5-18-2007

Automated Discovery, Binding, and Integration Of GIS Web Services

Lev Shulman
University of New Orleans

Automated Discovery, Binding, and Integration Of GIS Web Services

A Thesis

Submitted to the Graduate Faculty of the
University of New Orleans
in partial fulfillment of the
requirements for the degree of

Master of Science
in
The Department of Computer Science

By
Lev Shulman
B.S., University of New Orleans, 2004
May 2007

ACKNOWLEDGEMENT

This thesis is the result of two years of work whereby I have been accompanied and assisted by many people. I am highly indebted to my advisor, Dr. Mahdi Abdelguerfi, for providing the guidance and direction that was substantial to this research. I would also like to thank Dr. Nauman Chaudhry and Dr. Shengru Tu, for their teaching provided fundamental knowledge crucial to understanding and designing data driven applications. Finally, I would like to acknowledge Kevin Shaw, Dr. John Sample, and the entire NRL DMAP team for their support.

Table of Contents

List of Figures

List of Tables

Abstract

The last decade has demonstrated steady growth and utilization of Web Service technology. While Web Services have become significant in a number of IT domains such as eCommerce, digital libraries, data feeds, and geographical information systems, common portals or registries of Web Services require manual publishing for indexing. Manually compiled registries of Web Services have proven useful but often fail to include a considerable amount of Web Services published and available on the Web.

We propose a system capable of finding, binding, and integrating Web Services into an index in an automated manner. By using a combination of guided search and web crawling techniques, the system finds a large number of Web Service providers that are further bound and aggregated into a single portal available for public use. Results show that this approach is successful in discovering a considerable number of Web Services in the GIS(Geographical Information Systems) domain, and demonstrate improvements over existing methods of Web Service Discovery.

Keywords: GIS Web Services, Web Services, Web Crawling, Web Map Services, XML Data Management, Automated Discovery of Web Services, Web Service Portal

Chapter 1 Introduction

As the number of Web Services has grown significantly over the last several years, Web Service portals, catalogs, and registries have sprouted to provide the user with a single access point or index from which to search and browse for Web Services of interest. The greater the size of the index, the greater is its value to the user as a service portal. This type of index typically grows by allowing Web Service providers to manually publish their services, or by employing an operator responsible for seeking out new Web Services of interest on the web and configuring the services into the index.

By taking advantage of Web Services that adhere to well-defined open standards, such as a common XML schema, we present the design and implementation of a Web Service Portal System that is fully capable of search, discovery, and integration of Web Services in a fully automated manner using scalable web crawling techniques. By acquiring the set of all valid HTTP URLs on the web, and validating each to a common XML schema, we could discover all such Web Services. However, such an approach is clearly impractical as the time and resources required to crawl the entire web for all possible HTTP URLs are out of scale for all but the most sophisticated major systems. Therefore, the preferred method of selecting URLs for validation should be scalable as well as intelligent enough to select URLs which are more likely to validate to a Web Service. By using domain knowledge and fine tuning some parameters of the web crawl, we can significantly reduce the number of possible HTTP URLs to test allowing the crawling process to perform on a tenable scale.

We focus on GIS Web Services for this research, as geospatial data is often most useful when aggregated together in a portal or index for browsing and search. This document is organized as follows. Chapter 1 provides an introduction. Chapter 2 contains some background on Web Services in general, GIS Web Service standards, some Geospatial Portals available on the web today, and previous research on discovery of Web Services. Chapter 3 follows with a discussion of using the Google APIs to generate a set of seed URLs which are in turn fed to the crawler in order to find more URLs to validate to a Web Service XML Schema. Chapter 4 depicts the validation process for the crawler's findings, specifically validation to the OpenGIS Consortium Web Mapping Service Standard Schema, and their integration into a unified Web Mapping Service Portal. In Chapter 5, results are presented which demonstrate the effectiveness of our approach in comparison to other research. Chapter 6 outlines the implementation and automated binding to an existing Naval Research Laboratory GIDB Web Map Service Portal. Finally, Chapter 7 provides an overall conclusions and discussion on future enhancements to this research.

Chapter 2 Background

This chapter provides a review of Web Service technology and XML. It follows with an overview of the SOAP/WSDL/UDDI SOA model of web service technology, and introduces OpenGIS Web Services with a description of the OpenGIS Web Mapping Service is provided. The chapter follows with a brief overview of the Google API and its relevance to this work, and concludes with a survey of previous research into automating search and discovery of OpenGIS Web Services.

2.1 Web Service Technology

According to the World Wide Web Consortium (W3C)[1], a Web Service is a software system designed to support interoperable machine to machine interaction over a network. In general, a Web Service is an API that can be accessed over a network, such as the Internet, and executed on a remote system hosting the service. Such an API is typically defined using the XML language [2].

2.1.1 Extensible Mark-up Language (XML)

XML is a general purpose W3C recommended mark-up language widely employed in a variety of software applications for the web. Its primary purpose is to facilitate the sharing of data across different information systems, and its popularity is due to its platform independence, self-describing format, and a variety of standardized tools to

parse and generate XML content. Of particular interest to this research, an advent of XML is validation to an XML Schema. An XML Schema employs a rich data-typing system allowing for detailed constraints on the logical structure on an XML document, and can serve as a set of rigid specifications for the structure and content that an XML document may contain. For an XML document to validate to an XML Schema entails that its content passes the set of specifications laid out in the schema. Another way of describing the relationship is to refer to an XML document that validates to an XML Schema as its instance document. This feature is relevant to this research as when we discuss the automated discovery of Web Services, we imply those Web Services whose XML definitions validate to a common XML Schema.

2.1.2 SOAP/WSDL/UDDI

A common usage of the term Web Service relates to SOAP formatted XML message envelopes and have their APIs described via the XML derived Web Service Definition Language Schema(WSDL)[3]. In this model, a Web Service is advertised as an XML document that validated to a particular WSDL Schema. A WSDL describes the service interface, bindings, protocols, and other details necessary to bind to and communicate with the service. In this model, UDDI is a protocol for registering and discovering metadata for a Web Service to a registry.

We define a Web Service domain as a collection of published Web Services that validate to a common standard WSDL Schema, and formulate a problem addressed in

this research as follows: how do we automatically find Web Services within a particular Web Service domain given the WSDL Schema for that domain? For example, several online news media providers such as the New York Times, the Washington Post, and the Houston Chronicle each provide a Web Service for user applications to bind to and retrieve news feeds on a daily basis. The functionality is very similar for each provider, and each newspaper site's Web Service is an XML Document that validates to a common WSDL Schema. With automatic discovery and integration, a portal system in the online news media domain can be programmed to automatically find and integrate Web Services from other online news media providers that validate to the same WSDL schema, forming an aggregate of news media from a single access point or portal.

2.1.3 OpenGIS Web Services

The Open Geospatial Consortium (OGC) provides a set of Web Service specifications for a variety of applications related to geospatial applications and geographical information systems such as the Web Mapping Service (WMS), Web Feature Service(WFS), and the Web Coverage Service(WCS)[4]. The OGC Web Services paradigm is analogous to its SOAP/WSDL/UDDI counterpart in that an OGC Web Service provides an XML document that consists of service interface descriptions, along with the details of their bindings. Such a document is called a Capabilities document by the OGC, and serves as an API used to generate server and client code. Each Web Mapping Service, for example, advertises a Capabilities XML document that validates to a public standard Web Mapping Service XML Schema published by the OGC, similar to a WSDL in the

SOAP/WSDL/UDDI model. A Web Mapping Service Client program retrieves such a document, scans it, and then is able to issue requests for map layers featured within the document.

Here is an example of a simple Web Mapping Service Capabilities document with two map layers:

```
<WMT_MS_Capabilities version="1.1.1" updateSequence="0">
	<Service>
		<Name>OGC:WMS</Name>
		<Title>NRL GIDB Portal: GIDBImageServer</Title>
		<Abstract>WMS-based access to NRL's GIDB Portal. Try NRL's GIDB Portal
System at http://dmap.nrlssc.navy.mil.</Abstract>
		<KeywordList>
			<Keyword>GIDB</Keyword>
			<Keyword>DMAP</Keyword>
		</KeywordList>
		<OnlineResource xmlns:xlink="http://www.w3.org/1999/xlink"
xlink:type="simple"
xlink:href="http://columbo.nrlssc.navy.mil/ogcwms/servlet/WMSServlet/GIDBImageServer.wms?
" />
		<ContactInformation>
			<ContactPersonPrimary>
				<ContactPerson>Kevin Shaw</ContactPerson>
				<ContactOrganization>NRL</ContactOrganization>
			</ContactPersonPrimary>
			<ContactPosition>software developer</ContactPosition>
			<ContactAddress>
				<AddressType>postal</AddressType>
				<Address>NRL Code 7440.2</Address>
				<City>Stennis Space Center</City>
				<StateOrProvince>MS</StateOrProvince>
				<PostCode>39529</PostCode>
				<Country>USA</Country>
			</ContactAddress>
			<ContactVoiceTelephone>+1 228 688-4197</ContactVoiceTelephone>
			<ContactFacsimileTelephone>+1 228 688-
4853</ContactFacsimileTelephone>

	<ContactElectronicMailAddress>kshaw@nrlssc.navy.mil</ContactElectronicMailAddress>
		</ContactInformation>
		<Fees>none</Fees>
		<AccessConstraints>none</AccessConstraints>
	</Service>
	<Capability>
		<Request>
			<GetCapabilities>
				<Format>application/vnd.ogc.wms_xml</Format>
					<DCPType>
						<HTTP>
							<Get><OnlineResource
xmlns:xlink="http://www.w3.org/1999/xlink" xlink:type="simple"
xlink:href="http://columbo.nrlssc.navy.mil/ogcwms/servlet/WMSServlet/GIDBImageServer.wms"
/></Get>
						</HTTP>
					</DCPType>
			</GetCapabilities>
			<GetMap>
				<Format>image/jpeg</Format>
				<Format>image/png</Format>
				<DCPType>
					<HTTP>
						<Get>
```

```
                                                        <OnlineResource
xmlns:xlink="http://www.w3.org/1999/xlink" xlink:type="simple"
xlink:href="http://columbo.nrlssc.navy.mil/ogcwms/servlet/WMSServlet/GIDBImageServer.wms"
/></Get>
                                        </HTTP>
                                </DCPType>
                        </GetMap>
                </Request>
                <Exception>
                        <Format>application/vnd.ogc.se_xml</Format>
                </Exception>
                <VendorSpecificCapabilities />
                <Layer>
                        <Title>GIDBImageServer</Title>
                        <SRS>EPSG:4326</SRS>
                        <Layer>
                                <Name>:1</Name>
                                <Title>Georgia Aerial Imagery</Title>
                                <LatLonBoundingBox minx="-180.0" miny="-90.0" maxx="180.0"
maxy="90.0" />
                                <ScaleHint min="0" max="1000000000" />
                        </Layer>
                        <Layer>
                                <Name>:0</Name>
                                <Title>NASA Blue Marble</Title>
                                <LatLonBoundingBox minx="-180.0" miny="-90.0" maxx="180.0"
maxy="90.0" />
                                <ScaleHint min="0" max="1000000000" />
                        </Layer>
                </Layer>
        </Capability>
</WMT_MS_Capabilities>
```

The above Capabilities Document applies to a Web Map Service(WMS) that returns Map Images to the client, and in this case the WMS server advertises only two layers of maps—Georgia Aerial Imagery and NASA Blue Marble. There are WMS Servers on the web today that advertise hundreds and hundreds of such map layers. Therefore, it is easy to see how a portal that automatically finds and aggregates such services can provide a rich and valuable resource to the GIS user. For this research, while our approach applies to a variety of Web Service domains, we focus mainly on the Web Mapping Service domain.

2.1.3 Web Map Service (WMS)

Access to a WMS Server [13] is most often advertised by a typical HTTP URL such as

http://www.wmsServerHost.com/path

Requests to the WMS server are constructed by configuring key-value pairs to the query part of the URL. For example, a URL of the form

http://www.wmsServerHost.com/path?REQUEST=GetCapabilities&Service=WMS

should return a XML WMS Capabilities document that adheres to a WMS Capabilities Schema as published by the Open Geospatial Consortium. Similar to the example above, the Capabilities document should describe the map layers that the WMS server provides, along with the WMS server's contact info and metadata.

Fetching a map from a WMS server can be as simple as typing a WMS URL request in an internet browser to retrieve the image. For example,

http://dmap.nrlssc.navy.mil/ogcwms/servlet/WMSServlet/GIDBImageSERVER.wms?REQUEST=GetMap&SERVICE=WMS&LAYERS=NASA_BLUE&BBOX=-180.0,-90.0,180.0,90.0&WIDTH=800&HEIGHT=600&FORMAT=image/png.

This HTTP URL would return a NASA Blue Marble PNG map image of 800 pixels width and 600 pixels height for the geographic region bounded by -180 and 180 longitude and -90 and 90 latitude. The WMS server host is "dmap.nrlssc.navy.mil", and the URL path is "/ogcwms/servlet/WMSServlet/GIDBImageServer.wms?". The various map request parameters in the query part of the URL,

"REQUEST=GetMap&SERVICE=WMS&LAYERS=NASA_BLUE&BBOX=-180.0,-90.0,180.0,90.0&WIDTH=800&HEIGHT=600&FORMAT=image/png", are simple HTTP key-value pairs that specify the geometric bounding box of the map image, pixel width and height, etc.. There are a number of user-friendly WMS client applications that provide a variety of mapping capabilities such as zoom in/out, map image transparency, layering, etc.., by configuring the HTTP key-value pairs according to user actions.

2.2 Google APIs

The Google API[5] is relevant to this research because its functionality allows the proposed system to query the Google Indexes of online resources and retrieve a set of URLs. Without going into much detail of the Google API, much in the same manner as you would if you manually typed in "Web Service" into a www.google.com page and hit the "Search" button, the Google API allows us to write a program that can query the Google indexes with search strings such as "Web Map Service", "maps", or "GIS", and get back a set of related URLs without manual entry.

2.3 Previous Research

Previous research on the discovery of Web Services is somewhat limited. Currently, Web Services compiled in registries or portals mostly require manual entry or registration. For Web Services adherent to the SOAP/WSDL/UDDI model, UDDI [14] provides the client/server protocol for publishing and querying the registry, and is most

commonly used inside a company or enterprise to dynamically bind client systems to implementations. For GIS Web Services, the OGC defines a Catalog Service Interfaces[6], but this standard has had a relatively low level of success and implementation[17]. Most registries are ad hoc manually compiled indexes such as the US Geological Survey's National Map[7] and the National Spatial Data Infrastructure Celaring-house portal.

Automating the search and discovery process of GIS Web Services has been investigated by several research groups. The Refractions Research OGC Survey[9] employs the Google APIs to search the internet for possible OGC Web Services. With this approach, the Google APIs are queried with search strings such as "inurl:Request=GetCapabilities". This returns a set of URLs with "Request=GetCapabilities" as a substring, and clearly is likely to return some URLs that point to OGC Web Service Capabilities documents. However, as in Figure 1**,** many OGC Web Service providers advertise by simply posting a URL of the form

http://OGCWebServiceHost.com/path?

instead of

http://OGCWebServiceHost.com/path?Request=GetCapabilities

Such URLs would not be hit by the Google API query, and though this approach has high probability of finding OGC Web Services, many such services won't be discovered. The Mapdex map server index[8] uses a similar approach using the Google APIs to find OGC Web Services as well as non-standard ArcIMS Services(a commonly used proprietary solution similar to WMS).

Figure 1

http://mesonet.agron.iastate.edu/wms/comprad.php?	NEXRAD radar	Info	US
http://www.geographynetwork.com/servlet/com.esri.wms.Esrimap?	Various vector and raster datasets from ESRI	Info	Global
http://clearinghouse1.fgdc.gov/scripts/ogc/ms.pl?	DCW data		Global
http://globe.digitalearth.gov/viz-bin/wmt.cgi?	Many Environmental layers	Info	Global
http://www2.demis.nl/mapserver/Request.asp?	General data	Info	Global

This snippet from a web page listing WMS, shows WMS URLs that would not be discovered using the Google API with search string "Request=GetCapabilities".

Chapter 3 Web Service Discovery

3.1 Reducing the Search Space of URLs

While the number of available WMS Servers on the web has been growing at an impressive rate for the past couple of years, searching for new WMS sources is often reduced to simply "googling" to find WMS servers on the internet. We have devised an automated approach to discovering Web Services such as WMS by reformulating the problem and considering the following assertions:

- Every WMS Server has a published XML document that validates to a common XML schema as published by the Open Geospatial Consortium
- Every WMS Server will return a valid OGC WMS Capabilities Document when issued a HTTP GET to its URL with the "Request=GetCapabilities" string attached

With this in mind, we can solve the problem by gathering the set of all HTTP URLs on the web, and for each URL, append the "Request=GetCapabilities" string and test for validation to the OGC WMS Schema. If validation is a success, and the HTTP Response is an instance of the Schema, we have found a WMS. Clearly, this approach is untenable because crawling the entire web for all possible URLs would take an impractical amount of execution time and resources even for the most advanced systems. Some filter is needed to scale down the set of URLs to be validated.

An appropriate filter would reduce the set of URLs to those related to the Web Service Domain, such as http://www.washingtonpost.com and http://www.nytimes.com for a Web Service domain related to news media, or http://www.opengis.com and http://www.esri.com for the GIS Web Service domain. By using the Google API to query the Google web indexes with search strings like “maps” or “WMS maps”, we acquire a set of URLs likely related to the GIS Web Service domain. These URLs are then fed to a web crawler as seed URLs to gather more URLs in the domain.

3.2 Crawling For GIS Web Services

Typically, a web crawler program is fed a set of seed URLs to crawl for more seed URLs. Given a seed HTTP URL, the crawler fetches the URL HTTP Response, and traverses the content to find pointers to other HTTP URLs. The method of extraction varies in complexity depending on the objective of the crawl. If the objective is as simple as crawling through HTML links embedded in HTML <href> elements, the crawler would completely ignore downloads of file types such as Word documents, audio files, Excel spreadsheets, executables, etc.. If the objective is to find all embedded HTTP URLs, the crawler can be programmed with tools to read Excel or Word formats for text and employ pattern matching to find URLs. The approach can be as granular as parsing the raw binary data of the HTTP Response to find URLs if necessary, although this would consume a greater amount of resources. Unless there are filters and limits to the potential search space of URLs for a crawl job, the crawler may crawl the web indefinitely or until the program runs out of memory.

The web crawler employed within the system is the scalable and highly configurable Java Heritrix Open Source Web Crawler[10]. Heritrix can be configured and reprogrammed to carry out highly specialized crawl jobs by fine tuning several parameters of a crawl job. Here are a few of interest:

Scope: for each discovered URI, the crawl scope decides if it is within the scope of the current crawl. Here are some levels of scope provided with Heritrix:

- **Broad Scope:** allows for limiting the depth of the scope, but has no constraints on the hosts, domains, or URI paths crawled.
- **Domain Scope:** limits discovered URIs to the set of domains of the crawl job's seeds. For example, given a seed "http://www.uno.edu", domain scope would allow URIs with cs.uno.edu and math.uno.edu to be fed as seeds for further crawling.
- **Host Scope:** limits discovered URIs to the set of hosts of the crawl job's seeds. Given a seed "http://www.uno.edu", the host scope would not allow cs.uno.edu or math.uno.edu
- **Path Scope:** limits discovered URIs to a section of the paths on hosts defined by the seeds.

Focus: defines a set of filters for URIs to be considered within scope. A filter may set constraints on the size of a download from a URI, set a regular expression to match for discovered URIs, or set constraints on the types of data to be downloaded. An

HTTP Response usually contains some key-value pairs as metadata in a section called the HTTP Header, which can be fetched separately prior to downloading the whole response. The MIME type, such as 'image/png' or 'text/xml' or 'application/msword', is typically included in the header to indicate what type of file the response is sending to the client. Because we want to crawl for XML Web Services, crawl focus is significant in limiting the seed space to URIs that return only text or XML and avoiding downloading irrelevant types of data such as video or audio.

Exclude: defines a set of filters for URIs to be considered out of scope.

Depth: the number of link hops the crawl job can make from its initial seed URIs. This parameter can substantially impact the time it takes to complete a crawl job. Consider that a crawl job with Broad Scope and no depth limit will go on indefinitely.

To avoid unnecessary crawls, it is important to seed the crawler with URIs that will have high probability of crawling relevant hosts and domains. The crawl jobs for our system are fed with seed URIs gathered from queries to the Google APIs, using search strings like "WMS Maps" or "OpenGIS Web Services". To improve the performance of the Web Crawler and limit unnecessary downloads, the crawler focus is configured to exclude data types such as images, video streams, executables, etc., and include relevant data types such as HTML, XML, Word, Text, Excel, etc..

Note that our approach is somewhat superior to that which relies solely on the Google Search API to find WMS sources because it will find sources whose URIs do not include the query string “Request=GetCapabilities” to retrieve the Capabilities Document. For example, our crawler was able to find the WMS Server advertised by the URI http://mesonet.agron.iastate.edu. Because http://mesonet.agron.iastate.edu?Request=GetCapabilities is not explicitly advertised anywhere on the web and therefore not indexed by Google, the Google API approach does not find this WMS server.

Chapter 4 Web Service Validation

4.1 Eliminating Duplicates

As the crawler carries out its tasks, the discovered URIs are stored in a database for later validation. In addition to ensuring that there are no duplicate URIs, the system checks for host aliases by resolving to IP. For example, the crawler has found two different URIs http://dmap.nrlssc.navy.mil/ogcwms/GIDBImageServer.wms and http://columbo.nrlssc.navy.mil/ogcwms/GIDBImageServer.wms. The hosts dmap.nrlssc.navy.mil and columbo.nrlssc.navy.mil actually map to the same IP 55.345.23.22. The system resolves both URIs to http://55.345.23.22/ogcwms/GIDBImageServer.wms, thereby eliminating duplicates.

Every online WMS Server must respond to a request for its Capabilities document, which is made by an HTTP GET request to the server URL with the query string "REQUEST=GetCapabilities&SERVICE=WMS", such as http://wmsHost.com/path?REQUEST=GetCapabilities&SERVICE=WMS
The response is an XML document that validates to the OGC Web Mapping Service XML Schema. With this in mind, every URL gathered by the crawler is a valid WMS source only if it responds correctly to the Capabilities query with a valid Capabilities document. A validation component of the system traverses the list of the crawler's findings by appending the query string and attempting to validate the HTTP Response.

Note that by performing hostname resolution to IP, we can eliminate URL duplicates that point to the same location. However, we have not taken into account the case where two URLs point to two different locations but return the same content. Figure 2 illustrates this point.

Figure 2

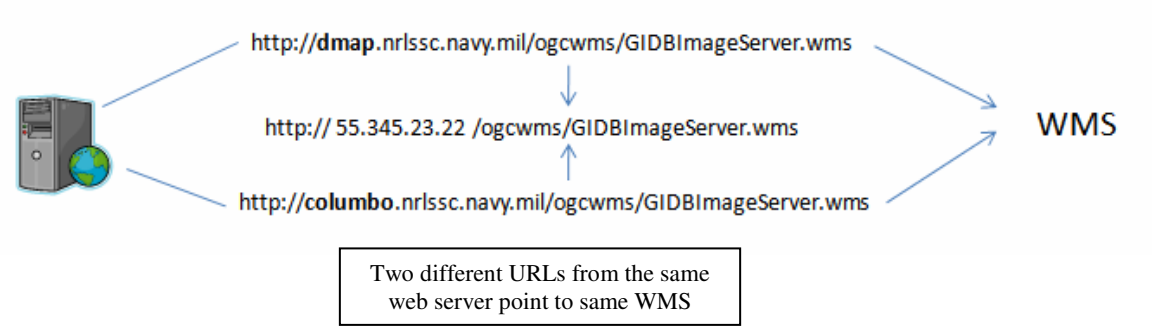
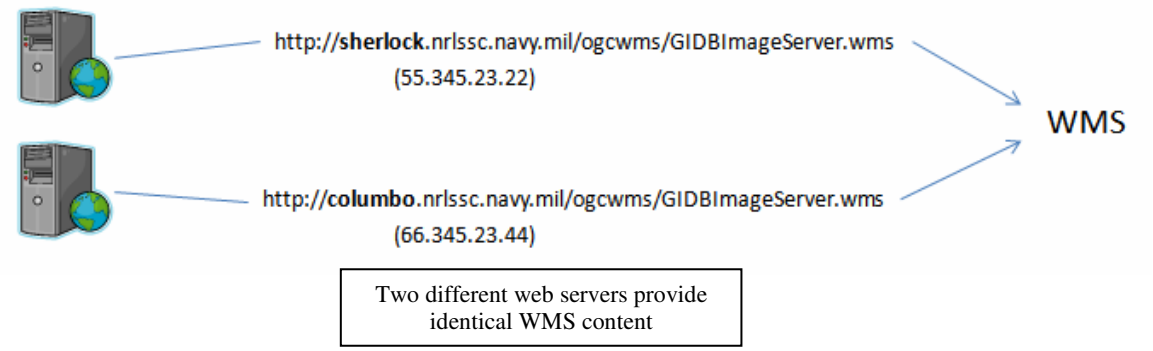


Figure 3



It is possible for two different URLs to refer to the same WMS. For example, Figure 3 depicts a scenario in which the sherlock.nrlscc.navy.mil WMS provides the same collection of maps as columbo.nrlssc.navy.mil WMS. For our portal to index both of these sources in such a scenario would introduce a level of redundancy. The simple solution to detect such redundancy would be to test for string equality between the two

entire XML Capabilities documents from the two hosts. However, testing for XML document equivalence by simply comparing the bare text does not entirely solve the problem. The system must be able to detect such redundancy given the structure or 'canonical' representation[11] of an XML document, not just its text representation. For example, consider the two very simple XML files in Figure 4.

Figure 4

File A

```
<fruit>
    <cherry color="red" size="small"/>
    <watermelon color="green" size="large"/>
</fruit>
```

File B

```
<fruit>
    <watermelon size="large" color="green"/>
     <cherry size="small" color="red"/>
</fruit>
```

The text representations of the two files would not be equal because the order of child elements for the <fruit> element is different. Notice also that the order of attributes for the <cherry> and <watermelon> elements is different between the two files. Though string equivalence would fail, the two files are clearly logically equivalent. The two files in Figure 4 are different, but represent the same collection of fruit. Similarly, if our web crawler finds two WMS sources offering the same map content, the system indexes a single source.

To test for logical XML document equivalence, we recognize that a well-formed XML document is a tree of elements. The system implements logical equivalence between two XML elements, and tests for document equivalence by recursively applying the equivalence test of the root element.

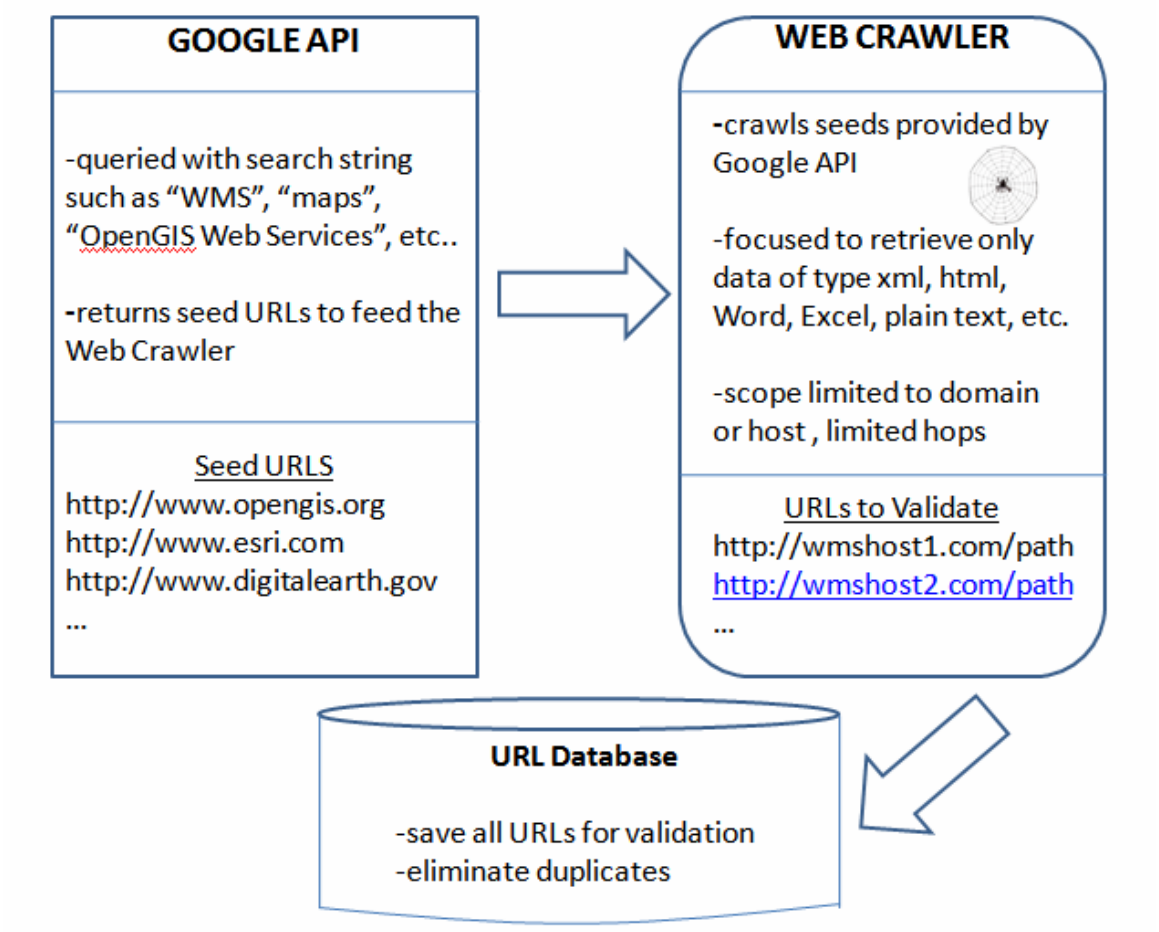


Figure 5: Web Service Discovery process

4.2 Quality of Service

Once the set of found URLs has been processed, the system discards those URLs that do not validate to the OpenGIS Web Map Service Schema. At this stage, the system has acquired a set of potential WMS sources. Note, however, that simply because the system has found links to valid WMS XML Capabilities documents, does not entail that each pertains to an actual online WMS Server. There are likely XML Capabilities documents published on the web that serve as examples. It is also possible that the server itself is down or inactive. This thesis document contains an example Capabilities XML text for a WMS with two layers, and the XML contains an HTTP URL for a dummy WMS. You

are now probably aware that if this particular thesis document was published on the web, the system crawler might very likely find this document and extract the URL as a potential WMS source!! To handle such cases, some quality of service should be introduced for each candidate WMS source found by the crawler.

All of the WMS sources indexed in the system are periodically invoked to check availability of service. A multi-threaded program was implemented for this purpose, where a thread attempts to fetch a generic map image from the WMS. The thread sequentially invokes each map layer advertised in the Capabilities document until an image is returned. Once a map is successfully retrieved, the system closes the thread. This at least assures that the Capabilities document found by the crawler did not point to a dummy WMS, but finer granularity can be configured into the system to ensure better quality of service.

In our research, of the number of WMS sources indexed that were at some point assured to have a back end server, at any given time approximately 2% to 3% are down and inactive. Observations show that these inactive services are not necessarily recurring. In other words, the system's periodic checks for availability of service show that the set of inactive servers at a given point in time may change at the next check. For these reasons, the system does not discard a found WMS source if it is off line.

Chapter 5 Results

The result of using our approach to facilitate the search and discovery for Web Map Services is promising [15, 16]. Using the Google APIs in conjunction with a focused web crawler has discovered scores of services published on servers scattered all over the globe. Currently the index houses around 1400 WMS sources, for a total of close to 330,000 various map layers. With the advent of the system's index, today the GIDB brokers access to the largest number of Web Map Services on the web.

Web Services allow for a great level of interoperability, and as an example of the utility and flexibility of this automated system, its index has been integrated into the existing Naval Research Lab GIDB Portal[12](see chapter 6 below). The GIDB Portal publishes WMS access to all of the system's findings on the web, and fully integrates with many popular GIS products such as ESRI, and popular mapping applications like the UDIG WMS map client, NASA's World Wind application, and Google Earth. Currently, GIDB is a leading provider of electronic map content.

We compare the results of our approach to that of the Refractions Research WMS survey. Table 1 shows a comparison between the two methods of WMS search and discovery. Our approach found twice as many WMS Servers and thirty-four more unique hosts, each of which may provide multiple Web Mapping Services. Considering the ratio of WMS servers and unique hosts, our approach is somewhat superior. The primary reason why using the Google API in conjunction with a web crawler is more efficient than using the Google API alone is depicted in this work.

Table 1

	GDIB WMS Crawler	Refractions Research Google Search
WMS Servers	761	309
Unique Hosts	174	140
Servers Uniquely Found By Method	436	84

The performance of the crawler itself depends highly on the configuration of the crawl jobs. For this research, the WMS survey results were compiled by running several crawl jobs on one machine over a period of approximately twelve days. About 15 different search strings were used to query the Google APIS, such as “maps”, “WMS”, “Web Mapping Service”, “OpenGIS Web Services”, etc.. The crawls were highly focused to avoid extraneous downloads, of domain scope, and limited to 3 and 4 hops. The crawls executed on a 3.0GHz processor with 1GB memory. From this configuration, it is more evident how scalable and adaptable this approach when faced with limited computing resources. For example, when attempting to run the crawls with 6 to 7 hop limits, the system ran for two weeks, while estimating only ~15% completion.

Chapter 6 Integration with Existing Applications

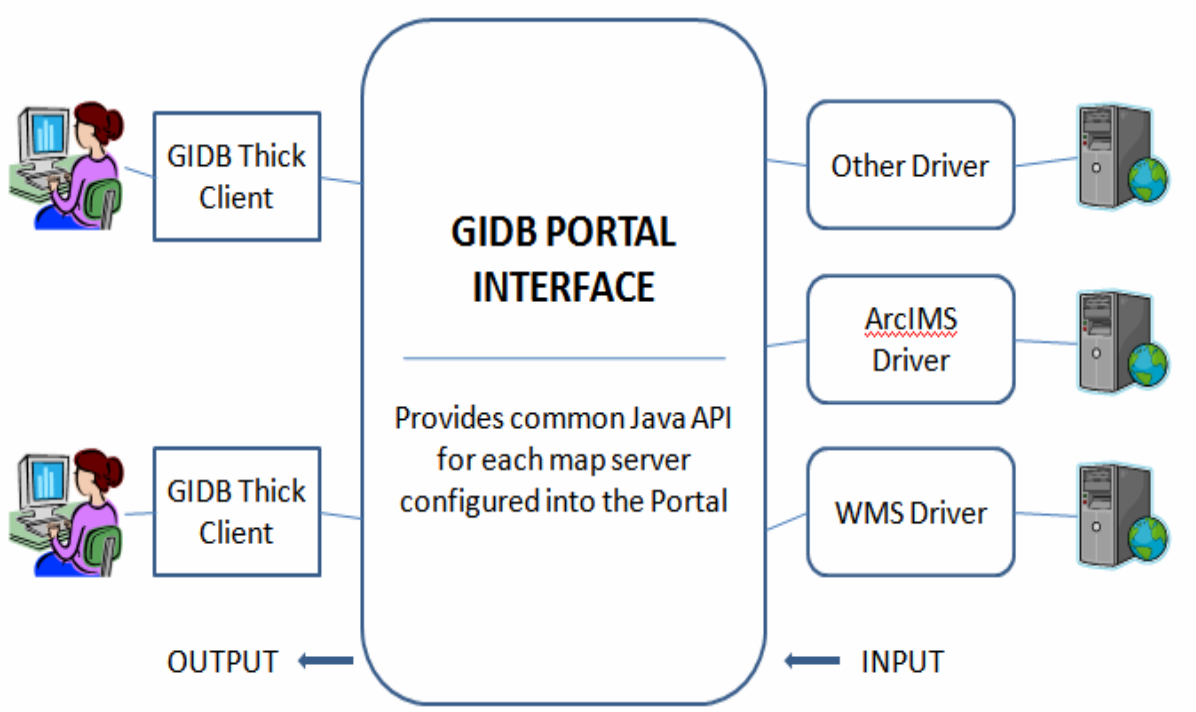
The utility of the system proposed in this work lies not only in providing access to a vast array of Web Map Services, but in its interoperability with other existing applications and system. A major benefit of utilizing Web Service technology is the flexibility of integrating systems by implementing to a well defined, highly interoperable interface. The Naval Research Lab's GIDB Portal [12] is an example of an existing application that has been successfully integrated with the automated WMS search/discovery system depicted in this work.

6.1 Naval Research Lab GIDB Portal

The NRL GIDB Portal System brokers access to thousands of maps from hundreds of various internet map servers such as JPL's NASA Blue Marble server, ArcIMS servers, Satellite Imagery Servers, National Weather Service server, as well as WMS servers. While the various map servers may have client applications specifically designed to work with the particular type of map server, the GIDB Portal provides the user access to all of the map server through a single mapping client application. Because the map servers have different implementations and access formats, drivers are programmed to convert map requests for a particular map server to the GIDB Portal Interface Java API. The major benefits of using the GIDB Portal to access maps online are the copious amount of maps the Portal brokers, and homogeneous access to a variety of heterogeneous map servers via thick/thin client applications provided by NRL.

Of particular interest to this work is the GIDB portal WMS Driver component, which converts GIDB Portal map requests to WMS map requests, which are in turn relayed to various WMS servers brokered by the GIDB Portal. The system architecture is shown in Figure 6**.**

Figure 6



6.1.1 WMS Driver

To add a new map server to the GIDB Portal, a Driver component must be implemented according the JAVA API provided by NRL. The JAVA API is a series of Java interfaces, the methods of which are to be implemented in the Driver component to map request/response communication between the GIDB Portal and the map server. For

example, the GIDB Java API contains an interface with the method cutRasterForAOI, which fetches a map image to the client application

```
public RasterImageData cutRasterForAOI(long[] idPath, BoundingBox bb,
                  int width, int height, int quality,
                  long scale, boolean legend) throws
                  Exception;
```

The method arguments are as follows:

- idPath—unique identifier for a map layer in the GIDB Portal
- bb—this is a geographic lat/lon bounding box such as (-90,90)(-90,90)
- width—width of the image to be returned in pixels
- height—height of the image to be returned in pixels
- quality—this parameter is irrelevant to our discussion
- scale—the scaling factor
- legend—include the legend in the image returned

Note that these parameters are similar to the parameters needed to construct a WMS map request. Consider the xml snippet from our sample WMS Capabilities document:

```
<Layer>
       <Name>:1</Name>
       <Title>Georgia Aerial Imagery</Title>
       <LatLonBoundingBox minx="-180.0" miny="-90.0" maxx="180.0" maxy="90.0" />
       <ScaleHint min="0" max="1000000000" />
</Layer>
```

The snippet describes a layer available from the WMS, and we can issue a map request to the WMS by constructing the following URL

http://dmap.nrlssc.navy.mil/ogcwms/servlet/WMSServlet/GIDBImageSERVER.wms?REQUEST=GetMap&SERVICE=WMS&LAYERS=NASA_BLUE&BBOX=-180.0,-90.0,180.0,90.0&WIDTH=800&HEIGHT=600&FORMAT=image/png

Building a WMS driver for the GIDB Portal entails mapping the parameters of WMS requests such as the URL above to the arguments of GIDB Java API methods such as the cutRasterForAOI() method.

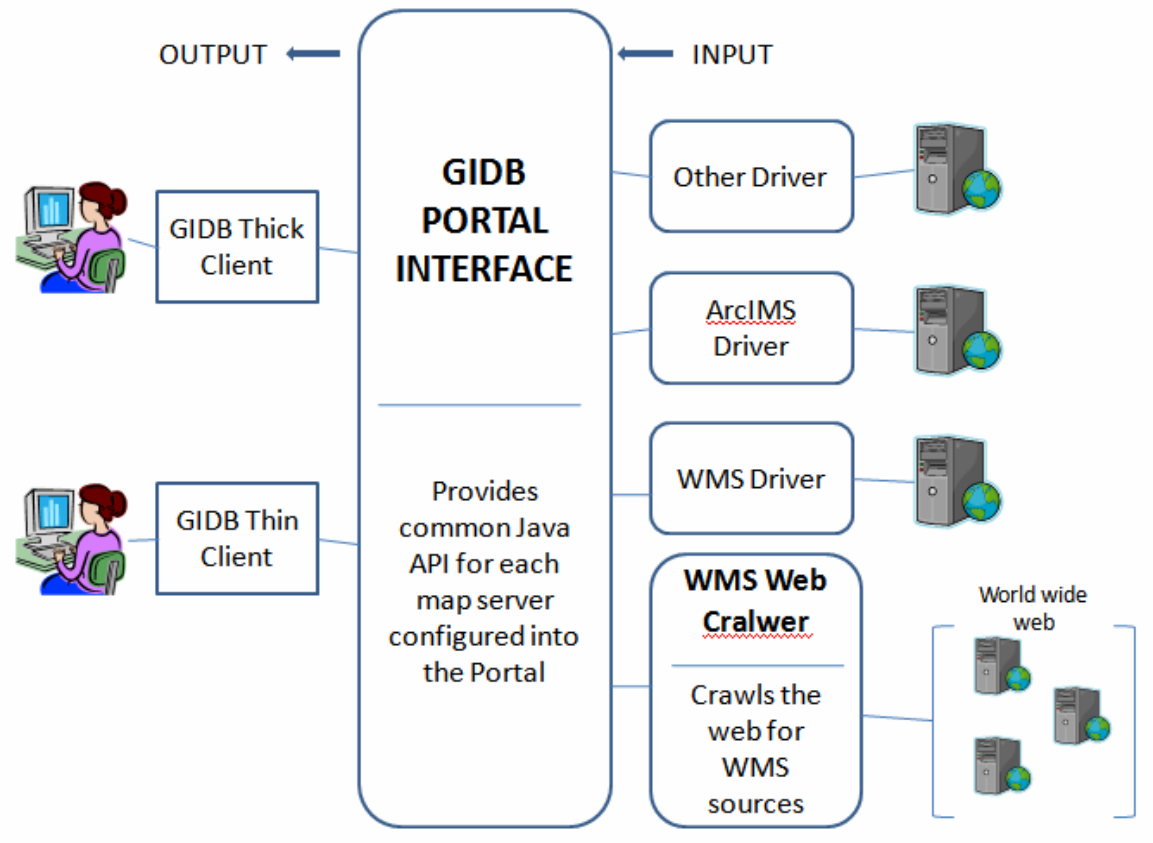
The GIDB WMS Driver converts GIDB Portal map requests to WMS Map requests by constructing the appropriate WMS URLs. Essentially, this system component behaves as a WMS client that reads the Capabilities document of a WMS Server configured into the GIDB Portal. Based on the contents of the document, the WMS Driver advertises the particular WMS Server's map content to the GIDB Portal user. As the user selects to view a particular map layer, a GIDB Portal fetch map request is issued and processed into a WMS fetch map request to the selected WMS Server. The only piece of information that is needed for the WMS driver to configure a WMS Server into the GIDB Portal is the WMS Server URL as described above. Communication from the WMS Driver to the WMS Server is performed by configuring HTTP query key-value pairs to the corresponding WMS Server URL and fetching the HTTP response.

6.1.2 Integration with the GIDB Portal

To integrate our system with the GIDB Portal, the system was implemented to make its list of WMS sources available to the WMS Driver. The WMS Driver was programmed to periodically check the list for new sources. If a new WMS source has been discovered by our system, the WMS Driver fetches its URL and makes access available through the GIDB Portal. In this manner, the GIDB Portal grows as our system grows by

automatically searching and discovery new WMS sources. Figure **7** demonstrates the revised diagram of the GIDB Portal System integrated with the system addressed in this work.

Figure 7



Chapter 7 Conclusion

7.1 Overview

Web Service portals that broker access to Web Services of a particular interest provide highly interoperable, platform independent means of binding and integration to a vast array of services from a single aggregated source. The greater the number of Web Services compiled by such a portal, the greater its utility and benefit to the user. Most search and discovery methods for such Web Services are limited to manual publishing to a registry or require an operator to manually search for new sources on the web and configure the findings to the portal index.

This work has presented a highly scalable, practical approach to designing and implementing a system capable of automated search and discovery of Web Services that adhere to well defined standards and a common schema. By employing a topic driven, highly focused web crawler, the system is scalable and flexible enough to meet the constraints of fairly limited computing resources, and demonstrates interoperability to fully integrate with a modern applications for automated binding. Overall, the system is capable of search, discovery, and integration of Web Services in a fully automated manner, thereby reducing the need for manual search and publishing of such services to a registry.

7.2 Contribution to Research

In addition to coming up with the original concept for the unique way of searching and discovering GIS Web Services depicted work, the following is a broad list of contributions to this research:

- System Design
    - Conceptualized the various functions of system components and assessed the feasibility of implementation
    - Analyzed the various versions of the WMS and WFS GIS Web Service format, and designed QOS methods to implement in the system
    - Researched and designed the system component responsible for tracking duplicate sources indexed by the system
    - Fully designed the PostgreSQL geospatial database responsible for the indexing of the Web Crawlers Findings
    - Fully designed the aggregation and generation of all sources in the index into one singe web service
- System Implementation
    - Reprogrammed and reconfigured the Heritrix Open Source Java code to suit the objectives of the system.
    - Fully implemented all data connections and flow between the system components.
    - Created a Java package for validating the discovery of Web Services, tracking duplicates within the index, and QOS of sources

- Fully implemented the WMS connector code needed to integrate the Crawler findings with the GIDB Portal.
- Created a user friendly Web Interface for browsing the index by test keywords as well as geospatial criteria.

7.3 Future Work

While the system has been highly successful at finding WMS sources, there is room for improvement. In generating the seed URL search space, the system uses a linear approach and considers each URL to have the same probability crawling to a WMS as the next. In other words, as the crawler is configured to execute another general web survey of WMS sources, it has no memory of its previous findings. However, our results show that a number of WMS sources are often published as a group on the same host or domain. In order to increase the probability of success for a seed URL, the system should maintain a history of previously crawled hosts or domains and assign weights accordingly. For example, consider the following six WMS sources for the Charleston County North Carolina maps:

http://ergmap.er.usgs.gov/OGCConnector/servlet/OGCConnector?servicename=charleston_county

http://ergmap.er.usgs.gov/OGCConnector/servlet/OGCConnector?servicename=charleston_1_foot

http://ergmap.er.usgs.gov/OGCConnector/servlet/OGCConnector?servicename=charleston_1_meter

http://ergmap.er.usgs.gov/OGCConnector/servlet/OGCConnector?servicename=charleston_geology

http://ergmap.er.usgs.gov/OGCConnector/servlet/OGCConnector?servicename=charleston_hydro

http://ergmap.er.usgs.gov/OGCConnector/servlet/OGCConnector?servicename=charleston_roads

The six URLs are Web Map Services all from the same host ergmap.er.usgs.gov. The system is configured to run a new crawl survey every month. Suppose USGS publishes a new WMS. At the start of the survey, the crawler will give no preference to crawling host ergmap.er.usgs.gov over host www.cs.uno.edu, although er.usgs.gov clearly has more probability of producing new results. The system can be reconfigured to run two types of crawl surveys—a general survey of the web, and a survey of all domains which have produced results in the past. The domain specific survey would require dramatically less amount of resources and time, and could be configured to reiterate every day, whereas the general survey would run every month.

It is relevant to reiterate that the solution outlined in this research is not only to the problem of finding GIS web services, but can be extended to the general problem of finding XML content of a particular interest that validates to a common XML Schema. While the system is capable of finding such content without any requirements on the content provider, there are ways to make discovery more efficient by introducing a few demands on the server side. When the system starts a crawl on a base URL returned by the Google API, such as http://www.host.com/path?, it first retrieves the contents of the top of the host hierarchy http://www.host.com/index.html, and further crawls down the host's tree of URLs. An HTML document can contain a wealth of meta data, invisible to the web user. The crawler can be programmed to avoid downloading and extracting unnecessary URLs from the host if the host index.html page contains a simple meta data element such as:

```
<web crawl metadata>
        <Web Service href="http://www.host.com/path1? />
```

```
        <Web Service href="http://www.host.com/path2? />
        <Web Service href="http://www.host.com/path3? />
</web crawl metadata>
```

This would eliminate the need to crawl the entire host. By introducing such a protocol between the client web service discovery system and the server web service content provider, this research can be extended to construct a well defined framework for a much more efficient method of automated web service discovery.

References

1. World Wide Web Consortium, Web Services Activity, http://www.w3.org/2002/ws
2. T. Bray, et al, Extensible Markup Language (XML) 1.0, W3C Recommendation, Feb. 2004.
3. E. Christensen, et al, Web Services Description Language(WSDL) 1.1, W3c Note, March 2001.
4. Open Geospatial Consortium Specifications, January 2007. http://www.opengis.org/standards
5. Google API Specifications, March 2006. http://code.google.com/
6. OpenGIS Catalog Service, Jan 2002. http://www.opengeospatial.org/standards/cat
7. National Spatial Data Infrastructure Portal. http://geodata.gov
8. Mapdex. http://mapdex.org
9. Refractions Research OGC Web Services Survey, March 2006. http://www.refractions.net/white_papers/ogcsurvey.
10. Heritrix Open Source Web Crawler, March 2006. http://crawler.archive.org/
11. XML Canonicalization Requirements, June 1999. http://www.w3.org/TR/1999/NOTE-xml-canonical-req-19990605
12. J. Sample and F. McCreedy, "Distributed Geospatial Intelligence Integration and Interoperability through the GIDB Portal System, " Chapter 7, *Net-Centric Approaches to Intelligence and National Security*, Springer, 2005. Pp 55-81
13. J. Beaujardier, Web Map Service Implementation Specifications, Version 1.3, Aug. 2004: http://portal.opengis.org/files/index.php?artifact_id=5316

14. UDDI Version 2.03 Data Structure Reference, UDDI Committee Specification, July 2002
15. John T. Sample, Roy Ladner, Lev Shulman, Elias Ioup, Fred Petry, Elizabeth Warner, Kevin Shaw, Frank P. McCreedy, "Enhancing the US Navy's GIDB Portal with Web Services," IEEE Internet Computing, vol. 10, no. 5, pp. 53-60, Sept/Oct, 2006.
16. Lev Shulman, Elias Ioup, John T. Sample, Kevin Shaw, Mahdi Abdelguerfi, "Automated Web Service Discovery", SWIIS Workshop, 2007.
17. Paul Ramsey, "Simple Web Services Catalogues", http://geotips.blogspot.com/2005/10/simple-web-services-catalogues.html

Vita

Lev Shulman was born in Moscow, Russia. He received his Bachelor of Science degree from University of New Orleans, in Spring 2004, with a major in Computer Science. In the Fall of 2004, he was awarded the Louisiana Board of Regents Masters Fellowship, and started a Masters program in Computer Science at the University of New Orleans. He currently resides in New Orleans, LA.