SOAP Web Services

Brian Suda

Master of Science
Computer Science
School of Informatics
University of Edinburgh
2003

Abstract

SOAP based web services are designed with a common XML-based protocol. The goal is to allow for a machine readable document to be passed over any and/or multiple connection protocols to create a decentralized, distributed system. This project demonstrates an RPC client-server model service that provides bibliography information. Any SOAP aware program could use this service by sending the proper values to the server and in return getting back a full listing of bibliographic information. This paper also discusses the advatages and disadvantages of the SOAP protocal, how it compares to various other distributed system protocols such as; CORBA, XML-RPC, and JAVA-RMI, and how SOAP fits into the distributed acritecture.

Acknowledgements

Special thanks to Henry Thompson as my Advisor, he answered all my questions and queries, to Alexander Holt for his help with the old COGSCI BibTeX database, and to Jay, Mary, Mary Pat, and others for their many eyes in proofing chapters of this paper.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Brian Suda*)

Table of Contents

Chapter 1

Introduction to Web Services

1.1 What is a Web Service?

The term "Web Service" was and still is quite a buzzword. The definition ranges from the quite loose "any services that is available over the web" to the more concrete. The World Wide Web Consortium (W3C)[1] defines a web service as the following:

> The World Wide Web is more and more used for application to application communication. The programmatic interfaces made available are referred to as Web services.(Haas, 2002)

The "Web" in web services is actually a misuse: the term "Internet Services" would be more appropriate. Web refers to HyperText Transfer Protocol (HTTP) and the World Wide Web, whereas the word "Internet" refers to the larger network of computers on multiple protocols. A web service can use any of these protocols to pass a message, not just HTTP.

Web services have been around since at least 1999, making them a relatively new technology that has gotten lots of press and praise. There is no secret behind web services that will instantly make everything better or work together. The most important factor to the success and popularity of web services is the fact that its backbone is XML.

[1] The W3C is the standards body that makes recommendations regarding internet protocols.

1.2 What is XML?

XML is an acronym for eXtensible Mark-up Language and was developed and finalized by the W3C in 1998. XML is a well formed, tree structured, plain text document that is human readable and machine consumable. It is a lighter and more flexible version of its predecessor Standard General Mark-up Language (SGML)[2].(ISO, 1986) XML provides syntax for document mark-up and provides syntax for declaring the structure of documents.

Originally designed to meet the challenges of large-scale electronic publishing, XML is also playing an increasingly important role in the exchange of a wide variety of data on the web and elsewhere. XML is at the heart of web services and this is what gives them their many strengths. XML is now over five years old, and more and more practical applications for its mark-up are being discovered each day.

1.3 What is SOAP?

SOAP is designed to be a new protocol for the decentralised, distributed environment, which utilises the power of the Internet and XML to pass typed information between nodes. Originally, SOAP stood for Simple Object Access Protocol, but was later changed simply to SOAP with version 1.2, because it did not directly use Objects.

SOAP is fundamentally a stateless, one-way message exchange paradigm between SOAP nodes, from a SOAP sender to a SOAP receiver. By combining one-way exchanges with features provided by the underlying transport protocol and/or application specific information, SOAP can be used to create more complex interactions such as request/response, request/multiple response, etc.(Don Box, 2000)

SOAP is a lightweight protocol that is platform independent, transport independent, and operating system independent, all because it is built using time testing systems like the HTTP protocol and text mark-up in XML.

SOAP is a W3C recommendation, which means that it is a technical report that is the end result of an extensive consensus building inside and outside of the W3C about

[2]SGML was published in 1980 by ANSI and in October 1985 a draft ISO standard was published. SGML was derived from Generalized Mark-up Language (GML) created in 1969.

a particular technology or policy.

There are two types of SOAP requests. The first is the Remote Procedure Call (RPC) style request similar to other distributed architectures. This is usually synchronous; the client sends a message and waits to get a response or fault message back from the server. The second type of SOAP request is the document request. In this case, a full XML document is passed to/from the client and server, in side a SOAP message.

1.3.1 RPC

SOAP-RPC is an implementation of a Remote Procedure Call (RPC). In this case, a function on a remote machine is called as if it were a local function. All of the marshalling and unmarshalling of data is handled by SOAP and is performed in XML. RPC-style web services are tightly coupled and interface-driven. The clients invoke the web service by sending parameters and receiving return values. RPC-style web services follow call/response semantics; therefore, they are usually synchronous, which means that the client sends the request and waits for the response until the request is processed completely.(James Snell, 2001)

In Appendix B.1 there is an example RPC SOAP message.

1.3.2 Document

With a document style message the client and/or server passes an XML document as the body of the SOAP message instead of parameters.

An example of this is an invoice. The document is marked-up with both XML and a schema, which is common to both the sender and receiver. The sender is the consumer, which makes a function call to the web service with some parameters, and in return the vendor sends back, not data results, but an XML document that contains all of the information that a normal invoice would have, the difference being that it is marked-up with XML, and is machine-readable.

Document style messaging has other advantages over a remote procedure call, which is meant to be relatively static, and any changes to the RPC interface would

break the contract between the service and the application. Changing the RPC description would cause all of the applications that rely on a specific method signature of SOAP-RPC to break. Good design dictates that the method signature of an RPC message service should never change. With document messaging, the rules are less rigid and many enhancements and changes can be made to the XML schema without breaking the calling application, because what is sent is an XML document rather than a structured return value.

Web service applications should be able to use a guaranteed delivery mechanism to improve its reliability, scalability, and performance. Since a document message is usually a self-contained XML file, it is better suited for asynchronous processing and can be placed directly into the queue. The reliability of the application is improved because the message queue guarantees the delivery of the message even if the target application is not currently active, performance is improved because the web application simply delivers the document to a queue and is then free to perform other tasks, and scalability is improved because the document is offloaded to one or more instances of an application that handles its processing.(James Snell, 2001)

1.4 Description of a SOAP Message

A SOAP message is an XML document that has been standardised and agreed upon. Figure 1.1 shows the basic outline of a SOAP message. The root element is the envelope tag, which contains two more elements: body and header.

The body element provides a simple mechanism for exchanging mandatory information intended for the ultimate recipient of the message.(Don Box, 2000) Here, the message is contained in an XML format. The body element can contain an XML document or if it is an RPC request, it contains structured return data or arguments, or some fault for error reporting.

The header element is a sort of dumping ground for tags. The header can contain zero to many custom tags and was purposely left open for flexibility in future applications. Currently, tags with attributes such as the `mustUnderstand` are placed here. (see Section 3.1.7). As the SOAP message travels from server to server, these tags are

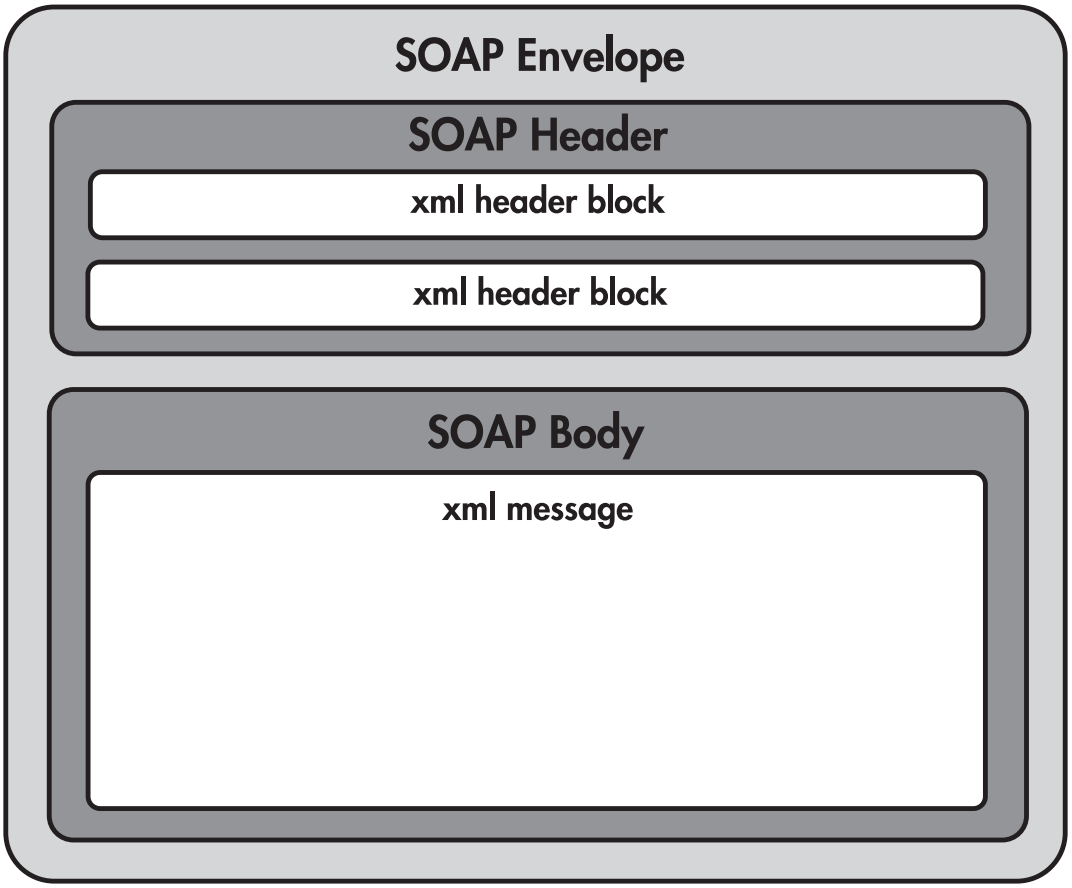

Figure 1.1: Diagram of a SOAP Message

read and possibly acted upon. Other tags in the header can be instances of transaction or session ID tags to create a state, although they can be anything. Thus, SOAP has the flexibility to deal with situations that have not been created or encountered yet.

1.5 Why are Web Services Interesting?

Web services are interesting for lots of reasons, in lots of different domains. The Internet gives you a web-sized library of possible components and services to use. It can tie islands of data, devices, businesses, and people together.

Today's web usage is browser-oriented. Users browse for information in vast

databases, and it is presented in user-friendly HTML displays. Once that data is rendered into HTML, it is very difficult to manipulate and use for anything other than display. Web services are not browser-oriented; they are more like websites with no user interface, webpages for machines. The data from a web service is returned in XML format, which is easy to manipulate and use for many things.

1.5.1 B2B Services

Business to Business services are at the heart of every company. Sharing data with your business partners has always been possible, but it was costly and time consuming. The set up of Virtual Private Networks (VPN), static IP's for specific businesses to access, or a complete systems of usernames and passwords for each business partner was a business in and of itself, but it allowed for special access to things like querying inventory, buying or selling parts at the discounted prices, or access to confidential information only available to select individuals. Inevitably, each company was using different accounting, tracking, and fulfilment software, which made passing and converting the data only slightly faster than retyping on each system.

Web services help by streamlining communications between all of the different data sources. Now any vendor can gain access to the information in a much more structured process. This allows smaller companies, which were previously not worth spending the time to set up and maintain within the previous system, to gain many of the same benefits as the larger companies. While a small business does not need the same large scale accounting packages, it is still able to interact with them through a SOAP interface.

There are still several hurdles that web services must considered before transactions, reliability, accountability, and security, make it successful. Other such hurdles that SOAP must overcome are mentioned later in Section 3.2.

1.5.2 B2C Services

Business to Consumers services allow for more interaction with end-users. If you have information that you want to share with a small group of vendors or partners, then

there are lots of “work arounds” that could be put in place. This was never feasible with consumers, because you could have many, many, more consumers than vendors or partners. It would be a nightmare to manage thousands of consumers accessing the information. With SOAP it is possible to create specialised functions that gives them access to specific data. This creates a paradigm shift in the way data is gathered, used, and accessed for consumers.

Amazon.com, for example, would be willing to spend a lot of money allowing partners and vendors to access their system. It does this because the cost of setting up this system saves them money in the long term. Setting up a similar system for consumers on an individual basis would certainly be too expensive for most companies. With SOAP, Amazon.com now has services for individual consumers, which allow anyone to query the Amazon.com database for books and other merchandise. Since this was accomplished with SOAP, Amazon.com avoided writing several different interfaces in different languages for different platforms. With the power of SOAP in the hands of Amazon.com’s consumers, it is now possible for individuals to pull data into their own websites and to build applications that searched Amazon.com in ways that were not previously thought of, thus increasing sales, and off-setting the initial investment of the creation of the SOAP web service.

1.5.3 Device to Device Services

Since the SOAP protocol is transport-independent, it is ideal for device to device communication as well. If you have to write software for a device that has several different ways of moving data including USB, Infrared, Bluetooth, TCP/IP, and more, the problem can be broken down into two simple steps. First, the software must be written to convert the message into a SOAP message, which will be sent over the transport protocol. Second, each transport protocol must be written, but instead of writing a proprietary marshalling format for each of the protocols and an unmarshalling format for each receiver, all the messages can be SOAP messages. This enables the programmer to write the data packaging code only once, which each of the multiple protocols will use that as the data encoding. Then each receiver will only have to be able to decode a SOAP message, instead of each proprietary format.

1.6 What SOAP is NOT

SOAP is not an end-all be-all architecture.

A major design goal for SOAP is simplicity and extensibility. This means that there are several features from traditional messaging systems and distributed objects that are not part of the core SOAP specification.(Don Box, 2000) Such features include:

- Distributed garbage collection
- Boxcarring or batching of messages
- Objects-by-reference (which requires distributed garbage collection)
- Activation (which requires object-by-reference)

SOAP was designed to be a simple message passing protocol. It will not solve every problem. Later, in Chapter 3, I discuss all of the strengths and weakness of SOAP and future plans for the protocol. After reviewing all the pros and cons, it will be evident that SOAP has its place for some activities; however, in other areas different protocols will excel.

SOAP is not the silver bullet that will revolutionise the way distributed systems are designed and it will not replace all other protocols, as some may say.

There has been a lot of hype about web services and what they can do. Some of the claims have not been fulfilled, but someday they may be. Until then, all of the options must be taken into consideration for each situation, and SOAP will be the best solution for only some of these problems.

Chapter 2

Alternative Distributed Systems

2.1 CORBA

CORBA is another distributed language created in 1991 that is used to access information across a network. CORBA 1.0 uses a language-independent way of communications, but the ORB that passes the requests to and from the client and server was left up to individual vendors to implement. This lead to confusion about CORBA and the misconception that it is has a closed protocol. With CORBA 2.0 a new protocol was introduced called Inter-ORB Protocol (IIOP) which runs over TCP/IP, among other network protocols.(Cohen, 2001)

The CORBA ORB can save the state of an object and make it persistent, which is something that is not yet possible with SOAP. If SOAP wants to support some state mechanisms and atomic transactional requests, new features will need to be defined and implemented.

Both SOAP and CORBA have open specifications and are not proprietary. The difference is that CORBA is a complete architecture whereas SOAP is only the message passing protocol. CORBA consists of both the encoding and transport protocols, where SOAP uses XML simply for encoding and is transport independent.

CORBA was designed for speed. SOAP was not, so the same information represented as a SOAP-RPC request is about four times the size of an equivalent CORBA message. The overhead time it takes to actually parse the SOAP-RPC message must

also be taken into consideration. Depending on the needs of the system, these differences can be either pros or cons.

SOAP-Document style is good for data that has a lot of depth but is loosely structured, whereas, CORBA is optimised for parameter passing, not document passing. SOAP-Document style can actually out perform a similar CORBA request, because of the structured document attributes of XML.

2.2 JAVA RMI

Java RMI is Java's Remote Method Invocation protocol. This is how Java implements a distributed system. Java RMI has an Interface Description Language (IDL) similar to CORBA, which is the equivalent of SOAP's Web Service Description Language (WSDL) document. The Java IDL can be generated from the Java code itself or written by hand. The IDL interface for a Java application is Java, making it easy to understand and use.

The major draw back to Java RMI is that it is a Java only solution. Both the client and server must be running Java. It is possible to make Java calls out to external code, but that still requires a Java wrapper class. This makes it difficult to interface with legacy or existing systems.

Java RMI is also only a client-server design paradigm and is not very well suited to peer-to-peer systems. As personal computer become even more powerful, the idea of P2P networks will only increase. This is something that does not affect SOAP.

Since Java RMI is a Java only solution you do get all the benefits and features of the Java Security Manager, which SOAP does not have. It is also possible to utilise Java's ability to serialize and deserialize objects, therefore gaining the ability to pass objects rather than just parameters.

Java RMI also uses its own `rmi://` protocol to transport message back and forth between applications, making it transport dependant.

2.3 XML-RPC

XML-RPC was designed in 1999 by UserLand as an XML-based RPC. The specification was written and has not changed since, making it a dependable protocol to use. XML-RPC and SOAP accomplish many of the same things in slightly different ways.

> XML-RPC is a Remote Procedure Calling protocol that works over the Internet.
>
> An XML-RPC message is an HTTP-POST request. The body of the request is in XML. A procedure executes on the server and the value it returns is also formatted in XML.
>
> Procedure parameters can be scalars, numbers, strings, dates, etc.; and can also be complex record and list structures.(Winer, 1999)

2.3.1 Similarities and Differences of SOAP and XML-RPC

XML-RPC is very simple, whereas SOAP can be as simple or as complex as needed. XML-RPC only supports basic variable types and custom types as structs or arrays of basic types. SOAP, on the other hand, supports basic types, multiple references within the same document, and user-defined types through the use of XML Schemas. SOAP is Unicode compliant, whereas XML-RPC is only ASCII compliant. To get around Unicode in XML-RPC, information is base64 encoded. This is how binary data is packaged in both protocols. With XML-RPC parameter passing, order is important. When executing an operation such as division, as XML-RPC message must have the variables in the correct order, otherwise, there might be an unexpected result. SOAP uses named parameters, so whether the numerator or divisor comes first or second is irrelevant. The remote application gathers the variables by name and correctly executes the procedure. Finally, the XML-RPC specification is only for XML over HTTP, but it is not impossible to conceive XML-RPC over another protocol. SOAP is transport protocol independent and can be used over any number of protocols.

Chapter 3

Advantages and Disadvantages of the SOAP protocol

Many of SOAPs's strengths are also its weaknesses. For SOAP to be simple, flexible, and extensible, it had to make trade-offs, such as the size of the message verse the ability to look inside it.

3.1 Strengths of SOAP

3.1.1 Heterogeneous Environments

SOAP is heterogeneous because of its ability to works on any platform, any operating system, in any computing environment, with any programming language, and over any protocol. It is this heterogeneousness that has made SOAP a popular middleware between legacy systems. A perl script client can invoke and retrieve data from a web service that is a front end to a mainframe system with the ability to extract data that was previously locked away.

There are a few large-scale real world applications of this technology. The University of California Berkeley began to implement web services for many of their legacy systems on a test basis in mid-2002.

On the front end, users were able to access either by dialling a single number or use an IP address from any device to receive their emails, voice mails, faxes, calendaring

and scheduling information. The system also gave callers the ability to get personal data our of back-end systems such as accessing medical or enrolment information, as well as class availability.

All of these systems existed on the university campus before, but as distinct silos of information. Web services made it possible to connect the existing networks into a single provisioning platform.(Schwartz, 2002)

3.1.2 XML-Based

By having an XML-based protocol you instantly get all the advantages of XML for free. This includes well-formedness, document structures, human readable text, and it is machine consumable. Web services have sometimes been called webpages for machines. None of this would be possible without XML.

XML is not only at the heart of SOAP, but also Web Service Description Language (WSDL) files, Universal Description, Discovery Integration (UDDI) registries, and other web service extensions.

It must be said that there is a debate about XML being the de facto standard for the SOAP message packaging. Since SOAP is so flexible at the transport level, it would seem logical to keep SOAP flexible at the encoding level as well. Constituents of other mark-up languages, such as DARPA Agent Mark-up Language (DAML)[1], have contested that XML should not be the only way to encode a message.

3.1.3 Platform Independent

SOAP is the message passing protocol and is not tied to any single platform. This is one of the points that add to the heterogeneous nature of web services. If SOAP were tied to a single platform it would impossible to make an RPC request to a server, which you know nothing about, and expect it to work. This allows a client and server, written in any language, on any platform, to be able to communicate. One issue that arises in dealing with different platforms and different operating systems is the size of basic variable types, such as; integers, floats, etc. SOAP avoids these problems by

[1]DARPA is the Defence Advanced Research Project Agency, they have been working on DAML aimed at facilitating agent interaction on the web.

using XML Schema to define some basic types that are built on. Knowledge of these basic types allows each platform the ability to interoperate by packaging the variables into the proper format and mapping them to their own internal type system. This is discussed more in Section 3.2.1.

3.1.4 Transport Independent

Since SOAP is just the messaging protocol it can be passed over any transport protocol. This includes both synchronous and asynchronous protocols such as HTTP, SMTP, BEEP, Jabber, and others. Allowing SOAP and web services the flexibility to adapt to the future of the Internet. As protocols are replaced, upgraded, or deprecated SOAP will find new ways to continue sending message across the Internet.

A good analogy of transport independence, and a SOAP message, comes from the real world post office. When you write a letter to a friend you both agree on the protocol, in this case the English language. In the case of SOAP the XML Schema relating to the SOAP specifications. After you have written your letter you place it in an envelope and address it to the receiver. With SOAP you have constructed the XML document and you have the URI to the destination, which is the vendor's service. You are not concerned directly with how your message gets to the end point, just so long as it does within certain parameters: time, cost, etc. With your physical letter you drop it into the mailbox and wait for the post to deliver it. The post could travel from the office to the sorting station by car, and from that location by train or plain or truck to the next stop, where it is loaded to another vehicle. This repeats itself until it reaches the final destination. Your letter is transport independent because it can be carried by any number of protocols, in this case trains, plains, automobiles, and/or pony express. SOAP messages work in the same way, part of the message may travel over your local network, from there it might travel wirelessly, then switch protocol and switch back before finally reaching the vendor's service. Since you are not concerned about and cannot always control the infrastructure between you and the receiver, this design allows for the most universal way to communicate between the consumer and vendor.

3.1.5 Plain Text Packets

SOAP messages are sent in plain text with a special header in version 1.1 of

```
Content-Type: text/xml
SOAPAction: ``http://example.com''
```

and in SOAP version 1.2 this has been changed to

```
Content-Type: application/soap+_xml; action=``http://example.com''
```

Making this change allows firewalls and packet sniffers to look into these messages to verify their safety and validity. This transparency is one of the major advantages over other distributed protocols. Network administrators were never able to look into a CORBA message, subsequently; they became potential security risks.

On most networks the availability of ports is limited by the firewall. The HTTP protocol is an older safer plain-text protocol that the firewall can parse and manage based on content-types. This makes HTTP a good candidate for SOAP to use as the transport level. The HTTP port is open on most firewalls for messages to pass through and with the new content-type it is possible for firewalls to flag SOAP packets and examine them more carefully if necessary. The ability to examine the packet is possible because SOAP messages are plain text, if they were proprietarily encoded there would be no way to determine if the packet in question posed any security threat.

3.1.6 Interoperability

SOAP's ability to work between platforms and operating systems allows for a great degree of interoperability. Many companies are using SOAP to connect to legacy systems, pulling the data out as SOAP messages that can be send and consumed by other applications. SOAP becomes the glue between different interfaces of applications that never could previously interoperate. It is capable of doing this because the data is encoded in XML.

XML is basically a Unicode text file with structured mark-up. Since ASCII is a standard across platforms and XML Unicode can be represented in ASCII, it therefore can be read over multiple platforms, increasing interoperability.(Martin Dürst, 2003)

3.1.7 Must Understand

The SOAP mustUnderstand global attribute can be used to indicate whether a header entry is mandatory or optional for the recipient to process. Tagging elements in this manner assures that this change in semantics will not be silently ignored by those who may not fully understand it.(Don Box, 2000) This attribute prevents messages from getting passed to the next node; when in fact the current node did not understand the tag and did not do any work. If a node gets a SOAP message with a tag that has a mustUnderstand attribute that it does not understand, the message is not forward and a mustUnderstand fault is sent back to the client. Since the header of a SOAP message is where additional information can be placed there needed to be away to determine if the node could understand and process these tags. If the node was unable, then a fault needed to be thrown so that the message is not passed along or acted on accidentally.

A typical entry in a SOAP header might be a sessionID for a login. This is an ideal candidate for the mustUnderstand attribute. If the receiving node did not understand the tag with the sessionID and continued to complete the login, this is a potential security risk and can cause fatal errors. With the mustUnderstand fault this node could return an error instead of unknown data.

3.1.8 Just In Time Discovery

Just-In-Time (JIT) discovery and binding is the ability to build ad hoc applications from smaller pieces. Instead of “hard-wiring” remote applications together, which require high costs across all applications during maintenance, applications can be “soft wired” or “loosely coupled”. This is the ability to find and bind to services in a dynamic, ad hoc fashion. When a service gets too costly in terms of time or money, others can be found, replacing services and providers. Replacements are possible because of the open standards and wide adoption of web services. When applications are “soft wired” and mobile, JIT becomes an important way to find services and use them with little hassle. SOAP accomplishes these things through the use of a service description language (see Chapter 4) and discovery services (see Chapter 5).

To review how all three parts make up a complete Just-In-Time web service, refer

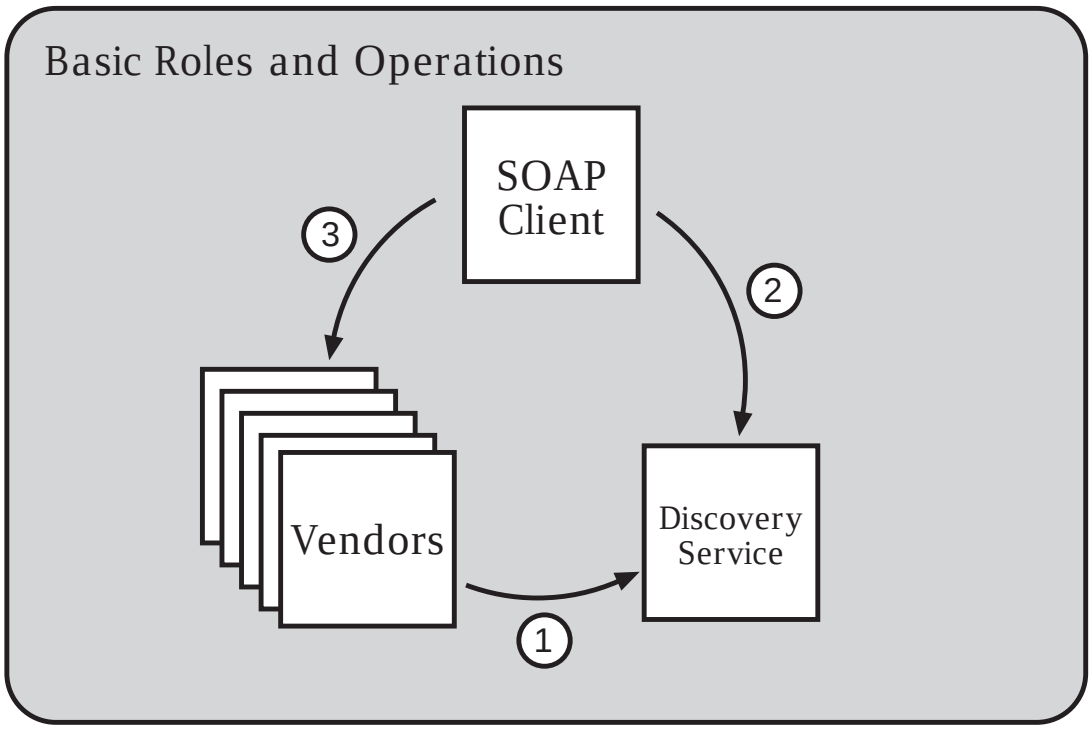

Figure 3.1: The Three Parts of a Web Service

to Figure 3.1. In Step 1, the vendors publish their available services to some sort of central repository. This could be a UDDI server or another commonplace system that describes services to vend. In Step 2, the client application queries the central server for a list of vendors that offer the service needed. In Step 3, the service is invoked from the client to the service vendor. At some point the WSDL file was read to get all the information about binding to the web service. The WSDL file could have been retrieved from the vendor before Step 3 and after the URI was sent by the discovery service in Step 2, or the central server could have also supplied the client with a copy of the WSDL during Step 2.

3.1.9 Robustness

Even though SOAP is designed for simplicity and flexibility, it is more robust than other XML-Based systems, mainly XML-RPC. SOAP allows you to declare other

namespaces in your document, something XML-RPC cannot. With this it is possible to add an XML Schema, which can define custom data types.

SOAP has the ability to scales to larger applications, both in transport and messaging. Since SOAP can use the HTTP protocol as the transport layer, it has the ability to scale with the size of the network. The message protocol is also scalable to larger applications, because of XML Schema custom data types, the extensibility of the header tag, and the ability to layer other XML technology onto SOAP.

All the other strengths of SOAP: Just-In-Time, mustUnderstand, Plain Text Packets, and others all add to the robustness of the protocol.

3.2 Weaknesses of SOAP

3.2.1 Big-endian, Little-endian Issues

SOAP must define some common data types such as string, integer, date, etc. These data types are not always the same between different programming languages and different operating systems. To get around this, SOAP allows for custom data types through the use of XML Schemas. This is a strength because it allows for the construction of custom types, but is a weakness because it also adds overhead, confusion, and creates potential problems of implementation.

3.2.2 Packet Sizes

SOAP's strength in having plain text messages becomes a weakness when dealing with size. A SOAP message has a lot of overhead because of the XML and the protocol design. This can become troublesome when dealing with lots of data because of bandwidth limitations or throughput speed of the network. An equivalent CORBA packet is a fourth the size; so if you are using the full bandwidth using a CORBA application, expect to use four times that when switching to a SOAP environment.

Along with larger messages come the issues of marshalling, unmarshalling, and parsing through all that XML once received, and it is not just the sender and receiver that have to do work, it is all the nodes in between. This all adds overhead and delay

to the protocol.

3.2.3 Implementation Issues

SOAP is currently at version 1.1 and version 1.2 is currently a draft. Different vendors interpreted early specifications slightly differently. For Instance, Apache and Microsoft's .NET both implement a `BigDecimal` data type. However they are not compatible.(Cohen, 2001)

Today things are much better and interoperability has been restored, but with new versions of applications and SOAP, there are always potentials for some disagreement.

To help combat vendors from interpreting the implemented standards in incompatible ways, or only partially implementing the standard, the Web Service Interoperability Organization (WS-I) group was formed. They have three primary goals to improve web service interoperability:(Organization, 2002)

- Provide implementation guidance and education to help customers with Web services adoption.
- Promote consistent and reliable interoperability among Web services across platforms, applications, and programming languages.
- Articulate and promote a common industry vision for Web services interoperability to
  - Ease customer decision making,
  - Grow industry adoption of Web services and
  - Ensure the continued evolution of Web services technologies.

3.2.4 Security Issues

SOAP alone does not support any sort of security. The common thing to do is to send the data over HTTPS, but this only secures your data at the network level and avoids addressing the bigger issue of securing content. Currently, there are a few drafts with the W3C for secure XML protocols that can be used within SOAP.

Some of the security and privacy issues being addressed are accessibility, confidentiality, authentication, authorization, data and content integrity, and non-repudiation. (Clabby, 2002) All of which are being dealt with by the W3C, because they realise that these element are crucial to building a secure computing environment. To achieve a secure, trustworthy environment, the WS-Security specification has been brought to Organization for the Advancement of Structured Information Standards (OASIS) to address these shortcomings. OASIS is also handling several other security related web service specifications including a web service endpoint model (WS-Policy), a trust model (WS-Trust), secure conversations (WS-SecureConversation), authorization (WS-Authorization), a privacy model (WS-Privacy), and federated trust (WS-Federation) specifications.

The header section of the SOAP message envelope is commonly used to stick authentication information, unique identifiers, but this information is passed as plain text during a normal run of the protocol and can not be guaranteed to be secure.

None of this prevents web services from serving up weather data, calendar information, or other data, but security is very important before web services can be adopted in enterprise applications.

Web services, in their current state of support, are an excellent application integration technology. Web services can glue together applications running on two different messaging product platforms, enable database information from one application to be made available to others, and enable internal applications to be made available over the Internet. Web services can also be used between two business partners who already have business agreements in place.

For point-to-point applications, SSL is already being used for security functionality. Web services are being used to offer interfaces to applications that were once browser-driven, like Google and Amazon.com.

Web services are also being developed as utilities, or pay-per-use program components, like the auto buyers VIN history service that returns the registration and salvage history of a vehicle. One benefit of this model is that its infrastructure can borrow from the experience of the telephony utilities industry, especially on user-driven service provisioning, usage tracking, and billing. Web service deployments have shown

that existing assets used within a company can readily become revenue-generating assets.(Kreger, 2003)

3.2.5 Versioning Issues

SOAP does not define a traditional versioning model based on major and minor version numbers.(Don Box, 2000) As SOAP matures and new versions are written there is currently no field, attribute, or tag to describe which version of the SOAP specification you are implementing. This will become a problematic as parts of the protocol are either depreciated or new features added. The consumer and vendor will have to make sure they agree on the correct protocol version before message passing, so they understand the complete messages each are receiving. If the consumer and vendor are using different versions, the SOAP application MUST respond with a `VersionMismatch` faultcode message, as described in the W3C recommendation.(Don Box, 2000)

3.2.6 Message Path

With the current version of SOAP there is no way to specify a path the message should travel. As with normal Internet traffic, the message is routed over the network according to several factors, including speed and availability. It is important to understand what routes the messages are taking. This improves reliability if you can mandate the path the message must follow through reliable servers. If the path is known, faults are easier to detect and diagnose. If a transaction is several messages long and a single message is lost in transport the whole operation somehow must be rolled back to a previous state. With the message path known, it is possible to create an audit trail of the individual message, therefore giving both the consumer and vendor a higher level of trust.

WS-Reliability and HTTP-R are attempts at once and only once reliable messaging. The HTTP-R specification has been available since July 2001, but has yet to make its way into any sort of daily use.

There are attempts to route SOAP messages through a known system of nodes, each of which have some previous knowledge of some of the actions intended. This

is important because of the `mustUnderstand` (see Section 3.1.7) attribute for header elements. It is possible to define a tag called "logging" and set it to `mustUnderstand`. Therefore, every server that is in the message path must implement "logging" otherwise the message is sent back with a fault. If you could specify the path the message travels, then there are less worries that a single node in the chain will throw a fault because it is unable to understand or implement a tag.

3.2.7 Latency

Latency is a problem that distributed systems are not able to get around. Low-bandwidth connections, congested routers, and overloaded servers have always been problems, but web services have yet another bottleneck with the SOAP XML parser. As a SOAP message is passed from node to node in the chain, each node is responsible for looking at the SOAP header and implementing any tag with the `mustUnderstand` attribute. This also adds the overhead of parsing each message and possibly acting on it before passing it along to the next node.

Whenever you are using part of a network that you are not in control of there is no guarantee of the quality of the service. This is not a unique problem to SOAP as a distributed environment, other implementations, such as CORBA, have optimised their messages for speed to off-set latency, where as SOAP optimises for openness and interoperability.(Asaravala, 2002)

3.2.8 No Objects

The "O" in SOAP originally stood for "Objects", but with version 1.2 the acronym was dropped. It is not possible for SOAP to pass a reference to a remote object as a parameter, as it is with some other distributed languages. This solves a lot of programming problems for developers of applications that manage SOAP requests, because they do not have to deal with things like distributed garbage collection. This trade-off does limit some of the capabilities of the client application. Without objects it is difficult to get persistence, or for multiple clients to access the same object, rather than multiple instances of the same object. Not supporting distributed objects was something that

was decided in the SOAP recommendation, so that it could be a simple and extensible language.(Don Box, 2000)

3.2.9 Reliability and Trust

When you are letting an unknown source tell you some fact, you must weigh the reliability of that source before believing the fact yourself. Even with the simplest of examples, getting the current temperature, you must assign some level of trust to the vendor. With the get current temperature example, the vendor could simply be guessing and the consumer would be none the wiser. When web services become more important, such as; searching for the cheapest bank loan, stock quotes, logging reservations, or other transactional requests, you are assigning a level of trust to the vendor that they will be supplying the goods, services, or merchandise promised. There is a potential to lie when the vendors are not the same style brick and mortar business and cannot be as easily found.

Trust also ties in with security. Do you trust the vendor you are dealing with to not sell your data or to protect their system from being broken into and your data stolen?

3.2.10 Ontology

XML itself does not have any sort of ontology. This is not necessarily a bad thing, because the code is lighter and less complex. Problems do occur when the tag names could be interpreted in more than one way. With a proper ontology, terms are defined and scoped with a namespaces to fully define each tag name so no ambiguity exists. Terms such as; title, size, cost, etc. all could be interpreted in different ways. Title could be interpreted as a prefix to a name (Lord, Lady, Sir), it could be a property deed, or the name of a document. XML has no inherent way to define the actual meaning of the tag name. This is why the XML Schema must be agreed upon before the transmission, so both parties know what type of document to expect and to better understand the values that are being passed.

3.2.11 Statelessness

SOAP messages are fundamentally one-way transmissions from a sender to a receiver, but SOAP messages are often combined to implement patterns such as request/response. (Don Box, 2000) SOAP messages can use HTTP, which is by design stateless; to hold some sort of state between individual SOAP messages something must be added to the SOAP protocol. This is where the W3C has left the header field open for additional tags to be added, possibly to contain some unique identifier or sessionID that will be used to represent state between the client and server. Figure 1.1 shows a SOAP message how the header tag is associated with it.

Chapter 4

Service Description

4.1 Web Service Description Language

Web services can be simple or complex, so a standardised way of expressing all the different types of services needed to be drafted. It is possible to describe a web service by creating a well-written document that is publicly accessible that written in prose, about how to implement and bind to the service. This is adequate, but it does not allow Just-In-Time services to be invoked. Web Service Description Language (WSDL)[1] was created to solve this problem.

4.2 What is WSDL?

WSDL stands for Web Service Description Language. It is an XML file that describes the technical details of how to implement a web service, more specifically the URI, port, method names, arguments, and data types. Since WSDL is XML, it is both human-readable and machine-consumable, which aids in the ability to call and bind to services dynamically.

[1] WSDL is currently a W3C note, version 1.1

4.3 Describing Interfaces

Most distributed system programming languages have some sort of interface description language. CORBA has IDL, Java has its interface, and web services have WSDL. With these interfaces it is possible to generate client and server stubs around which to write the code. There are several programs that will take a WSDL file and generate Java stubs to use in a project. This helps maintain continuity, prevents errors between the vendor and consumer code, and speeds the development time for creating distributed applications.

4.4 Description and Service Mismatch

Since a WSDL file simply describes the service and is separate from the code, it is possible that the service could change and not be reflected in the WSDL file, causing problems when the service is later invoked. Some web service programming environments try to prevent the WSDL file from getting "stale" by generating the WSDL dynamically from the code itself, thus preventing errors by not divorcing the two files. Until all programming environments generate the WSDL dynamically, there is a chance for a drift between the WSDL file and the web service.

4.5 Description of a WSDL Document

A WSDL file is made up of several different sections, each describing a part of the interface for a web service. Figure 4.1 shows the format of a WSDL file: the items with a (?) are optional and the items with a (*) can contain zero to many entries.

The WSDL file is an XML file with a definition tag as the root element. The next possible element is the import tag. While this approach is not yet fully recognised by all WSDL-enabled tools, the import tag would allow you to point to an external XSD file. This proposed XML Schema would replace the types tag in describing all the elements. Currently, it is common to embed the XML Schema into the WSDL document, thus not requiring the import tag to be used.

The next tag is the types tag, which defines user specific types. For example, in Appendix D the WSDL document to describe the BibTeXDB service contains a user-defined type called `BibTeXDBResults`. This new type is equivalent to a "C" structure, or a Java Object, and it contains all the values for a BibTeX citation as strings. There is a second user-defined type to define an array of the `BibTeXDBResults`, which is what is returned by the search function.

The message tag binds a user-defined type or standard type to a variable. These become the inputs and outputs for the operations tags in the portType tags. The portType tags can have zero or more children of type operation. The operation tag is equivalent to the name of the function that resides on the server. The operation tag, then, has input, output, and fault tags that define the arguments to the function and what is returned from the RPC.

The binding tags looks as if they are repeating much of the same information as the portType tags do, but there is additional information not relevant to the input and output. This additional information is in the SOAP namespace and describes the transport; in the case of Appendix D, HTTP encodes the style of the variables, the style as RPC or Document (see Section 1.3.1), and adds a Universal Resource Name (URN) for each operation.

Finally, the service tag contains information about the location of the actual service. There is a child tag called port, which is not the same port as TCP ports, but that is so called for the binding to the binding tag. The port tag also has a child tag called address that is in the SOAP namespace. This address tag has a value of the Universal Resource Indicator (URI) of where the service is located.

With all of this information it is possible to automatically build stubs for Java classes or to even to invoke the service dynamically.

4.6 IBM WSDL Only Client

IBM and others have created applications that can call simple web services without ever writing a single line of client code. This works by invoking their application and passing a WSDL file as one of the parameters, along with the web service function

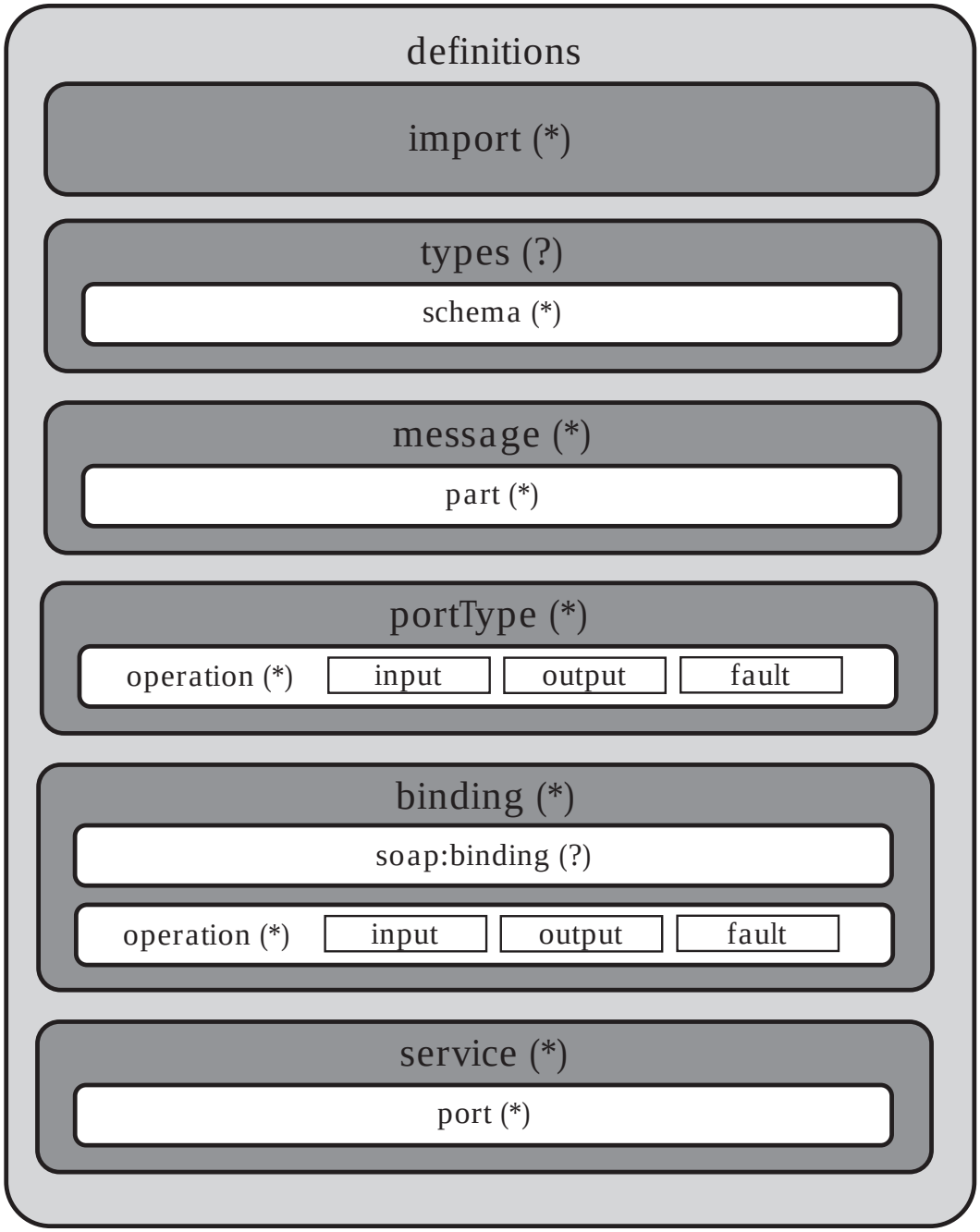

Figure 4.1: Sample WSDL Document

name, arguments, etc. Since the WSDL file contains all the information about where to locate and bind to a service, the application can parse the WSDL file and make a connection to pass a custom made SOAP message. The WSDL file contains all the function names and data types for the service, so the application is able check that the function and parameters being passed are valid. If all the data validates, the application can open an HTTP connection with the remote web service, then create and send the custom valid SOAP message based on the original parameters. The remote web service executes and returns the correct data type. This too, is expected by the application because the WSDL file defines all the return types as well. The returned data is then displayed for the user to view or logged to a file. By creating this application, there is no need to create any client code that is specific to a given service. The same application can be used to call any SOAP web service saving time and money in rewriting client code.

4.7 BibTeXDB WSDL

For my project I wrote the WSDL file that can be seen in Appendix D. It lists the three main functions that are available (Add, Edit, Search) and what arguments are passed to each, along with all the available information about the service location.

I created a Java class that contained variables and functions related to a BibTeX citation. To represent this class object in WSDL, I had to use the ComplexType feature along with the SOAP array feature to build a representation of the data that is passed to and from the server and client.

4.8 WSDL's Flexibility

My project did not use some of the more advanced features of WSDL. Since the WSDL file itself is written in XML, it is very flexible and extensible. One of the many things possible is operation overloading, the ability to define the same function name with different parameters, and defining new variable types through XML Schema.

Operation overloading is common in higher level programming languages. This is

where multiple functions or mathematical operators have the same name but different arguments. It is up to the compiler or runtime engine to decide which function to use; in the case of Java, it is based on the function signature.

The ability to create new data types in SOAP is crucial. WSDL does not aim to create a standard for XML data typing. Instead, WSDL is specifically designed for maximum flexibility and not tied exclusively to any one data typing system. This allows for extensibility, interoperability between different operating systems and platforms, and more complex data types such as structures and arrays.

This flexibility does come at a cost: the WSDL file becomes very complex and verbose, seemingly defining things several times, and becoming less human-readable. WSDL files are not easy to code by hand, even for simple services. As time progresses, better toolkits will become available to generate WSDL files automatically rather than by hand or the WSDL file will be generated automatically from the code itself.

Chapter 5

Service Discovery

5.1 How to Discover a Service

As part of one of the three steps in Just-In-Time computing, discovery has the most options. Invocation is conducted by your application via some transport, most likely HTTP. Description is handled by the WSDL file, but the question remains, how to find that description and its service? Several ways have emerged to do this, but it is not clear which will be the most effective. The first is Universal Description, Discovery and Integration (UDDI). IBM, Microsoft, and a few others with the intent of setting up a hierarchy to discover services designed it. The second is Advertisement and Discover of Services Protocol (ADS), which is distributed, much less structured, and relies on consistency. Thirdly, WS-Inspection is another option for discovering web services. It describes all the different methods of gathering the service description and lets the client to choose the best option.

5.2 BibTeXDB Discovery

My project is designed only for the COGSCI department and was never meant for worldwide consumption; therefore discovery is outside the scope of this project. Discovery, in this instance, would be word of mouth, which is another valid kind of discovery, but does not contribute to Just-In-Time computing. Since discovery is an important

part of all distributed environments it is worth mentioning the different ways in which discovery is handled for web services.

5.3 UDDI

UDDI acts like Domain Name Server (DNS), but instead of resolving names to locations, it resolves names to services or vendors. UDDI is not affiliated with the W3C, but it has been submitted to the OASIS standards body for review. Several large companies including Microsoft and IBM currently run UDDI registers, but it is possible to set-up a private UDDI registry for your own company's intranet or extranet. The listings in the registry can therefore be public or private, depending on how the UDDI registry is published.

A positive aspect of UDDI is the definition of an ontology for web services. A problem with UDDI, is that it is centralised by design, both in the single tree ontology and in the design based fundamentally on a central registry, with inter-registry operation as a secondary thing.(Berners-Lee, 2002) Taxonomies, ontologies, and identifier systems play an important role within UDDI. It is often through categorization and identification that searchers of UDDI find the businesses and services that meet a particular need.(Daveid Ehnebuske, 2001)

When you search a UDDI register for a web service, you do so using SOAP messages. Hence, you use a web service to find web services. Since the UDDI registry is a web service, it gains all the benefits of XML, SOAP, and Just-In-Time computing.

The UDDI registry is broken down into three main parts, or pages. These pages each perform a different function; their names are White Pages, Yellow Pages, and Green Pages.

5.3.1 White Pages

The UDDI white pages follow the traditional real world equivalent. They contain basic information about the business such as name, telephone, URL, other basic contact information and an overview of the key services they provide. This is how the publisher

of a web service register's itself to the database of vendors. This allows consumers to search the UDDI database for matching vendors by name or location.

5.3.2 Yellow Pages

Yellow pages are categorised based on existing business taxonomies of classification, including where the business operates. This allows consumer applications to search for types of web services in the UDDI database for use in Just-In-Time computing.

5.3.3 Green Pages

The green pages are designed to contain technical information about the web service and how to interface with it. This was designed without the use of WSDL and attempts to achieve the same tasks, but what regularly happens is that the green pages become a pointer to a WSDL file.

5.4 Advertisement and Discovery of Service Protocol

UDDI requires the vendor to update the UDDI registry with the necessary information, and certain checks and balances must be put in place to prevent you from editing other vendors' information. ADS works on the opposite principal. Similar to web crawlers, a service crawler would scour the Internet for predefined files that you generate and control on your own website. `Robot.txt` files are what web crawlers use to spider a website, while the web service crawlers will look for an equivalent file called `svcadvt.xml` in the root directory of a website. This file advertises all services that are offered and allows the crawler to compile a centralized list of services that can be searched similar to any current website search directory. The other option is to place a new meta tag into your HTML that points to the WSDL file.(Vasudeva, 2001)

```
<meta
  name="serviceDescriptionLocation"
  content="http://example.com/bibtexdb.wsdl"
/>
```

The different methods of advertisement are relevant to different situations. A search engine or portal may make use of both advertisement methods in different ways. The `svcsadvt.xml` file provides a consistent starting point for performing a search for services provided by a site, while the second method may be used during the standard indexing of web pages, allowing for the categorization of services based upon contextual information.(William A. Nagy, 2001) With either method the web service crawler will find your advertised services and save them back to its central database. This will create a sort of search engine for web services that is updated with each crawl, whereas the UDDI registry is only updated when the vendor change their entries. ADS has also been suggested as a way to keep the UDDI server up to date by feeding the results found on the website back into the registry for each corresponding vendor.

5.5 Web Service Inspection

A WS-Inspection Document is another client-based way to advertise services. It is a single starting point to an aggregation of pointers to multiple service description documents. A WS-Inspection documents is an XML file that is fairly easy to write and maintain. It is just a pointer which may point to a variety of service description document formats. WS-Inspection documents may be created which allow for the consumer of the document to pick and choose from the available descriptions and to access only those that it is able to understand. As new description formats arise, new references can be added to existing WS-Inspection documents without requiring a modification to be made to the base WS-Inspection schema.(Keith Ballinger, 2001)

The WS-Inspection Document in Appendix F outlines two different ways to access the service description. The first description is for the WSDL file that describes all the functions, arguments, and ports needed to invoke the service. The second description is for a UDDI registry. With this information you can determine the service vendor and information about the service from the registry's yellow, white, and green pages.

It will be a matter of time before any one way of service discovery is widely accepted. Not only are there technical hurdles, but social hurdles as well.

Chapter 6

MSc Project Description

6.1 BibTeXDB Description

My project was to design and construct a BibTeX database for the COGSCI department that is accessible using SOAP web service protocol. The project was written in Java and uses HTTP as the transport mechanism. Having this as a centralized service will reduce redundancy and allow users to search for other BibTeX sources that someone else may have entered into the database.

The Java client has three main functions, the ability to query for citations, add a BibTeX citation, and edit an existing BibTeX citation in the database through the web service.

The Java servlet application that is used to vend the web services uses an XML file to store all the BibTeX citations. This is transparent to the consumer and Java client because SOAP is just the encoding. It acts as the middleware and is independent of the underlying methods of data storage and retrieval.

A Web Service Description Language (WSDL) file has also been created so the client application knows the available functions and parameters. With this document it is possible to generate skeleton code and stubs for the distributed service, or to invoke the service dynamically.

This project was taken up in Java, but it is possible to demonstrate the ability to make web service requests through a heterogeneous system by the use of additional

languages such as perl, for a simple client.

6.2 BibTeXDB Web Service

Figure 6.1 shows the flow of data through the BibTeXDB web service. Step 1 is when the client application sends a SOAP message to the Tomcat web server listener. Once the web server gets the message it knows that it is a SOAP message from the content-type and passes it to the Apache SOAP module to parse the XML data. The Apache SOAP module parses the message and determines all the arguments, the function to call, and which class to invoke. Step 3 is the act of actually calling the Java class and passing the arguments to the proper function. Between Steps 3 and 4 is some of the Java code that I wrote for the project. This code accesses the XML database and Adds, Edits, or Searches for entries. When the function has completed, it returns the proper return values back to the Apache SOAP module in Step 4. The module then marshals the data back into a SOAP Response message, or a fault to be returned to the client. A sample SOAP Response message can be seen in Appendix B.2. This message is passed to the Tomcat web server to be sent back to the client as Step 6. At this point the original message has been received by the server, parsed, handled, and the results returned to the client so that it can do something with this data. In the case of this project the client application will do some sort of reporting, either by telling the user that the data has been successfully added or edited, or in the case of searching, the results are printed to standard out.

The portion in the larger grey box is all part of the remote server that the vendor manages. The client is the other part of the diagram and can be replaced by any number of different or multiple clients. This is what gives SOAP such power in the distributed application field. The ability to have clients of different types, languages, or platforms is something that is not possible with other distributed implementations.

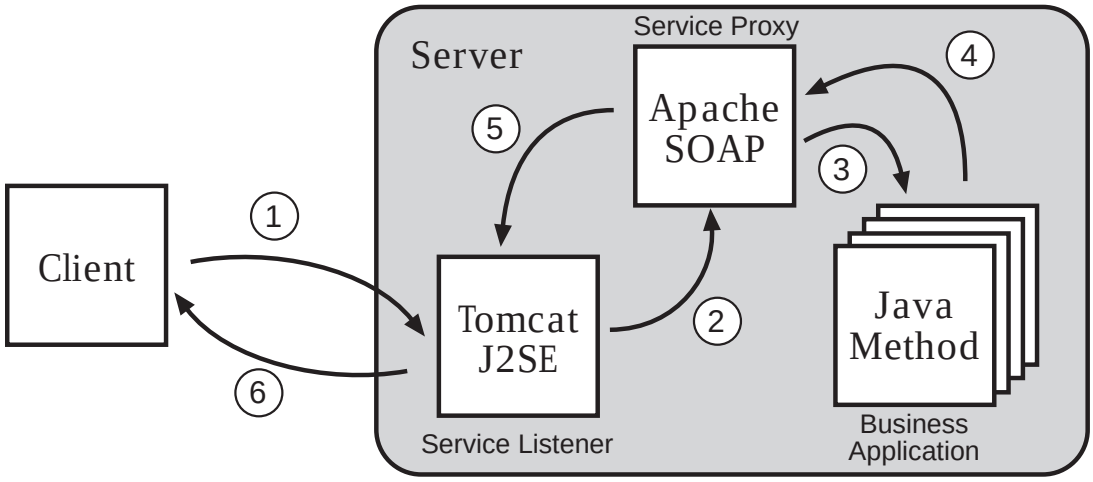

Figure 6.1: Diagram of the flow of data through the BibTeXDB Web Service.

6.3 Design and Implementation

Appendix E shows the interface for the client application, it takes all the different BibTeX types as parameters for an Add or Edit function call and returns feedback about the function's success.

There is a second command line program which allows the consumer to search the database for relevant entries. The search takes a query string and the results are printed to standard out.

The COGSCI department had a BibTeX database of entries that numbered close to 20,000. This BibTeX database was taken and converted into an XML file that the Java server servlet could reference as a database. After converting the file, it was validated using RXP(Tobin, 1997) to check for any errors or anomalies. The final XML file has a BibTeXDB tag as the root element and one or more citation tags as children. Each citation tags must have one key tag for unique identification. There are several optional tags that correspond to all the possible BibTeX types that may also be appear once as children of a citation tag. The following is a citation from the database:

```
<bibtexdb>
...
<citation>
  <entry>InCollection</entry>
  <key>9138</key>
  <address>Amsterdam</address>
  <author>David N Lee</author>
  <booktitle>Tutorials in motor behaviour</booktitle>
  <editor>G E Stelmach and J Requin</editor>
  <publisher>N Holland</publisher>
  <title>Visuo-motor co-ordination in space-time</title>
  <year>1980</year>
</citation>
...
</bibtexdb>
```

The resulting XML file is quite large, and will only to grow with use, so I choose to parse the XML with a SAX parser. Since the XML file is working as a database a unique identifier was added to the BibTeX XML citation called `Key`. This `Key` value severed several purposed. Since it was just an incremental integer, it prevented any collisions from people trying to add an entry with the same value. It also gave a unique identifier to search on or edit at a later date.

There are two different ways to parse XML the first is to use a DOM tree, which reads the entire XML file into a tree structure. DOM trees are great for moving through the tree, but require lots of memory to hold each node and links within the tree. A SAX parser works by only reading one tag at a time and not holding state or knowledge of the location within the tree.(Peter Murray-Rust, 2000) SAX is much faster than DOM because of this, and since the service would be constantly searching through the database for matches, speed is the most important factor.

SOAP-RPC style encoding was chosen for all three functions. It is intuitive to use RPC for the Add and Edit, because both simply return integer values for the new key and error code respectively. The Search function could have been implemented as a Document style encoding, but was not for several reasons. Document style encoding is best for asynchronous connections, but the BibTeXDB uses HTTP, which is synchronous. It does make some sense for the search function to return an XML document of all the search results. This would actually be smaller than the current

return values. Since `BibTeXResults` is a defined data type there are empty tags being returned in the array. (See Appendix B.2) With Document style those empty tags could be omitted because they could be set to optional in the XML Schema. With the extra overhead of passing empty tags the client code can automatically put the SOAP response directly into local variables, whereas if an XML document where returned that would have to be parsed and then put into local variables. Thus moving the work from the server back to the client. An advantage of parsing the XML document locally is that if it were to change in the future such as; adding more tags, this would not break the client. If would either omitting these tags or use them properly, but it won't break the application. The current client that is waiting to receive the SOAP-RPC response would not be able directly converted the new XML tags into local variables, it would break and throw an exception because it would not be able to unmarshal these extra tags into the old class.

Besides returning a XML document to the client to parse, being a synchronous connection, and having a well-defined BibTeX vocabulary; SOAP-RPC makes a better encoding style for the search function and an open document format.

6.3.1 Searching

Searching is the bulk of the operations that the web service has to perform and this is where most of the effort was spent.

XML has several special characters that need escape sequences, specifically the ampersand (&) and the less than greater than sign (<). The less than sign is escaped by the sequence < since it starts with the ampersand that too must be escaped to &. When converting the BibTeX database to XML these characters had to be changed so not to break the XML, but needed to be returned as their original form when printed as a BibTeX citation. This required a bit of pre-processing and post-processing avoiding any complications.

The next big problem with a BibTeX citation is that it has the ability to define special characters by surrounding the letters with brackets ({X}). This needed to be stripped out during searching to see if the query string matched any of the text and put back into the output if it did match.

The searching algorithm implemented is rather simple, both the query string and the XML database text are converted to uppercase, all special BibTeX command sequences are stripped out, and special XML characters are replaced. Then the query string is compare to each of the possible elements in a BibTeX citation; author, publisher, notes, etc. If a match is found, the BibTeX citation, in its original form, is placed into an array, which is returned to the consumer after the entire XML database file has been searched. If you search for the term "XML" you will get results if the title of the entry contained the letters XML, and/or XML appeared in the notes, or in any other field. Currently, it is better to error on the side of returning too many results rather than being too restrictive.

6.3.2 Java Beans

Earlier in section 3.2.8, it was discussed how SOAP does not use Objects, but my web service passes a `BibTeXResults` object to the vendor and as a return value, gets an array of `BibTeXResults` objects. To pass anything besides the basic data types in Java to the web service, you must write your own Serialize and Deserialize functions. These are functions written that convert a class or object into XML and back. So it is true that SOAP does not pass objects, but instead passes representations of the data in the objects, so that it can be used to create new objects. Apache SOAP comes with a built-in serialization class called `BeanSerializer`. This class can take a Java Bean and automatically serialize and deserialize it. In Appendix C the XML Deployment Descriptor has a tag called mappings. This tag describes the Java Bean, the URN, the Qualified Name, and the Serialize/Deserialize classes that should be to convert the Java Bean to XML and back. For this project the `org.apache.soap.encoding.soapenc.BeanSerializer` is used to convert the Java Bean.

My `BibTeXResults` class was converted into in to a Java Bean so that it could be automatically serialized and deserialized by the web service listener. To turn the class into a Java bean, accesser functions were added to `getVariableName()` and `setVariableName()`. Other accesser functions were written to format the private variables of the Bibtex Java Bean to output to XML, plain text, and BibTeX format.

This allows for the results array to be outputted to standard out.

For this project, the client and server were both written in Java, but one of the advantages of SOAP, is that it is language independent. Even though Java Beans were used in this project it is not necessary that the client implement them. The WSDL file describes the object `BibTeXResults` as a set of Strings, which can be implemented in any language. The Serialize function for the Java Bean creates an XML representation of the Java Bean; this XML can be consumed in another language as a structure, list, array, or other data type specific to the language.

6.4 SOAP Web Service Listener

The Java class that provides the web service is running in a Java servlet container, through the use of Tomcat's J2SE environment. With technology like Common Gateway Interface (CGI)[1], each server request creates a new instance of the CGI application, eating memory and resources. A Java Virtual Machine (JVM) would do a similar thing with each instance of a Java class, allocating space on the heap, using memory and resources. The J2SE environment helps to prevent this by creating a common area where all the classes run, helping to alleviate many of the problems that CGI and the JVM share. There is still a JVM, but it does not start-up and shutdown with each instance of the class. The J2SE is a constantly running a JVM where classes can start and stop, but the JVM continues to run, thus cutting some overhead and increasing response times from the server.

By default, the Tomcat server runs off port 8080 on localhost. All these settings can be changed in the config files, for testing and debugging purposes I changed the port settings to 8081. This is because of an application call `TcpTunnelGui`. With this tool it is possible to tunnel one port to another. The Java applications would send the SOAP message on port 8080. This would be captured by TcpTunnelGui and forwarded to port 8081. The web service would then send the reponse back to the TcpTunnelGui on port 8081, which was then forwarded back to the consumer on port 8080. This may seems like an extra step, but the TcpTunnelGui captures all the traffic it forwards.

[1] CGI is an acronym for Common Gateway Interface, it was developed so that HTML could be dynamically served in real-time.

Once captured it can be printed to the screen for debugging purposes. I used this tool to watch and capture my SOAP messages. To execute the TcpTunnelGui run the following command at the prompt:

```
java org.apache.soap.util.net.TcpTunnelGui 8080 localhost 8081
```

The command is made-up of several parts, the first part, `java`, is the path to Java, the second part, `org.apache.soap.util.net.TcpTunnelGui` is the name of the Java class, the thrid part `8080` is the listening port followed by the server name, in this case `localhost`, and finally the port to forward the traffic to, `8081`.

After the set-up and debugging of the web service listener is complete, your Java class must be associated with some RPC name. When the client sends a message to the listener, it also sends a URN (Universal Resource Name). With this information it is possible for the Java server to invoke the correct Java class, pass any arguments, and return any resulting values. Tomcat has two ways to register your class with a URN, the first is through a web interface running as part of the server administration package. The second is to create an XML document that describes the class, functions available, parameters, etc. This XML file is called a Service Descriptor Document; Appendix C.2 has the service descriptor for the BibTeXDB class. Once that is registered, any SOAP message that the listener receives will be passed to the correct class and the listener will wait for a return value that it will marshal back into XML for a SOAP message.

These returned values are sent back to the client application, where it is parsed and assigned to some variables in the particular language. This project was built in Java, so the returned value is an Object that is cast into the proper data type.

Both the Add and Edit functions are conducted through the same Java swing interface. Both functions require the same parameters, which are made up of one or more of the standard BibTeX values. The Swing application gives feedback as to whether the operation was successful or if there were any problems. To invoke the BibTeX Editor, type the following command at the prompt:

```
java bibtexdb.edit http://localhost:8080/soap/servlet/rpcrouter
```

The command line is made up of three parts, the first `java` is the path to the java executable. The next part `bibtexdb.edit` is the name of the package and the program to invoke. This will start the swing application for adding and editing entries. The final argument `http://localhost:8080/soap/servlet/rpcrouter` is the URI of the SOAP listener. In this case, Tomcat was running locally on port 8080. The path `soap/servlet/rpcrouter` is the path to the listener that will unmarshal the actual SOAP messages and invoke the correct Java class.

The Add function returns an integer value or a SOAP fault. If a SOAP fault is return then the fault message must be inspected to find where the actual error occurred. Otherwise, an integer value is return, which represent the key value of the newly added BibTeX citation.

The Edit Function returns as integer value or a SOAP fault. The integer is an error code that describes any problems that are not related to SOAP. If the Edit function returns a “0” then the edit was successful, a return of “1” means the key value supplied could not be found. The client application takes these values and prints an appropriate String for the user rather than the error code.

The Search function is invoked slightly different because it is not a swing application, but instead it is a command line application. Search takes the same parameters as the Add and Edit application; path to java, package to invoke, and URI to the SOAP listener, but a fourth argument is added as well. This argument is the term or terms to queried. The command would look like the following if you were searching for the term XML.

```
java bibtexdb.search http://localhost:8080/soap/servlet/rpcrouter XML
```

If there is a need to narrow the search more terms can be added to the query by simply placing all the terms in quotes as follows.

```
java bibtexdb.search http://localhost:8080/soap/servlet/rpcrouter
 ''XML W3C''
```

The spaces act as an “AND” operator and will return any results that contain all of the search terms. The more information provided the more refined the search results become.

The search function will return an array of a class called `BibTeXResults`. This contains all the BibTeX values such as author, publisher, notes, etc. Appendix B.2 contains a sample response with a BibTeXDB array. This data is then transformed by the client application to output it in the proper format. If no records are found then the array is empty and nothing is printed to standard out. This is because the program could be used as part of a longer UNIX command where the results from one application are piped as input into the next. If the next application in the pipe after `bibtex.search` is expecting some BibTeX citations and gets a string saying “no results found” this could cause a problem. So it was best left to not print anything except null or an expected list of BibTeX citations.

If there is a fault during processing SOAP will return the appropriate fault code and fault message. This will help determine where the error occurred, what the problem was, and possibly how to correct it.

Chapter 7

Conclusion

The BibTeXDB web service system was completed successfully, but has plenty room for improvement. I learned a lot about the intricacies of SOAP, what it means to be a web service, the role distributed systems play in computing, and how the Internet will shape the way services are vended in the future.

7.1 Semantic Web and Web Services

The Semantic Web is not a new Internet, but an extension of the current one, in which information is given a well-defined, structured meaning, better enabling computers and people to work in cooperation.(Tim Berners-Lee, 2001) This is currently being designed in parallel with the current web. SOAP web services are part of the current web and not intended to be part of the next generation Semantic Web. SOAP uses XML technology, which is not currently powerful enough to express full relations needed in the Semantic Web. Instead, Resource Description Framework (RDF), which is a form of XML, will be the basis for message passing, querying, and data storage.

SOAP web services are used to conduct transactions, pass documents, and to make remote function calls on systems that are heterogeneous. The Semantic Web will have a similar more powerful system using DAML and RDF. DAML provides a basic infrastructure that allows a machine to make the same sorts of simple inferences that human beings do. It's just a start, but is a critical foundation for a web of information that machines can draw upon.(Pease, 2002) This does not mean SOAP is already a doomed

technology, rather the Semantic Web is years away, if ever, from construction. SOAP solves many problems that exist today and tomorrow, and because of this SOAP will be used for a long time to come. SOAP is an XML based technology, just like many of the technologies in the Semantic Web and instead of dropping SOAP it probably will be integrated or evolve to fit into the Semantic Web.

7.2 Improvements and Future Work

7.2.1 Satellite Projects

There are several smaller projects outside the scope of this MSc project that could be worked on and integrated.

When a LaTeX file is compiled using BibTeX it creates an .AUX file for the bibliography. It is possible to create a small utility that parses the .AUX file and grabs the list of all the cite references. It could then invoke the BibTeXDB web service and gather all the BibTeX citations into a .BIB file that is included in the final run of LaTeX. This would build your bibliography on the fly from the citation references in the document.

7.2.2 Better Searching

Currently the search only matches text string. Future implementations could utilise the power of regular expressions to search for text strings, allowing for more robust results. It would also be possible to limit the fields the search is conducted on, to only author, key, or title, etc. Limiting the fields enables searches of the database for all authors named “Joe Smith” and not have any results that contain “Joe” or “Smith” in any of the other fields.

Since the search string is parsed at the space character it is not possible to search for “New York” without getting results like; “The New English Dictionary, printed in York”. The current search implementation parses at the spaces and will search for “New” and do a separate search for “York” and if an entry contains both terms it is returned in the query. Another problem with parsing at the spaces is that a search containing “New” will find “New York” with a space after “New” and it will also find

“Newcastle” because it contains the substring “New”. To avoid finding “Newcastle” it should be easy, just search for “New ” with a trailing space, but because the query is parsed on spaces this is not possible with the current implementation. To improve the search capabilities, future implementation could accept a regular expression string as input and use that to search the database for results. Using regular expressions might be more processor intensive than the substring match and could increase the time and utilisation of the application.

Currently, when querying on multiple search terms, the search returns results that match all of the search terms. The space character between the words acts as an “AND” operator and will find entries that contain all of these strings or substrings. The search algorithm could be changed so that an entry would only need to match any the search terms to be returned. Thus turning the space character into an “OR” operator.

With some more advanced algorithms it could be possible to emulate a query that is more related to an SQL statement, where “AND” and “OR” operators are intermixed and wildcard characters are accepted also.

7.2.3 Internationalisation

This ties in with better searching; certain words in the English language are borrowed from other languages, for example “resume”. It is sometimes spelled with or without the accents in English. As the database grows and different users, with different language backgrounds, add new entries, each will be slightly different. Therefore, the system should be implemented to do allow for characters outside the 26 common letters used in the English language, and should also search for their English equivalence. Search results for “Vitæ” should contain entries with Vitæ and Vitae allowing for different spellings.

7.2.4 Result Format

Currently, when a search is performed the results are returned as a result object, which is then cast into a `BibTeXDBResult` object array. This is then printed to standard out in BibTeX format. Future versions of the software could use the BibTeX output as

the default, but additional flags could be passed as arguments on the command line to return the results as plain text, XML, BibTeX, MLA format, or additional formats. This MSc project is assuming the entries will be used with LaTeX, but with additional styles of outputting the data could be incorporated into different applications.

7.2.5 Java Subclasses

In my undergraduate I learned to program in C/C++, it wasn't until this year that I had to use more than the basic knowledge of Java for a large project. Only after most of the code was written was it evident that it should have been written differently. It should have been written so that it is more understandable, uses the Object-Oriented paradigm, and is designed with future editing in mind, specifically the code for the web server listener when accessing the XML database. I used SAX to parse the XML file and to search for matching entries, add new entries, and to find and edit existing ones. The SAX parser works through the use of several predefined functions that are used to determine the start and end of XML tags and the XML document. These functions were part of the base BibTeXDB class so each of my three functions (Search, Add, Edit) had to use these SAX functions to parse the XML database. Problems arose in determining exactly, which of my three functions called the SAX function and what exactly to do inside the SAX function that my functions needed. This became complicated and I essentially used global variables by making private variables in the base class to tell the SAX function which of my three functions called it.

This could have all been solved if I had made the SAX functions its own class, then each of my three functions could have instantiated their own specific SAX functions rather than trying to put code for all three of my functions inside one SAX function. At the point I realised this I was too far along to fix it, the does code works properly – it is just not the cleanest.

Another problem with using a SAX parser is that it only knows about the current tag. It does not know which node in the tree this tag represents, so it is difficult to keep state. This was a problem with the search function because as the search moved through each element of the BibTeX citation the application needed to also remember the tags in the citation it has already seen just in case the last tag matched the search

criteria and the whole citation needed to be returned to the consumer.

7.2.6 Database Options

For this project specification, it was determined that an XML database of BibTeX citations was to be used. There are several advantages to this, because now with a master XML file of entries it is possible to use other tools, such as XSLT to search and manipulate the data.

There are downsides to using an XML file as well; one being that there is no locking or atomic transactions. Using a different database method to store the BibTeX citations could solve this. A Relational Database could be used to hold the data. The benefits of doing this are that Java has a Java Database Connectivity (JDBC) Manager to open connections to a database and with a pre-built database manager, such as MySQL, PostGreSQL, or others, things like file locking and atomic transactions are part of the system. The downside to these systems is that it is another piece of software needs to be installed and managed, whereas the XML file is just a file on disk.

7.2.7 Speed and Scalability

During most of the project I used a subset of the actual database for testing. This subset was only the first 15 of the $\tilde{2}$0,000 entries. This allowed me to easily check and see entries being updated, added, and searched. Toward the end of the project I started to scale-up the number of entries being parsed, so that I could test for a wider range of errors and more specific searches. The SAX parser is a fast parser, much faster than the DOM parser because DOM holds each node in memory and creates a tree structure data type, which requires lots of memory. Even with the SAX parser's speed, it must look at each tag's contents and compare it to the search string to determine if it is a match. As the XML file grows in size, so does the work the SAX parser must do. The final XML database I tested has over 210,000 lines, almost 500,00 words, is more than 6MB on disk, and takes too long to return an answer.

Since Java requires a virtual machine and has a garbage collection system, the speed of the application is not as fast as one written in C. Even with that in mind, I

think most of the time is spent parsing the XML and is not the overhead of the Java language. That overhead would be a constant and not dependant on the size of the XML file. So the time to search the database is directly proportional to the number of lines in the database.

If nothing can be changed and the delay between request and response is too large there are a few things that should be done to help the user. One thing that can be added is a system status; this could take the form of an hourglass or progress bar. The first item in the list of "Ten Usability Heuristics" is Visibility of System Status. The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.(Nielsen, 1994) Another option would be to create a new thread in the application that handles the SOAP request/response. By doing this, the application is free to do other things while you wait for the SOAP response.

7.2.8 Multi-User

There are a few problems that could arise in the environment that have not been built-into this system. There is currently no provision for file locking of the XML database, this could cause problems if multiple users are editing or adding entries to the database. Some of the data could be lost as BibTeX citations are written to the file.

It has also been suggested that a multi-tiered systems be implemented so that a level of rights management can be imposed. The ability to search and add, but not edit entries for base users or possibly only being able to edit entries that you have added to the system. This would require adding additional tags to the database that remember who last edited and who added the data on what date. Higher-level users could delete entries or edit anyone's entry along with inheriting lower-level user's rights of adding and searching. How the tiers are separated is out of the scope of this MSc project, but adding this functionality would require an authentication system built on top of the current system. Building this would mean that all sensitive operations would also require a session ID or session key to be adding to the function parameters or passed in the SOAP header that is checked with each function call. Since HTTP is stateless, every message passed will require the authentication information to be added, be it name and password or a token, with each message.

Appendix A

Software Used

To construct a webservice I had to install, configure, and run several different pieces of software. This is a list of the different packages, what their purpose was and how they were setup.

- Tomcat 4.0 Servlet/JSP Container
- Java 1.4.1
- Xerces Java Parser
- JavaMail™ API 1.3 release
- Javabeans™ Activation Framework 1.0.2 Release
- Microsoft Windows 98 Operating System
- RXP an XML parser

Java 1.4.1 is the newest version of Java and is the most feature rich. It was used because many of the other Java related items required the most up-to-date software to be installed. The Javabeans Activation Framework was installed so that it was possible to use Java Beans with application. The Xerces Java Parser was install to help parse the SOAP messages that were being sent to and from the server. RXP was used to validate the BibTeXDB XML file that was created. Finally, Tomcat was set-up on port 8081 to be the SOAP listener.

Appendix B

Sample SOAP RPC request

B.1 SOAP RPC Request

```
POST /soap/servlet/rpcrouter HTTP/1.0
Host: localhost:8080
Content-Type: text/xml; charset=utf-8
Content-Length: 446
SOAPAction: ""

<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
>
<SOAP-ENV:Body>
<ns1:Search
  xmlns:ns1="urn:suda:bibtexdb"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
>
<query xsi:type="xsd:string">New York</query>
</ns1:Search>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

B.2 SOAP RPC Response

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 1747
Date: Sat, 06 Sep 2003 23:56:26 GMT
Server: Apache Tomcat/4.0.6 (HTTP/1.1 Connector)
Set-Cookie: JSESSIONID=F9824E9D7530755D1491AB32560189A2;
Path=/soap
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
>
<SOAP-ENV:Body>
<ns1:SearchResponse
  xmlns:ns1="urn:suda:bibtexdb"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
>
<return
  xmlns:ns2="http://schemas.xmlsoap.org/soap/encoding/"
  xsi:type="ns2:Array" xmlns:ns3="urn:suda"
  ns2:arrayType="ns3:bibtexdbresults[1]"
>
<item xsi:type="ns3:bibtexdbresults">
<address xsi:type="xsd:string">New York</address>
<annote xsi:type="xsd:string"></annote>
<author xsi:type="xsd:string">
  Jerry A. Fodor and T. G. Bever and M. F. Garrett
</author>
<booktitle xsi:type="xsd:string"></booktitle>
<chapter xsi:type="xsd:string"></chapter>
<edition xsi:type="xsd:string"></edition>
<editor xsi:type="xsd:string"></editor>
<entryType xsi:type="xsd:string">Book</entryType>
<howpublished xsi:type="xsd:string"></howpublished>
<institution xsi:type="xsd:string"></institution>
<journal xsi:type="xsd:string"></journal>
<key xsi:type="xsd:string">14</key>
<month xsi:type="xsd:string"></month>
<note xsi:type="xsd:string"></note>
```

```
<number xsi:type="xsd:string"></number>
<organization xsi:type="xsd:string"></organization>
<pages xsi:type="xsd:string"></pages>
<publisher xsi:type="xsd:string">McGraw Hill</publisher>
<school xsi:type="xsd:string"></school>
<series xsi:type="xsd:string"></series>
<title xsi:type="xsd:string">
  The Psychology of Language
</title>
<type xsi:type="xsd:string"></type>
<volume xsi:type="xsd:string"></volume>
<year xsi:type="xsd:string">1974</year></item>
</return>
</ns1:SearchResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

B.3 SOAP Fault Message

```
HTTP/1.1 500 Internal Server ErrorContent-Type: text/xml;
charset=utf-8
Content-Length: 4115
Date: Mon, 07 Jul 2003 17:36:01 GMTServer:
Apache Tomcat/4.0.6 (HTTP/1.1 Connector)
Set-Cookie: JSESSIONID=92DAFC3A84D3DBEABE6BB4CEB47DBCFD;
Path=/soap
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
>
<SOAP-ENV:Body>
<SOAP-ENV:Fault>
<faultcode>SOAP-ENV:Server</faultcode>
<faultstring>
Exception from service object:
allrefs.xml (The system cannot find the file specified)
</faultstring>
<faultactor>/soap/servlet/rpcrouter</faultactor>
```

```
<detail>
<stackTrace>
        java.io.FileNotFoundException:
allrefs.xml (The system cannot find the file specified)
        at java.io.FileInputStream.open(Native Method)
        at java.io.FileInputStream.<init>(FileInputStream.java:103)
        at java.io.FileInputStream.<init>(FileInputStream.java:66)
        at java.io.FileReader.<init>(FileReader.java:41)
        at bibtexdb.bibtexService.Search(bibtexService.java:146)
        ...
        at java.lang.Thread.run(Thread.java:536)
</stackTrace>
</detail>
</SOAP-ENV:Fault>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Appendix C

Tomcat 4.0 JSP/Servlet Container

C.1 Instructions

Tomcat is a Java servlet container that acts as the web service listener. When a SOAP request comes in, Tomcat forwards the message to the correct Java class. To do this each Java class has to be registered with the server. The Following is a Deployment Descriptor, it is an XML document that contains all the instructions to regisiter the service with the Tomcat server.

C.2 Deployment Descriptor

```
<isd:service
   xmlns:isd="http://xml.apache.org/xml-soap/deployment"
   id="urn:suda:bibtexdb">
  <isd:provider type="java"
                scope="Application"
                methods="Search Add Edit">
    <isd:java class="bibtexdb.bibtexService"/>
  </isd:provider>

  <isd:mappings defaultRegistryClass="">
    <isd:map
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:x="urn:suda"
      qname="x:bibtexdbresults"
      javaType="bibtexdb.ResultsBean"
      xml2JavaClassName=
```

```
        "org.apache.soap.encoding.soapenc.BeanSerializer"
      java2XMLClassName=
        "org.apache.soap.encoding.soapenc.BeanSerializer" />
  </isd:mappings>

  <isd:faultListener>
    org.apache.soap.server.DOMFaultListener
  </isd:faultListener>
</isd:service>
```

Appendix D

Web Service Description Language (WSDL)

D.1 WSDL File to Describe the BibTeXDB Web Service

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
WSDL File for the BibTeXDB Web Service
Brian Suda
brian.suda@ed.ac.uk
-->
<definitions name="bibtexdbDescription"
        targetnamespace="urn:suda:bibtexdb"
        xmlns:tns="urn:suda:bibtexdb"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
>
  <types>
     <xsd:schema xmlns="http://www.w3.org/2001/XMLSchema"
       targetNamespace="urn:suda:bibtexdb">
      <xsd:complexType name="BibTeXDBResults">
        <xsd:all>
          <xsd:element name="Key"          type="xsd:string" />
          <xsd:element name="Type"         type="xsd:string" />
          <xsd:element name="Booktitle"    type="xsd:string" />
          <xsd:element name="Title"        type="xsd:string" />
          <xsd:element name="Author"       type="xsd:string" />
```

```
          <xsd:element name="Publisher"    type="xsd:string" />
          <xsd:element name="Editor"       type="xsd:string" />
          <xsd:element name="Address"      type="xsd:string" />
          <xsd:element name="Institution"  type="xsd:string" />
          <xsd:element name="Organization" type="xsd:string" />
          <xsd:element name="School"       type="xsd:string" />
          <xsd:element name="Month"        type="xsd:string" />
          <xsd:element name="Year"         type="xsd:string" />
          <xsd:element name="Journal"      type="xsd:string" />
          <xsd:element name="Volume"       type="xsd:string" />
          <xsd:element name="Series"       type="xsd:string" />
          <xsd:element name="Number"       type="xsd:string" />
          <xsd:element name="Chapter"      type="xsd:string" />
          <xsd:element name="Pages"        type="xsd:string" />
          <xsd:element name="Edition"      type="xsd:string" />
          <xsd:element name="Howpublished" type="xsd:string" />
          <xsd:element name="Annote"       type="xsd:string" />
          <xsd:element name="Note"         type="xsd:string" />
        </xsd:all>
      </xsd:complexType>

      <xsd:complexType name="BibTeXDBResultsArray">
        <xsd:complexContent>
          <xsd:restriction base="soapenc:Array">
             <xsd:attribute
               ref="soapenc:arrayType"
               wsdl:arrayType="typens:BibTeXDBResults[]"
             />
          </xsd:restriction>
        </xsd:complexContent>
      </xsd:complexType>

    </xsd:schema>
  </types>

<!-- variables passes in function -->
<message name="SearchQuery">
  <part name="query" type="xsd:string" />
</message>
<message name="SearchResults">
  <part name="return" type="tns:BibTexDBResultsArray" />
</message>
```

```
<message name="AddQuery">
  <part name="AddEntry" type="tns:BibTeXDBResults" />
</message>
<message name="AddResult">
  <part name="return" type="xsd:integer" />
</message>

<message name="EditQuery">
  <part name="EditEntry" type="tns:BibTeXDBResults" />
</message>
<message name="EditResult">
  <part name="return" type="xsd:integer" />
</message>

<!-- These are function names and their input/output -->
<portType name="bibtexInterface">
  <operation name="Search">
     <input  message="tns:SearchQuery">
     <output message="tns:SearchResults">
  </operation>
  <operation name="Edit">
     <input  message="tns:EditQuery">
     <output message="tns:EditResult">
  </operation>
  <operation name="Add">
     <input  message="tns:AddQuery">
     <output message="tns:AddResult">
  </operation>
</portType>

<!-- These are the interfaces -->
<binding name="bibtexBinding" type="tns:bibtexInterface">
  <soap:binding
style="rpc"
transport="http://schema.xmlsoap.org/soap/http"
  />
  <operation name="Search">
    <soap:operation soapAction="urn:Search" />
    <input>
      <soap:body
        use="encode"
```

```
        namespace="urn:Search"
        encodingStyle="http://schema.xmlsoap.org/soap/encoding/"
      />
    </input>
    <output>
      <soap:body
        use="encode"
        namespace="urn:Search"
        encodingStyle="http://schema.xmlsoap.org/soap/encoding/"
      />
    </output>
  </operation>
  <operation name="Edit">
    <soap:operation soapAction="urn:Edit" />
    <input>
      <soap:body
        use="encode"
        namespace="urn:Edit"
        encodingStyle="http://schema.xmlsoap.org/soap/encoding/"
      />
    </input>
    <output>
      <soap:body
        use="encode"
        namespace="urn:Edit"
        encodingStyle="http://schema.xmlsoap.org/soap/encoding/"
      />
    </output>
  </operation>
  <operation name="Add">
    <soap:operation soapAction="urn:Add" />
    <input>
      <soap:body
        use="encode"
        namespace="urn:Add"
        encodingStyle="http://schema.xmlsoap.org/soap/encoding/"
      />
    </input>
    <output>
      <soap:body
        use="encode"
        namespace="urn:Add"
```

```
        encodingStyle="http://schema.xmlsoap.org/soap/encoding/"
      />
    </output>
  </operation>
</binding>

<!-- service location -->
<service name="bibtexService">
  <port name="bibtexPort" binding="tns:bibtexBinding">
    <soap:address
       location="http://localhost:8081/soap/servlet/rpcrouter"
    />
  </port>
</service>

</definition>
```

Appendix E

BibTeXDB Client

This is the screen capture of the Java Client. This client allows you to Add or Edit and entry by filling in the corresponding fields.

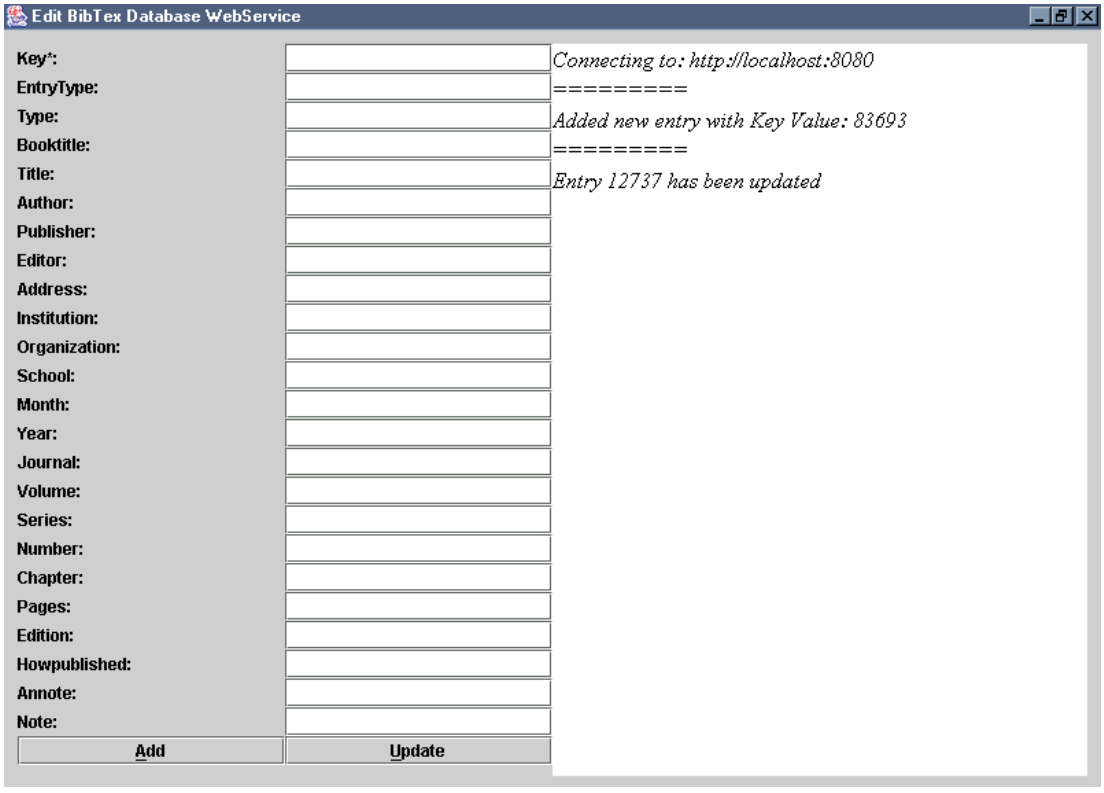

Figure E.1: Figure: BibTeX Java Client

Appendix F

WS-Inspection Document

This is a sample Web Service Inspection Document, it points to a WSDL file that describes the service fully and it points to a UDDI registery that contain information about the provider and service.

```
<?xml version="1.0"?>
<inspection
  xmlns="http://schemas.xmlsoap.org/ws/2001/10/inspection/"
  xmlns:wsiluddi="http://schemas.xmlsoap.org/ws/2001/10/inspection/uddi/"
>
  <service>
    <abstract>BibTeXDB with two descriptions</abstract>

    <!-- Sample WSDL Description -->
    <description
      referencedNamespace="http://schemas.xmlsoap.org/wsdl/"
      location="http://example.com/BibTeXDB.wsdl"
    />

    <!-- Sample UDDI Description -->
    <description referencedNamespace="urn:uddi-org:api">
       <wsiluddi:serviceDescription
         location="http://www.example.com/uddi/inquiryapi"
       >
         <wsiluddi:serviceKey>
           4FA28580-5C39-11D5-9FCF-BB3200333F79
         </wsiluddi:serviceKey>
       </wsiluddi:serviceDescription>
```

```
    </description>
  </service>

  <!-- links to other WS-I Documents -->
  <link
    referencedNamespace="http://schemas.xmlsoap.org/ws/2001/10/inspection/"
    location="http://example.com/moreservices.wsil"
  />
</inspection>
```

Bibliography

Asaravala, A. (2002). Can Public Web Services Work? Internet. http://www.newarchitectmag.com/documents/s=2453/na1102d/.

Berners-Lee, T. (2002). Web Services: Program Integration across Application and Organization boundaries. Internet. http://www.w3.org/DesignIssues/WebServices.html.

Clabby, J. (2002). Web Service Gotchas. Internet. http://www.ibm.com/developerworks/webservices/library/ws-gotcha.html.

Cohen, F. (2001). Myths and misunderstandings surrounding SOAP. Internet. http://www.ibm.com/developerworks/webservices/library/ws-soap.html.

Daveid Ehnebuske, IBM & Barbara McKee, I. (2001). Versioning Taxonomy and Identifier Systems. Internet. http://uddi.org/pubs/tn-taxonomy-versioning-v1.05-Draft-20010906.pdf.

Don Box, David Ehnebuske, G. K. A. L. N. M. H. F. N. S. T. D. W. (2000). Simple Object Access Protocol (SOAP) 1.1. Internet. http://www.w3.org/TR/2000/NOTE-SOAP-20000508/.

Haas, H. (2002). Web Services Activity. Internet. http://www.w3c.org/2002/ws/.

ISO (1986). Information processing - Text and office systems - Standard Generalized Markup Language (SGML). Internet.

James Snell, Doug Tidwell, P. K. (2001). *Programming Web Services with SOAP*. O'Reilly & Associates, Inc. http://www.oreilly.com/catalog/progwebsoap/.

Keith Ballinger, Peter Brittenham, A. M. W. A. N. S. P. (2001). Specification: Web Services Inspection Language (WS-Inspection) 1.0. Internet. http://www.ibm.com/developerworks/library/ws-wsilspec.html.

Kreger, H. (2003). Fulfilling the Web Services promise. *Communications of the ACM*, 46(6):29–ff.

Martin Dürst, A. F. (2003). Unicode in xml and other markup languages. Internet. http://www.unicode.org/reports/tr20/.

Nielsen, J. (1994). Ten Usability Heuristics. Internet. http://www.useit.com/papers/heuristic/heuristic_list.html.

Organization, T. W. S. I. (2002). The Organization and its Work. Internet.

Pease, A. (2002). Why use DAML? Internet. http://www.daml.org/2002/04/why.html.

Peter Murray-Rust, D. M. (2000). SAX. Internet. http://www.saxproject.org.

Schwartz, E. (2002). UC Berkeley to use Web Services to deploy unified messaging system. Internet. http://www.infoworld.com/article/02/03/01/020301hnberkeley_1.htm.

Tim Berners-Lee, James Hendler, O. L. (2001). The Semantic Web. Internet.

Tobin, R. (1997). RXP - an XML parser available under the GPL. Internet. http://www.cogsci.ed.ac.uk/ richard/rxp.html.

Vasudeva, V. (2001). A Web Services Primer. Internet. http://webservices.xml.com/pub/a/ws/2001/04/04/webservices/.

William A. Nagy, Franciso Cubera, S. W. (2001). The Advertisement and Discovery of Services (ADS) protocol for Web Services. Internet. http://www.ibm.com/developerworks/web/library/ws-ads.html.

Winer, D. (1999). XML-RPC Specification. Internet. http://www.xmlrpc.com/spec.