to Windows et. al.

The user was shown Microsoft Outlook 2002 opened with a single e-mail message, the one from Alisa, in the Inbox. The user was then shown the Contacts folder and Ben's contact information card in it. The user was then told how to create a new e-mail message, either by right-clicking on a contact information card and choosing "New message to contact" or by clicking on the New toolbar command and typing in the contact's name or e-mail address. The user was also shown how to attach a picture to a message by drag-and-drop, and how to file a message into a folder also by drag-and-drop. The user was told that he or she could use menus and toolbars as he or she felt appropriate.

Next, the user was shown Paint Shop Pro 7.0 and told how to rotate a picture.

Then, the user was shown where to find the one and only Celine Dion's song in Windows Media Player.

The user was finally told that a list of shortcuts had been made for his or her convenience and collected at the bottom right corner of the screen, but he or she was welcomed to use any other program on the computer as he or she needed.

Subject in Driver's Seat

The subject was then told to start performing the tasks given in either Haystack or Windows first, and then in the other environment second. A screen-capture software was run to record all mouse and keyboard actions.

The subject was allowed to ask questions during this stage. Questions were answered by the study coordinator if they were deemed to not influence the points being tested. For example, if the subject was not sure where to find the Attach command for an e-mail message because he or he used Eudora instead of Outlook, then his or her inquiry for that command would be answered. Note that in cases where it was not clear whether a question would influence the points being tested, the study coordinator used his own discretion to decide whether to answer the question. There were simply too many paths that could be taken in a complex environment as Microsoft Windows XP; not all could be predicted *a priori* such that responses to the subjects' answers could be answered without unintentional bias for or against Windows.

Concluding Survey

When the subject had finished performing the given tasks on both environments, he or she was asked to complete the survey shown in Appendix B. This was the last stage of the user study session.

7.4 User Study Results and Discussion

Twenty three (23) subjects, 12 male and 11 female, who saw the user study's ads posters came to participate in the study. One subject quit the study when told to perform the tasks first on Haystack. Two subjects did not demonstrate the prerequisite skills on Windows and relevant applications. One subject's data was accidentally lost. The results reported herein came from the performance of nineteen (19) subjects, 9 male and 10 female.

7.4.1 General Observations

Several unexpected events occurred during the user study:

- Some subjects did not comprehend all the tasks even if they were asked to read the requests thoroughly and even after they said they had no question regarding the tasks. As a result, some bias would be expected against the first environments in which they performed.

- Subjects interpreted the goals of some tasks differently than intended (e.g. some subjects made shortcuts to the whole contact information card of Celine Dion instead of just to the map to her home).

- Some subjects carried out tasks out of order (e.g. the subjects resized the picture before attaching it to the message to Ben). They took shortcuts that would speed up their performances.

- Some subjects skipped some tasks as if they forgot to do those tasks (e.g. they classified Alisa's message into only one category).

- Some subjects were reluctant to browse the file system because it seemed private to the study coordinator.

The timings might have been inaccurate due to a number of interruptions:

- Subjects felt a need to compose proper e-mail messages and spent time formulating the contents of the messages to Ben even if not explicitly asked for.

- Virus alert dialog boxes popped up in Microsoft Outlook when attachments were opened, demanding subjects to read and respond to them.

- One subject was having lunch during the study.

- One subject answered his cellular phone during the study.

Overall, the subjects seemed comfortable with using drag and drop and context menus in Haystack. Some subjects had great difficulties in composing new e-mail messages addressed to Ben in Haystack.

7.4.2 Numerical Data

Timings

Table 4 shows the times it took the subjects to perform the user study tasks, completely or incompletely, on Haystack and on Windows. Each subject can only be compared against him- or herself and not against another subject since subjects differ widely in their familiarity with computers and software. The performance of each subject is calculated as the ratio of the time it took him or her to perform the tasks on Haystack to the time it took on Windows, regardless of whether he or

she skipped any task in either environment. This performance metric is labeled "Duration ratio" in the table. The average of these ratios, 0.68, indicates the subjects were able to perform the given tasks on Haystack in about 70% of the time they took to perform the same tasks in Windows. Again, task skipping is not taken into account in this figure. (The standard deviation of these duration ratios is 0.21.)

The average time it took to complete the tasks on Haystack is 12.73 minutes ($\sigma = 6.68$), on Windows is 18.90 minutes ($\sigma = 8.24$). The ratio of these averages is 0.67, which is almost equal to the average of ratios.

Table 5 shows the subjects' genders and which environments they used first. Gender did not seem to influence performance. However, there is a clear difference in performance between those who used Haystack first and those who used Windows first. The average of duration ratios for those who used Haystack first is 0.79, and for those who used Windows first is 0.57. (The ratios of averages are 0.79 and 0.54 respectively.) This is a noticeable learning effect: there is a clear bias against the first environment. Regardless, Haystack still outperforms Windows.

Table 4. Timings of subjects' performances on two environments

	Haystack				Windows					
User ID	Start	End	Duration	**Duration (minutes)**	Start	End	Duration	**Duration (minutes)**	**Duration ratio**	Session total time in minutes
1	0:00:10	0:11:43	0:11:33	**11.55**	0:00:06	0:14:17	0:14:11	**14.18**	**0.81**	25.73
2	0:00:37	0:21:06	0:20:29	**20.48**	0:00:07	0:23:53	0:23:46	**23.77**	**0.86**	44.25
3	0:00:12	0:09:31	0:09:19	**9.32**	0:00:17	0:14:13	0:13:56	**13.93**	**0.67**	23.25
4	0:01:40	0:21:16	0:19:36	**19.60**	0:00:13	0:21:40	0:21:27	**21.45**	**0.91**	41.05
5	0:00:07	0:10:08	0:10:01	**10.02**	0:00:30	0:32:39	0:32:09	**32.15**	**0.31**	42.17
6	0:00:16	0:20:56	0:20:40	**20.67**	0:00:11	0:16:52	0:16:41	**16.68**	**1.24**	37.35
7	0:00:06	0:08:01	0:07:55	**7.92**	0:00:03	0:15:00	0:14:57	**14.95**	**0.53**	22.87
8	0:00:22	0:33:06	0:32:44	**32.73**	0:00:12	0:45:13	0:45:01	**45.02**	**0.73**	77.75
9	0:00:10	0:10:19	0:10:09	**10.15**	0:00:06	0:18:09	0:18:03	**18.05**	**0.56**	28.20
10	0:00:09	0:11:08	0:10:59	**10.98**	0:00:09	0:17:07	0:16:58	**16.97**	**0.65**	27.95
11	0:00:34	0:07:09	0:06:35	**6.58**	0:00:05	0:10:51	0:10:46	**10.77**	**0.61**	17.35
12	0:00:05	0:08:03	0:07:58	**7.97**	0:00:12	0:10:53	0:10:41	**10.68**	**0.75**	18.65
13	0:00:02	0:08:32	0:08:30	**8.50**	0:00:03	0:13:08	0:13:05	**13.08**	**0.65**	21.58
14	0:00:04	0:05:19	0:05:15	**5.25**	0:00:03	0:13:09	0:13:06	**13.10**	**0.40**	18.35
15	0:00:08	0:06:24	0:06:16	**6.27**	0:00:08	0:12:38	0:12:30	**12.50**	**0.50**	18.77
16	0:00:10	0:17:14	0:17:04	**17.07**	0:00:06	0:17:26	0:17:20	**17.33**	**0.98**	34.40
18	0:00:15	0:13:04	0:12:49	**12.82**	0:00:10	0:24:57	0:24:47	**24.78**	**0.52**	37.60
19	0:00:23	0:16:07	0:15:44	**15.73**	0:00:09	0:24:57	0:24:48	**24.80**	**0.63**	40.53
20	0:00:06	0:08:26	0:08:20	**8.33**	0:00:17	0:15:13	0:14:56	**14.93**	**0.56**	23.27
								Average	**0.68**	31.64

Table 5. Subjects' performances with respect to first environment used

User ID	Windows	Haystack	Duration ratio	Gender	Use first
2	23.77	20.48	0.86	f	Haystack
4	21.45	19.60	0.91	f	Haystack
6	16.68	20.67	1.24	m	Haystack
8	45.02	32.73	0.73	m	Haystack
10	16.97	10.98	0.65	f	Haystack
12	10.68	7.97	0.75	f	Haystack
14	13.10	5.25	0.40	m	Haystack
16	17.33	17.07	0.98	m	Haystack
19	24.80	15.73	0.63	m	Haystack
1	14.18	11.55	0.81	m	Windows
3	13.93	9.32	0.67	f	Windows
5	32.15	10.02	0.31	m	Windows
7	14.95	7.92	0.53	f	Windows
9	18.05	10.15	0.56	f	Windows
11	10.77	6.58	0.61	f	Windows
13	13.08	8.50	0.65	f	Windows
15	12.50	6.27	0.50	m	Windows
18	24.78	12.82	0.52	m	Windows
20	14.93	8.33	0.56	f	Windows

Figure 46 shows a different view of the data in Table 4 and
Table 5. Each subject's performance is plotted as a point with the x coordinate equal to the subject's time on Windows and the y coordinate equal to the subject's time on Haystack. The solid diagonal line divides the plane into two halves: the upper half contains points of subjects who performed faster on Windows, and the lower half contains points of subjects who performed faster on Haystack. The diagonal dashed line has the slope of 0.7 and represents the average duration ratio discussed above.

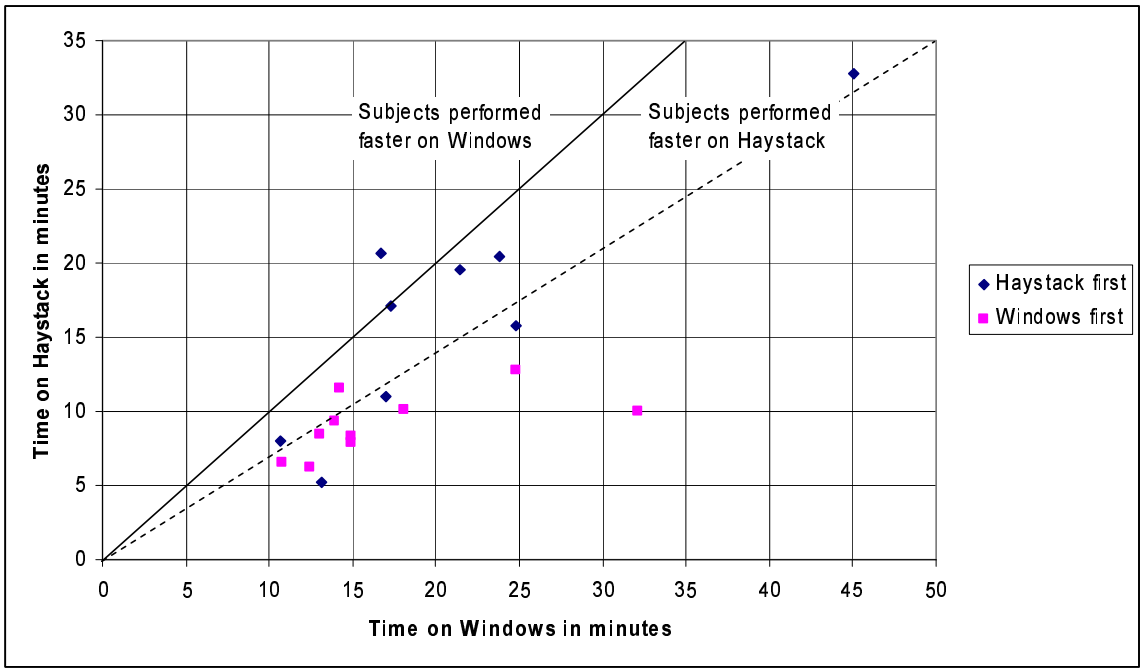

Figure 46. Comparison of subjects' performances on Haystack and Windows

Task Skippings

Table 6 shows whether subjects skipped certain tasks. There are four possibilities for each task on a given environment: the task was completed and not skipped ("no"), the task appeared to have been forgotten ("yes (forgot)"), the task was tried until an access denied error was encountered[5] ("yes (access denied)"), and the task was tried but not completed ("yes"). The subjects skipped more tasks on Windows than on Haystack. In particular, task #9, which required multiple categorization of Alisa's message, was skipped by 9 subjects on Windows, but only by one on Haystack. Likewise, task #10, which required identifying the photographer of Celine Dion's portrait, was skipped by 12 subjects on Windows, but by none on Haystack. Figure 47 shows a plot of the data in Table 6.

[5] The study coordinator forgot to grant the subjects access permission to the folder containing the song by Celine Dion.

Table 6. Recordings of whether subjects skipped particular tasks

User ID	Skip Task #2?		Skip Task #6?		Skip Task #9?		Skip Task #10?	
	Haystack	Windows	Haystack	Windows	Haystack	Windows	Haystack	Windows
1				yes (access denied)	yes (forgot)	yes (forgot)		yes
2		yes		Yes		yes		yes
3				yes (access denied)		yes		yes
4		yes (forgot)		yes (forgot)				yes
5		yes	yes (forgot)	yes (forgot)				yes
6						yes		
7						yes		
8		yes (forgot)		yes		yes		yes
9								yes
10								
11								yes
12				yes		yes		yes
13	yes			yes		yes		yes
14								
15						yes (forgot)		
16								yes
18								yes
19								
20								
yes	1	2	0	4	0	7	0	12
yes (forgot)	0	1	1	2	1	2	0	0
yes (access denied)	0	0	0	2	0	0	0	0
no	18	15	18	11	18	10	19	7

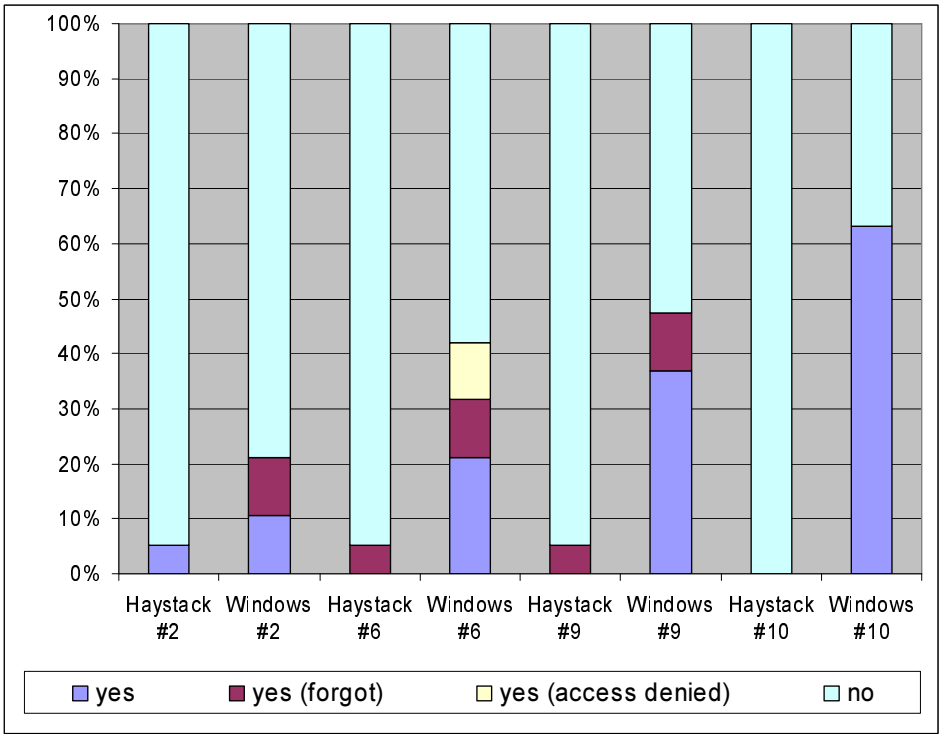

Figure 47. Breakdowns of subject pool based on whether particular tasks were skipped

Multiple-choice Survey Responses

Table 7 lists the subjects' responses to the multiple-choice questions in the survey. Note that most subjects have had more than five years of experience using Windows and the World Wide Web. However, this is not an accurate indication of their competence levels, as evidently contradicted by the wide discrepancies among the times it took they to complete the user study tasks.

Table 7. Subjects' responses to multiple-choice survey questions

	Years of Experience		Question 3.1		Question 3.2		Question 3.3		Question 3.4		Question 3.5		Question 3.6	
User ID	Windows	Web	Frequency	Easier	Frequency	Easier	Frequency	Easier	Frequency	Easier	Frequency	Easier	Frequency	Easier
1	>5	2-5	Often	Equal	Sometimes	Haystack	Sometimes	Windows	Often	Equal	Once or twice	Equal	Often	Haystack
2	>5	2-5	Sometimes	Haystack	Sometimes	Equal	Never	Haystack	Sometimes	Haystack	Sometimes	Haystack	Never	Haystack
3	>5	>5	Often	Haystack	Sometimes	Haystack	Often	Equal	Often	Equal	Never	Haystack	Never	Haystack
4	2-5	2-5		Haystack		Haystack		Equal		Equal		Haystack		Haystack
5	>5	>5	Often	Haystack	Sometimes	Haystack	Sometimes	Equal	Sometimes	Haystack	Sometimes	Equal	Sometimes	Haystack
6	>5	>5	Often	Equal	Once or twice	Haystack	Never	Haystack	Often	Haystack	Never	Haystack	Sometimes	Haystack
7	>5	>5	Often	Haystack	Never	Haystack	Once or twice	Equal	Sometimes	Equal	Sometimes	Haystack	Never	Equal
8	>5	>5	Sometimes	Equal	Often	Equal	Sometimes	Haystack	Sometimes	Equal	Once or twice	Haystack	Once or twice	Haystack
9	<1	>5	Often	Equal	Sometimes	Equal	Once or twice	Haystack	Often	Equal	Often	Haystack	Once or twice	Haystack
10	1-2	>5	Sometimes	Haystack	Once or twice	Equal	Once or twice	Haystack	Often	Haystack	Sometimes	Haystack	Once or twice	Equal
11	>5	>5	Often	Equal	Never		Never		Often	Windows	Sometimes	Equal	Often	Equal
12	>5	>5	Often	Equal	Often	Haystack	Often	Haystack	Often	Equal	Often	Equal	Sometimes	Haystack
13	2-5	>5	Sometimes	Haystack	Sometimes	Equal	Often	Equal	Often	Equal	Often	Equal	Once or twice	Haystack
14	>5	>5	Sometimes	Equal	Once or twice	Haystack	Sometimes	Equal	Often	Haystack	Never	Haystack	Sometimes	Equal
15	>5	>5	Often	Equal	Often	Haystack	Often	Equal	Sometimes	Haystack	Sometimes	Haystack	Sometimes	Haystack
16	>5	>5	Often	Windows	Sometimes	Haystack	Often	Equal		Equal	Often	Equal	Never	Haystack
18	>5	>5	Often	Windows	Once or twice	Equal	Sometimes	Equal	Never	Haystack	Sometimes	Equal	Never	Haystack
19	>5	>5	Often	Windows	Often	Windows	Often	Haystack	Often	Haystack	Sometimes	Haystack	Sometimes	Haystack
20	2-5	2-5	Often	Haystack	Often	Haystack	Often	Equal	Often	Equal	Sometimes	Haystack	Sometimes	Haystack

Figure 48 shows how often the subjects need to perform particular types of task outside the user study:

3.1: The highest fraction of the subjects need to revisit previously encountered information **often**.

3.2: The highest fraction need to rotate or resize embedded images **sometimes**.

3.3: The highest fraction need to attach a photo already embedded within a larger piece of information to an e-mail message **often**.

3.4: The highest fraction need to collect together items of several types **often**.

3.5: The highest fraction need to classify an item in more than one way **sometimes**.

3.6: The highest fraction need to locate additional information related to some pieces of information on their local computer **sometimes**.

These responses serve as an indication that the tasks chosen for this user study were realistic.

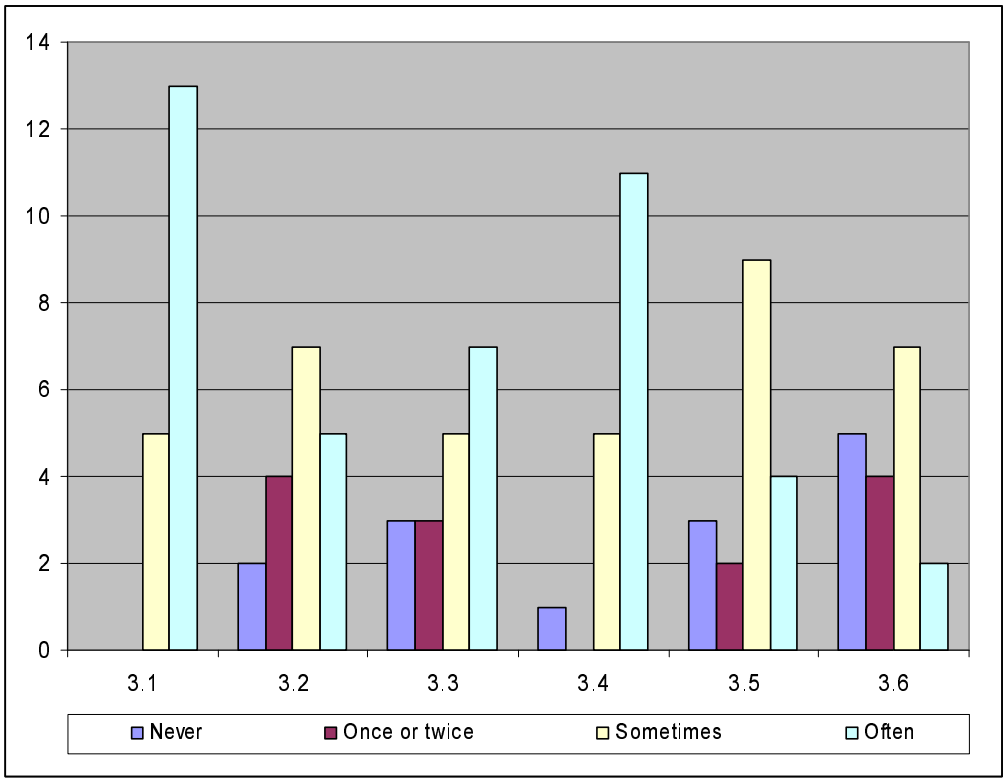

Figure 48. Subjects' responses to how often they need to perform particular types of task

Figure 49 shows the subjects' responses to whether Haystack or Windows was easier, or both were equally easy or hard, for particular types of task. The highest fraction of the subjects felt that Haystack was easier for manipulating embedded pictures, multiple categorization, and locating

additional information. The highest fraction felt that both environments were almost equally easy or hard for attaching embedded photos to messages and for collecting together items of different types. There was no type of task in which the highest fraction of subjects felt that Windows was easier.

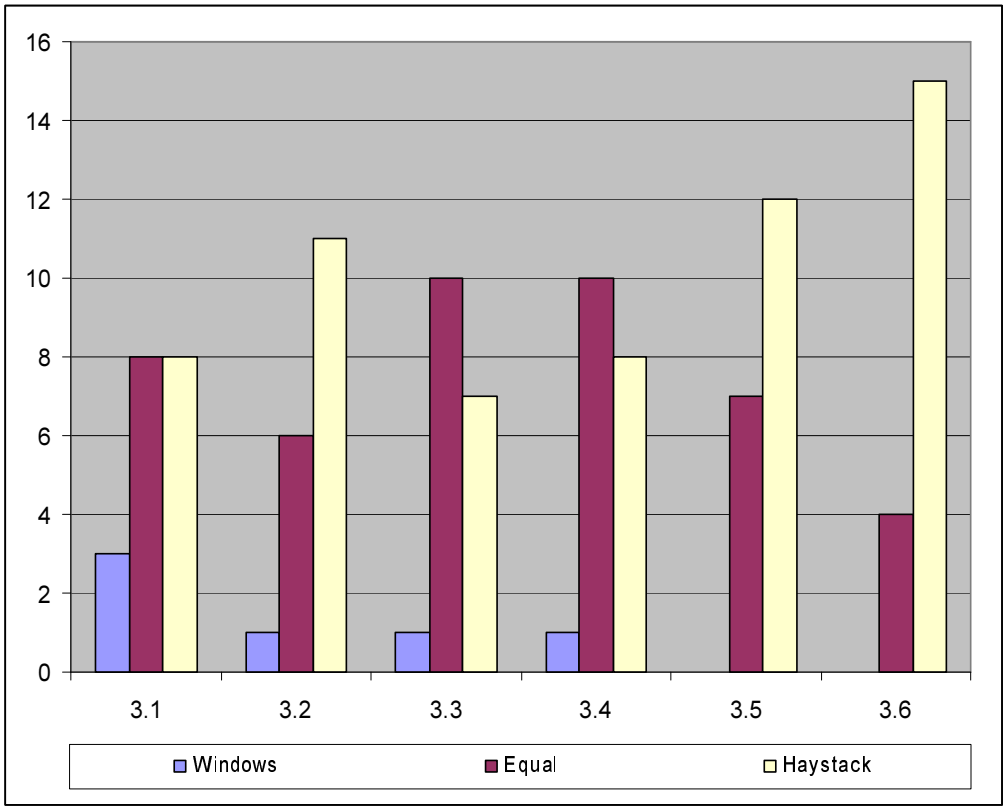

Figure 49. Subjects' answers to whether Haystack or Windows was easier, or both were equally easy, for particular tasks indicated in six survey questions

7.4.3 Written Responses

Transcribed survey written responses are included in Appendix C. There are four comments resonated by many subjects:

1. In Haystack, there is no need to open a separate program for editing images.

2. Drag and drop just seems easier in Haystack.

3. In Windows, the subjects had to consciously think about copying when they needed to classify an item in more than one way.

4. Many subjects felt that it was impossible to find picture information in Windows.

Note that the subjects' performances and responses might have been drastically different were all functionalities left intact in Haystack. The subjects might have had just as hard a time finding information in Haystack as in Windows.

7.4.4 Conclusion

It is worth reiterating that this user study was a preliminary attempt to evaluate the Haystack platform. The study had several flaws that were recognized and minimized but could not be fixed entirely due to the nature of user interface testing. The results are positive and they indicate that Haystack, as set up for the study, was easier for some realistic information management tasks as compared to Microsoft Windows XP and relevant applications. No statistical significance is claimed for these results. They simply serve as encouraging indication that "we are on the right path".

CHAPTER 8

Future Work

With encouraging indications of the Haystack UI's usability, this chapter briefly discusses improvements to be made to the current UI framework.

8.1 Component Architecture

The component architecture of the Haystack UI framework needs to incorporate versioning information and needs to make use of such information for resolving evaluators from prescriptions. Policies and mechanisms for importing, exporting, and upgrading components will need to be devised and implemented.

Security and privacy issues need to be addressed over the entire Haystack platform while considering specific needs of the UI framework. Note that perhaps the greatest challenge here lies in the use of RDF itself: the fine-grained nature of RDF makes it difficult to implement access control mechanisms inasmuch as the boundaries between objects are blurred when they are represented in RDF.

Investigations are needed on the topic of user programming. It may be beneficial to allow power users to interact with Haystack components directly to express and customize computations for their own purposes.

8.2 Views

In the current implementation of the UI framework, view requests can only contain specifications for view classes. There is a need to specify more expressively the desired characteristics of views, some examples of which are listed in section 5.2.2.

View resolution also needs improvements. For example, it is conceivable that a view request can be shipped to a different machine to be resolved if that machine understands best the specifications in that view request, or if the information required to resolve the view is restricted to that machine only. Currently, view requests are only resolved locally. In addition, view resolution has not been made dynamic in the sense that once a view request has been mapped to a view, the mapping is not changed while the view is still being shown. This lack so far can be overlooked because the rendering abstraction and the data computation abstraction keep dynamic the most frequently changing aspects of the presentations.

Views of information can also be in the form of speech as well as natural language text. The latter involves formulating output text strings in complete, grammatically correct sentences such as "Your appointment with Dr. Kane has been postponed to next Thursday afternoon at 2 p.m. at your mother's request." Note that this example illustrates several subtle signs of intelligence of the system; for examples:

- The string "your mother" is used instead of the name of the person who is the mother of the user;
- The string "next Thursday afternoon at 2 p.m." is used instead of something less humanly such as "Thursday, May 1 2003, 14:00:00 EST"; and
- The string "at your mother's request" is used instead of "at a request of your mother", showing some level of mastery of the English language;

Finally, the term "view" subtly implies that the UI is read-only. This is not true with the current UI, although the UI is not as editable as it should be. In many systems, there is a clear distinction between the view mode and the edit mode (or the run time and the design time). We wish to blur this distinction by allowing users to edit not only user information but also UI layouts on the fly at runtime. This capability will let users customize Haystack UI flexibly to their needs.

8.3 UI Elements

Ozone's text handling capabilities fall short in several aspects. First, advanced text styles (e.g. small caps, drop cap), full justification of text, international text support (e.g. vertically flowing text), and custom text wrapping support (e.g. around images) are currently missing. Second, there is no uniform support for text selection and text editing. Currently, a piece of text can only be selected or edited if it is inside a text box. Third, punctuation marks are not handled properly since the `slide:Text` element does not distinguish punctuation marks from other characters. In certain cases, punctuation marks fall onto new lines rather than stay with the preceding words. Fourth, sorting is not supported on compound text output: that is, if a piece of text on the screen is rendered by several UI elements, it is put together at the pixel level, is not represented as a whole internally, and hence cannot be sorted by. For example, an onscreen text string such as "the authors of paper X are Y and Z" is painted by six different UI elements that render the following six substrings individually: "the authors of ", "paper X", " are ", "Y", " and ", and "Z". This is because the second, fourth, and sixth substrings are "expanded" at runtime from view requests made for three different objects: the paper and the two authors. An effect of these late expansions is that the entire string seen on the screen is never stored in its entirety internally.

In addition to text functionalities, various graphics capabilities of the framework need improvements. First, there is no support for zooming. If text editing is allowed everywhere, for uniformity, zooming will be needed everywhere rather than just within document editors and viewers as provided in current information management environments. Second, printing should also be supported. Together, zooming and printing require the UI framework to abandon its pixel-based coordinate system in favor of a coordinate system that is more flexible. This change would be a major overhaul, not a minor re-factoring. Third, zooming requires icons to scale gracefully at any level of magnification. This in turn requires vector graphics. Fourth, animations should be supported to allow exploration of more dynamic UI concepts. This list of graphics capability improvements can go on and on.

Input event handling is also currently lacking. Events such as keyboard accelerators (shortcuts) are currently not supported. The scheme for routing keyboard events is underdeveloped in the UI framework. Unlike mouse events which are fired against very specific UI elements on the screen, keyboard events are fired to the whole environment, which then routes the events to appropriate UI elements based on the concept of an abstract keyboard focus. However, certain keyboard events are not get routed to the UI element having keyboard focus if they are deemed to be keyboard

accelerator events. A keyboard accelerator event is rather sent to whichever UI element that has registered itself to handle that accelerator.

A multi-tool paradigm may be investigated. The current UI framework only accepts mouse and keyboard events for input. It can be made to handle more input devices, such as pen, multiple mice, multiple keyboards, touch, etc. There needs to be a unified paradigm for binding physical input devices and input events to abstract tools in the system. A user can use several tools at the same time to accomplish complex tasks. Convenience should be modeled so that each tool is made easily accessible where its use is appropriate.

Recognition logic can be added for accepting gesture and speech input. Recognition-based user interfaces are more complicated than event-based user interfaces in that the formers have to divine high-level user intentions from streams of low-level input events. Some difficulties involved in recognition-based user interfaces are summarily discussed in [33].

8.4 Data Computations

Currently, a data request might have a syntax very different than that of an Adenine expression while they both express the same computation. The syntax of the Adenine expression is always much simpler. There is a high probability that the syntax of the data request can be simplified to resemble the Adenine expression.

Not all UI elements currently understand data requests and are able to work with data producers. Even if a UI element can understand a data request, it only chooses to interpret data requests assigned to only a few of the attributes of its rendering requests. All UI elements should be made capable of understanding data requests attached to any attribute of their rendering requests so that the UI can be responsive to any change to rendering requests. Before this can be done, however, more framework support is required to make data producers easier to interact with.

Appendix A – User Study Instructions

Please read the following scenario and the list of tasks thoroughly. Then let the user study coordinator know when you have finished reading. These instructions will be available to you throughout the study—there is no need to memorize them.

Scenario:

You are starting to work on a group project with your friend Alisa. This project involves collecting information about famous Canadian singers and writing an essay about them.

Alisa has managed to get some contact information for Celine Dion, a famous Canadian singer, who is incidentally one of your favorites. Alisa has sent you an e-mail message with a **contact information card** attached.

Please perform the following 12 tasks **in sequence**.

If you cannot perform a particular task after trying as much as you deem necessary, **go on to the next task**.

1. From Alisa's e-mail message, **view the contact information of Celine Dion**. Note that the contact information card includes a portrait of her and the map to her home.
2. You intend to conduct a face-to-face interview with Celine, hopefully tomorrow. As a result, you want a quick way to return to the map later when you need to drive to her home. **Create that quick way**. You will be asked to return to the map later.
3. The photograph of Celine Dion is in a wrong orientation. **Rotate that photo** to orient it correctly. **When this task is done, the photograph that you see in the contact information card should be right-side up.**
4. Another friend of yours, Ben, is also a fan of Celine Dion. You want to send her photograph to him. **Compose a new e-mail message and attach the (properly oriented) photo**.

5. After attaching the photograph, you remember that it is actually very large. To be courteous to Ben who is on dialup, **make sure that you send the photo at only 50% of its original size**. That is, reduce the image to be sent to half of its width and half of its height. Don't actually send the message.
6. Now that you have some information about Celine Dion, you decide to include her information in your project. Since you already have **one** of her songs, you want to collect it together with other information in your project. **Go to your collection of songs and organize Celine's song into your project**.
    a. In Windows, all information in your project is stored in the folder **C:\Canadian-Singers-Project**, accessible through a shortcut on the Desktop. Or you may choose any other means available for collecting the information together.
    b. In Haystack, all information in your project is classified into the category labeled "**Canadian Singers Project**" shown on the right pane.
7. The e-mail message from Alisa is also relevant in your project. Also **put it in your project**.
8. You will need to **include Celine's (properly oriented) photograph in your project** as well.
9. Your friend Alisa's e-mail message also talks about a plan for the weekend and gives you some tips about interviewing with IBM for a summer internship. **File the message away** taking into account these two points.
    a. In Haystack, there is a category labeled "Fun" and a category labeled "Internship".
    b. In Windows, Outlook has two e-mail folders labeled "Fun" and "Internship".
10. **Find out the name of the photographer who took Celine's photograph**, as he might have taken other pictures of her or of other famous Canadian singers.
11. **Browse the photographer's collection of photographic work**.
12. **Quickly return to the map to Celine's home**.

Appendix B – User Study Survey

1. For how many years have you used Microsoft Windows (any version)? Select one:

   ❑ < 1 year ❑ 1 – 2 years ❑ 2 – 5 years ❑ > 5 years

2. For how many years have you used the World Wide Web? Select one:

   ❑ < 1year ❑ 1 – 2 years ❑ 2 – 5 years ❑ > 5 years

3. For each of the six (6) tasks below:

   - select one option to indicate whether you have ever ***had a need*** to do that task or something similar to it ***outside this user study*** (choose one among "never", "once or twice", "sometimes", and "often");
   - select one option to indicate whether it is easier for you to have done the task in Windows or Haystack, or to indicate that both environments are almost equally easy or hard;
   - if one environment is easier with respect to that particular task, briefly explain how or why it is easier in your opinion, drawing from your experience during this user study.

   Tasks:

   1. Creating a quick way to return to a piece of information that you have previously encountered (as done with the map in steps #2 and #12 in user study instructions).

      ❑ never ❑ once or twice ❑ sometimes ❑ often

      ❑ almost equally easy or hard ❑ Windows easier ❑ Haystack easier

If one is easier, why or how? _____

_____

_____

2. Rotating or resizing a picture embedded within a larger piece of information such as a contact information card or an e-mail message (steps #3,5).

❑ never ❑ once or twice ❑ sometimes ❑ often

❑ almost equally easy or hard ❑ Windows easier ❑ Haystack easier

If one is easier, why or how? _____

_____

_____

3. Attaching a photo already embedded within a larger piece of information to an e-mail message (e.g. attaching Celine's portrait in her contact information card to the message to Ben, step #4).

❑ never ❑ once or twice ❑ sometimes ❑ often

❑ almost equally easy or hard ❑ Windows easier ❑ Haystack easier

If one is easier, why or how? _____

_____

_____

4. Collecting together items of several types (e.g. putting various things into the project, steps #6,7,8).

❑ never ❑ once or twice ❑ sometimes ❑ often

❑ almost equally easy or hard ❑ Windows easier ❑ Haystack easier

If one is easier, why or how? _____

_____

_____

5. Classifying an item in more than one way (e.g. classifying Alisa's message into both Fun and Internship categories, step #9).

❑ never ❑ once or twice ❑ sometimes ❑ often

❑ almost equally easy or hard ❑ Windows easier ❑ Haystack easier

If one is easier, why or how?_____

_____

_____

6. Locating additional information related to some pieces of information on your computer (such as identifying authors of photographs, steps #10,11).

❑ never ❑ once or twice ❑ sometimes ❑ often

❑ almost equally easy or hard ❑ Windows easier ❑ Haystack easier

If one is easier, why or how?_____

_____

_____

Do you have any other comment about this user study or your experience in performing the tasks it includes?

_____

_____

_____

_____

_____

_____

_____

_____

_____

(End of Survey)

Appendix C – Survey Written Responses

USER 1

3.2: With Haystack, I just rotated inside. In Windows, I had to open a separate program.

3.3: With Haystack, I ended up making mistakes (attaching extra files) that I couldn't fix. In Outlook, it's much easier to remove unwanted files.

3.6: It was impossible to find picture info in Windows. In Haystack, all I had to do was click.

Overall comment: Part of the reason I've had so much difficulty in this study is because I use Eudora as opposed to Outlook. Regardless, Haystack still seems easier (except when it comes to deletion), yet just as flexible as Windows.

USER 2

3.1: In Haystack it was easier to just drag and drop info, as opposed to copying/pasting it in Outlook.

3.3: It was easier to drag and drop in Haystack.

3.4: Drag and drop in Haystack worked faster than organizing materials from different Windows programs.

3.5: Haystack was easier than new version of Windows; this task was easier to do in older version of Windows.

3.6: In the Haystack version, the source info on the photograph was easier to access (couldn't find it in Windows).

Overall comment: I was more experienced in older version(s) of Outlook; new version seems harder to use. (Also, I had very little experience with Media Player.)

Once I got used to Haystack, it seemed easier to use, more intuitive than Windows (though I had some problems with ???, mostly with tasks I normally never do).

USER 3

3.1: You could just click on it in the margin; though I imagine with lots of files on your computer Windows is easier because it has such a strong organizational system in place.

3.2: You did not have to export the image to a new application.

3.3: In Windows, I would normally just forward the email; but in Haystack it was easier to attach it.

3.4: Haystack was easier, but it bothered me that the file didn't actually move; it was just a series of shortcuts being created.

3.5: I think had I known how to do it in Windows, it would have been fine, but it seems this is what Haystack is designed for.

3.6: Perhaps I was just ??? with Windows.

Overall comment: I think Haystack seems like great software, but I would be wary of things getting cluttered in a way they don't in Windows. Windows is harder to use and more time consuming because it is basically like a filing cabinet. Haystack seems more like a notice board.

USER 4

Overall comment: This new user interface is great, but for new users, it seems there are too much information /or options.

USER 5

3.1: Full status appears to exist in object pointer structure… So one click or a double click and your back.

3.2: There were fewer options to choose from.

3.4: There are constraints in Windows that do not permit universally qualified "pointers" or references.

3.6: Once one knows the presentation "format", very easy.

Overall comment: The differentiation of validity of test due to "users" a priori knowledge. The tasks were simple from users view but quite complicated computationally! Good luck…

USER 6

3.2: With Haystack, I do not have to look for a photo editor. Luckily, there was a shortcut in Windows on the desktop.

3.3: With Windows, I would have to look for the file name. Drag + drop did not work for me to attach.

3.4: One click. No looking for project folder.

3.5: It was impossible in Windows without making a copy of the message.

3.6: Right click accomplishes this in Haystack. In Windows there are many options and pulldown menus at the top.

Overall comment: Normally, I would know where the files are (i.e., if it were my computer).

This time, I needed to find the file location from media player and then browse there. Normally, I would go directly there and have less trouble.

Icons need to be clickable in Haystack. I never understood the difference between left & right panes.

USER 7

3.1: Everything is right in the "favorites" list, always open. W/ Windows you have to navigate through folders and click a lot of buttons.

3.2: No need to open a photo editing program.

3.5: I couldn't figure out how to do it in Outlook…

USER 8

3.3: Was able to locate the Haystack message easier since it all shows without overlapping on the screen.

3.5: Could not recopy or recreate message using Windows

3.6: Also, could not find the info in Windows as opposed to Haystack's being shown on the photograph.

Overall comment: No.

USER 9

3.2: Haystack: easier to do the resizing

Windows: easier to save a copy of both the original and the resized version

3.3: I liked having all my current info simultaneously available on the screen – made drag and drop/check boxes much simpler

3.5: (often – but not usually within an application – usually just a file)

easier because can just select and click check box; which of Haystack of Windows uses len space to do this?

3.6: because Haystack was created to display this info very easily – I don't even known where this info is in Windows.

Overall comment: Note: Although I have used Windows @ work and during labs (6.182; 12.307) my day-to-day computer use is usually Athena (Unix) and Mac OS X.

For simple tasks I would definitely prefer Haystack to Windows once the response time improves. But I'm not sure I would like it very much for more complex tasks – e.g. it wasn't clear how to find general system information, etc.

USER 10

3.1: Just drag it to the side of the screen instead of saving it in a different place or creating a shortcut.

3.3: Again the dragging instead of messing with save windows and searching for it.

3.4: The files were pretty much easily accessible to start off with. In Windows I had to go searching for them before I could start gathering.

3.5: Could drag or check the same object in multiple places instead of making sure you copy it to the right place or copy it instead of moving it. (I had trouble finding in Windows copy to command.)

Overall comment: Unfamiliar Haystack tasks were easier to figure out in general than in Windows tasks. I like the Haystack layout. I don't have much experience with music programs or altering photos.

USER 11

3.3: never use address books – images always separate files

3.4: Sometimes using multiple sources would get clumsy/cluttered w. Haystack interface.

3.5: shortcuts

3.6: ??? in PSP if you don't know the program (look how easy it is for read/alter 103 tags)

Haystack seems too perfectly set up for these tasks. As if giving a random tasks it would not be easy to do.

Overall comment: PSP is not really an easy/intuitive program for these tasks.

Don't know about anyone else, but I've never worked with email this way (trying to edit in message, etc.) and I don't think I'd want to…

USER 12

3.2: You don't have to open a separate prog. to edit pictures

3.3: I can't seem to just drag and drop in Windows like I did in Haystack.

3.6: Haystack makes the info/properties really easy to find—just by right click

Overall comment: I could have done better in Windows if I was familiar with where all the files are stored. But overall I think Haystack made the tasks slightly easier to perform.

USER 13

3.1: It was more straightforward (click + drag).

3.6: Clicking once presented more information at a time, so less searching involved, therefore less wasted time.

Overall comment: I found Haystack easier b/c I tend to be a more visual learner. Haystack had concise information on various aspects on ONE screen, so it saved time + frustration in figuring out the tasks.

USER 14

3.2: It was all done within one program.

3.4: It was much easier to find information. I had trouble finding a way to more Celine's song.

3.5: Didn't force me to copy the image over and over again.

Overall comment: I really enjoyed using Haystack. I usually use Windows Explorer on my own computer so I like the feel of the browser applied to a computer and not just the Internet.

USER 15

3.2: Haystack will perform the rotation/resize without loading an external program—built-in functionality

3.4: Categories aren't limited to a certain type of media and doesn't obviously clutter the directory tree.

3.5: The groups are sufficiently abstracted away from the workings of the OS that arbitrary membership seems to be more intuitive—a long way of saying "it's just easier"

3.6: Haystack seems to be intelligent enough that it can recognize what operations one might want to perform—much better than Windows

USER 16

3.1: I used to use Windows, that's all. Beside of that, almost equally easy.

3.2: You don't need to use another program to do it.

3.6: Because in Haystack you see an author.

Overall comment: But generally interface of Haystack now is not very user-friendly. Sometimes it's a long way to see how you can do simple things, e.g., send an e-mail.

USER 18

3.1: I'm used to using Windows + never used Haystack before today.

3.4: Haystack seemed more intuitive.

3.6: I couldn't locate the function in Windows to get the add'l info (I could in Haystack).

USER 19

3.1: Because I have more experience with M.W.

3.2: Because I was working with this program before.

3.3: It's easier because you need just drag the photo to the attachment line

3.4: It's easy a little bit because you can drag and dropp function.

3.5: You just need click the button "fun" and "internship" and information will be classified. In M.Windows it's terrible!!!

3.6: I found information very quickly because it was located on the same screen. In Windows it takes 10 min to find it.

Overall comment: I found that Haystack is easy in operating in general.

USER 20

3.1: Windows gives the option of having necessary files open and minimized in order to return. In Haystack I didn't need to keep it open, because I had folders explored and could see files I need.

3.2: We almost always have to rotate or resize pictures when sending e-mails, and Windows allows you to do so only by using specific programs. Haystack had it in right place.

3.3: In both programs I just drag and drop pictures.

3.4: I would say I still drag and drop, but with Haystack I didn't need to open the folder before droping there. In Windows I use "move to folder" button.

3.5: Didn't need to copy before moving.

3.6: Haystack opened picture properties with picture. Didn't need to go to specific program and check image properties.

Overall comment: Study is good in identifying weak spots of Windows, with which each of us is dealing dayly. But I really see not a substitute to Windows, but an improved Outlook Express or other e-mail programs. The overall impression about Haystack was positive.

References

[1] *BrownSauce RDF Browser*. http://brownsauce.sourceforge.net/.

[2] *Common Lisp Interface Manager (CLIM): Release 2.0*. Symbolics, Inc., 1994.

[3] *CORBA*. http://www.corba.org/.

[4] *Haystack*. http://haystack.lcs.mit.edu/.

[5] *IBM Lotus software*. http://www.lotus.com/.

[6] *IsaViz Overview*. http://www.w3.org/2001/11/IsaViz/.

[7] *Microsoft COM Technologies*. http://www.microsoft.com/com/default.asp.

[8] *OEone Operating Environment*. http://www.oeone.com/.

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

[10] *Resource Description Framework (RDF) Schema Specification*. http://www.w3.org/TR/1998/WD-rdf-schema/.

[11] Ambrosio, Ana Paula. Introducing semantics in conceptual schema reuse. In *Proceedings of the 3rd International Conference on Information and Knowledge Management*, pages 50–56, 1994.

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

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

[14] Buxton, W., Lamb, M. R., Sherman, D., Smith, K. C. Towards a comprehensive user interface management system. In *Proceedings of the 10th Annual Conference on Computer Graphics and Interactive Techniques*, pages 35–42, 1983.

[15] Craig, David T. *Canon's Cat Computer: The Real Macintosh*. http://archaic-apples.shauny.de/files/cat/canon.html.

[16] Dourish, Paul, Edwards, W. Keith, LaMarca, Anthony, and Salisbury, Michael. Using properties for uniform interaction in the Presto document system. In *Proceedings of the 12th Annual ACM Symposium on User Interface Software and Technology*, pages 55–64, 1999.

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

[18] Frakes, William and Terry, Carol. Software reuse: metrics and models. *ACM Computing Surveys* (CSUR), 28(2), June 1996.

[19] Gabriel, Richard P., White, Jon L., and Bobrow, Daniel G. CLOS: integrating object-oriented and functional programming. *Communications of the ACM*, 34(9):29–38, 1991.

[20] Halasz, Frank G. Reflections on "Seven Issues": Hypertext in the Era of the Web. *ACM Journal of Computer Documentation*, 25(3), August 2001.

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

[22] Hayes, Philip J., Szekely, Pedro A., and Lerner, Richard A. Design alternatives for user interface management systems based on experience with COUSIN. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1985.

[23] Hill, Ralph D. Supporting concurrency, communication, and synchronization in human-computer interaction—the Sassafras UIMS. *ACM Transactions on Graphics (TOG)*, 5(3):179–210, July 1986.

[24] Hudson, Scott E. and Mohamed, Shamim P. Interactive specifications of flexible user interface displays. *ACM Transactions on Information Systems (TOIS)*, 8(3):269–288, July 1990.

[25] Huynh, David. *Haystack's User Interface Framework: Tutorial and Reference*. http://haystack.lcs.mit.edu/documentation/ui.pdf.

[26] Kahan, José and Koivunen, Marja-Ritta. Annotea: an open RDF infrastructure for shared Web annotations. In *Proceedings of the 10th International Conference on World Wide Web*, pages 623–632, 2001.

[27] Klein, Daniel. Developing applications with the Alpha UIMS. *interactions*, 2(4):48–65, October 1995.

[28] Krueger, Charles W. Software reuse. *ACM Computing Surveys (CSUR)*, 24(2), June 1992.

[29] Li, Wen-Syan and Holowczak, Richard D. Constructing information systems based on schema reuse. In *Proceedings of the 5th international conference on Information and Knowledge Management*, pages 197–204, 1996.

[30] Manheimer, J. M., Burnett, R. C., and Wallers, J. A. A case study of user interface management system development and application. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 127–132, 1989.

[31] Meerten, L. and Pemberton, S. The Ergonomics of Computer Interfaces – Designing a System for Human Use. *CWI Report CS-R9258*, December 1992, CWI Amsterdam.

[32] Möller, Ralf. *Symbolics Lisp Machine Museum*. http://kogs-www.informatik.uni-hamburg.de/~moeller/symbolics-info/symbolics.html.

[33] Myers, Brad, Hudson, Scott E., and Pausch, Randy. Past, Present, and Future of User Interface Software Tools. *ACM Transactions on Computer-Human Interaction*, 7(1):3–28, March 2000.

[34] Palay, A. J., Hansen, W., Kazar, M., Sherman, M., Wadlow, M., Neuendorffer, T., Stern, Z., Bader, M., and Peters, T. The Andrew toolkit: An overview. In *Proceedings on Winter 1988 Usenix Technical Conference*, USENIX Assoc., Berkeley, CA, 9-21.

[35] Pittman, Jon H. and Kitrick, Christopher J. VUIMS: a visual user interface management system. In *Proceedings of the 3rd annual ACM SIGGRAPH Symposium on User Interface Software and Technology*, pages 36–46, 1990.

[36] Platt, David S. *The Essence of OLE: A Programmer's Workbook*. 1st ed. Prentice Hall, 1996.

[37] Price, Roger. Beyond SGML. In *Proceedings of the 3rd ACM conference on Digital Libraries*, pages 172–181, 1998.

[38] Puerta, Angel and Eisenstein, Jacob. Towards a General Computational Framework for Model-Based Interface Development Systems. In *Proceedings of the 4th International Conference on Intelligent User Interfaces*, pages 171–178, 1998.

[39] Quan, Dennis. *Metadata Programming in Adenine*. http://haystack.lcs.mit.edu/documentation/adenine.pdf.

[40] Quan, Dennis, Karger, David R., and Huynh, David. RDF Authoring Environments for End Users. *International Workshop on Semantic Web Foundations and Application Technologies (SWFAT)*, 2003.

[41] Raskin, J. *The Humane Interface*. Addison-Wesley, 2000.

[42] Roberts, Donald Bradley. Practical Analysis for Refactoring. PhD Thesis. University of Illinois at Urbana-Champaign, 1999.

[43] Shneiderman, Ben. Direct Manipulation for Comprehensible, Predictable and Controllable User Interfaces. *In Proceedings of the 2nd International Conference on Intelligent User Interfaces*, pages 33–39, 1997.

[44] Sibert, John L., Hurley, William D., and Bleser, Teresa W. An object-oriented user interface management system. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, pages 259–268, 1986.

[45] Singh, G. and Green, M. A high-level user interface management system. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 133–138, 1989.

[46] Storey, Margaret-Anne, Best, Casey, Michaud, Jeff, Rayside, Derek, Litoiu, Marin, and Musen, Mark. Demonstration: SHriMP views: an interactive environment for information visualization and navigation. *Conference on Human Factors and Computing Systems*, pages 520–521, 2002.