

Learning the Process of World Wide Web Data Retrieval

by

Ryan A. Manuel

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

January 24, 2005

Copyright 2005 Massachusetts Institute of Technology. All rights reserved.

Author _____
Department of Electrical Engineering and Computer Science
January 24, 2005

Certified by _____
David R. Karger
Thesis Supervisor

Accepted by _____
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Learning the Process of World Wide Web Data Retrieval

by

Ryan A. Manuel

Submitted to the Department of Electrical Engineering and Computer Science
on January 24th, 2005, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

We develop a method for extracting and internalizing web site form submissions which we refer to as web operations. To begin the process, a user performs a sample submission of the form. From that submission, our system determines all of the necessary information to store the web operation. Through a simple user interface the user can view and modify the web operation to the extent that he wants or needs to. With the operation now stored, the user can invoke the operation without browsing to the web site on which the operation was originally contained. By utilizing the web site information extraction techniques contained in the Haystack information management system, we give the user the option to extract information off of web operation results pages. Thus, when using our system to the fullest extent, a user can invoke web operations and view and make use of the results without viewing any web pages.

Thesis Supervisor: David R. Karger
Title: Associate Professor

Acknowledgements

I would like to thank my advisor, David Karger, for all of his continual support. Without his advice, feedback, and ideas, this thesis would not have been possible. Also, I would like to give my thanks to Andrew Hogue who took the time to help me learn about both his wrapper induction system and the Haystack system in general. Lastly, I would like to thank my family for their constant support throughout my life, especially throughout my years at MIT.

Table of Contents

1 Introduction.....	8
2 Related Work	11
2.1 Learning Web Operations.....	11
2.2 Haystack.....	12
2.3 Operations	13
2.4 Wrapper Induction	13
3 User Interface.....	15
3.1 Storing Web Operations.....	15
3.2 Editing and Viewing Stored Web Operations	19
3.2.1 Viewing the Operation with Hidden Parameters.....	21
3.3 Invoking Stored Web Operations.....	23
4 Web Operation Characterization.....	26
4.1 Inputs of a Web Operation.....	26
4.2 Outputs of a Web Operation.....	28
5 System Design.....	30
5.1 Learning a Web Operation	30
5.1.1 Listening to Posts and Gets	31
5.1.2 Learning Parameters of the Web Operation.....	32
5.1.3 Acquiring Information From the HTML	33
5.1.4 Learning to Format the Results.....	36
5.2 Storing a Web Operation	37
5.2.1 Storing Parameters	37
5.2.2 Storing How to Invoke the Operation.....	38
5.2.3 Storing Result Identification.....	38
5.3 Invoking a Stored Web Operation	39
5.3.1 Constructing the String.....	39
5.3.2 Sending the String to the Appropriate Address.....	40
5.3.3 Gathering the Results	40
6 System Implementation and Architecture.....	41
6.1 Listening for Web Operations	41
6.2 Remembering the Operation	42
6.3 Storing the Operation Within Haystack.....	43
6.4 Invoking the Operation	44
6.5 Improvements to Operations	46
7 Testing Results	50
7.1 Web Operation Survey	50
7.2 System Successes	51
7.3 System Failures	54
7.3.1 Operations with Intermediate Pages.....	54
7.3.2 Confusion About Parameters' Purposes.....	55
7.3.3 Form Components Manipulated by JavaScript.....	56
7.3.4 Results Pages that Differ Based on Inputs.....	56
8 Conclusions.....	58

8.1	Contributions.....	58
8.2	Future Work.....	58
8.2.1	Incorporating Other Parameter Types.....	59
8.2.2	Cascading Operations Together.....	59
8.2.3	Allow Multiple Wrappers per Operation.....	59
8.2.4	Cope with Operations with Intermediary Pages.....	60
8.2.5	Organization for Web Operations.....	60
8.2.6	Mechanism for Updating Web Operations.....	60
8.2.7	Integration with Haystack on Other Systems.....	61
8.2.8	Making Renaming Parameters Easier.....	61
A	List of Sites to Test.....	62
B	Test Results.....	64

List of Figures

Figure 1: Context menu for remembering web operations. This menu is displayed on a results page after the user submitted a query to Google.	16
Figure 2: UI continuation for remembering web operations	17
Figure 3: A results page wrapped with the wrapper google	18
Figure 4: A view of the web operation google_query	19
Figure 5: Expanded view of a web operation parameter	20
Figure 6: View of the complete web operation google_query_Original	23
Figure 7: An example search for a web operation	24
Figure 8: Context menu giving the user the option to "Perform operation"	24
Figure 9: UI continuation for the web operation google_query	25
Figure 10: A results collection for a web operation.....	25
Figure 11: http://www.google.com/ - an example initiating web page.....	27
Figure 12: An example results page.....	29
Figure 13: An example checkbox parameter	48
Figure 14: An example select parameter	49
Figure 15: Form from the web site http://www.amazon.com	51
Figure 16: Invocation of an Amazon.com query	52
Figure 17: Invocation of an advanced Google query	53

List of Tables

Table 1: List of sites tested	63
Table 2: List of sites tested and comments about their success	64

Chapter 1

Introduction

Many applications become more useful when more information is available to them. For example, most e-mail applications will autocomplete e-mail addresses based on information stored in an address book. The more contact information stored in the e-mail program, the more the user can utilize the autocomplete feature of the e-mail program. Unfortunately, getting information into systems like an e-mail program may involve tedious data gathering from a variety of sources.

One of these sources, the internet, is an enormous repository of information. Unfortunately, this data is not represented so that it can be accessed by automated systems in a standardized way. This can create a hassle for users and applications that want to utilize this vast repository of information to its fullest extent.

One application whose goal is to utilize such a repository of information is Haystack [8]. Haystack is an application with the key feature that all objects within the system have semantic data that the user can access at any time. As one method of gathering information, Haystack attempts to extract data from the internet through a process called wrapper induction. The method that wrapper induction uses to extract data off of web sites utilizes the fact that most individual web sites present their information in the form of similarly structured records on a web page. Once Haystack has identified the structure for a particular type of record on a particular website, it can extract that record and others like it off of the page and incorporate them into Haystack's database.

Extracting data from a web page is only part of the process of getting it from the internet and into an application like Haystack. Getting to the web pages that contain the data is also an important part of this process. Most web sites that contain records that can be wrapped and recognized by Haystack present the records when the user performs some sort of query or other web operation.

The goal of our system is to incorporate this concept of a web operation into Haystack. In order to accomplish this, the user needs a mechanism to tell the system about a certain web operation. Our system accomplishes this by having the user perform a sample invocation of the operation. Then, from this sample, the system determines all the necessary information to perform this operation in the future.

Once the user can store and invoke web operations from within Haystack, he can make use of the features that Haystack offers. For example, a user can organize his operations in whatever way he sees fit. Also, the user can add his own descriptions for his operations. Then in the future, he can use some of Haystack's searching features in order to find web operations based on those descriptions. In short, the user can find and invoke operations from within Haystack without having to travel to the operations on the internet or deal with the extraneous clutter that is contained on many web sites.

The process that is used to incorporate web operations into Haystack is fairly straightforward. The system observes the user's actions until it notices that a web operation has been invoked. At this point the system gathers information about the operation and allows the user to specify if he wants to store the operation. To store the operation, the user must provide two pieces of information. First, the user specifies a name for the operation so that the user can find and perform the operation from within Haystack in the future. Second, the user can potentially specify a wrapper for the results page so that when the operation is invoked in the future, the resulting page can be wrapped and appropriate records can be extracted. If the user does not specify a wrapper, the results page will be displayed to the user.

Once the user has successfully stored a web operation, they can then, through a simple user interface, view and change any of the parameters of the operation as well as other relevant information about the operation. At the minimum the user will possibly want to change different properties of an operation's parameters so that performing the operation in the future is easier. Advanced users may want to change other aspects of the operation to suit their needs.

With the web operation stored within Haystack, the user can now find the operation easily within Haystack and perform it whenever he wants. When the operation is invoked within Haystack, the user is presented with parameters to fill out. After the user fills out the parameters and submits the operation, the user is then presented with one or more Haystack objects that are the result of the web operation.

In this thesis, we first describe in Chapter 2 some of the related work and existing systems of which our system makes use. In Chapter 3, we give a demonstration of the user interface features of our system. Then, in Chapter 4, we give a brief description of how web operations work. This includes what is involved in invoking them and what the results of these operations typically consist of. In Chapter 5, we briefly go over a top-level design of our system. The design includes what information is necessary to store an operation, how our system will go about acquiring that information, and how we will extract any results. In Chapter 6, we go into the specifics of the architecture and implementation of our system and how we went about incorporating it into the existing Haystack framework. Chapter 7 discusses several successful and unsuccessful test cases. In Chapter 8, we discuss our conclusions and several ideas for future work. Appendix A contains a table of web sites that we tested and Appendix B contains comments about how our system performed on those sites.

Chapter 2

Related Work

2.1 Learning Web Operations

Some web browsers have already attempted to learn web operations. Browsers like Mozilla [2] or Safari [1] offer to remember values that users have filled in when completing forms within web pages. Once the user chooses to remember these parameter values in this fashion, he can visit the web sites that contained those operations and the browser will fill in the previous values for the parameters. Thus, all the user needs to do to invoke the operation is press submit. This speeds up the operation process, but, unlike our system, the user still has to visit the web sites that contain the operations in order to invoke them. Thus, they may have to remember the web site that contains the operation or manually search through bookmarks that contain web site titles in order to find the operation. By internalizing the operations in our system, we allow the user to search for the operation itself as opposed to searching for or remembering the web site on which it was contained.

OmniWeb [4] has also added a feature to learn web operations. This feature works by creating shortcuts for web operations. In order to create these shortcuts, the user selects a textbox in a form by clicking inside one of the textboxes. Then, the user can use select to remember this form and give it a name. From then on, the user will be able to type in the name for this form and a search term and invoke the form. One of the benefits of this system is that the user interface for specifying which form is to be learned is fairly simple. Also, this browser allows the user to perform the web operation without necessarily visiting the web site that contains the operation. However, this system counts

on the fact that the web operation in question only has one parameter. This is necessary because, to invoke a learned operation, the user types in the name of the learned operation and a value for the single parameter in the search pane of the browser. Because of this interface there would be no way to distinguish multiple values for parameters if the system allowed multiple parameter operations.

One of the benefits of our system over these other systems is that we allow users to perform web operations without worrying about the internet at all. Like users of the OmniWeb browser, users of our system do not have to visit the web site containing the web operation in order to invoke it. In addition, users do not have to deal with the extraneous information that is quite common on search result pages. They also do not have to worry about extracting any information off of results pages. In our system, we use the wrapper induction technique existing in Haystack to identify and extract information off of results pages in the form of records that Haystack will understand.

2.2 Haystack

As mentioned in Chapter 1, we will be extending some of the capabilities of Haystack for our system. Haystack is a system that is based on the concept of the semantic web [5]. In a semantic web system like Haystack, there is a network involving all objects in the system. In the network, properties of the individual objects are the links that connect different objects together. For example, a document object within Haystack may have an author property that links the document to a person object. This concept, combined with appropriate views of these objects, gives users the ability to perform powerful tasks. These tasks are generally presented to the user in the form of context menus. For example, when viewing an e-mail message, the user can right click on the message and perform the typical tasks that are available in most e-mail clients like replying to or forwarding the message.

The power of Haystack comes from the fact that everything presented to the user is treated as an object. Thus, in the e-mail example the user can treat the objects in the

“To:” or “From:” fields as person objects rather than just e-mail addresses. This means they can right click on these objects and perform tasks on these objects. These tasks might include adding these people to an address book or creating a reminder to contact this person.

2.3 Operations

General operations are already a concept that Haystack has used in its implementation. Haystack lets developers create normal operations in an RDF [3] language. The developer specifies parameters for the operation in this language as well as an action to take place with those parameters. We want to extend this concept to include our notion of web operations, allowing users to create and store web operations within Haystack while it is running. Currently, however there is no way for a user to create an operation from within Haystack itself while it is running. Thus, our system must find a way to let users create operations without having to specify them in RDF.

In addition, our system allows web operations to benefit from some of the operation-specific features of Haystack. When a user invokes an operation within Haystack, he is presented with a continuation [9] for that operation. In continuations, the user is presented with inputs for each parameter. This presentation takes place off to the side as an in-progress command. In this manner the user can fill out the necessary parameters while performing other tasks within Haystack. In addition, this in-progress feature of operations allows users to drag and drop objects themselves as parameters. This style of operation presentation is something we hope to utilize when incorporating web operations into Haystack.

2.4 Wrapper Induction

In order for our system to be most useful, our system needs a way to extract information off of web operation results pages. Andrew Hogue’s wrapper induction technique [7] is a

means for extracting information off of web pages and is already implemented within Haystack. Thus, it seems a likely candidate to choose for information extraction.

Hogue's wrapper induction technique makes use of the fact that most records that are found within web pages are structured similarly in terms of their HTML. For example, in a directory on a web site, directory entries may be laid out in a similar fashion. They will all have names, addresses, and phone numbers located in the same place in the HTML structure of each HTML record.

The wrapper induction system implemented in Haystack has the user identify a sample record by highlighting it on the web page. The system then represents that example as an HTML tree. It then searches the rest of the web page for trees that have similar structure using a metric called tree edit distance [10]. Tree edit distance is a metric similar to string edit distance [6]. It is essentially the number of operations (e.g. insertion or deletion) that must take place in order to transform one tree into another. Once similarly structured trees are found, the system assumes that these trees are also records.

One of the benefits of this system is that it allows the user to specify a class for the records within the page. For example, with a web directory, different entries on the page can be treated as people. By highlighting specific nodes within one of the example records, the user can specify different nodes within the tree to be different properties for that class. In the web directory example, they might specify that certain elements represent the name or phone number of a person. In this manner, the system can extract records off of the page, give them a class, and populate that record with certain properties. This feature is very beneficial since it allows more data to be brought into the system.

Chapter 3

User Interface

This thesis plans to discuss a method for learning and storing web operations. This method would not be as effective without a powerful user interface to help implement it. There are three key parts that are needed for this interface: a mechanism to learn and store the web operations, a mechanism to edit and view these web operations, and a mechanism to invoke the stored web operations.

3.1 Storing Web Operations

In our system, in order to store a web operation, the user must first invoke a web operation. After a web operation has been invoked through the embedded web browser, the user can right click and, through a context menu, select to “Remember Web Operation” (see Figure 1). Another option would have been to have the user specify that he wanted to remember the operation on the initiating page. We felt that this might have been a little more complex than having the user choose to remember the operation after invoking it. This is because the user would have to perform an action at the start of the operation and at the end of the operation rather than just at the end of the operation.

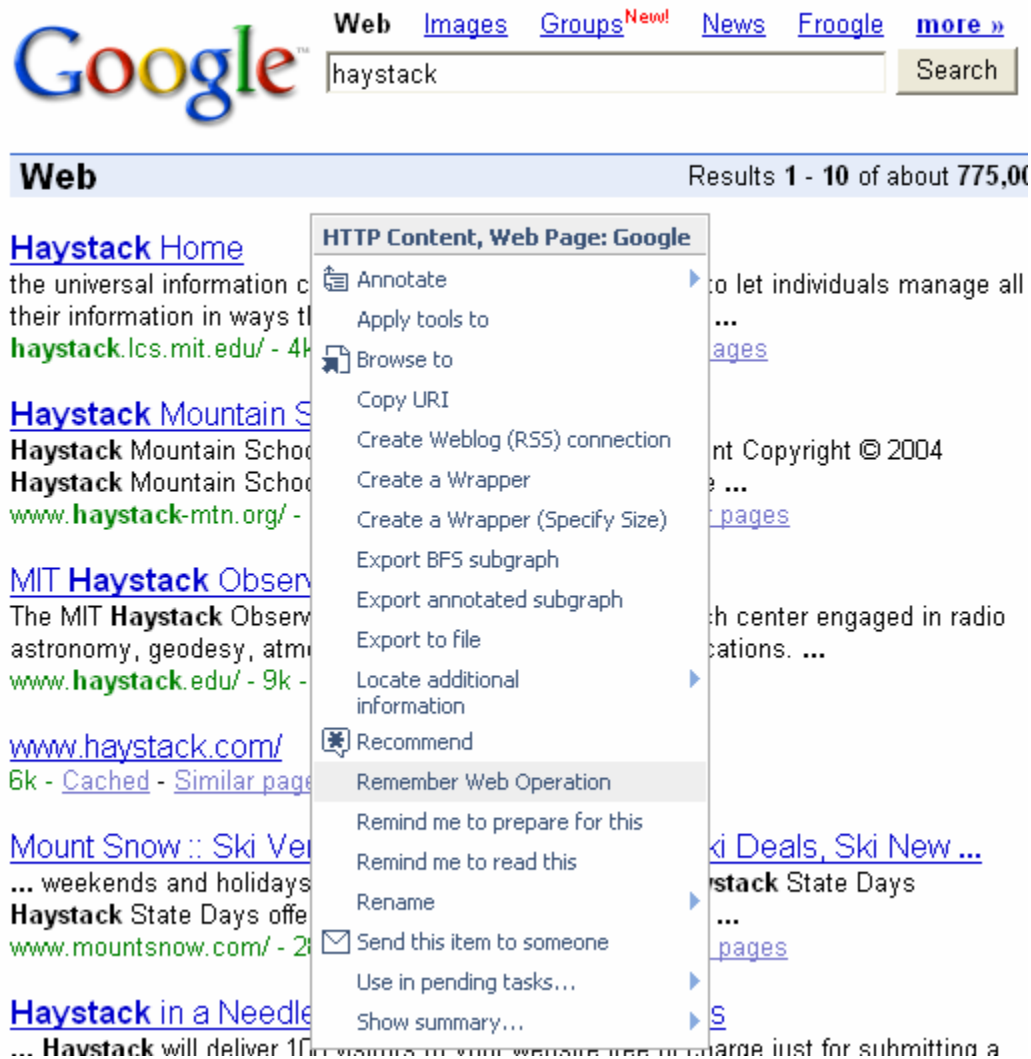


Figure 1: Context menu for remembering web operations. This menu is displayed on a results page after the user submitted a query to Google.

After the user specifies that he wants to remember the operation, a Haystack continuation (see Figure 2) appears off to the side requesting further information from the user. The first piece of information required from the user is a name for the given operation so that the user can find and invoke the operation later.



Figure 2: UI continuation for remembering web operations

The second piece of information is what wrapper to use on the results page when presenting the results to the user. Haystack presents the user with all of the existing wrappers associated with the results page. The user then can select which wrapper he wants from the list. Also, since continuations allow the user to perform other tasks before completing a given continuation, if the user has not created the appropriate wrapper on the results page, he has the option to create one on the spot.

Wrapper creation is specified in greater detail in Hogue’s paper describing his wrapper induction algorithm; however, I will briefly go over the basics. To create a wrapper, the user selects an example record from a certain page, right clicks, and then selects to “Create a Wrapper”. After giving the wrapper a name and a semantic type, the user clicks “OK” and the wrapper is applied to the page. A page that has been wrapped is shown in Figure 3. When a user clicks on one of the records, they can either perform operations with the record itself, the wrapper, or the web page. Since the name of the wrapper is given next to the word “Wrapper:” in this menu, the user can determine the

name of a given wrapper on the page. This is important so that the user can determine which wrapper they want to associate with the current web operation.



Figure 3: A results page wrapped with the wrapper google

3.2 Editing and Viewing Stored Web Operations

After creating the operation, it is important to be able to view and edit the operation. There are several reasons for this, both for the developer's benefit and for the user's benefit. In Figure 4, we see a view of the web operation. The view shows the title of the operation as well as other key properties of the operation. There is also a section that shows all of the web operation parameters. Lastly, at the bottom of the view we have a view of the initiating page for the web operation.

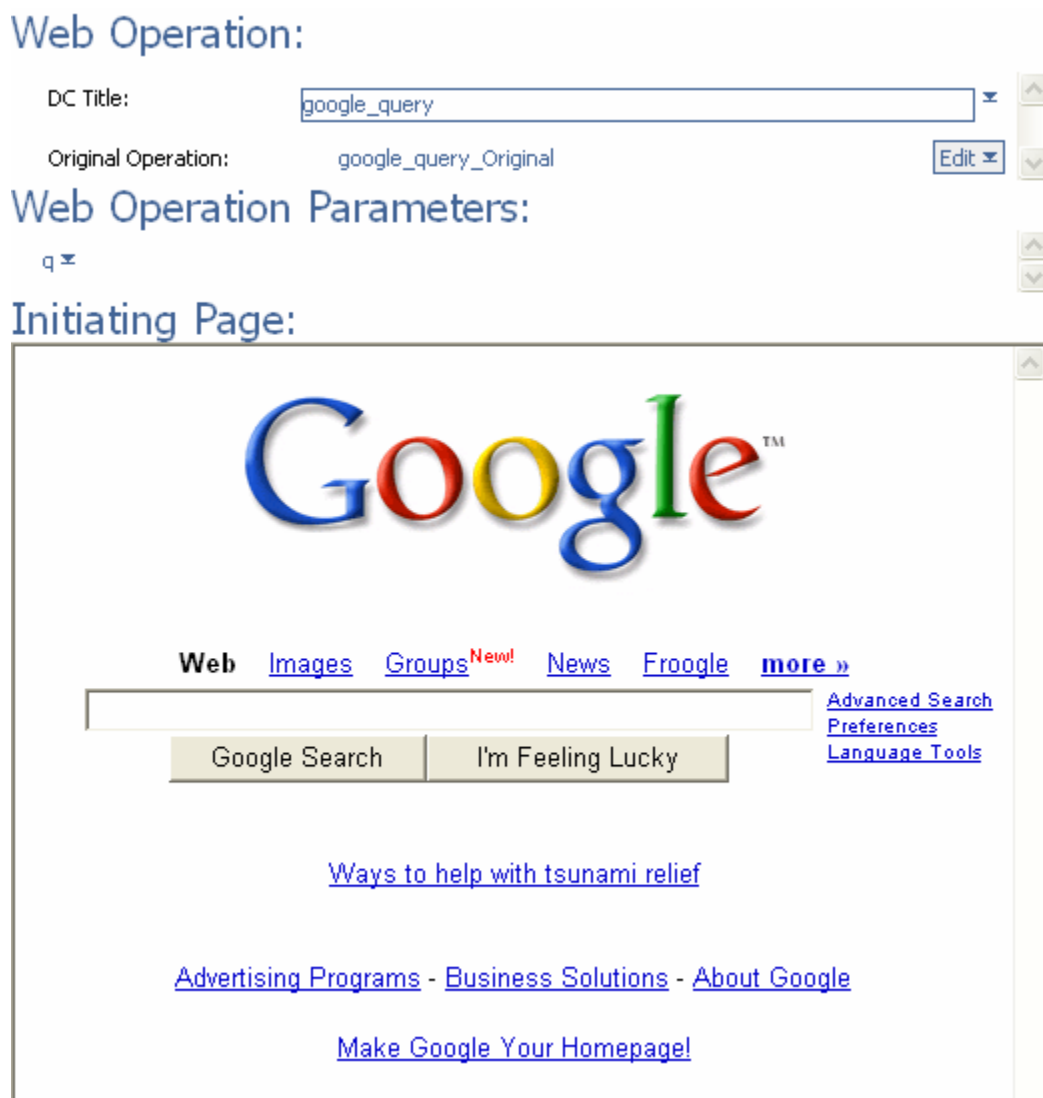


Figure 4: A view of the web operation google_query

First of all, as Figure 4 shows, some parameter names are not human readable. For example, on the Google website, the parameter name for the search box is “q”. For the average user, if they were presented with a form to fill out and “q” was the name of one of the parameters, they might have a hard time figuring out what was supposed to go in that box. For this reason, it is important for a mechanism to be in place for the user to be able to change names for parameters when invoking the operation in the future.

As shown in Figure 4, when first shown to the user, all of the parameters are shown in a collapsed form. This is done so that if there are many parameters, they will be able to fit better in the space provided. This parameter view can be changed to an expanded version by clicking the icon next to the parameter name. An expanded view of a parameter from the advanced search of Google can be seen in Figure 5.

Web Operation Parameters:

Site to search on 	
Display Name:	<input type="text" value="Site to search on"/> 
Original Name:	as_sitesearch
Original Value:	
Parameter Type:	default
Right Text:	e.g. google.com, .org More info SafeSearch
Left Text:	return results from the site or domain

Figure 5: Expanded view of a web operation parameter

In this view, the user is presented with certain properties about the parameter. These properties include a display name, an original name, an original value, and a parameter type. We also have spots for the text that is to the right and left of the parameter on the original web page that contained the operation. This information, combined with the view of the initiating page and the original value of the parameter, will help the user decide a new name for the parameter. As can be seen in Figure 5, if the user changes the display name for the parameter, in this case to “Site to search on”, then the parameter’s title will change. In this manner, the user can select an appropriate human-readable name for a parameter.

3.2.1 Viewing the Operation with Hidden Parameters

Some parameters on web sites are not shown to the user. They have the type “hidden”. Hidden parameters need to be handled differently than normal parameters. They should not be presented to the user but should still have their appropriate values used when invoking this operation. In Haystack there is a related concept that is occasionally used with normal operations. This concept is the notion of currying operations.

In normal operations, when a user has specified values for one or more parameters, the user can choose to save that operation in its partially completed form. The user gives it a different name than the original operation and that new operation is saved as a separate operation. When the user invokes this operation in the future, he is presented only with the parameters that have not been filled out when he saved the original operation.


Our system mimics this behavior with hidden parameters. Once the entire operation has been created with all of the parameters (including the hidden ones), our system then creates another operation that represents the original operation curried with the hidden parameters’ values already filled in. Thus, when the user wants to invoke the operation in the future, he will only be presented with the parameters that are not hidden. The hidden parameters will still be stored and used appropriately when the user submits the operation.


One concern that comes up with this approach is the naming of the two different operations: the original operation and the curried operation. In a normal setting the original operation already exists and has a name, and the user gives the curried operation a name when they curry it. With our system, however, neither operation exists prior to the user creating them. To make things easier on the user, our system only requests that the user specify a single name for the operation. Since the curried operation will more than likely be the operation that is used more often by the user, our system gives that operation the name the user specified. Our system then gives the complete operation the

name that consists of the name specified by the user appended with “_Original”. Thus, these two operations can now be visibly distinguished from each other.

When viewing the web operations, advanced users may want to view the original operation. They may want to view and edit some of the hidden parameters if they understand more about what these parameters are used for. From Figure 4, we can see that the original operation is one of the properties shown to the user. Thus, to view the operation the user can right click and browse to this operation. In our example, if the user does this he is presented with the operation shown in Figure 6. As can be seen in Figure 6, the user now has an additional parameter that was originally hidden from the user. In this view we have already changed a parameter’s name from “q” to “Search”

Web Operation:

DC Title: 

Original Operation: None specified; [click here to add](#) Edit 

Web Operation Parameters:

Search 

hl 

Initiating Page:



Figure 6: View of the complete web operation google_query_Original

3.3 Invoking Stored Web Operations

To invoke a stored operation, the user must first find the appropriate operation. There are numerous ways to do this within Haystack including performing a search for the name of the operation. When the user finds the operation, he can invoke it by simply clicking on the name of it.

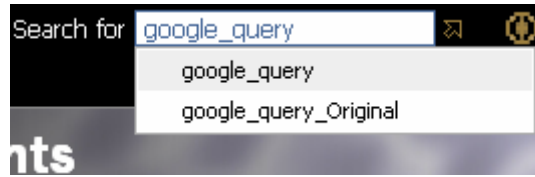


Figure 7: An example search for a web operation

Another way to invoke the operation is possible if the user is currently viewing the operation. If that is the case, the user can right click and select “Perform operation” (see Figure 8). In addition, the user may add the operation to a collection within Haystack so that he can browse to that collection later and either view or perform the operation.

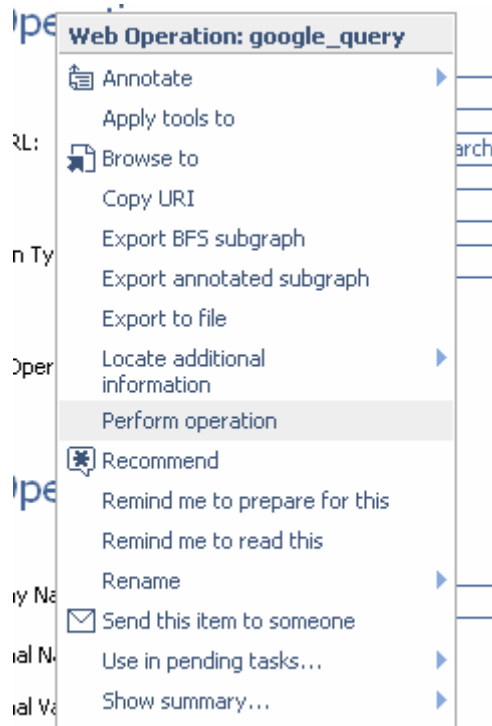


Figure 8: Context menu giving the user the option to "Perform operation"

Once the user invokes the operation he is presented with a continuation with all of the parameters for the operation. A continuation for our “google_query” web operation can be seen in Figure 9. Note how “Search”, our new name for the parameter, is the name presented to the user as opposed to “q”.

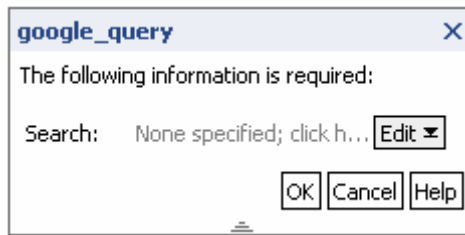


Figure 9: UI continuation for the web operation google_query

Once the user is finished filling out the continuation, he presses “OK” and the system invokes the operation. After the system obtains the results page and applies the wrapper, the system then presents the results to the user. These results can either be a collection, a single resource, or the results page itself if no extractable objects are found. An example results collection is shown in Figure 10.

urn:duMMp11jTSoV1A4			
Change view ▼ Change layout ▼			
Type	Title ▼	Date	
Web Page	MA DOR HomepageThe Official We...	No items in list	
Web Page	Mass.gov - The Official Web Si...	No items in list	
Web Page	Massachusetts Bar AssociationS...	No items in list	
Web Page	Massachusetts College of Artma...	No items in list	
Web Page	Massachusetts Department of Ed...	No items in list	
Web Page	Massachusetts Office of Travel...	No items in list	
Web Page	Massachusetts Registry of Moto...	No items in list	
Web Page	University of Massachusetts Am...	No items in list	
Web Page	University of Massachusetts Bo...	No items in list	

Figure 10: A results collection for a web operation

Chapter 4

Web Operation Characterization

In order to learn and store a web operation, it is first necessary to characterize what a web operation consists of. Typically on the web, an operation consists of inputs on a web page and results on a separate web page.

4.1 Inputs of a Web Operation

The inputs of a given web operation are typically found on a single web page which I will refer to as the initiating web page. A sample initiating web page is shown in Figure 11. An initiating web page can have multiple operations stored on it. To distinguish between operations, web pages use the HTML tag “form” to surround each operation. The attribute “action” specifies which URL to send the inputs to. The attribute “method” specifies which method to use to send the inputs, either post or get. If the form is a post form, there is also an attribute “enctype” which specifies the encoding method for the arguments in the form.

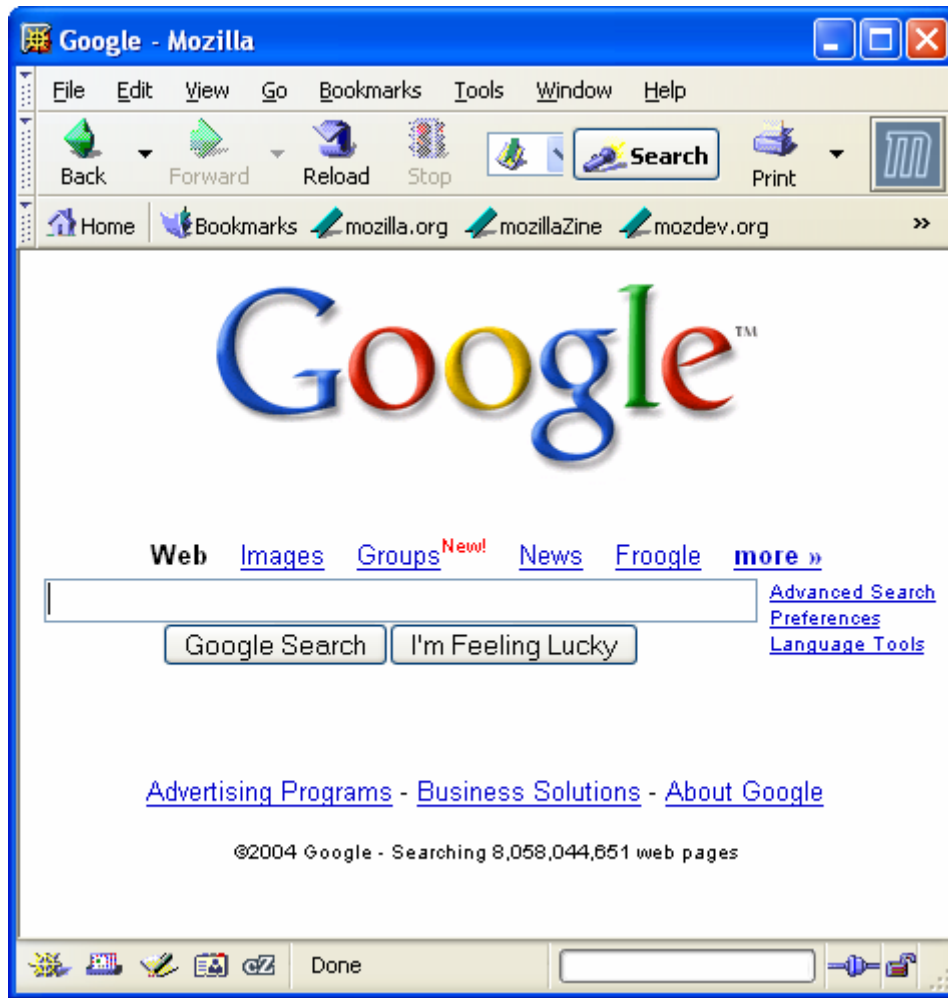


Figure 11: <http://www.google.com/> - an example initiating web page

To specify the actual inputs to the operation, web pages use tags such as “input” and “select.” Inputs can be of several types such as a text box, menu, or radio button. As mentioned earlier, inputs of the type “hidden” are inputs whose values will get sent to the server but are not visible to the user on the web site. A special input of type “submit” produces a button which, when pressed, submits all of the inputs to the URL specified by the “action” attribute of the surrounding “form.” Below is an example of a form with inputs:

```
<form method="get" action="/search">
  <input name=query value="">
  <input name=name value="">
  <input type=submit value="Search">
</form>
```

When the user presses the submit button, the website constructs a string consisting of all of the input names and values. Each input name and value pair is constructed in the following pattern: *name=value*. The pairs are separated by the character ‘&.’ Thus, if the user enters “haystack” and “Ryan” in the above sample form and presses submit, the following string is created: *query=haystack&name=Ryan*. If the “method” attribute is set to get, then the browser takes the link specified by the “action” attribute appends a ‘?’ and the string just created and then navigates to the resulting string just as if it were a normal link. If the “method” attribute is set to post, then the browser sends the string of input names and values to the server via an HTTP post transaction.

4.2 Outputs of a Web Operation

The outputs of a web operation are typically found on one web page which from now on I will refer to as the results page. An example results page is shown in Figure 12. As mentioned earlier, the information on a results page is typically laid out in a standardized fashion for each given operation. This information can be thought of as one or more records representing certain classes of data such as a person, a travel itinerary, or a search result. Each of these records is typically structured similarly. For example, a person record will have information like name, address, and phone number located in a certain place in the hierarchy of each record within the web page.

Web

[Haystack Home](#)

Haystack is a tool designed to let every individual manage all of their information in the way that makes the most sense to them. ...

[haystack.lcs.mit.edu/](#) - 4k - [Cached](#) - [Similar pages](#)

[Haystack Mountain School of Crafts](#)

Haystack Mountain School of Crafts, ... Us. Site Map. Content Copyright © 2004 **Haystack** Mountain School of Crafts. What's Going On... The ...

[www.haystack-mtn.org/](#) - 7k - Dec 8, 2004 - [Cached](#) - [Similar pages](#)

[MIT Haystack Observatory](#)

The MIT **Haystack** Observatory is an interdisciplinary research center engaged in radio astronomy, geodesy, atmospheric sciences, and radar applications. ...

[www.haystack.edu/](#) - 9k - Dec 8, 2004 - [Cached](#) - [Similar pages](#)

[www.haystack.com/](#)

6k - [Cached](#) - [Similar pages](#)

[Mount Snow :: Ski Vermont's All-Season Resort, Ski Deals, Ski New ...](#)

Welcome to Mount Snow, Vermont home of five Mountain faces, extensive tree areas and top rated parks and pipes. Save money with our vermont ski vacation deals.

[www.mountsnow.com/](#) - 27k - Dec 8, 2004 - [Cached](#) - [Similar pages](#)

[Haystack in a Needle - Web Marketing Consultants](#)

... **Haystack** will deliver 100 visitors to your website free of charge just for submitting a quote request by December 7th, 2004! New Article: Overture vs. ...

[www.haystackinaneedle.com/](#) - 19k - Dec 8, 2004 - [Cached](#) - [Similar pages](#)

[The Haystack Group, Inc. ~ Retained Executive Search ~ Vinalhaven ...](#)

Welcome. to the cyberhome of The **Haystack** Group, Inc. ... Please forward resumes or inquiries to the following: John A. Gasbarre, President The **Haystack** Group, Inc. ...

[www.haystack-group.com/](#) - 18k - [Cached](#) - [Similar pages](#)

Figure 12: An example results page

Chapter 5

System Design

5.1 Learning a Web Operation

In order to learn a web operation, there are a couple of different techniques that could be used. One technique is to watch the network traffic associated with the browser until the system detects a sample get or post associated with the given operation. With this technique, the system can gather just enough information to recreate the operation by resending a post or a get. It can gather parameter names, the type of operation, and the action URL. In addition, the system can gather information like sample parameter values. As mentioned in the UI section, these values may be beneficial to users when they try to rename parameters.

Another technique is to learn the web operation by extracting information from the HTML of the initiating page. Again, with this technique, the system can gather enough information to recreate the operation. It can also gather information that may be beneficial to the user when trying to rename parameters or when they invoke it again. This information might include surrounding text, ranges for certain parameters, or types of certain parameters.

There are pros and cons to each technique of web operation learning. In both cases, the system can gather all information it needs to recreate the operation. With the get/post listening method, however, the user may have problems performing the operation again. This is because the parameter names it acquires from the post or get may not be human readable and thus, the user may not realize what the parameters are for. While the user

may be able to tell what a parameter is used for by looking at sample parameter values, this may not be enough information to completely distinguish what a parameter is used for.

With the HTML extraction method, this problem may be fixed by giving the user access to some of the text surrounding the input. However, with the HTML extraction method, there is an added difficulty of specifying which operation the user wants to learn on a given page. In addition, if JavaScript is used to manipulate the form on the page while the user is filling it out or if it is used in the actual submission of the form, the system may have extracted incorrect information about the operation by just looking at the parameters in the HTML of the page. With the get/post listening method, the system knows exactly what operation and what parameters the user is using in the given invocation of the operation. Thus, even if JavaScript has manipulated the HTML on the initiating page or been used to submit the form, the system still has an idea of what parameters are used when the actual submission takes place.

Thus, both methods of operation learning are lacking the ability to learn different kinds of information about web operations. In an effort to acquire as much information as possible about web operations, our system observes both the get or post and the HTML of the initiating page.

5.1.1 Listening to Posts and Gets

To learn information about an operation, our system observes the post or get that is associated with it. With that information, the system can break up the post or get and determine the parameters for the operation as well as sample values for those parameters.

Posts are relatively easy to listen to. Just after the user submits a post form on a web page, the web browser sends the post data to the server. Most web browsers have an event that fires just before navigating to a web page, and this event contains any post data

that is being sent. For example, the event BeforeNavigate2 is used in Internet Explorer to capture this data. Our system makes use of this event and listens for posts in this manner.

Gets, on the other hand, are not quite as straightforward to listen to. Unlike posts, gets are sent to the server just like a normal URL. Thus, it can potentially be hard to distinguish a normal URL from a get request to the server. At first, one might think that it would be possible to determine a get request based on the structure of the URL. Since gets contain a '?' followed by a sequence of parameter assignments, it might be possible to distinguish normal URLs from gets based on this structure.

However, there is a problem with this strategy. Some websites will send requests to the server that look like gets in order to retrieve ads. Thus, it is not possible to determine a get request resulting from a form submittal just by the structure of the URL. Ad retrievals, however, are initiated after the main page has been requested. Thus, to detect if a URL is likely to be a get request, our system first checks to see if it is structured like a get request and then checks to see if it is the first URL requested after a document has completely loaded. The document that has been completely loaded can now be thought of as the initiating page.

5.1.2 Learning Parameters of the Web Operation

Once our system has listened to the post or get associated with a given web operation, the next step in learning the operation is to determine what parameters are used in the web operation. With the information from a sample get or post, it is relatively simple to acquire a list of parameters. The necessary information is contained within the query string for a web operation. The query string is located either within the post data field in the post transaction or the part of the URL after the '?' in a get request.

Once this string is obtained, the application can then break up the string into parameter name and value pairs since each pair is separated by a '&'. Once this is done, the application can obtain the parameter names since they come before an '=' in the

parameter name and value pairs. Our system also remembers all of the values that the user submitted in the sample operation invocation. These values can give the user an idea of what types of values are valid for certain inputs when they invoke the operation later on.

5.1.3 Acquiring Information From the HTML

In order to make an operation invocation easier for the user, our system provides supplemental information about some of the parameters in the operation. For example, some websites use other forms of input than just a normal text box like a checkbox or menu. Then based on the user's selection, a certain string is used as the appropriate submitted value to the server. The user is typically unaware of these values and may not be able to guess an appropriate value for these kinds of input. As an example of this, consider a menu input for a color that has three values: red, blue, and green. These values are displayed to the user in the drop down menu. However, the values that are submitted for these values might be r, b, and g. If users were presented with a normal text box to input a color, they might not know to enter r, b, or g instead of red, blue, or green.

As mentioned earlier, however, in addition to the post or get, our system uses the HTML from the initiating page to help determine information about parameters that have ranges of inputs that the user can choose from. Typically, the HTML contains the appropriate submit values associated with each possible variant of these inputs. Thus, an application can find all of the ranged inputs for a given operation in the HTML of the originating page, and from that HTML, the application can determine appropriate ranges for those inputs.

In order to get information about parameters from the operation, our system first must determine which operation is the correct operation within the originating page. This is not trivial since occasionally there are multiple operations on the same page. One might think at first that it is possible to determine which operation on the page is correct by

simply looking at the “action” attribute of the “form” tag. It is not that easy, however, since some sites will use the same “action” URL for multiple operations on the same page. For example, the Google advanced search page has 3 out of 4 operations that use “/search” as their “action” URL.

Another possible method that one might consider is to determine the web operation by detecting which submit button was clicked to submit the operation. The system could then determine which form the submit button was contained within and use that form’s HTML to determine information about parameters. Unfortunately, the web browser that we are embedding within Haystack does not allow detection of these events in this manner.

Thus, in order to accurately identify the correct operation, we use a more complex process. First, from within the initiating web page, we first identify all of the forms that contain the correct “action” URL. Then we go through all of the inputs for each operation and see which operation has all of the inputs from the get or post that we listened to earlier. Once we find an operation that meets both of the above criteria, we can be relatively certain we have the correct operation. It is conceivable, however, that there could be two or more operations that meet all of the above criteria. For example, there could be two forms on the same page that use the same “action” URL and have the same names for all of the parameters on the page. We choose to ignore cases like these, however, because HTML developers tend not to do this, because it will make their job harder on the server side.

Once the appropriate web operation is found on the initiating page, our system proceeds to extract important information about the parameters for that operation. The first bit of information that we deduce is the parameter’s type. Currently, the only types that our system gives special attention to are “hidden”, “select”, “radio”, and “checkbox” inputs.

Inputs of type “hidden” have their “type” attribute set to “hidden” and are not visible to the user on the web page. As mentioned earlier, we want to have a similar functionality

for these inputs in our system. Thus, we remember that these inputs are “hidden” and we remember their value so that we know what value to send to the server. We also leave the option open for the user to change some of these “hidden” values if the web site changes something or if an advanced user has knowledge about how these “hidden” values are used.

Inputs of type “select” have “select” as their tag name. They are drop-down menus that are presented to the user. Each of the possible values in the menu have a display value that is shown to the user and a submit value which is submitted to the server and unknown to the user. Since the submit values are unknown to the user, our system must present a similar type of interface where we have display values shown to the user which represent submit values that are sent to the server. Thus, for “select” inputs, we must learn the range of display values and the submit values that they represent. A similar method will be used for radio buttons.

“radio” inputs have their “type” attribute set to “radio” and are treated similarly to “select” inputs. When an input of type “radio” is present on a page, there is typically a group of radio inputs that all have the same “name” attribute. Each input has some text that is visible next to the radio button and a “value” that is submitted to the server if that button is selected when the user presses submit. Thus, to help the user in future invocations of the operation, we store the range of text values that are visible to the user and the submit values they represent. This means that from a Haystack data model perspective, a “radio” input is treated in the same way as a “select” input. They are an enumerated type consisting of display values and submit values that those display values represent. If a radio input is not selected during the sample invocation of the operation, the name of that input will not appear in the post or get. Thus, when searching the HTML, if a radio input is found that has a name that is not identified in the list of parameters learned from the get or post, our system adds that input and all of its possible values to the list of parameters.

“checkbox” inputs have their “type” attribute set to “checkbox” and only submit a name/value pair if the checkbox is checked. Thus, “checkbox” inputs must be handled somewhat cleverly. Like “radio” inputs, if the checkbox was not checked on the sample invocation of the operation, then the checkbox’s name will not be available in the post or get that our system observed. Thus, as with “radio” inputs, when the system is searching HTML and discovers a “checkbox” input whose name does not appear in the list of parameters it knows about, it will add that input and identify it as having a parameter type “checkbox”. The name for that input is obtained through the HTML and the value that is submitted when the box is checked is either the “value” attribute for the input or “on” if no “value” attribute is specified. Appropriate display values for “checkbox” parameters may possibly be extracted from the text that is directly next to the checkbox. Our system takes this additional step for checkboxes because the user might not be able to tell what this input is for otherwise. This is because both the name for the input and the original value for the input may not be intelligible to the user.

In addition to the information mentioned above, our system also makes an attempt to gather the surrounding text for all of the parameters. We look at the nodes that are directly to the left and directly to the right of the input in order to get this text. This text is used later to help the user identify what the parameter is used for in the operation. The nodes that are directly to the left and right of the input may not always contain text but in general we believed that this would be a good method to help determine the purpose of parameters.

5.1.4 Learning to Format the Results

As mentioned earlier, we use Haystack’s wrapper induction method to extract results off of a results page. In our system, a user uses this tool to specify a record on a results page of a sample invocation of an operation. When a user invokes this operation in the future, the system can then pick out the results through wrapper induction.

5.2 Storing a Web Operation

When storing a web operation, our system needs to store enough information to recreate the operation in the future. This information includes information about parameters, how to invoke the operation, and what to do with the results.

5.2.1 Storing Parameters

The information about parameters that our system stores includes all of the information needed to send off queries to the server and any appropriate supplemental information determined from the HTML or post or get that will be used to help the user in future invocations of the operation.

Clearly the names used by the web operation in the post or get are necessary to invoke this operation again. This name is discovered by our system in the get or post query string in most cases. In the case of a checkbox or a radio button that the user did not use in the sample invocation of the operation, our system uses the name found for the input in the HTML since the name will not be included in the query string as mentioned in section 5.1.3.

Also, later in our system, a user can specify a more appropriate name to display to the user when invoking the operation again since the names of inputs on a given page may not be completely intelligible. For this reason, it is a good idea to store the original value given to the input and the surrounding text so that the user may know what a typical input is for this parameter when specifying a new name for the parameter.

Also, supplemental data about the parameter should be stored. The input's type should be stored. In our system, this includes "hidden" for hidden inputs, "checkbox" for checkbox inputs, and "select" for both radio and select inputs. We use select as the type for both radio and select inputs since they will be treated similarly when invoking the operation in the future. Thus, they are the same type as far as our system is concerned.

Any other type will fall under the “default” type and will be treated as a text box parameter.

Other supplemental data that is stored concerns the “checkbox” and “select” input types. For “checkbox” inputs, our system stores the value to send to the server if the checkbox is checked. For “select” inputs, our system stores pairings of select values presented to the user and submit values that will be submitted to the server.

5.2.2 Storing How to Invoke the Operation

It is also necessary to store information about how to invoke the operation again. First, this information includes the appropriate URL to send a post or get request to. This can be determined by obtaining either the portion of a get request preceding a ‘?’ or the entire URL to which a post is sent.

Also, clearly it is necessary to store whether the operation is a post or a get. This information is necessary to specify how the operation will be invoked in the future. Whether or not an operation is a get or post can be determined when the operation is first observed which is described in more detail in section 5.1.1. If the operation is a post operation, then it is also necessary to keep track of the attribute “enctype” for the operation which specifies the encoding type that will be specified in the headers that will be sent to the server in the post transaction.

5.2.3 Storing Result Identification

As mentioned earlier, when repeating this operation, the application needs to know how to identify a record on a results page. Haystack has a method for storing the wrapper induction method used to identify records on a web page so this is what our system uses for a storage mechanism to let the application know how to identify a record in future invocations.

5.3 Invoking a Stored Web Operation

To invoke a stored operation, our application will take inputs from the user and reconstruct an appropriate string using input names and values. This string will then be sent to the appropriate address in the manner of either a get or post. Then, the results will be extracted from the results page and displayed to the user.

5.3.1 Constructing the String

When a user invokes a web operation, he will be presented with a list of parameters. He will then specify values for all of those parameters. For normal parameters, our application will just use the value specified when constructing the string to send to the server.

If the parameter is a “checkbox”, then our system will behave similarly to the way that normal web operations behave. If the checkbox is checked then our system will use the value that was specified in the HTML as the submit value which our system learned earlier. If the checkbox is not checked then the system will not use the parameter at all.

If the parameter is either a “radio” or “select” input, then our system will need to determine the correct value to submit based on the value that the user selects. The system accomplishes this by looking through the select/submit pairings that the system learned from the HTML of the initiating page. Once it finds the pairing that contains the select value that the user has selected, it determines the submit value that matches the appropriate select value.

If the parameter is a “hidden” input, then the user will not have specified a value for this input because this parameter will not have been presented to the user. In this case, our system simply uses the value that was either extracted from the HTML or extracted from the post or get.

Once our system has presented the parameters to the user and determined the appropriate values for all of the parameters, it constructs a string of the form:

$$name_1=value_1\&name_2=value_2\&\dots\&name_n=value_n$$

Here, the names represent the names of the inputs in the HTML and not the names necessarily visible to the user and the values are the values determined by the system given the user's inputs.

5.3.2 Sending the String to the Appropriate Address

The next step is to send the constructed string to an appropriate address. If the operation involves a get, the application should append the constructed string to the URL stored in section 5.2.2 with a '?' on the end. Then, the application should browse to the resulting string just like any other URL.

If the operation is a post, the application should take the string constructed in section 5.3.1 and construct a post transaction with it. The URL stored in section 5.2.2 will be the URL to which the transaction is sent. The post data is the string constructed in section 5.3.1.

5.3.3 Gathering the Results

As mentioned earlier, once the get or post is sent to the server, the application should apply the stored wrapper to the results page. Once the wrapper is applied, the system can determine what to display to the user based on how many records are found on the page. If one record is found then that record should be displayed to the user. If multiple records are found, then they should be added to a collection and displayed to the user. Lastly, if no records are found or no wrapper is specified, then the user should be navigated to the results page. That way the user can be presented with information that might be on the results page to indicate what might have gone wrong with the given web operation.

Chapter 6

System Implementation and Architecture

6.1 Listening for Web Operations

A key feature of our system is its ability to tell when a user has performed a web operation. Without this ability the system would not be able to begin the process of remembering the operation for future invocations.

As mentioned earlier, our system currently only works with the Internet Explorer browser that is embedded within the Windows version of Haystack. Internet Explorer uses an event called the BeforeNavigate2 event to specify any information that it will send to the server before it actually performs the navigation with that information. The information specified in this event that is of interest to our system includes the URL for the page being navigated to, any additional headers that may be necessary for some post transactions, and post data if the operation being performed is a post transaction.

In Haystack, our system taps directly into this event through the Internet Explorer browser. Our system, then, looks at each BeforeNavigate2 event until it sees one that represents a web operation. As mentioned earlier, this is accomplished by looking at the post data for post operations or the URL for get operations.

Whenever our system sees what it thinks is a web operation, it saves state for that operation in case the user decides to remember it in the future. Since some of the processing that is involved on determining parameter names and values from the query

string may be time consuming if the operation involves many parameters, our system only does this processing if the user specifies that he wants to remember this operation.

Thus, our system needs to keep track of all of the state that is necessary to remember the operation should the user decide to in the future. This data includes all of the pertinent information in the BeforeNavigate2 event like the URL, headers, and post data. Another important thing to save is the form in the HTML of the initiating page that represents the current operation. This is necessary in order to capture all of the supplemental information, described in section 5.1.3, which might not be available in the URL or post data. Since none of the operation processing that determines parameter names and the action URL takes place until the user specifies that he wants to remember the operation, our system has no way of knowing which form on the initiating page is the form that represents the operation that was just invoked. To deal with this, our system stores all of the forms on the initiating page as DOM tree structures until the user specifies that he wants to remember the operation.

6.2 Remembering the Operation

With all of this information stored, our system can now remember and store the specific operation that the user has just invoked whenever the user wants to. Once the user specifies that he wants to remember the operation, our system takes the information just stored while listening to this operation and converts it to a structure that represents one specific operation.

In order to create the structure that represents the operation, our system uses many of the techniques described in chapter 5 to extract information from the state stored in section 6.1. First, the system determines if the operation is a get or a post by determining if there is any post data. If there is, then the operation is a post and if not, then the operation is a get. Next, the system extracts the action URL from the URL stored in section 6.1. Then, the system extracts all of the parameter names and values from either the URL or the post data. With this information, the system can now go through all of the forms that are

stored in section 6.1 and determine which form represents the current operation. Once that form is determined, the system then extracts all of the supplemental information about the operation's parameters.

6.3 Storing the Operation Within Haystack

Once the system has gathered all of the information about the current operation, the system stores the operation in Haystack's database. Since one of the goals of the system is to treat web operations like Haystack operations, we chose to store the web operations as Haystack operations.

Normally, a user specifies operations prior to Haystack running. The user specifies these operations by specifying parameter names for the operation and a body of code that specifies what actions should be performed when the operation is invoked. Since the web operations and parameters have certain properties like action URLs and original parameter values that are unique to web operations, we decided to subclass operations and parameters so that they can be treated specially.

The Haystack schema for operations specifies that parameters are properties for operations. We will use this similar idea. In extending operations to web operations, we add properties that include the action URL, headers if it is a post operation, the operation's type, the initiating web page, and the wrapper that will be used to extract records off of the results page. We also specify that web operations have web operation parameters as properties instead of normal operation parameters. The schema for normal parameters in Haystack has properties like whether or not the parameter is optional. We extend these parameters to web operation parameters by adding properties like parameter type, original value, original name, surrounding text, display name, and a possible range of display and submit values if it is a select parameter.

As for creating the operation, we needed a way to specify a web operation on the fly. When we want to store a recently learned operation in Haystack's database we first create

a Haystack web operation resource. We then specify all of the web operation parameters for that operation. In both cases we include all of the appropriate properties for the web operation and web operation parameters.

When determining the body of the operation, we had to create a generic method body that would represent any web operation that might be stored. To do so, in the method body, we first query Haystack for all of the parameters of the given operation. Most operations within Haystack do not need to explicitly query for all of the parameters in their operation body, because they are able to define variables that are used to represent all of the parameters within the operation body. Since we need to have an operation body that represents all possible web operations and different web operations will have different numbers of parameters, we have to query to dynamically determine the parameters for the given operation. Once we have the parameters we can go about invoking the operation. This will be discussed more in section 6.4.

6.4 Invoking the Operation

As mentioned earlier, when a user wants to invoke a web operation within Haystack they first fill in values for that web operation. Our system queries and then iterates through each of the parameters and reconstructs a query string with those parameter values.

Some processing may be necessary for some of the parameter values depending on the parameter type. Parameters of type “default” do not need any type based processing since the values that users enter are valid values for those parameters.

Parameters of type “checkbox” have their value set to “true” if the box is checked and nothing if the box is not checked. So for “checkbox” values our system will use the value that it determined from the HTML if the value is set to “true”. It will ignore the parameter entirely if the parameter has no value.

“select” parameters will have their value set to whatever value the user selected from the drop down menu. As mentioned earlier, this value is not a valid value in terms of what the server will be able to recognize. Thus, for “select” parameters our system searches through the display/submit value pairs associated with that parameter to find the one that matches the value the user selected and uses this value in the query string.

One more piece of processing needs to take place on parameter values. Parameter values need to be “safe” in terms of the characters that they contain. As such, it is incorrect for parameter values to contain characters like spaces, semicolons, or ampersands. Thus, the parameter values need to be canonized in order to make them “safe” before sending them to the server. This process involves changing any “unsafe” character into a percent sign followed by the character’s ASCII value. Thus, for example, a semicolon changes to “%3B” and an ampersand changes to “%26”. As a special case, spaces can be changed to plus signs in addition to their value of “%20”.

Once the query string is actually constructed it is time for the system to actually perform the operation. In order to do this, our system essentially fires a BeforeNavigate2 event on an instance of the embedded browser. One thing to note about this process is that the embedded browser that we use is not visible to the user.

If the operation is a “get”, the system places the query string after the action URL and a “?” to create the URL for the event. Otherwise, the action URL is used by itself. Any headers that were stored when listening to the sample operation are placed in the header field. Finally, if the operation is a “post”, then the system uses the query string as the post data.

Once we fire this event, the request gets sent to the server and a document is received. The only way to tell that this document has been received is to subscribe to the DocumentComplete2 event for the browser. Thus, we subscribe to this event, and when the document is complete we begin the process of extracting results off of the document that we just received.

In order to extract the results off of the results page, we make use of some of the code for Hogue’s wrapper induction. First, we take the wrapper that the user specified when learning the operation and apply that wrapper to the results page. This process yields a set of matching records. Then, for every record that is returned we use Hogue’s wrapper system to begin the process of creating Haystack resources for those records and adding those resources into Haystack. First, the system creates a generic resource. Then, the system assigns that resource the type that is given by the semantic type associated with the stored wrapper. Then, based on any properties of the class that are specified within the wrapper, the system creates a set of statements to invoke on the just created resource to add the different properties that are specified by the wrapper on the results page.

Next, Haystack browses to the resulting record or records. If there is a single record this is straightforward. Our system simply navigates to that record. If there are multiple records however, we take each record created and add it to a collection. Then, we browse to that collection. Lastly, if no records are found or if the user has not specified a wrapper, we show the results page to the user.

6.5 Improvements to Operations

In order for our system to work more effectively, we needed to make several changes to the operation type within Haystack. One of those changes had to do with currying operations. Previously when a user curried an operation within Haystack, the resulting operation existed as an entirely new and separate operation with no way to “uncurry” or in other words, access the original operation. This could create some problems for advanced users of our system if they decided that they needed to change some of the “hidden” parameter options. Since our system essentially curries those “hidden” parameters, there would be no way for the user to access them.

To fix this problem, we created a new property for operations: the “Original Operation” property. The system populates this property with the current operation when a user

curries an operation. Here is an example to illustrate how this helps a user. Suppose a user curries an operation with three parameters by first currying one parameter to create operation A and then currying another parameter to create operation B. The user can then access the originating operation from operation B by first accessing the “Original Operation” property of operation B to get operation A and then accessing the “Original Operation” property of operation A.

Another important feature that we added to Haystack’s operation type has to do with how operations are presented to the user. Before our system was implemented, there were only two ways to specify how parameters would be presented to the user. The user could either specify that a parameter was unique or not. If the parameter is a unique parameter then only one value can be entered for that parameter. Otherwise, the user can add as many values for the parameter as he wants to. The unique parameter option is what we wanted to use for textbox parameters. This option will essentially create a textbox for that parameter and only allow the user to specify one value for the parameter.

The two existing display options, however, were not powerful enough to encompass all of the parameter types that are available for web operations. We needed a way for Haystack to present “checkbox” or “select” parameters. Thus, we added this functionality to Haystack.

We decided that the best way to have Haystack recognize that a parameter should be viewed as a checkbox would be to have the range of the parameter be a boolean. The range of a parameter determines what values are possible for the parameter. Thus, it makes sense that a checkbox interface most adequately matches a parameter whose possible values are either “true” or “false”.

In our system, when a user specifies that a parameter has a range consisting of boolean values, that parameter will be presented to an operation invoker as a checkbox. When the box is checked and the user submits the operation, the value for that parameter in the

method body will be set to “true”. Otherwise the value will be nonexistent. An example of what a checkbox parameter may look like in a continuation is seen in Figure 13.

The following information is required:

English: ☐

Figure 13: An example checkbox parameter

“select” parameters are a little more difficult for Haystack to recognize. We decided to create a new property for operation parameters that could specify possible values for that parameter. We use this attribute of the parameter to distinguish that a given parameter is a “select” parameter. Since “select” parameters will have a finite number of possible display value possibilities, we decided that the developer can specify those possibilities in the possible values field in order to specify that a parameter is a “select” parameter. Another possibility for specifying “select” parameters would have been to set the range to the possible values for that parameter. We did not want to use this method however because we also wanted to have a notion of range for these parameters in addition to the possible values field. In that way, the user can specify a value for the parameter that is in the range but may not necessarily be in the possible values for the parameter.

There are a couple of ways to specify the possible display values. The first and most straightforward way is to specify the values in the form of a list. The other way to specify these values would be to specify a data source. Data sources are used in Haystack to tie a set of data to some manipulation of resources that are currently within Haystack. For example, a query data source is way of specifying a set of data that is the current result of a query within Haystack. Thus, if the user specifies that the range of a parameter is either a list or a data source, the system would know to present that parameter to the user with a drop down menu mechanism. Clearly, for our system, the data source method of specifying “select” parameters is not as useful as the “list” method but we added both methods to Haystack since it made logical sense. An example of what a select parameter may look like in a continuation is seen in Figure 14. As of right now, radio buttons do not exist as an input method within Haystack. In the future, when they

are available, it might be beneficial to give the user or developer the choice to specify that they want to have a “select” parameter displayed as either a radio button or a drop down menu.

The image shows a web form interface. At the top, there is a text label 'Wrapper: No items in list' followed by a button labeled 'Select...' with a small upward-pointing arrow icon. Below this, the text 'Select Value:' is displayed. Underneath, a dropdown menu is shown with the word 'google' selected. To the right of the dropdown menu is a vertical scrollbar. At the bottom of the form, there is a button labeled 'Add Value'.

Figure 14: An example select parameter

Chapter 7

Testing Results

7.1 Web Operation Survey

In order to test the system, we needed to come up with a set of web sites that contained operations that we could try to learn. When choosing these sites we tried to keep several things in mind. First, we wanted to make sure that we had representative set of sites that tested all of the features of our system.

Also, since Hogue's wrapper induction is an integral part of our system, we used many sites that that we knew for sure worked well with that system. However, if wrappers did not work perfectly on the site, we did not count that as a total failure. There are several reasons for this. First, since the main part of our system is learning web operations and performing them again later, we can still test those parts of the system without necessarily worrying about whether or not records are extracted. Also, since the system can work without wrapper induction by just returning the results page rather than extracted records, it is not a not a complete failure if the sites we deal with do not handle the wrapper induction technique very well. Lastly, as long as certain APIs of the wrapper induction system are not changed, improvements can be made in the future to the wrapper induction technique and these improvements will help the record extraction side of our system as well.

A table of sites we tested, as well as pertinent information about why we chose those sites, is located in Appendix A. The results for those tests and presumed reasons for any failures are included in Appendix B.

7.2 System Successes

In general our system performs fairly well. With simple queries on sites that wrapper induction works properly, our system works as we expected it to. The above chapters have all dealt with an example of such a site.

More complex queries work as well. For example, the form on Amazon.com shown in Figure 15 has two inputs, one of which is a select input.


A screenshot of the Amazon.com search form. It features a blue header with the word "SEARCH" in white. Below the header is a white search input field containing the text "All Products" and a blue downward arrow. To the right of the input field is a red circular button with the text "GO!". Below the search field is a yellow banner with the text "FREE Super Saver Shipping on orders over \$25!" and a blue link that says "Restrictions apply".

Figure 15: Form from the web site <http://www.amazon.com>

In Figure 16, we see that this operation has been successfully learned. When the user tries to perform the operation again he is presented with two inputs. The select input correctly displays all of the values in the range. Also, this operation is a “Post” operation so it is beneficial for testing that part of our system. When the user presses “OK”, we are presented with the appropriate search results. Thus, we can see that these aspects of our system are working properly.

The screenshot shows a dialog box titled "amazon" with a close button (X) in the top right corner. Inside the dialog, the text "The following information is required:" is displayed. Below this, there is a "Search:" label followed by a text input field. Underneath the search field, there is a "Type:" label, the text "No items in list", and a "Select..." button. Below the "Select..." button, the text "Select Value:" is displayed. Underneath this, there is a list of categories with checkboxes: "All Products", "Apparel", "Arts & Hobbies", "Auctions", "Automotive", "Baby", "Beauty Beta", and "Books". To the right of this list is a vertical scrollbar. Below the list, there is an "Add Value" button. At the bottom of the dialog, there are three buttons: "OK", "Cancel", and "Help".

Figure 16: Invocation of an Amazon.com query

As an example with many parameters we next look at the advanced Google Search. Here, we had to perform a lot of cross referencing with the initiating page in order to rename the parameters. We did this by looking at the right and left text for each parameter and attempting to match it up with the text surrounding the inputs on the initiating page. After being reasonably sure about the parameter's use on the page, we could rename it. An idea to make this process easier is discussed in section 8.2.8. Once we successfully renamed the parameters and then invoke the operation we are presented with the continuation in Figure 17. After submitting the query with various combinations of parameters, we were reasonably certain that all aspects of this operation were working correctly.

google_advanced

The following information is required:

all of the words:

at least one of the words:

containing a number over:

containing a number under:

exact phrase:

number of results:

No items in list

Select...

Select Value:

10 results

100 results

20 results

30 results

50 results

Add Value

pages written:

No items in list

Select...

Select Value:

Chinese (Simplified)

Chinese (Traditional)

Croatian

Czech

Danish

Dutch

English

Estonian

Add Value

results from site or domain:

No items in list

Select...

Select Value:

Don't

Only

Add Value

results of the file format:

No items in list

Select...

Select Value:

Adobe Acrobat PDF

Adobe Postscript (.eps)

Microsoft Excel (.xls)

Microsoft Powerpoint (.ppt)

Microsoft Word (.doc)

Rich Text Format (.rtf)

any format

Add Value

results where terms occur:

No items in list

Select...

Select Value:

anywhere in the page

in links to the page

in the URL of the page

in the text of the page

in the title of the page

Add Value

return results of the file format:

No items in list

Select...

Select Value:

Don't

Only

Add Value

safesearch:

No items in list

Select...

Select Value:

Filter using SafeSearch

No filtering

Add Value

updated in the:

No items in list

Select...

Select Value:

anytime

past 3 months

past 6 months

past year

Add Value

without the words:

OK

Cancel

Help

Figure 17: Invocation of an advanced Google query

As a side note, the invocation in Figure 17 provides a good example of why currying operations might be beneficial. There are so many parameters in this operation that it might be tedious for the user to have to fill out all of them every time. One of the benefits of currying is that the user could curry the operation with several parameters that the user tends to keep constant every time he performs the operation. Also, it is possible

for the user to carry several versions of this operation. For example, one version of the operation could involve all of the site specific search parameters and another could involve various language parameters.

7.3 System Failures

Even though our system works reasonably well in the above cases, there are some examples where certain aspects of our system do not perform as well. These examples fall into the following categories.

7.3.1 Operations with Intermediate Pages

When a user invokes an operation on some websites, one or more intermediate pages are briefly shown to the user before the results page is shown to the user. For example, on the Expedia web site (<http://www.expedia.com>), once a user searches for a travel itinerary, an intermediary page shows up that says that it is searching for flights before the results page shows up. This type of behavior can create problems for our system.

First of all, when our system recognizes that an operation has taken place, it tries to gather supplemental information about the operation from the page that the user was viewing just before the operation took place. When a site behaves like Expedia, however, our system might be fooled into thinking that an operation has taken place between the intermediary page and the results page because of the structure of the URL on the result's page. Thus, it would attempt to extract supplementary information from the intermediary page instead of the initiating page.

In addition, our system has problems invoking the operation again when a web site behaves in this fashion. Currently, our system waits until a `DocumentComplete2` event fires after the operation is invoked. If there are intermediate pages in between the initiating page and the results page, then there will be a `DocumentComplete2` event that fires for the intermediate page. This will confuse our system into thinking the

intermediate page is the results page. Some potential fixes to this problem are discussed in section 8.2.4.

7.3.2 Confusion About Parameters' Purposes

Another problem with our system arises when the user leaves several fields blank in the form on the initiating page. This tends to happen on sites like the advanced Yahoo search (<http://search.yahoo.com/search/options?fr=fp-top&p=>) where there are many fields that accomplish a certain task for the user. In the advanced Yahoo Search example, there are four fields that can be used to input search terms. Typically, when a user performs an advanced Yahoo search, he will only fill out one of these fields leaving the remainder blank.

When a user does this and attempts to remember this operation using our system, he will be presented with all of the parameters of the form as normal. However, there might be confusion as to what each of the different parameters is for. As mentioned earlier, the initial display names for parameters our system creates are meaningless. Also, since the user has no reference of an original value for the parameters he left blank, he has further confusion regarding the purpose for these parameters. Our system has attempted to fix this issue by looking at text to the left and right of parameters on a web page. This will not necessarily fix all problems, however. There might be cases where parameters do not have text on either side or have exactly the same text.

Another source of confusion for parameters arises with a specific scenario involving select parameters. If two select parameters have the same range, then again the user may have trouble realizing what the specific purpose of either of these parameters is. An example of this is on the advanced Yahoo search page. The values “any part of the page”, “in the title of the page”, and “in the URL of the page” are in the ranges for four select parameters on that page.

This example also shows another problem of confusion with parameters. These parameters are used in a sentence that involves two different parameters. For example, on the Yahoo page, the select boxes that specify where to search for a term are tied to the text boxes that specify the search term. The problem that arises here is that there is no way to group these two inputs together when we invoke the operation via our system.

7.3.3 Form Components Manipulated by JavaScript

Another problem that can arise with some websites happens when there is some JavaScript on the page. Some websites will change parameter ranges based on the selections a user has made. For example, on the Monstertrak job searching page if a user selects a certain job category, then a menu listing job subcategories becomes populated with subcategories related to the selected job category.

This type of website will create a problem because the ranges stored for the operation in our system are the ranges that exist on the web site at the time the sample operation was invoked. Thus, in the Monstertrak example if a user were to attempt to invoke the operation with a different job category selected, the subcategories listed would not be what they should be.

7.3.4 Results Pages that Differ Based on Inputs

The final problem we mention here has to do with results pages. On some websites, the format for the results page will change based on the inputs to the operation. For example, on the internet movie database website (<http://www.imdb.com>), if the search terms match multiple items in the database, the user is presented with a list of all of the matching items. If the search terms match only one item, then the user is presented with a view of just that item. Thus, the format of results page will be different depending on how many matches exist for the input.

This can create problems for our system since we only record one wrapper per operation. Thus, there is no way in our system to specify a wrapper that will handle two types of formats of results pages. This is not a huge problem, however, since if a wrapper fails in one of these formats, the user will just be presented with the results page. Thus, the system will still accomplish something useful for the user.

Chapter 8

Conclusions

8.1 Contributions

In this system, we have attempted to create ways to make retrieving information off the internet easier. We did this by attempting to learn and store web operations for future use. As discussed in section 2.1, some web browsers have previously attempted to automate web operations. None of them have attempted to either allow operation invocation outside of the internet or combine the operation process with information extraction.

By adding our system to Haystack, we have tried to incorporate some of the benefits for Haystack operations into our system for web operations. By doing this, users can perform web operations easier by currying them. In addition, by giving the user the power to perform web operations and extract the results without having to visibly access the internet, we give them the power to extract information from the internet more efficiently.

We have tried to make the process of learning, storing, and performing web operations as easy as possible through a simple user interface. To aid in this process, we try to acquire as much semantic data as possible from the web pages containing the web operations. We also try to help the user make performing web operations easier by letting them modify aspects of the parameters like their display names.

8.2 Future Work

There are several extensions to our system and Haystack in general that will make our system more powerful and useful. We outline a few of them below.

8.2.1 Incorporating Other Parameter Types

Currently, our system only handles parameters of the type “hidden”, “select”, “radio”, and “checkbox”. These are not the only parameter types that are available for forms however. For example, some forms use inputs of type “file” and “password” as well. “file” types send files to the server. “passwords” are similar to normal text fields except the text is not displayed so that the user can read it as he types it in. The “password” type input can be implemented using a normal text input if one does not want to worry about the hidden nature of the “password” type. However, this feature of “password” types is important so that eavesdroppers will not be able to tell what a user’s password is.

8.2.2 Cascading Operations Together

One future improvement that will benefit both Haystack and our system involves cascading operations together. For example, a user might want to perform a search on Google and then use the resulting collection in another operation. To make this process easier in the future, Haystack might allow the user to cascade these two operations together. Thus, in the future the user can simply invoke one operation and behind the scenes, Haystack will perform the query and perform the next operation on the resulting collection automatically.

8.2.3 Allow Multiple Wrappers per Operation

As mentioned earlier one of the problems in our system is that sometimes results pages will have multiple formats. Thus, one improvement that can be made to this system is to allow the user to specify multiple wrappers for the results page. Then, if a user invokes

that operation again, the system will pick which wrapper to use in the given situation based on which wrapper works on the given page.

8.2.4 Cope with Operations with Intermediary Pages

Our system will currently not cope with web operations that have intermediary pages. There are many web operations on the internet that use this method to implement their forms. Thus, in order to make our system as powerful as possible, in the future it would be beneficial to fix this problem with our system. One way to potentially fix this would be to let the user either specify both the initiating page and the results page explicitly or specify how many pages are in between the initiating page and the results page.

8.2.5 Organization for Web Operations

One possible addition to our system would be to provide some kind of organization for all of the web operations in the system. Hogue's wrapper induction system provides organization for its wrappers by associating them with the web pages they are applied to. Our system could potentially have a similar organization structure built on top of it. One benefit of this could include exposing our system to multiple invocations of an operation. These additional invocations could provide additional sample values for parameters. In addition, being exposed to multiple invocations could allow the system to hide parameters that a user typically leaves blank or enters the same value for every time.

8.2.6 Mechanism for Updating Web Operations

Web sites may change their format over time. Consequently, some parameter names that are valid for a web site today may not be valid in the future. Also, wrappers that worked on results pages for the operation today may not work in the future. Thus, in the future, it would be nice to have a mechanism and interface to update stored web operations to work with any new format that a web site might create.

8.2.7 Integration with Haystack on Other Systems

Currently our system only works in conjunction with an embedded Internet Explorer browser on the Windows version of Haystack. In the future it will be beneficial to have this system working on all versions of Haystack. It is possible for our system to work with other browsers on other platforms. We designed the system so that to extend the system, a developer would only have to implement a few methods in an interface for each browser that he wants this system to work with.

8.2.8 Making Renaming Parameters Easier

Earlier we mentioned how parameter names are unintelligible until the user renames them. We thought of a possible way to make that process easier in the future. First, the user would select the parameters in the view of the web operation. Based on which parameter was selected, the system would then highlight that parameter on the initiating page. Thus, the user would know instantly what each parameter is without having to perform any cross referencing with the parameter's left and right text.

Appendix A

List of Sites to Test

Below is a list of sites that we used to test our system. We list the URL for the operation, the type of operation, and the types of parameters that are used in the operation. We also list whether the wrapper induction system, in its current state, will allow the results pages to be wrapped.

Web Sites	URL	Operation Type	Parameter Types Tested	Wrapper Induction Worked
Google Simple Search	http://www.google.com/	Get	Normal, Hidden	yes
Google Advanced Search	http://www.google.com/advanced_search?hl=en	Get	Normal, Hidden, Select, Radio	yes
Yahoo Simple Search	http://www.yahoo.com	Get	Normal, Hidden	yes
Yahoo Advanced Search	http://search.yahoo.com/search/options?fr=fp-top&p=	Get	Normal, Hidden, Checkbox, Select, Radio	yes
Internet Movie Database	http://www.imdb.com	Get	Normal, Select	no
MIT Course Search	http://student.mit.edu/catalog/extsearch.cgi	Post	Normal, Checkbox, Select, Radio	yes
EBay	http://www.ebay.com	Get	Normal, Hidden	yes
Amazon.com	http://www.amazon.com	Post	Normal	yes
Barnes and Noble	http://www.barnesandnoble.com/	Post	Normal, Hidden	no
Expedia.com	http://www.expedia.com	Post	Normal, Hidden, Select	no
Mozilla bugzilla	http://bugzilla.mozdev.org	Get	Normal	yes
weather.com	http://www.weather.com	Get	Normal, Hidden, Select	No

Table 1: List of sites tested

Appendix B

Test Results

Below is a table of sites that we used to test our system and comments on whether the system worked well or not. If the system did not work well, we specify exactly what went wrong. Most of the comments about what went wrong refer back to section 7.3.

Web Sites	Comments
Google Simple Search	All aspects of the system work well
Google Advanced Search	Possible confusion about parameters' purposes
Yahoo Simple Search	All aspects of the system work well
Yahoo Advanced Search	Possible confusion about parameters' purposes
Internet Movie Database	Results pages differ based on inputs
MIT Course Search	Results pages differ based on inputs
EBay	JavaScript submitted the form
Amazon.com	Results pages differ based on inputs
Barnes and Noble	Operations has Intermediate Pages
Expedia.com	Operations has Intermediate Pages
Mozilla bugzilla	JavaScript submitted the form
weather.com	JavaScript submitted the form

Table 2: List of sites tested and comments about their success

Bibliography

1. Apple – Safari. <http://www.apple.com/safari/>.
2. Mozilla Suite – The All-in-One Application Suite.
<http://www.mozilla.org/products/mozilla1.x/>.
3. RDF Site Summary (RSS) 1.0 specification. <http://web.resource.org/rss/1.0/spec>, 2001.
4. The Omni Group – Applications – OmniWeb.
<http://www.omnigroup.com/applications/omniweb/>.
5. T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 284(5):35, May 2001.
6. T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, second edition, 2001.
7. A. Hogue. Tree pattern inference and matching for wrapper induction on the World Wide Web. Master’s thesis, Massachusetts Institute of Technology.
8. D. Quan, D. Huynh, and D. Karger. Haystack: A platform for authoring end user semantic web applications. In *Proc. 2nd International Semantic Web Conference*, 2003.
9. D. Quan, D. Huynh, D. Karger, and R. Miller. User Interface Continuations. In *Sixteenth ACM Symposium on User Interface Software and Technology (UIST)*, Vancouver, B.C., November 2003.
10. K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Computing*, 18(6):1245-1252, December 1989.