

A SERVICE-ORIENTED APPROACH TO APPLICATION DEVELOPMENT

A Major Qualifying Project Report

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

in Computer Science

by

Robert Darneille

Gary Schorer

Date: 10/10/2007

Approved

Prof. M. J. Ciaraldi, Major Advisor

ABSTRACT

Web services are becoming an increasingly popular tool in large software development environments. This project assessed several existing frameworks for exposing Java classes as web services, looking for an effective, consistent handling of many complex data types. One framework was chosen and used as the base for a skeleton Eclipse project which enables developers to quickly and easily expose Java classes as web services and secures access to those services to only a certain set of authorized applications.

TABLE OF CONTENTS

1. Introduction.....	1
2. Background.....	3
2.1 Service Frameworks	3
2.2 SOAP	4
2.3 Spring.....	5
2.4 WSDL.....	7
3. Methodology	11
3.1 Service Framework Selection.....	11
3.1.1 Determine Available Frameworks	12
3.1.2 Determine Relevant Characteristics.....	12
3.1.3 Create Common Testing Base	12
3.1.4 Deploy Testing Base Using Frameworks.....	13
3.1.5 Document and Analyze Results.....	13
3.2 Skeleton Implementation	13
3.3 Skeleton Documentation.....	14
3.3.1 Versioning Recommendations.....	14
3.3.2 Client Stub Creation.....	15
3.4 Skeleton Testing	15
4. Results & Analysis	17
4.1 Service Framework Selection.....	17
4.1.1 Determine Available Frameworks	17
4.1.2 Determine Relevant Characteristics.....	18
4.1.3 Create Common Testing Base	19
4.1.4 Document and Analyze Results.....	20
4.1.4.1 XFire.....	21
4.1.4.2 Axis2	22
4.1.4.3 CXF	25
4.1.4.4 Conclusions.....	26
4.2 Skeleton Implementation	28
4.2.1 Authentication Protocol.....	29
4.2.2 Acegi Configuration.....	29
4.3 Skeleton Documentation.....	31
4.3.1 Versioning Recommendations.....	31
4.3.2 Client Stub Creation.....	33
4.4 Skeleton Testing	35
5. Future Work	36
Appendix A: Framework Comparison Form.....	37
Appendix B: XFire Analysis Results	38
Appendix C: Axis2 Analysis Results	60
Appendix D: CXF Analysis Results.....	83

FIGURES

Figure 1 Example Spring Bean declaration	6
Figure 2 WSDL aggregation diagram	9
Figure 3 Data types and method signatures of the service framework test base	20
Figure 4 Snippets required for XFire configuration.....	21
Figure 5 Snippets required for Axis2 configuration.....	23
Figure 6 Example code produced by the Axis2 client generator	34
Figure 7 Example code produced by the XFire client generator	34

TABLES

Table 1 Relevant service framework characteristics and their rankings	28
---	----

1. INTRODUCTION

Web services are becoming an increasingly popular tool in large software development environments. They encourage code reuse by encapsulating the logic involved in repetitive tasks in such a way that it is readily accessible by any developer working with any language on any platform. Web services also help encourage loose coupling by minimizing the number of places where pieces of software have to interact. For example, by creating a web service to handle searching for people in an employee directory, there is now only one point of connection to the database for searching. This means that if the database must be migrated or changed, there is only one place where the code for connecting to and interacting with it must be updated; all other applications can continue accessing the service oblivious to the changes in the underlying logic.

Web service development at Lincoln Laboratory has gone on in a largely independent fashion. There are several different groups of developers creating their own web services in their own ways. This diversity in the tools and techniques used to create services makes communication and sharing of knowledge and experience extremely difficult. A developer in one group may have found a solution that fits their needs perfectly, although it may not necessarily work for most other cases. There has thus far been no comprehensive research into the capabilities of the available service frameworks which are used to create these web services.

To address this problem within the ICS group, Lincoln Laboratory initiated this survey into the available technologies and techniques associated with creating and maintaining web services. Specifically, the goal of this project is to create a service skeleton which can be used as a development base for new Java applications. The skeleton establishes one particular method for exposing web services that requires a minimal amount of configuration on the part of the developer.

The skeleton also establishes one particular method for securing access to these services from remote applications, such that a given application must be authorized to use a given service before it is able to make use of its operations. It also conceals the specifics of the authentication, minimizing the amount of configuration required by both

the service developer and other developers who wish to use the service in their applications.

Along with creating this skeleton code, research was also conducted into best practices for web service versioning. Recommendations were created about how to version web services, as well as defining what should constitute a version change and what kinds of effects various version changes would have on the people using a service. To further ease the widespread adoption of services, research was also conducted into service client generators, which create easy-to-use code that abstracts away the nuances of marshalling between XML and Java and communicating with the service over the network. Aside from making it easier to use any given service, a universally accepted client generator would provide increased consistency in connecting to services in general, encouraging adoption of services as a whole, and not just one particular service.

2. BACKGROUND

2.1 Service Frameworks

There is a wide array of frameworks available for exposing Java applications as web services via Apache Tomcat. A more detailed analysis of the functionality of each of these frameworks is presented in section 4.1.4.

The first service framework examined in our research – and the only one currently being used by the ICS group in the Laboratory – was XFire. XFire is a Codehaus project, which claims to “[make] service oriented development approachable through its easy to use API and support for standards.”¹ Among its listed features are: “Support for important Web Service standards - SOAP, WSDL, WS-I Basic Profile, WS-Addressing, WS-Security, etc.”, “JSR 181 API to configure services via Java 5 and 1.4 (Commons attributes JSR 181 syntax)”, “Spring, Pico, Plexus, and Loom support.”, and “Client and server stub generation.”¹ It provides for use and configuration both by an XML configuration and a Java API.

The second service framework examined was Apache Axis2. According to their website, “Apache Axis2 not only supports SOAP 1.1 and SOAP 1.2, but it also has integrated support for the widely popular REST style of Web services. The same business logic implementation can offer both a WS-* style interface as well as a REST/POX style interface simultaneously.”² Axis2 also boasts its own web-based administration console for deploying and undeploying services without having to restart the entire server.

The third framework examined was Apache CXF. CXF is the continuation of XFire after it merged with Celtix. CXF also supports configuration by XML and APIs, much like XFire. It supports an even broader set of standards, as well as providing REST/HTTP binding. According to its website, “Apache CXF is an open source services framework. CXF helps you build and develop services using front-end programming APIs, like JAX-WS. These services can speak a variety of protocols such as SOAP,

¹ Taken from the official XFire website, located at <http://xfire.codehaus.org/>

² Taken from the official Axis2 website, located at <http://ws.apache.org/axis2/>

XML/HTTP, RESTful HTTP, or CORBA and work over a variety of transports such as HTTP, JMS or JBI.”³

2.2 SOAP

The official W3C SOAP specification states, “SOAP ... provides the definition of the XML-based information which can be used for exchanging structured and typed information between peers in a decentralized, distributed environment.”⁴ The specification continues to add that,

“SOAP is fundamentally a stateless, one-way message exchange paradigm, but applications can create more complex interaction patterns (e.g., request/response, request/multiple responses, etc.) by combining such one-way exchanges with features provided by an underlying protocol and/or application-specific information. SOAP is silent on the semantics of any application-specific data it conveys, as it is on issues such as the routing of SOAP messages, reliable data transfer, firewall traversal, etc. However, SOAP provides the framework by which application-specific information may be conveyed in an extensible manner. Also, SOAP provides a full description of the required actions taken by a SOAP node on receiving a SOAP message.”

Simply put, SOAP messages are XML documents that contain three basic elements:

- The SOAP envelope – the root of the document, which must be present. There are no attributes defined for the envelope element, but every envelope must contain exactly one body element.
- The SOAP header – an optional element that is used “to pass information in SOAP messages that is not application payload.” This could consist of things such as authentication information for the application/user sending the message, or any other kind of data that is independent of the message being sent. The contents of

³ Taken from the official CXF website, located at <http://incubator.apache.org/cxf/index.html>

⁴ Taken from the official SOAP specification, located at <http://www.w3.org/TR/2007/REC-soap12-part0-20070427>

the header elements are undefined by the SOAP standard; their semantics are completely application-dependent. There are, however, attributes that can be specified in the header which do have significance in the SOAP standard. The `mustUnderstand` attribute requires the header be processed or return a fault. The `relay` attribute requires the header be passed on to the next SOAP processor (if any exists).

- The SOAP body – contains the actual payload of the application and is a required element. The request/response being sent is specified inside the body. The contents of the body elements are undefined by the SOAP standard; their semantics are completely application-dependent. The body element is also used to relay fault information if an error occurs in the processing of the SOAP message. The format for the elements specifying this fault is defined in the SOAP standard.

2.3 Spring

The Spring Framework is used heavily in this project. “Spring is a layered Java/J2EE application framework”⁵ that provides a wide range of services, the full extent of which is well beyond the scope of this document. As such, this will only address the relevant features used in this project.

Perhaps the most basic feature – and the one most commonly used in this project – is dependency injection and relationship management. As opposed to defining relationships manually in Java and hard-coding relationships between classes, Spring users are encouraged to program to interfaces and develop their POJOs (Plain Old Java Objects) independently. Spring can then be used to “inject” an instance of one class into another class. This, when combined with loosely coupled classes, enables the developer to plug classes in and out without needing to edit references in their Java and recompile it.

As an example, imagine you have a class called `Person`. Since people own cars, your `Person` object has a reference to another object called `vehicle`. If you were

⁵ Taken from the official Spring Framework website, located at <http://www.springframework.org>

programming to interfaces, this would be of type Car, where Car is an interface implemented by all of your objects that represent cars. You could then make a class called Subaru, which implements Car. Then to create a person who owns a Subaru using Spring, you could do the following:

```
<bean class="Person">
  <property name="vehicle">
    <bean class="Subaru">
      ...
    </bean>
  </property>
</bean>
```

Figure 1
Example Spring Bean declaration

Then as more classes implementing Car were added, you could change the type of vehicle owned by this particular person from Subaru to something else. Since Spring manages all the relationships, there are no actual references in the Java hard-coding and forcing a relationship between two objects.

Another important feature of Spring that factored heavily into the development of this project was its singleton management. By default, all objects declared in Spring are singletons. This can be changed by passing adding the property scope="prototype" to their bean definitions, but in this particular case, this behavior was desirable. By declaring the objects that comprise a service in a Spring bean file, it provides the ability to have one service implementation object that is universally accessible by both internal application logic and remote service calls. For example, suppose there were a web application made up of JSPs that ran alongside a service and needed to use its functionality. These JSPs could request the service object from Spring and they would receive the same instance of the object that was being used by service calls. While this may appear trivial at first, it enables in-memory changes to be made by either a remote application calling the service, or a JSP accessing it as a local application, in such a way that both are aware of the change.

set of operations. For the purposes of this project, all communication is conducted with SOAP via HTTP. However, it is possible to declare other communication protocols and other transfer protocols (such as SMTP).

- **Services** – The service declaration attaches an address to a specific binding, telling the client where to send its requests.

WSDL Structure

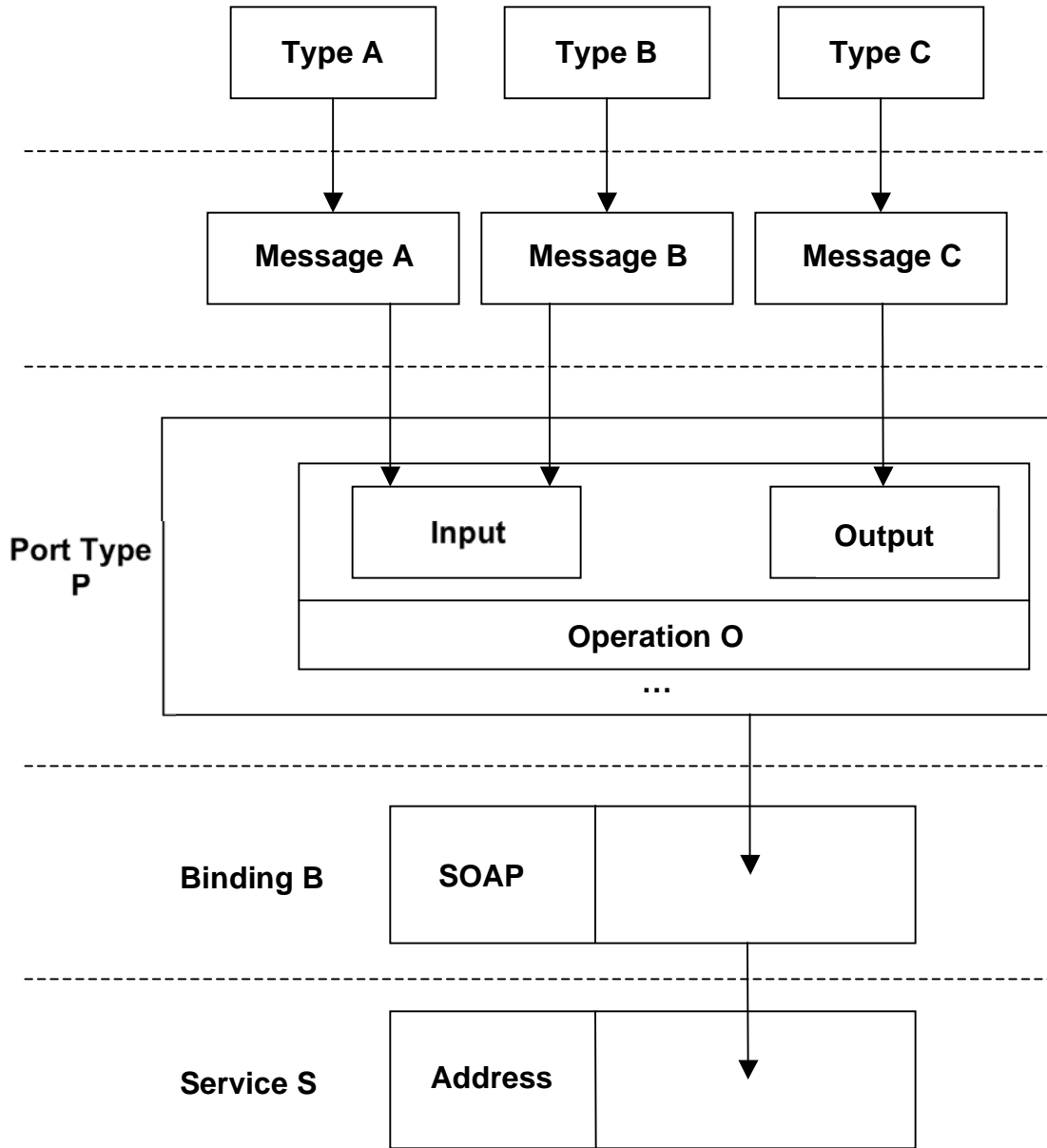


Figure 2
WSDL aggregation diagram

As evidenced by figure two (2), the WSDL document forms itself as an aggregation of smaller components. Each layer of the document adds some extra level of meaning without interfering with the previous layers it depends upon. This makes its components reusable at almost every phase. For example, once types have been created, they can be used by any number of message components. Those messages can then be assembled in any order on any number of operations as either inputs or outputs. At each level, a little bit more information is added, until the document reaches the service level, where it is capable of providing all the information necessary for a client to interact with the service.

Note that this is not a comprehensive documentation of the full functionality of the WSDL specification. These are only the parts that are relevant to this particular project. For the full details of the specification, please see the official reference page located at <http://www.w3.org/TR/wsdl>.

3. METHODOLOGY

The purpose of this project is to create a code skeleton and enabling documentation which can be used by Java developers to quickly create new web services capable of running inside Apache Tomcat, and access to which is secured and limited to a specific set of applications. To meet this goal, four steps were created:

1. **Service Framework Selection** – Service framework selection involved examining the existing service framework products that were available and determining which one would best meet the needs of the Laboratory.
2. **Skeleton Implementation** – With the components that would comprise the skeleton selected, they would then have to be assembled and made operational.
3. **Skeleton Documentation** – After the skeleton had been implemented and deemed operational, its usage instructions and capabilities needed to be documented as thoroughly as possible.
4. **Skeleton Testing** – If time permitted, “real-world” testing of the skeleton would be conducted. The purpose of this testing was to determine how easily Laboratory employees can put the skeleton to use in creating and exposing services.

3.1 Service Framework Selection

In any service-oriented application structure, the most important piece of infrastructure is the framework used to deploy parts of the application as services. Given the importance of this particular component, it was important to find one that meets the specific needs of the Laboratory. In order to do this, a process was created which would allow for an effective, efficient comparison of the various available frameworks.

The first step in this process was to determine the list of candidate frameworks from which the final selection would be made. After that, the set of relevant characteristics needed to be chosen – these would form the basis on which all of the frameworks’ effectiveness was judged. Given this list of relevant characteristics, a test “base” was developed which was capable of testing for the presence of each

characteristic. After the base was created, it was then deployed using each of the available frameworks. The results of this test deployment were then documented and analyzed, enabling us to determine which framework best meets the needs of the Laboratory, and which, if any, of those needs remain unmet.

3.1.1 Determine Available Frameworks

The first step in the process was a general survey of the available service frameworks. One of the given requirements in this case was that the framework be able to run inside a servlet framework environment, specifically Apache Tomcat. Additionally, the framework must be able to integrate with the security provider already chosen by the Laboratory: Acegi Security (also known as Spring Security). Laboratory staff were initially consulted to determine which frameworks, if any, were already in use. With a base list constructed, it could then be expanded through research on the Internet.

3.1.2 Determine Relevant Characteristics

The base characteristic list would consist of those properties that would affect a general transmission of data across any medium between any types of applications. Additional consulting was done with Laboratory staff to determine any additional characteristics that might be applicable to this particular environment. During this process, the importance of each characteristic was also discussed to determine which characteristics would carry the most weight in the final analysis of the testing results.

3.1.3 Create Common Testing Base

Given this set of characteristics, which each framework needed to possess, a common base was then created to test for the presence of these characteristics. Since the primary focus of this research was exposing applications as services, the test base itself was an application. This application's method signatures and data types (the only parts relevant to services) closely mirrored a typical application based on *de facto* Laboratory

standards. Of particular concern were things such as commonly used data formats (ArrayList objects as opposed to primitive Object arrays) and naming conventions.

3.1.4 Deploy Testing Base Using Frameworks

With a common testing base created, it was then deployed as a service using each framework. This uniform testing base eased the process of comparing the tools by ensuring that each framework was being compared to each other framework based on the same set of criteria. A uniform process was created to ensure that frameworks were compared in such a way that each relevant characteristic was thoroughly tested against the example.

3.1.5 Document and Analyze Results

After deploying the common testing base in each framework, the results of the test were documented using a uniform format to facilitate comparisons. With the list of relevant characteristics for each framework fully documented, conclusions were drawn with regards to which framework(s) will most effectively meet the needs of the Laboratory. Ultimately, a formal, written recommendation was produced documenting which framework is the best option and why. This documentation also contains notes of any characteristics which are not adequately met by the proposed solution.

3.2 Skeleton Implementation

With a service framework and security provider selected, the two then needed to be integrated together and configured such that they can be easily redistributed and setup. Per one of the Laboratory's initial requirements, the security provider needed to be setup in such a way as to be able to retrieve lists of allowed applications from the Laboratory LDAP server. The security provider also needed to be configured to forward successfully authenticated requests to the service framework in such a way that they could be processed and results returned to the user transparently. Both the service framework and the application calling the service had to be blind to the specific implementation of the

security provider. For all intents and purposes, the framework was not supposed to know that the request it is receiving has been handled by the security provider at all. This loose coupling ensured that both the security provider and the service framework can be swapped in and out if needed with minimal impact on each other. The security provider needed to require an absolute minimum amount of developer setup (ideally, none at all). The service framework had to similarly require a minimal amount of developer setup. Ideally, the minimum required configuration in the case of the framework would be for the developer to only need to somehow register their service class (or classes) with it.

3.3 Skeleton Documentation

Along with the physical skeleton code, there needed to be documentation explaining, at a bare minimum, how to use it. The documentation in this particular case needed to be more comprehensive than that, and extend beyond simple usage instructions. A list of known bugs or limitations of the skeleton must also be provided to ensure that developers know precisely what they can and cannot do with it.

However, beyond this purely technical documentation, there also needed to be “best practice” recommendations that cannot be enforced in the skeleton. Specifically, these “best practice” recommendations must address issues of versioning in web services (both generally, and as it applies to this particular skeleton) and enabling other developers to access a service via the creation and distribution of client “stubs.”

3.3.1 Versioning Recommendations

The Laboratory does not currently have any standards or best practices of its own associated with web service versioning. As such, this section of the documentation was informed mostly by research into best practices published by other organizations or researchers. The results of this research were then discussed with Laboratory employees to arrive at a standard which enables meaningful versioning of services without negatively impacting any other existing Laboratory standards or the functionality of the skeleton.

3.3.2 Client Stub Creation

The creation of client stubs for distribution to other developers is one technique for easing adoption of services by other Laboratory employees. These client stubs abstract out the networking nuances associated with accessing web services and create objects that can be accessed like any other local in-memory object in the program, hiding all web service-related aspects of the functionality.

There are a variety of tools for automating the creation of these stubs. They generally read the WSDL defining the service in question, and then create a stub client, which can access all of the operations provided by the service. It is our general recommendation that client stubs be generated for all services. They serve not only to ease adoption of a particular service by other individuals, but also make it easier for the developer to connect to his or her own services. As part of this recommendation, we examined available client stub generators and determined which one generates the easiest to use client stubs.

Since the target language for this project is Java, only Java client generators were examined. The process for selecting a “best” client generator closely mirrored the process by which a service framework was selected earlier. First, a list of all available client generators was compiled. A common service was then passed into each generator; in this case, the common service was the example service constructed earlier by exposing the common test base through the chosen service framework. The resulting clients were then compared, with the primary criteria for selection in this case being ease of use and reliable conversion of data types.

3.4 Skeleton Testing

With all the necessary documentation written, the skeleton was then tested in the actual Laboratory environment. The purpose of this step was to ascertain the effectiveness of the skeleton in terms of its ability to enable Laboratory Java developers to quickly create new web services. Originally, a list of potential Java web service developers was to be obtained from the Laboratory. These developers were to be contacted and asked if they would like to participate in this testing. Unfortunately, due to

time constraints, the scope of the testing was far narrower than originally intended. We were only able to get the skeleton to one developer (Peter Tecce) in time for testing. Although a more thorough testing would have been desirable, valuable input was still gathered about the documentation of the skeleton.

4. RESULTS & ANALYSIS

The overall goal of this project was to create a skeleton project and enabling documentation, which would enable Java developers to quickly and easily create web services. The individual objectives set to reach that goal were: to select a service framework for the skeleton, implement the skeleton, then document and finally test the skeleton. These objectives were met satisfactorily. The process of testing the skeleton was not completed as thoroughly as originally desired. However, it still provided valuable input with regards to the skeleton documentation. Despite this, the overall goal of the project was still met and an effective skeleton and documentation were produced which will enable Java developers to easily expose future applications as web services.

4.1 Service Framework Selection

The process of selecting a service framework was carried out in four steps. First, a list of viable candidate service frameworks was developed based first on input from Laboratory employees, then from research conducted on the Internet. Second, a list of relevant characteristics was established. This list was based on initial requirements provided by the Laboratory (such as the server the frameworks must be able to run in, Apache Tomcat) and then developed more fully by consultation with Laboratory employees. After having identified the relevant characteristics, a testing base was then created which, when exposed as a service, would test each framework for the presence of the relevant characteristics. Lastly, this testing base was exposed as a service in each framework, and the results were documented and analyzed. From these results, we determined XFire to be the most effective framework.

4.1.1 Determine Available Frameworks

Three major frameworks were quickly identified as possible candidates: Axis2, XFire and CXF. At a bare minimum, all of these frameworks provide the ability to quickly and easily expose Plain Old Java Objects (POJOs) as SOAP-based services. All three of these frameworks automatically generate WSDLs for the services they expose.

As a result, developers are able to setup existing Java without necessarily learning the nuances of WSDL creation.

XFire, the oldest of the three frameworks, does not provide many features beyond that of exposing basic services for use via SOAP. It does provide excellent Spring support via an included Spring-enabled version of the standard XFire Servlet.

Additionally, it has full support for the JSR181 (JAX-WS) annotations.

Axis2 has perhaps the widest set of operational features. Axis2 supports SOAP with attachments, as well as coming with an administrative console, which allows services to be deployed and restarted without necessarily restarting the entire framework. Aside from exposing services for use via SOAP, it also automatically exposes them via REST⁷, with no additional work required on the part of the developer.

CXF is the most recent of the three frameworks. It has moved from beta into release over the course of the writing of this document. CXF is the result of a merger between the XFire and Celtix teams. It has many of the same features as Axis2, including easy exposing of services for use via REST.

4.1.2 Determine Relevant Characteristics

The basic characteristics each framework must possess were stated at the outset of the project by the Laboratory. The framework must be able to expose Java classes as services for use via SOAP. It must also be able to run inside the Apache Tomcat Servlet engine.

After further consultations, a wider range of relevant characteristics was established. The first and most important characteristic is that of reliable and consistent marshalling and unmarshalling between Java classes and XML in accordance with the service's WSDL. The framework must also be able to handle a wide range of data types, specifically: all primitive Java types (including primitive arrays of primitive types,

⁷ REST (REpresentational State Transfer) is a technique for architecting software systems originally defined in the doctoral dissertation of Roy Thomas Fielding, available at http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm. However, the term "REST" is often used to refer to the ability of a web service to accept requests via standard HTTP URL requests. Parameters in a REST request are embedded in the URL (.../ServiceName?param1=val1¶m2=val2...) instead of being packaged in SOAP messages. This is the manner in which the term is used in this document. It is not used to refer to the true architecture style of REST, only to the more colloquial usage of the term, where it refers to HTTP URL-based requests to services.

standard Java classes, and user-defined custom types), all standard Java Collection types (including Java 1.5 generics) and any user-defined custom type (including ones with nested complex types, and all previously mentioned types). Another important characteristic is Spring support. Ideally, developers using the framework should be able to use standard Spring 2.0 beans in the creation of their services. Built-in support for REST is also desirable. The framework could take a single service class and simultaneously expose it for use with both SOAP and REST. Another important characteristic is support for the full set of JSR181 annotations. This standard provides for an effective means of easily configuring many important parts of a service through the use of simple Java annotations. The fact that it is a standard would also ease transition of the services between different frameworks when such a move is required. The same can be said for support for Spring: its increasingly wide adoption means services configured via Spring in one framework should be easy to migrate to another. The last relevant characteristic is the handling of faults/exceptions. In the event of an exception or error in the course of processing a request, the framework should be able to send such a message back using the standard SOAP Fault messages. Ideally, it will simply convert Java exceptions thrown by the services into SOAP faults.

A simple form was created for documenting a framework's support of each characteristic. It is available as Appendix A.

4.1.3 Create Common Testing Base

Based on the defined set of relevant characteristics, a common testing base was created, which aided in determining how many of those characteristics each framework meets. The primary focus of the test base is data type support. The test base should have a wide array of operations which return and take in as parameters as many data types as possible. For the purposes of testing fault handling, some operations should also generate faults for the framework to return. Some of the operations should also be annotated to test for support of the JSR181 annotations. Based on this, a simple directory/phonebook-style service was created with the following operations and data types:

```
Operations  
public HashSet<Person> getDirectory();  
public ArrayList<Person> getDirectoryList();
```

```
public Person[] getPeopleByName(String) throws Exception;
public Person getPersonById(String);
public String getNameById(String);
public void addPerson(Person);
public void addPeople(ArrayList<Person>);
public void addPeopleArray(Person[]);
```

Person

```
String id;
String firstName;
String lastName;
PhoneNumber phone;
ArrayList<String> emailAddresses;
```

PhoneNumber

```
Integer areaCode;
Integer firstThree;
Integer lastFour;
```

Figure 3
Data types and method signatures of the service framework test base

4.1.4 Document and Analyze Results

For each of the candidate frameworks, the testing base was deployed as a service. Each operation was then tested using a product called soapUI⁸. soapUI will generate SOAP request skeletons for each operation defined on a provided WSDL. The user can enter their data into the request, and then view the raw SOAP response. It is an extremely effective tool for simulating use of the service by a potential user when it is used in this manner. Since it can only base its requests on the WSDL, it has just as much information as a potential user would. It also shows the raw SOAP output of a service, which will expose any nuanced differences between the declared format in the WSDL and the actual result – differences which may be concealed by the particular parsing technique of automatically generated Java clients.

Additionally, the setup procedure required for exposing a service via the framework was documented both for future reference and for comparison purposes. The level of support for REST, Spring, JSR181 annotations were also documented, along with the handling of Java exceptions/faults.

⁸ soapUI, a product of eviware, “is a free and open source desktop application for inspecting, invoking, developing, simulating/mockng and functional/load/compliance testing of web services over HTTP.” It is available at the soapUI website: <http://www.soapui.org>

4.1.4.1 XFire

The first framework tested was XFire. From the perspective of the skeleton developer, there was some amount of research required to reach the desired level of functionality in terms of Spring and JSR181 annotation support. That being said, with that research done, the developer utilizing the skeleton need never encounter it. The resulting setup procedure is an extremely easy one. The developer must annotate the Java class they wish to expose as a service, and add a single Spring bean for it to one of the Spring context XML files. To expose an annotated class called `directoryservice.Directory`, the only necessary configuration is the addition of an annotation to the Java class, and one line of XML to the XFire configuration file.

Annotation

```
package directoryservice;  
  
@WebService  
Class Directory { ... }
```

XML Configuration

```
<bean class="directoryservice.Directory" />
```

Figure 4
Snippets required for XFire configuration

- **Spring Support** - XFire's Spring support is included by default. Part of the standard XFire distribution is a Spring-enabled Servlet called `XFireSpringServlet` which simply replaces the standard `XFireServlet` in the `web.xml` configuration file.
- **REST Support** - REST support is not included, and none is planned (most new feature support is being directed toward CXF).
- **JSR181 Annotation Support** - XFire fully supports all of the JSR181 annotations.
- **Fault Handling** - XFire handles conversion of Java Exceptions into SOAP Faults. Whenever any of the service operations throws an exception, XFire takes the message of the exception and sends it back as the message of a SOAP Fault.

Data Handling - XFire's data handling was almost ideal. Its weaknesses in this area can best be described as a tendency to create unnecessary XML elements. For example, to handle all requests that return collections of Person objects, XFire created an ArrayOfPerson type in the schema. This is not necessarily a bad thing, but ArrayOfPerson is synonymous with an unbounded sequence of Person types. Similarly, it created an ArrayOfString type which is nothing more than an unbounded sequence of stock xs:String types.

These minor caveats aside, XFire's handling of data types was exceptional. All collection-based objects were mapped to the same type within the schema. While they are not necessarily the same in the Java, they are very similar in purpose, and since they contain the same kind of information, there is no reason for them to not be represented by a single type in the XML. Everything XFire returned over the course of its testing was completely in line with what would be expected given the WSDL. No problems were encountered in any of the operations.

The full results of the testing are included as Appendix B. These include all of the SOAP requests and responses for all of the operations available in the testing base, as well as the setup procedure used to configure the framework.

4.1.4.2 Axis2

Axis2 required the greatest amount of work on the part of the skeleton developer to achieve the desired level of Spring support. The standard means of deploying services in Axis2 is to package them in a .aar file (which is identical to the zip format) and deploy them using the administration console. These .aar files depend on Axis2's larger structure and contain only information about this specific service, and as such cannot be deployed as .war files in another Servlet container. Furthermore, the .aar packaging is not conducive to Spring. As such, the skeleton had to be setup in such a manner as to not utilize this format. By mimicking Axis2's directory structure and using the same Servlet specified in the Axis2 .war distribution, a skeleton was constructed which operated similarly to XFire. However, along with declaring the services as Spring beans, users of the skeleton also have to write a relatively small amount of XML in a services.xml file

which Axis2 uses to determine which services to expose. The Spring bean definition and services.xml configuration for the test base were as follows:

```
Spring bean
<bean id="directoryService" class="directoryservice.Directory" />

services.xml
<service name="DirectoryService" scope="application">
  <description>Directory Service</description>
  <messageReceivers>
    <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-
only"
class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver"/>
    <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-
out" class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
  </messageReceivers>
  <parameter
name="ServiceObjectSupplier">org.apache.axis2.extensions.spring.r
eceptors.SpringServletContextObjectSupplier</parameter>
  <parameter name="SpringBeanName">directoryService</parameter>
</service>
```

Figure 5
Snippets required for Axis2 configuration

- **Spring Support** – Partial Spring support is possible using only standard Axis2 classes and configuration options. A fully Spring-enabled Axis2 setup (where the actual Axis2 Servlet handling the service requests is also available in the Spring context) is also possible with the creation of a custom Servlet.
- **REST Support** – By default, all services exposed via Axis2 are also exposed via REST. Their request format is:
`http://.../services/ServiceName/operationName?param1=value1¶m2=value2&...`
- **JSR181 Support** – Axis2 has nearly full JSR181 support. There are a few notable annotations, however, which it does not appear to support. The name attribute of the @WebResult annotation does not appear to be supported, nor does the endpointInterface attribute of the @WebService annotation.
- **Fault Handling** – Axis2 converts exceptions into SOAP faults in much the same way that XFire does. However, Axis2 also includes the stack trace of the exception in the message. This is probably not information that should be passed back to the user of the service.

Data Handling – Axis2’s data handling was somewhat less than ideal. Axis2 assumes that any complex type being passed in an operation is a standard Java bean. That is to say, that the class will have a series of private attributes which are manipulated using the standard set* and get* methods (e.g. to get the value of a string called firstName, one would call getFirstName).

With this as its starting point, Axis2 works backwards to determine what fields will be present in the XML schema. Normally, this would be a highly desirable and consistent behavior. However, it applies this technique to all complex types, including those present in the standard Java distribution – specifically, the Collection types. As a result, none of the Collection types are declared correctly in the WSDL. In the case of the ArrayList type, for example, the only operation it has that matches the standard Java bean format is isEmpty. As a result, when constructing the schema for this type in the WSDL, Axis2 creates an element which has only one property: a Boolean called “empty.”

This discrepancy is only present in the WSDL definition. Axis2 will actually return the contents of a Collection type. However, without being declared as such in the WSDL, there is no way for a potential user of the service to parse the results without having actually run the service and looked at the raw SOAP result. Furthermore, there is no way for the user to tell if the structure of the elements within the Collection has changed just from looking at the WSDL; they would be oblivious to the change until their service calls began to fail or return nonsensical results.

While not affecting the results returned by the service, this schema error does affect requests passed to Axis2. Collection types passed in a request in accordance with the structure Axis2 uses in its results will be ignored. No structure was found that would result in Axis2 correctly processing the contents of a collection in a request. No SOAP faults are returned, and the operation appears to complete successfully, but the contents of the message are never unmarshalled into the correct Java types.

The full results of the testing are included as Appendix C. These include all of the SOAP requests and responses for all of the operations available in the testing base, as well as the setup procedure used to configure the framework.

4.1.4.3 CXF

The CXF setup procedure was fairly straightforward. It has some amount of Spring support included by default, but there was no apparent way to get the full level of Spring support offered by a properly configured Axis2 or XFire. Configuration for REST required actual rewriting of the existing Java code. With that exception, CXF can be configured out of the box by simply creating the necessary `jaxws:endpoint` elements in the `applicationContext.xml` file.

- **Spring Support** – Partially included by default. The standard configuration process makes use of a standard Spring bean definition file. However, `jaxws:endpoint` elements must still be declared to expose the beans as services. There does not appear to be any way to make these endpoints reference standard Spring beans. As a result, some amount of Spring configuration is possible, but it is not as direct as Axis2 or XFire's.
- **REST Support** – Services can be exposed using REST out of the box. Additionally, the mapping of URLs to operation requests is configurable via the `@HttpResource` annotation. However, in order for this to work, the method must take in a single standard Java bean with setters for all the parameters defined in the annotation.
- **JSR181 Annotation Support** - CXF fully supports all of the JSR181 annotations perfectly.
- **Fault Handling** - CXF ideally handles conversion of Java Exceptions into SOAP Faults. Whenever any of the service operations throws an exception, CXF takes the message of the exception and sends it back as the message of a SOAP Fault.

Data Handling – CXF handles data similarly to Axis2 in that it does not wrap elements unnecessarily. For example, where XFire creates an `ArrayOfPerson` element, which is no more than an unbounded sequence of elements of type `Person`, CXF and Axis2 will simply use the sequence itself. However, CXF does generate more accurate WSDLs. It correctly interprets collection types as sequences and maps them accordingly. It also returns the collections reliably.

CXF's one weakness in regards to data handling is in passing sequences. It does not correctly process sequences passed as the parameter to a request. Sequences embedded in a complex type will be passed and processed correctly, but sequences passed as a parameter in a request will cause CXF to generate an exception and return a SOAP fault.

The full results of the testing are included as Appendix D. These include all of the SOAP requests and responses for all of the operations available in the testing base, as well as the setup procedure used to configure the framework.

4.1.4.4 Conclusions

With all of the above results having been collected, they were then compared to determine which of the frameworks would be the best fit for the skeleton. The single most important aspect of the frameworks was their performance in regards to data handling. In this regard, XFire performed the best. XFire understood both primitive arrays and the more complex parameterized Collection types. There were no problems encountered in returning and passing all of the data types tested. CXF and Axis2 returned results in a similar fashion. However, they both had unique, serious problems in their handling of certain Collection types. In the case of CXF, any operation that took in a sequence (a primitive array or Collection) as a parameter threw an exception. Axis2 does not generate any errors, but it does not properly pass the data into the operation. At this point, further comparison was purely academic. Axis2 and CXF could not map between certain standard Java types and XML, meaning they could not effectively fill the role of being the base of the skeleton. However, in the interest of documenting all relevant functionality, the other results are documented below.

The second most important area is that of WSDL generation. If the framework cannot produce reliable WSDLs, its ability to correctly handle many data types is severely limited. For example: Axis2 can return parameterized Collection types successfully. However, its WSDL declares that the only property that will be coming back is a Boolean called "empty."⁹ So even though it is capable of returning the data,

⁹ This is due to the process by which Axis2 constructs its XML types. Axis2 assumes all data types will be in the format of standard Java beans – that is, they will have private or protected properties and getters and setters that will be used to access those properties. The only method in the Collection classes that matches

potential users will have no way of knowing what that data will look like until it actually comes back. It also makes it impossible to tell if data types have changed just by looking at the WSDL. In this area, CXF produced perhaps the best WSDL's. They were similar to those generated by XFire in that it correctly covers all the data types returned by the service. However, it does not have the extra layer of wrapping that XFire uses (e.g. making an `ArrayOfPerson` type which is just an unbounded sequence of `Person` types). That being said, XFire's WSDL generation is in no way deficient. The wrapping objects add one extra level of hierarchy to step through when looking at certain complex types, but they do accurately reflect the way the data is returned.

The third most important area is that of fault handling. Operations will inevitably need to generate errors at some point in time, and there must be an easy way to translate errors that are meaningful in a Java environment (exceptions) into errors that are meaningful in a service environment (SOAP faults). All three frameworks behaved very similarly in this area. They all take the message of a Java exception and turn it into a message of a SOAP fault and pass it back to the user of the service. Axis2 goes one step further and includes the stack trace from the exception in the message as well. This is not detrimental to the exception to fault conversion, but it is unnecessary information that should not be passed back to the user in the first place.

Of significantly less importance is support for REST. While desirable, REST does not have the same power as SOAP for transferring data types. Passing complex types through REST is just as difficult as passing them through SOAP, so whenever complex types are involved, there is no effective difference between the two. Given that, SOAP's consistency between simple and complex types is preferable over the inconsistencies found in REST (URL-based passing for simple types, XML-based passing for complex types). Despite its relative insignificance, it is still worth documenting the frameworks' behavior in this regard. Axis2 has the best REST support of all. Operations are automatically exposed via REST with no additional configuration necessary. CXF's support is slightly less ideal. The user is required to create another endpoint definition in their configuration files. The user also has to create operations that take in a single

this standard format is `isEmpty()`, which returns a Boolean. As such, this is the only property Axis2 sees as being accessible.

complex type with setters for the various properties present in the URL in order to expose the operation via REST. However, XFire has no REST support of any kind, and none is planned.

Area of Concern	Ranking (Best→Worst)
Data Handling	1. XFire 2. Axis2/CXF
WSDL Generation	1. CXF 2. XFire 3. Axis2
Fault Handling	1. CXF/XFire 2. Axis2
REST Support	1. Axis2 2. CXF 3. XFire

Table 1
Relevant service framework characteristics and their rankings

All these things taken into consideration, XFire is clearly the best fit for this skeleton. Although it is not as fully featured as Axis2 and CXF, the features it lacks are not necessary to the essential functionality of services. It was the only one out of all three candidates that was capable of reliably passing and returning all data types tested in a manner that was consistent with its WSDL.

4.2 Skeleton Implementation

The process of implementing the skeleton consisted mostly of configuring the security provider (Acegi) to work with XFire and properly authenticate applications. XFire itself had already been setup properly from the framework comparison, so no additional configuration was needed in that regard. The process of actually implementing the skeleton was broken into two steps. First an authentication protocol was devised which would define the general way in which applications could be authenticated. With that authentication protocol created, Acegi could then be configured to implement it.

4.2.1 Authentication Protocol

As part of the process of generating the skeleton, an authentication protocol had to be created to handle authenticating remote applications. The authentication system aims only to authenticate applications and other services. It does not attempt to handle any kind of user authentication; that is something that must be managed by the application calling the service. Lastly, it does not aim to be a completely air-tight suite capable of securing access to services from every conceivable angle. The goal is only to restrict access to services such that they cannot be easily used without being registered for access. The authentication system is intended to be used as part of a usage monitoring system that would enable service developers to track which applications are using their services. This way, when a major change is planned, which will break backwards compatibility, they will have at the ready a list of all the applications currently using their service so that their developers can be informed of the change.

Given these requirements, a system was created which requires each party to contribute one token to a larger authentication call which will verify that the two are compatible. The calling application adds its token to the header of the SOAP request. Acegi (the service's security provider) takes this token and combines it with two other things: the token specified by the service in its configuration files, and the IP of the calling application. It then queries a SQL database (MySQL was used for testing purposes) specified in its configuration files to search for the presence of a record confirming that the specified application is allowed to access the service from the given IP.

Ideally, this would eventually be migrated to use an existing official Laboratory directory, such as LDAP. However, no such solution could be created and implemented in time for the completion of this project, so the temporary SQL database-based solution was created as a proof of concept.

4.2.2 Acegi Configuration

Since there appeared to be no existing Acegi filters meeting the needs of our authentication protocol, a new one had to be created. The filter was constructed in three steps. First, it retrieved tokens from incoming message headers. Second, it passed

authenticated requests along into XFire and returned errors for unauthenticated requests. Third, it checked the contents of the token provided by the application to determine if it had access to the service.

The first, and simplest, step of the process was creating a filter which could retrieve tokens from the SOAP header of an incoming message. A simple filter was created which extended the standard `AbstractProcessingFilter` (`org.acegisecurity.ui.AbstractProcessingFilter`). In its original incarnation, this filter simply searched the text of the message contents for a standard token placement (`Header/Security/Username`). This was later modified to use `XPath`, and the path it searches for was made configurable via the Acegi configuration file.

The second, and most complicated, step was changing the underlying `AbstractProcessingFilter` to pass requests on to XFire. A request authenticated by Acegi stops there unless something is setup to continue to pass it along to where it would normally have gone otherwise. The underlying filter had to be capable of passing these requests along to XFire, as well as returning errors of some kind when authentication failed.

Since these are errors pertaining exclusively to SOAP services, we can safely assume that the clients will understand the SOAP fault error format. For the sake of consistency, the XFire SOAP fault structure was used as a base. The filter simply converts messages from Acegi's `AuthenticationException` exceptions into the faultstring of a SOAP fault.

The case of redirecting to XFire on a successful authentication was somewhat more difficult. Since we were already using the `XFireSpringServlet` to enable us to use Spring configuration in our services, we were also able to use Spring to access the actual instance of XFire that was currently running. After examining the Javadocs and XFire source, a bean called `xfire.servletController` was located in the Spring context which enabled us to inject an `HttpRequest` object directly into XFire and have it pass it off to the correct service for processing.

Lastly, an authentication provider had to be created that would actually check the token that was being retrieved by the newly created filter. The actual implementation of the authentication provider was a relatively simple process, which just involved querying

a SQL database based on the token obtained by the filter, the address of the incoming request and another token specified in Acegi's configuration file. The actual process of determining what the authentication specification would be was distinct from the implementation and somewhat more complicated.

4.3 Skeleton Documentation

After the skeleton had been implemented and tested internally, it was then necessary to document its usage and to write recommendations related to service creation, usage and maintenance.

The first part of the documentation to be written was the usage instructions. This consisted of several parts. First, a basic "quick start" guide was created which covered the bare minimum amount of setup necessary to get a service running. Then, two additional sections were created with more advanced configuration options. One section dealt exclusively with JSR181 annotations used to configure the service's WSDL, documenting most of their relevant attributes and explaining how they work. Another section related to skeleton-specific configuration options, such as disabling access to WSDLs and changing the location of the remote application authentication database.

The second major component of the documentation consisted of the various¹⁰ recommendations made to service developers. These recommendations fell into two main categories: those related to versioning, and those related to client stub generation.

4.3.1 Versioning Recommendations

The first set of recommendations addressed were those pertaining to versioning. Problems of versioning arise quickly when working with web services. The primary versioning problems present in WSDL definitions revolve around the issue of backwards compatibility. As such, changes to WSDL definitions are grouped into two categories: those which break backwards compatibility, and those that do not.

According to IBM's "Best practices for Web services versioning," the changes that do not break backwards compatibility are the "[a]ddition of new WSDL operations to

¹⁰ Taken from the IBM "Best practices for Web services versioning", located at <http://www.ibm.com/developerworks/webservices/library/ws-version/>

an existing WSDL document” and the “[a]ddition of new XML schema types within a WSDL document that are not contained within previously existing types.” In both cases, more information is being added to the WSDL definition in the new version, but no existing definitions are modified in any way. The result in this case is that the existing definition’s components (schemas/types and operations) are a subset of the new definition’s components. For changes that fall into this category, version labeling is purely informational, as changes between the documents do not break backwards compatibility.

The other category of changes listed in IBM’s “Best practices for Web service versioning” is changes that break backwards compatibility. The changes that break backwards compatibility are “[r]emoving an operation”, “[r]enaming an operation”, “[c]hanging the parameters (in data type and order) of an operation”, and “[c]hanging the structure of a complex data type.” In all of these changes, new information may or may not be added. What characterizes these types of changes is the fact that they change or remove existing information. As such, the existing definition is not a subset of the new one, and requests that conform to the existing definition will not be compatible with a service that conforms to the new definition. At this point, version labeling is no longer purely informational; there is now a physical difference between the two versions and backwards compatibility is no longer guaranteed.

There are also soft changes that, while not physically breaking backwards compatibility, have resulted in such a substantial change to the meaning of a parameter or return value that the two different versions of the service are entirely incompatible. In the case of the IBM article, an example is cited with regards to stock prices. Suppose that previously, stock prices were stored as dollar amounts using floats. This can result in an undesirable level of precision, with more than two decimal places being present. As a response to this, the way stock prices were stored was changed to be cent values stored using integers. Both values are numbers, and may be understood and processed by both the service and its clients. However, their values are off by multiples of 100. This kind of soft change should also merit a major version change. The two data types may very well be compatible, but their meaning has changed in such a way that it would be dangerous to allow clients referencing the old version’s WSDL to access the new version’s services.

Having reached these conclusions, they were then written up, proofread and included in the skeleton documentation.

4.3.2 Client Stub Creation

With versioning recommendations having been written, work could then begin on choosing a client stub generator and documenting its usage. To ease adoption of services, it is strongly recommended that developers choosing to expose their applications as services also produce client stubs to facilitate communication with the services. There are a variety of tools available that will automate this process. Generally speaking, the tools take in a WSDL and will produce clients stubs capable of working with the operations defined in the WSDL. Developers wishing to connect to the service can then simply use the client stub as they would any other class and the nuances of actually creating and interpreting the SOAP requests and responses are abstracted out of view.

To determine which tool creates the “best” clients, a series of simple tests were conducted. As with the service framework selection process, a common base was chosen and clients were generated by each tool from the base. In this case, the base was the service created using XFire during framework testing. The clients were tested to determine how reliably they return the different data types, how easy it is to use the stub (i.e. how many nested complex types are necessary), and perhaps most importantly, whether or not SOAP headers can be added to the outgoing messages.

There were two client generators compared during testing: Axis2 and XFire. Both clients performed all required tasks sufficiently. They both provided a means of adding the SOAP headers required by the authentication system, as well as correctly handling all data types. Their only differences were in the structure of the client’s classes, as well as the specific syntaxes of operation calls.

The first client generator tested was Axis2. Axis2 generated a type for each type declared in the WSDL. For example, there is an `ArrayOfPerson` element defined in the WSDL, which contains an unbounded sequence of type `Person`. Axis2 created a Java type called `ArrayOfPerson`, which itself has a getter called `getPerson` which returns an array of Java `Person` objects. Aside from the wrapping resulting from XFire’s WSDL (for example, the wrapping of a sequence of `Persons` as an `ArrayOfPerson`), Axis2 adds no

extra wrapping. Every complex type is, at its lowest level, composed of standard Java types. To retrieve the area code of the phone number of the first person in the directory using Axis2, the user would have to call:

```
directory = client.getDirectory(new GetDirectory());  
directory.getPeople().getPerson()[0].getPhone().getAreaCode();
```

Figure 6
Example code produced by the Axis2 client generator

The second client generator tested was XFire. XFire similarly generated a Java type for each type declared in the WSDL. XFire differs, however, in its use of the JAXB data binding (Axis2 uses a proprietary data binding called AXIOM). The JAXB data binding results in an extra layer of wrapping around every element. As such, complex types, at their lowest level, are composed of JAXB elements which contain the real Java types. This means that every attempt to access part of a complex type will require an extra call to the JAXB `getValue` method. This may only be a minor inconvenience when retrieving values from service calls, but it becomes a more substantial hurdle when attempting to pass complex types to the service. Each JAXB element must be constructed with a class and namespace as well as a value. While not requiring substantially more work, it greatly reduces the readability of the code. Given the presence of another client generator (Axis2) which performs just as effectively, while not requiring these extra steps, there is no reason to use the XFire generator over Axis2.

```
directory = client.getDirectory();  
directory.getPerson().get(0).getPhone().getValue().getAreaCode().getValue();
```

Figure 7
Example code produced by the XFire client generator

Having decided on Axis2 as the better of the two client generators, instructions for using it were written up and included in the skeleton documentation.

4.4 Skeleton Testing

With the skeleton having been implemented and all relevant documentation having been written, it could then be tested by some of its potential users in the lab. Unfortunately, due to time constraints, only one developer was able to test the framework.

The technical mechanics of the framework behaved as expected, and no bugs were encountered in any of the underlying code that had been written. There was, however, a substantial amount of feedback with regards to the documentation.

First and foremost, the lack of an introduction section was made apparent. Having never been given a formal, spoken introduction to the skeleton, the test developer had no way of knowing what the skeleton was intended to accomplish or how it could benefit him. This resulted in the creation of an introduction section for the documentation outlining the reason for the skeleton and provided an overview of its capabilities. Also, the suggestion was made to include an example service the skeleton. A simple “Hello World” service was added with one operation. While simple in its method logic, it still provides a working example that shows what the JSR181 annotations actually look like in the Java.

5. FUTURE WORK

While the produced skeleton and its associated documentation are complete and functional, there is still work that could be done to improve upon them. This work is mostly concentrated in the area of application authentication.

Currently, the skeleton checks a custom database unique to this project to determine if an application has access to a service. Ideally, this would eventually be migrated over to an existing directory already in use for other purposes, preferably LDAP.

Also, this authentication process would hopefully be abstracted out into a service itself one day. The current system takes in two tokens and an IP address to determine if an application with token *X* has access to a service with token *Y*, when running from IP address *Z*. This could easily be converted into a service which takes in three parameters and performs the authentication itself, simply returning true or false based on whether or not the provided credentials authenticate properly.

While Acegi is completely capable of authenticating against almost any conceivable database, each distribution of the skeleton would need to be updated if the authentication database were to ever change in any way. While it is probably desirable to at least have a configuration setting in the Acegi that lets people change the address of the authentication service, it is certainly not desirable to have the connection to the database embedded in the Acegi.

There is also room for potential future work in regards to CXF. During testing, CXF failed to perform as effectively as XFire. However, since CXF is being worked on by many XFire developers, it is reasonable to assume that at some point, its basic functionality will be on par with that of XFire. Being that it provides additional functionality beyond the basic SOAP processing handled by XFire – such as exposing services via REST as well – it would be beneficial to move to this platform once it is capable of providing the same basic functionality as XFire. The only major hurdle with CXF currently is its inability to pass arrays and collections as parameters to operations. It is recommended that CXF be revisited at some point in the future to see if this problem has been fixed, and if so, a migration can then be considered.

APPENDIX A: FRAMEWORK COMPARISON FORM

Setup Procedure

Spring Support

REST Support

JSR181 Annotation Support

Annotation	Supported Attributes	Unsupported Attributes
@WebService		
@WebMethod		
@WebResult		
@WebParam		

Fault Handling

Data Type Handling

APPENDIX B: XFIRE ANALYSIS RESULTS

Setup Procedure

- 1) Annotate the service class using `@WebService` (above the class definition, optionally specifying a `serviceName`) and optionally, `@WebParam` (before each parameter in each method, specifying a name)
- 2) Drag XFire lib folder contents into `WebContent/WEB-INF/lib` in Eclipse.
- 3) Delete the included `spring-1.2.6.jar` and replace it with the Spring 2.0 `spring.jar`.
- 4) Setup `web.xml` and `xfire-servlet.xml` (see below)
- 5) Add beans for each service class to `application-context.xml` (see below)

Spring Support

Included. Instead of deploying using `XFireServlet`, deploy using `XFireSpringServlet`. Anything specified in the `contextConfigLocation` will then be scanned for beans with annotated classes to deploy as services. These beans are fully capable Spring beans.

REST Support

None. Additionally, no REST support is planned. XFire can only expose services capable of communicating via SOAP.

JSR181 Annotation Support

Full. Correctly responds to `@WebService` (`serviceName`, `endpointInterface`), `@WebParam` (`name`), `@WebMethod` (`operationName`, `exclude`) and `@WebResult` (`name`).

Fault Handling

Any exception thrown in the service will be translated into a SOAP fault by XFire. The faultstring of the fault will be set to whatever the message of the exception was and the faultcode will be set to `soap:Server`.

Required Configuration Files

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4"
xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
    <display-name>XFire</display-name>

    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>
            /WEB-INF/application-context.xml
            /WEB-INF/xfire-servlet.xml
        </param-value>
    </context-param>
    <listener>
        <listener-
class>org.springframework.web.context.ContextLoaderListener</listener-
class>
        </listener>
    <servlet>
        <servlet-name>XFireServlet</servlet-name>
        <servlet-
class>org.codehaus.xfire.spring.XFireSpringServlet</servlet-class>
        </servlet>
        <servlet-mapping>
            <servlet-name>XFireServlet</servlet-name>
            <url-pattern>/services/*</url-pattern>
        </servlet-mapping>
    </web-app>
```

xfire-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:aop="http://www.springframework.org/schema/aop"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-
2.0.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-2.0.xsd">

    <bean id="webAnnotations"
class="org.codehaus.xfire.annotations.jsr181.Jsr181WebAnnotations"/>

    <bean id="handlerMapping"
class="org.codehaus.xfire.spring.remoting.Jsr181HandlerMapping">
        <property name="typeMappingRegistry">
```

```

        <ref bean="xfire.typeMappingRegistry" />
    </property>
    <property name="xfire">
        <ref bean="xfire" />
    </property>
    <property name="webAnnotations">
        <ref bean="webAnnotations" />
    </property>
</bean>

<bean
class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping"
>
    <property name="urlMap">
        <map>
            <entry key="/">
                <ref bean="handlerMapping" />
            </entry>
        </map>
    </property>
</bean>

<import resource="classpath:org/codehaus/xfire/spring/xfire.xml" />
</beans>

```

application-context.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-
2.0.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop-2.0.xsd">
    <bean id="directoryService" class="directoryservice.Directory" />
</beans>

```

Resulting WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://directoryservice"
xmlns:ns1="http://data.directoryservice"
xmlns:soapenc12="http://www.w3.org/2003/05/soap-encoding"
xmlns:tns="http://directoryservice"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap11="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soapenc11="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
  <wsdl:types>
    <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
attributeFormDefault="qualified" elementFormDefault="qualified"
targetNamespace="http://directoryservice">
      <xsd:element name="getDirectoryList">
        <xsd:complexType/>
      </xsd:element>
      <xsd:complexType name="ArrayOfString">
        <xsd:sequence>
          <xsd:element maxOccurs="unbounded" minOccurs="0" name="string"
nillable="true" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>
      <xsd:element name="getDirectoryListResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element maxOccurs="1" minOccurs="1" name="out" nillable="true"
type="ns1:ArrayOfPerson"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="getPeopleByName">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element maxOccurs="1" minOccurs="1" name="name" nillable="true"
type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="getPeopleByNameResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element maxOccurs="1" minOccurs="1" name="out" nillable="true"
type="ns1:ArrayOfPerson"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="getPersonById">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element maxOccurs="1" minOccurs="1" name="id" nillable="true"
type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:schema>
  </wsdl:types>

```

```

</xsd:element>
<xsd:element name="getPersonByIdResponse">
<xsd:complexType>
<xsd:sequence>
<xsd:element maxOccurs="1" minOccurs="1" name="out" nillable="true"
type="ns1:Person"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:element name="getDirectory">
<xsd:complexType/>
</xsd:element>
<xsd:element name="getDirectoryResponse">
<xsd:complexType>
<xsd:sequence>
<xsd:element maxOccurs="1" minOccurs="1" name="out" nillable="true"
type="ns1:ArrayOfPerson"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:element name="getNameById">
<xsd:complexType>
<xsd:sequence>
<xsd:element maxOccurs="1" minOccurs="1" name="id" nillable="true"
type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:element name="getNameByIdResponse">
<xsd:complexType>
<xsd:sequence>
<xsd:element maxOccurs="1" minOccurs="1" name="out" nillable="true"
type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
attributeFormDefault="qualified" elementFormDefault="qualified"
targetNamespace="http://data.directoryservice">
<xsd:complexType name="ArrayOfPerson">
<xsd:sequence>
<xsd:element maxOccurs="unbounded" minOccurs="0" name="Person"
nillable="true" type="ns1:Person"/>
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Person">
<xsd:sequence>
<xsd:element minOccurs="0" name="emailAddresses" nillable="true"
type="tns:ArrayOfString"/>
<xsd:element minOccurs="0" name="firstName" nillable="true"
type="xsd:string"/>
<xsd:element minOccurs="0" name="id" nillable="true"
type="xsd:string"/>
<xsd:element minOccurs="0" name="lastName" nillable="true"
type="xsd:string"/>

```

```

<xsd:element minOccurs="0" name="phone" nillable="true"
type="ns1:PhoneNumber"/>
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="PhoneNumber">
<xsd:sequence>
<xsd:element minOccurs="0" name="areaCode" nillable="true"
type="xsd:int"/>
<xsd:element minOccurs="0" name="firstThree" nillable="true"
type="xsd:int"/>
<xsd:element minOccurs="0" name="lastFour" nillable="true"
type="xsd:int"/>
</xsd:sequence>
</xsd:complexType>
</xsd:schema>
</wsdl:types>
<wsdl:message name="getNameByIdResponse">
<wsdl:part name="parameters" element="tns:getNameByIdResponse">
</wsdl:part>
</wsdl:message>
<wsdl:message name="getNameByIdRequest">
<wsdl:part name="parameters" element="tns:getNameById">
</wsdl:part>
</wsdl:message>
<wsdl:message name="getPeopleByNameRequest">
<wsdl:part name="parameters" element="tns:getPeopleByName">
</wsdl:part>
</wsdl:message>
<wsdl:message name="getDirectoryRequest">
<wsdl:part name="parameters" element="tns:getDirectory">
</wsdl:part>
</wsdl:message>
<wsdl:message name="getPersonByIdRequest">
<wsdl:part name="parameters" element="tns:getPersonById">
</wsdl:part>
</wsdl:message>
<wsdl:message name="getDirectoryListResponse">
<wsdl:part name="parameters"
element="tns:getDirectoryListResponse">
</wsdl:part>
</wsdl:message>
<wsdl:message name="getDirectoryListRequest">
<wsdl:part name="parameters" element="tns:getDirectoryList">
</wsdl:part>
</wsdl:message>
<wsdl:message name="getPersonByIdResponse">
<wsdl:part name="parameters" element="tns:getPersonByIdResponse">
</wsdl:part>
</wsdl:message>
<wsdl:message name="getDirectoryResponse">
<wsdl:part name="parameters" element="tns:getDirectoryResponse">
</wsdl:part>
</wsdl:message>
<wsdl:message name="getPeopleByNameResponse">
<wsdl:part name="parameters" element="tns:getPeopleByNameResponse">
</wsdl:part>
</wsdl:message>

```

```

<wsdl:portType name="DirectoryServicePortType">
  <wsdl:operation name="getDirectoryList">
    <wsdl:input name="getDirectoryListRequest"
message="tns:getDirectoryListRequest">
    </wsdl:input>
    <wsdl:output name="getDirectoryListResponse"
message="tns:getDirectoryListResponse">
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="getPeopleByName">
    <wsdl:input name="getPeopleByNameRequest"
message="tns:getPeopleByNameRequest">
    </wsdl:input>
    <wsdl:output name="getPeopleByNameResponse"
message="tns:getPeopleByNameResponse">
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="getPersonById">
    <wsdl:input name="getPersonByIdRequest"
message="tns:getPersonByIdRequest">
    </wsdl:input>
    <wsdl:output name="getPersonByIdResponse"
message="tns:getPersonByIdResponse">
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="getDirectory">
    <wsdl:input name="getDirectoryRequest"
message="tns:getDirectoryRequest">
    </wsdl:input>
    <wsdl:output name="getDirectoryResponse"
message="tns:getDirectoryResponse">
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="getNameById">
    <wsdl:input name="getNameByIdRequest"
message="tns:getNameByIdRequest">
    </wsdl:input>
    <wsdl:output name="getNameByIdResponse"
message="tns:getNameByIdResponse">
    </wsdl:output>
  </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="DirectoryServiceHttpBinding"
type="tns:DirectoryServicePortType">
  <wsdlsoap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="getDirectoryList">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="getDirectoryListRequest">
      <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="getDirectoryListResponse">
      <wsdlsoap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="getPeopleByName">
    <wsdlsoap:operation soapAction=""/>

```

```

    <wsdl:input name="getPeopleByNameRequest">
      <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="getPeopleByNameResponse">
      <wsdlsoap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="getPersonById">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="getPersonByIdRequest">
      <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="getPersonByIdResponse">
      <wsdlsoap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="getDirectory">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="getDirectoryRequest">
      <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="getDirectoryResponse">
      <wsdlsoap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="getNameById">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="getNameByIdRequest">
      <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="getNameByIdResponse">
      <wsdlsoap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
<wsdl:service name="DirectoryService">
  <wsdl:port name="DirectoryServiceHttpPort"
binding="tns:DirectoryServiceHttpBinding">
    <wsdlsoap:address
location="http://localhost:8080/XFire/services/DirectoryService"/>
  </wsdl:port>
</wsdl:service>
</wsdl:definitions>

```

XFire Data Type Handling

Data Types

```
<xsd:complexType name="ArrayOfString">
  <xsd:sequence>
    <xsd:element maxOccurs="unbounded" minOccurs="0"
name="string" nillable="true" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="ArrayOfPerson">
  <xsd:sequence>
    <xsd:element maxOccurs="unbounded" minOccurs="0"
name="Person" nillable="true" type="ns1:Person"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Person">
  <xsd:sequence>
    <xsd:element minOccurs="0" name="emailAddresses"
nillable="true" type="tns:ArrayOfString"/>
    <xsd:element minOccurs="0" name="firstName" nillable="true"
type="xsd:string"/>
    <xsd:element minOccurs="0" name="id" nillable="true"
type="xsd:string"/>
    <xsd:element minOccurs="0" name="lastName" nillable="true"
type="xsd:string"/>
    <xsd:element minOccurs="0" name="phone" nillable="true"
type="ns1:PhoneNumber"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="PhoneNumber">
  <xsd:sequence>
    <xsd:element minOccurs="0" name="areaCode" nillable="true"
type="xsd:int"/>
    <xsd:element minOccurs="0" name="firstThree"
nillable="true" type="xsd:int"/>
    <xsd:element minOccurs="0" name="lastFour" nillable="true"
type="xsd:int"/>
  </xsd:sequence>
</xsd:complexType>
```

SOAP Operations

HashSet<Person> getDirectory

WSDL

Request:

```
<xsd:element name="getDirectory">
  <xsd:complexType/>
</xsd:element>
```

Response:

```
<xsd:element name="getDirectoryResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element maxOccurs="1" minOccurs="1" name="people"
        nillable="true" type="ns1:ArrayOfPerson"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Actual

Request:

```
<dir:getDirectory/>
```

Response:

```
<ns1:getDirectoryResponse xmlns:ns1="http://directoryservice">
  <ns1:people>
    <ns2:Person xmlns:ns2="http://data.directoryservice">
      <emailAddresses xmlns="http://data.directoryservice">
        <ns1:string>gcs@wpi.edu</ns1:string>
        <ns1:string>gary.schorer@ll.mit.edu</ns1:string>
      </emailAddresses>
      <firstName
xmlns="http://data.directoryservice">Gary</firstName>
      <id xmlns="http://data.directoryservice">2</id>
      <lastName
xmlns="http://data.directoryservice">Schorer</lastName>
      <phone xmlns="http://data.directoryservice">
        <areaCode>781</areaCode>
        <firstThree>981</firstThree>
        <lastFour>5356</lastFour>
      </phone>
    </ns2:Person>
    <ns2:Person xmlns:ns2="http://data.directoryservice">
      <emailAddresses xmlns="http://data.directoryservice">
        <ns1:string>gcs@wpi.edu</ns1:string>
        <ns1:string>gary.schorer@ll.mit.edu</ns1:string>
      </emailAddresses>
      <firstName
xmlns="http://data.directoryservice">Gary</firstName>
      <id xmlns="http://data.directoryservice">1</id>
```

```
        <lastName
xmlns="http://data.directoryservice">Schorer</lastName>
        <phone xmlns="http://data.directoryservice">
          <areaCode>781</areaCode>
          <firstThree>956</firstThree>
          <lastFour>2600</lastFour>
        </phone>
      </ns2:Person>
    </ns1:people>
  </ns1:getDirectoryResponse>
```

ArrayList<Person> getDirectoryList

WSDL

Request:

```
<xsd:element name="getDirectoryList">
  <xsd:complexType/>
</xsd:element>
```

Response:

```
<xsd:element name="getDirectoryListResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element maxOccurs="1" minOccurs="1"
name="people" nillable="true" type="ns1:ArrayOfPerson"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Actual

Request:

```
<dir:getDirectoryList/>
```

Response:

```
<ns1:getDirectoryListResponse xmlns:ns1="http://directoryservice">
  <ns1:people>
    <ns2:Person xmlns:ns2="http://data.directoryservice">
      <emailAddresses xmlns="http://data.directoryservice">
        <ns1:string>gcs@wpi.edu</ns1:string>
        <ns1:string>gary.schorer@ll.mit.edu</ns1:string>
      </emailAddresses>
      <firstName
xmlns="http://data.directoryservice">Gary</firstName>
      <id xmlns="http://data.directoryservice">2</id>
      <lastName
xmlns="http://data.directoryservice">Schorer</lastName>
      <phone xmlns="http://data.directoryservice">
        <areaCode>781</areaCode>
        <firstThree>981</firstThree>
        <lastFour>5356</lastFour>
      </phone>
    </ns2:Person>
    <ns2:Person xmlns:ns2="http://data.directoryservice">
      <emailAddresses xmlns="http://data.directoryservice">
        <ns1:string>gcs@wpi.edu</ns1:string>
        <ns1:string>gary.schorer@ll.mit.edu</ns1:string>
      </emailAddresses>
      <firstName
xmlns="http://data.directoryservice">Gary</firstName>
      <id xmlns="http://data.directoryservice">1</id>
      <lastName
xmlns="http://data.directoryservice">Schorer</lastName>
      <phone xmlns="http://data.directoryservice">
```

```
        <areaCode>781</areaCode>
        <firstThree>956</firstThree>
        <lastFour>2600</lastFour>
    </phone>
</ns2:Person>
</ns1:people>
</ns1:getDirectoryListResponse>
```

Person[] getPeopleByName(String)

WSDL

Request:

```
<xsd:element name="getPeopleByName">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element maxOccurs="1" minOccurs="1"
name="name" nillable="true" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Response:

```
<xsd:element name="getPeopleByNameResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element maxOccurs="1" minOccurs="1"
name="people" nillable="true" type="ns1:ArrayOfPerson"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Actual

Request:

```
<dir:getPeopleByName>
  <dir:name>a</dir:name>
</dir:getPeopleByName>
```

Response:

```
<ns1:getPeopleByNameResponse xmlns:ns1="http://directoryservice">
  <ns1:people>
    <ns2:Person xmlns:ns2="http://data.directoryservice">
      <emailAddresses xmlns="http://data.directoryservice">
        <ns1:string>gcs@wpi.edu</ns1:string>
        <ns1:string>gary.schorer@ll.mit.edu</ns1:string>
      </emailAddresses>
      <firstName
xmlns="http://data.directoryservice">Gary</firstName>
      <id xmlns="http://data.directoryservice">2</id>
      <lastName
xmlns="http://data.directoryservice">Schorer</lastName>
      <phone xmlns="http://data.directoryservice">
        <areaCode>781</areaCode>
        <firstThree>981</firstThree>
        <lastFour>5356</lastFour>
      </phone>
    </ns2:Person>
    <ns2:Person xmlns:ns2="http://data.directoryservice">
      <emailAddresses xmlns="http://data.directoryservice">
        <ns1:string>gcs@wpi.edu</ns1:string>
```

```
        <ns1:string>gary.schorer@ll.mit.edu</ns1:string>
    </emailAddresses>
    <firstName
xmlns="http://data.directoryservice">Gary</firstName>
    <id xmlns="http://data.directoryservice">1</id>
    <lastName
xmlns="http://data.directoryservice">Schorer</lastName>
    <phone xmlns="http://data.directoryservice">
        <areaCode>781</areaCode>
        <firstThree>956</firstThree>
        <lastFour>2600</lastFour>
    </phone>
    </ns2:Person>
</ns1:people>
</ns1:getPeopleByNameResponse>
```

Person getPersonById(String)

WSDL

Request:

```
<xsd:element name="getPersonById">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element maxOccurs="1" minOccurs="1"
name="id" nillable="true" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Response:

```
<xsd:element name="getPersonByIdResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element maxOccurs="1" minOccurs="1"
name="person" nillable="true" type="ns1:Person"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Actual

Request:

```
<dir:getPersonById>
  <dir:id>1</dir:id>
</dir:getPersonById>
```

Response:

```
<ns1:getPersonByIdResponse xmlns:ns1="http://directoryservice">
  <ns1:person>
    <emailAddresses xmlns="http://data.directoryservice">
      <ns1:string>gcs@wpi.edu</ns1:string>
      <ns1:string>gary.schorer@ll.mit.edu</ns1:string>
    </emailAddresses>
    <firstName
xmlns="http://data.directoryservice">Gary</firstName>
    <id xmlns="http://data.directoryservice">1</id>
    <lastName
xmlns="http://data.directoryservice">Schorer</lastName>
    <phone xmlns="http://data.directoryservice">
      <areaCode>781</areaCode>
      <firstThree>956</firstThree>
      <lastFour>2600</lastFour>
    </phone>
  </ns1:person>
</ns1:getPersonByIdResponse>
```

String getNameById(String)

WSDL

Request:

```
<xsd:element name="getNameById">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element maxOccurs="1" minOccurs="1"
name="id" nillable="true" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Response:

```
<xsd:element name="getNameByIdResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element maxOccurs="1" minOccurs="1"
name="name" nillable="true" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Actual

Request:

```
<dir:getNameById>
  <dir:id>1</dir:id>
</dir:getNameById>
```

Response:

```
<ns1:getNameByIdResponse xmlns:ns1="http://directoryservice">
  <ns1:name>Gary Schorer</ns1:name>
</ns1:getNameByIdResponse>
```

void addPerson(Person)

WSDL

Request:

```
<xsd:element name="addPerson">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element maxOccurs="1" minOccurs="1"
name="toAdd" nillable="true" type="ns1:Person"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Response:

```
<xsd:element name="addPersonResponse">
  <xsd:complexType/>
</xsd:element>
```

Actual

Request:

```
<dir:addPerson>
  <dir:toAdd>
    <data:emailAddresses>
      <dir:string>gcs@wpi.edu</dir:string>
      <dir:string>gary.schorer@ll.mit.edu</dir:string>
    </data:emailAddresses>
    <data:firstName>Gary</data:firstName>
    <data:id>0</data:id>
    <data:lastName>Schorer</data:lastName>
    <data:phone>
      <data:areaCode>781</data:areaCode>
      <data:firstThree>981</data:firstThree>
      <data:lastFour>5356</data:lastFour>
    </data:phone>
  </dir:toAdd>
</dir:addPerson>
```

Response:

```
<ns1:addPersonResponse xmlns:ns1="http://directoryservice"/>
```

void addPeople(ArrayList<Person>)

WSDL

Request:

```
<xsd:element name="addPeople">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element maxOccurs="1" minOccurs="1"
name="toAdd" nillable="true" type="ns1:ArrayOfPerson"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Response:

```
<xsd:element name="addPeopleResponse">
  <xsd:complexType/>
</xsd:element>
```

Actual

Request:

```
<dir:addPeople>
  <dir:toAdd>
    <data:Person>
      <data:emailAddresses>
        <dir:string>gcs@wpi.edu</dir:string>
        <dir:string>gary.schorer@ll.mit.edu</dir:string>
      </data:emailAddresses>
      <data:firstName>Gary</data:firstName>
      <data:id>0</data:id>
      <data:lastName>Schorer</data:lastName>
      <data:phone>
        <data:areaCode>781</data:areaCode>
        <data:firstThree>981</data:firstThree>
        <data:lastFour>5356</data:lastFour>
      </data:phone>
    </data:Person>
    <data:Person>
      <data:emailAddresses>
        <dir:string>gcs@wpi.edu</dir:string>
        <dir:string>gary.schorer@ll.mit.edu</dir:string>
      </data:emailAddresses>
      <data:firstName>Gary</data:firstName>
      <data:id>0</data:id>
      <data:lastName>Schorer</data:lastName>
      <data:phone>
        <data:areaCode>781</data:areaCode>
        <data:firstThree>981</data:firstThree>
        <data:lastFour>5356</data:lastFour>
      </data:phone>
    </data:Person>
  </dir:toAdd>
```

```
</dir:addPeople>
```

Response:

```
<ns1:addPeopleResponse xmlns:ns1="http://directoryservice"/>
```

void addPeopleArray(Person[])

WSDL

Request:

```
<xsd:element name="addPeopleArray">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element maxOccurs="1" minOccurs="1"
name="toAdd" nillable="true" type="ns1:ArrayOfPerson"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Response:

```
<xsd:element name="addPeopleArrayResponse">
  <xsd:complexType/>
</xsd:element>
```

Actual

Request:

```
<dir:addPeopleArray>
  <dir:toAdd>
    <data:Person>
      <data:emailAddresses>
        <dir:string>gcs@wpi.edu</dir:string>
        <dir:string>gary.schorer@ll.mit.edu</dir:string>
      </data:emailAddresses>
      <data:firstName>Gary</data:firstName>
      <data:id>0</data:id>
      <data:lastName>Schorer</data:lastName>
      <data:phone>
        <data:areaCode>781</data:areaCode>
        <data:firstThree>981</data:firstThree>
        <data:lastFour>5356</data:lastFour>
      </data:phone>
    </data:Person>
    <data:Person>
      <data:emailAddresses>
        <dir:string>gcs@wpi.edu</dir:string>
        <dir:string>gary.schorer@ll.mit.edu</dir:string>
      </data:emailAddresses>
      <data:firstName>Gary</data:firstName>
      <data:id>0</data:id>
      <data:lastName>Schorer</data:lastName>
      <data:phone>
        <data:areaCode>781</data:areaCode>
        <data:firstThree>981</data:firstThree>
        <data:lastFour>5356</data:lastFour>
      </data:phone>
    </data:Person>
  </dir:toAdd>
```

```
</dir:addPeopleArray>
```

Response:

```
<ns1:addPeopleArrayResponse xmlns:ns1="http://directoryservice"/>
```

APPENDIX C: AXIS2 ANALYSIS RESULTS

Setup Procedure

1. Drag Axis2 lib folder contents into WebContent/WEB-INF/lib in Eclipse.
2. Create the Axis2SpringServlet, in the package servlet. (see source below) This Servlet is required to gain access to the Axis2 Servlet via Spring.
3. Setup the web.xml file (see below).
4. Add beans for each service class to applicationContext.xml and service declarations to services.xml (see below) Note that the axis2 bean must be left in the applicationContext.xml file, as it is required for the Servlet to run.

Spring Support

None by default, possible with custom servlet creation. Services must still be declared in at least one services.xml file, they will not be automatically detected from Spring beans.

REST Support

All services are exposed via REST by default. Operations are of the format:

<http://.../services/ServiceName?param1Name=value1Name¶m2Name=value2Name>

JSR181 Annotation Support

Almost full. Correctly responds to `@WebService (serviceName)`, `@WebParam (name)`, `@WebMethod (operationName, exclude)`. Does not respond to `@WebService (endpointInterface)` and `@WebResult (name)`.

Fault Handling

Exceptions thrown in service operations will result in a SOAP fault being returned.

Faultcode is set to soapenv:Server, faultstring is set to the exception message, and detail is set to the stacktrace of the exception.

Required Configuration Files

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4"
xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
  <display-name>Axis2</display-name>
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/applicationContext.xml</param-value>
  </context-param>
  <listener>
    <listener-
class>org.springframework.web.context.ContextLoaderListener</listener-
class>
    </listener>
  <servlet>
    <servlet-name>AxisServlet</servlet-name>
    <servlet-class>servlet.AxisSpringServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>AxisServlet</servlet-name>
    <url-pattern>/services/*</url-pattern>
  </servlet-mapping>
</web-app>
```

applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:aop="http://www.springframework.org/schema/aop"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-
2.0.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-2.0.xsd">
  <bean id="axis2"
class="org.apache.axis2.transport.http.AxisServlet" />

  <bean id="directoryService" class="directoryservice.Directory"
init-method="initialize" />
</beans>
```

services/DirectoryService/META-INF/services.xml

```
<service name="DirectoryService" scope="application">
  <description>Directory Service</description>
  <messageReceivers>
```

```

        <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-only"
class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver"/>
        <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-out"
class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
    </messageReceivers>
    <parameter
name="ServiceObjectSupplier">org.apache.axis2.extensions.spring.receive
rs.SpringServletContextObjectSupplier</parameter>
    <parameter name="SpringBeanName">directoryService</parameter>
</service>

```

Source of servlet.AxisSpringServlet.java

```

package servlet;

import java.io.IOException;
import java.util.Enumeration;

import javax.servlet.ServletConfig;
import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;

import org.apache.axis2.AxisFault;
import org.apache.axis2.addressing.EndpointReference;
import org.apache.axis2.context.ConfigurationContext;
import org.apache.axis2.context.MessageContext;
import org.apache.axis2.context.SessionContext;
import org.apache.axis2.description.TransportInDescription;
import org.apache.axis2.transport.http.AxisServlet;
import
org.springframework.web.context.support.WebApplicationContextUtils;

public class AxisSpringServlet extends HttpServlet {
    /**
     *
     */
    private static final long serialVersionUID = -
8212869970214286587L;

    private AxisServlet axis2;

    private void initializeAxis() {
        if (axis2 == null) {
            axis2 = (AxisServlet)
WebApplicationContextUtils.getRequiredWebApplicationContext(super.getSe
rvletContext()).getBean("axis2");
        }
    }

    @Override
    public void init() throws ServletException {
        super.init();
        initializeAxis();
    }
}

```

```

        axis2.init();
    }

    public void destroy() {
        axis2.destroy();
    }

    public boolean equals(Object arg0) {
        return axis2.equals(arg0);
    }

    public EndpointReference getEPRForService(String serviceName,
String ip) throws AxisFault {
        return axis2.getEPRForService(serviceName, ip);
    }

    public EndpointReference[] getEPRsForService(String serviceName,
String ip) throws AxisFault {
        return axis2.getEPRsForService(serviceName, ip);
    }

    public String getInitParameter(String arg0) {
        return axis2.getInitParameter(arg0);
    }

    public Enumeration getInitParameterNames() {
        return axis2.getInitParameterNames();
    }

    public ServletConfig getServletConfig() {
        if (axis2 == null)
            return super.getServletConfig();
        else
            return axis2.getServletConfig();
    }

    public ServletContext getServletContext() {
        return axis2.getServletContext();
    }

    public String getServletInfo() {
        return axis2.getServletInfo();
    }

    public String getServletName() {
        return axis2.getServletName();
    }

    public SessionContext getSessionContext(MessageContext
messageContext) {
        return axis2.getSessionContext(messageContext);
    }

    public int hashCode() {
        return axis2.hashCode();
    }

```

```

    public void init(ConfigurationContext axisConf,
TransportInDescription transprtIn) throws AxisFault {
        initializeAxis();
        axis2.init(axisConf, transprtIn);
    }

    public void init(ServletConfig config) throws ServletException {
        super.init(config);
        initializeAxis();
        axis2.init(config);
    }

    public void initContextRoot(HttpServletRequest req) {
        axis2.initContextRoot(req);
    }

    public void log(String arg0, Throwable arg1) {
        axis2.log(arg0, arg1);
    }

    public void log(String arg0) {
        axis2.log(arg0);
    }

    public void service(ServletRequest arg0, ServletResponse arg1)
throws ServletException, IOException {
        axis2.service(arg0, arg1);
    }

    public void start() throws AxisFault {
        axis2.start();
    }

    public void stop() throws AxisFault {
        axis2.stop();
    }

    public String toString() {
        return axis2.toString();
    }
}

```

WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:ns2="http://directoryservice"
xmlns:ns1="http://data.directoryservice/xsd"
xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:ns0="http://util.java/xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
targetNamespace="http://directoryservice">
  <wsdl:documentation>DirectoryService</wsdl:documentation>
  <wsdl:types>
    <xs:schema xmlns:ax22="http://util.java/xsd"
attributeFormDefault="qualified" elementFormDefault="qualified"
targetNamespace="http://util.java/xsd">
      <xs:complexType name="AbstractCollection">
        <xs:sequence>
          <xs:element minOccurs="0" name="empty"
type="xs:boolean"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="AbstractSet">
        <xs:complexContent>
          <xs:extension base="ax22:AbstractCollection">
            <xs:sequence/>
          </xs:extension>
        </xs:complexContent>
      </xs:complexType>
      <xs:complexType name="HashSet">
        <xs:complexContent>
          <xs:extension base="ax22:AbstractSet">
            <xs:sequence>
              <xs:element minOccurs="0" name="empty"
type="xs:boolean"/>
            </xs:sequence>
          </xs:extension>
        </xs:complexContent>
      </xs:complexType>
    </xs:schema>
    <xs:schema xmlns:ax21="http://data.directoryservice/xsd"
attributeFormDefault="qualified" elementFormDefault="qualified"
targetNamespace="http://data.directoryservice/xsd">
      <xs:complexType name="Person">
        <xs:sequence>
          <xs:element minOccurs="0" name="emailAddresses"
nillable="true" type="xs:anyType"/>
          <xs:element minOccurs="0" name="firstName"
nillable="true" type="xs:string"/>
          <xs:element minOccurs="0" name="id" nillable="true"
type="xs:string"/>
          <xs:element minOccurs="0" name="lastName"
nillable="true" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
    </xs:schema>
  </wsdl:types>

```

```

        <xs:element minOccurs="0" name="phone"
nillable="true" type="ax21:PhoneNumber"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="PhoneNumber">
    <xs:sequence>
        <xs:element minOccurs="0" name="areaCode"
nillable="true" type="xs:int"/>
        <xs:element minOccurs="0" name="firstThree"
nillable="true" type="xs:int"/>
        <xs:element minOccurs="0" name="lastFour"
nillable="true" type="xs:int"/>
    </xs:sequence>
</xs:complexType>
</xs:schema>
<xs:schema xmlns:ns="http://directoryservice"
attributeFormDefault="qualified" elementFormDefault="qualified"
targetNamespace="http://directoryservice">
    <xs:element name="getPersonById">
        <xs:complexType>
            <xs:sequence>
                <xs:element minOccurs="0" name="id"
nillable="true" type="xs:string"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="getPersonByIdResponse">
        <xs:complexType>
            <xs:sequence>
                <xs:element minOccurs="0" name="return"
nillable="true" type="ns1:Person"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:complexType name="Exception">
        <xs:sequence>
            <xs:element minOccurs="0" name="Exception"
nillable="true" type="xs:anyType"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="Exception">
        <xs:complexType>
            <xs:sequence>
                <xs:element minOccurs="0" name="Exception"
nillable="true" type="ns:Exception"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="getPeopleByName">
        <xs:complexType>
            <xs:sequence>
                <xs:element minOccurs="0" name="name"
nillable="true" type="xs:string"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="getPeopleByNameResponse">

```

```

        <xs:complexType>
            <xs:sequence>
                <xs:element maxOccurs="unbounded" minOccurs="0"
name="return" nillable="true" type="ns1:Person"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="getNameById">
        <xs:complexType>
            <xs:sequence>
                <xs:element minOccurs="0" name="id"
nillable="true" type="xs:string"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="getNameByIdResponse">
        <xs:complexType>
            <xs:sequence>
                <xs:element minOccurs="0" name="return"
nillable="true" type="xs:string"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="getDirectoryListResponse">
        <xs:complexType>
            <xs:sequence>
                <xs:element minOccurs="0" name="return"
nillable="true" type="xs:anyType"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="getDirectoryResponse">
        <xs:complexType>
            <xs:sequence>
                <xs:element minOccurs="0" name="return"
nillable="true" type="ns0:HashSet"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
</wsdl:types>
<wsdl:message name="getDirectoryListRequest"/>
<wsdl:message name="getDirectoryListResponse">
    <wsdl:part name="parameters"
element="ns2:getDirectoryListResponse"/>
</wsdl:message>
<wsdl:message name="getPeopleByNameRequest">
    <wsdl:part name="parameters" element="ns2:getPeopleByName"/>
</wsdl:message>
<wsdl:message name="getPeopleByNameResponse">
    <wsdl:part name="parameters"
element="ns2:getPeopleByNameResponse"/>
</wsdl:message>
<wsdl:message name="Exception">
    <wsdl:part name="parameters" element="ns2:Exception"/>
</wsdl:message>
<wsdl:message name="getPersonByIdRequest">

```

```

        <wsdl:part name="parameters" element="ns2:getPersonById" />
    </wsdl:message>
    <wsdl:message name="getPersonByIdResponse">
        <wsdl:part name="parameters"
element="ns2:getPersonByIdResponse" />
    </wsdl:message>
    <wsdl:message name="getDirectoryRequest" />
    <wsdl:message name="getDirectoryResponse">
        <wsdl:part name="parameters"
element="ns2:getDirectoryResponse" />
    </wsdl:message>
    <wsdl:message name="getNameByIdRequest">
        <wsdl:part name="parameters" element="ns2:getNameById" />
    </wsdl:message>
    <wsdl:message name="getNameByIdResponse">
        <wsdl:part name="parameters"
element="ns2:getNameByIdResponse" />
    </wsdl:message>
    <wsdl:portType name="DirectoryServicePortType">
        <wsdl:operation name="getDirectoryList">
            <wsdl:input message="ns2:getDirectoryListRequest"
wsaw:Action="urn:getDirectoryList" />
            <wsdl:output message="ns2:getDirectoryListResponse"
wsaw:Action="urn:getDirectoryListResponse" />
        </wsdl:operation>
        <wsdl:operation name="getPeopleByName">
            <wsdl:input message="ns2:getPeopleByNameRequest"
wsaw:Action="urn:getPeopleByName" />
            <wsdl:output message="ns2:getPeopleByNameResponse"
wsaw:Action="urn:getPeopleByNameResponse" />
            <wsdl:fault message="ns2:Exception" name="Exception"
wsaw:Action="urn:getPeopleByNameException" />
        </wsdl:operation>
        <wsdl:operation name="getPersonById">
            <wsdl:input message="ns2:getPersonByIdRequest"
wsaw:Action="urn:getPersonById" />
            <wsdl:output message="ns2:getPersonByIdResponse"
wsaw:Action="urn:getPersonByIdResponse" />
        </wsdl:operation>
        <wsdl:operation name="getDirectory">
            <wsdl:input message="ns2:getDirectoryRequest"
wsaw:Action="urn:getDirectory" />
            <wsdl:output message="ns2:getDirectoryResponse"
wsaw:Action="urn:getDirectoryResponse" />
        </wsdl:operation>
        <wsdl:operation name="getNameById">
            <wsdl:input message="ns2:getNameByIdRequest"
wsaw:Action="urn:getNameById" />
            <wsdl:output message="ns2:getNameByIdResponse"
wsaw:Action="urn:getNameByIdResponse" />
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="DirectoryServiceSOAP11Binding"
type="ns2:DirectoryServicePortType">
        <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document" />
        <wsdl:operation name="getDirectoryList">

```

```

        <soap:operation soapAction="urn:getDirectoryList"
style="document"/>
        <wsdl:input>
            <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
            <soap:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="getPeopleByName">
        <soap:operation soapAction="urn:getPeopleByName"
style="document"/>
        <wsdl:input>
            <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
            <soap:body use="literal"/>
        </wsdl:output>
        <wsdl:fault name="Exception">
            <soap:fault use="literal" name="Exception"/>
        </wsdl:fault>
    </wsdl:operation>
    <wsdl:operation name="getPersonById">
        <soap:operation soapAction="urn:getPersonById"
style="document"/>
        <wsdl:input>
            <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
            <soap:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="getDirectory">
        <soap:operation soapAction="urn:getDirectory"
style="document"/>
        <wsdl:input>
            <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
            <soap:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="getNameById">
        <soap:operation soapAction="urn:getNameById"
style="document"/>
        <wsdl:input>
            <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
            <soap:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
</wsdl:binding>
<wsdl:binding name="DirectoryServiceSOAP12Binding"
type="ns2:DirectoryServicePortType">
    <soap12:binding
transport="http://schemas.xmlsoap.org/soap/http" style="document"/>

```

```

        <wsdl:operation name="getDirectoryList">
            <soap12:operation soapAction="urn:getDirectoryList"
style="document"/>
            <wsdl:input>
                <soap12:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap12:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="getPeopleByName">
            <soap12:operation soapAction="urn:getPeopleByName"
style="document"/>
            <wsdl:input>
                <soap12:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap12:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="Exception">
                <soap12:fault use="literal" name="Exception"/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="getPersonById">
            <soap12:operation soapAction="urn:getPersonById"
style="document"/>
            <wsdl:input>
                <soap12:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap12:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="getDirectory">
            <soap12:operation soapAction="urn:getDirectory"
style="document"/>
            <wsdl:input>
                <soap12:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap12:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="getNameById">
            <soap12:operation soapAction="urn:getNameById"
style="document"/>
            <wsdl:input>
                <soap12:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap12:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:binding name="DirectoryServiceHttpBinding"
type="ns2:DirectoryServicePortType">
        <http:binding verb="POST"/>

```

```

        <wsdl:operation name="getDirectoryList">
            <http:operation
location="DirectoryService/getDirectoryList"/>
            <wsdl:input>
                <mime:content type="text/xml" part="getDirectoryList"/>
            </wsdl:input>
            <wsdl:output>
                <mime:content type="text/xml" part="getDirectoryList"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="getPeopleByName">
            <http:operation
location="DirectoryService/getPeopleByName"/>
            <wsdl:input>
                <mime:content type="text/xml" part="getPeopleByName"/>
            </wsdl:input>
            <wsdl:output>
                <mime:content type="text/xml" part="getPeopleByName"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="getPersonById">
            <http:operation location="DirectoryService/getPersonById"/>
            <wsdl:input>
                <mime:content type="text/xml" part="getPersonById"/>
            </wsdl:input>
            <wsdl:output>
                <mime:content type="text/xml" part="getPersonById"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="getDirectory">
            <http:operation location="DirectoryService/getDirectory"/>
            <wsdl:input>
                <mime:content type="text/xml" part="getDirectory"/>
            </wsdl:input>
            <wsdl:output>
                <mime:content type="text/xml" part="getDirectory"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="getNameById">
            <http:operation location="DirectoryService/getNameById"/>
            <wsdl:input>
                <mime:content type="text/xml" part="getNameById"/>
            </wsdl:input>
            <wsdl:output>
                <mime:content type="text/xml" part="getNameById"/>
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="DirectoryService">
        <wsdl:port name="DirectoryServiceSOAP11port_http"
binding="ns2:DirectoryServiceSOAP11Binding">
            <soap:address
location="http://localhost:8080/Axis2/services/DirectoryService"/>
        </wsdl:port>
        <wsdl:port name="DirectoryServiceSOAP12port_http"
binding="ns2:DirectoryServiceSOAP12Binding">

```

```
        <soap12:address
location="http://localhost:8080/Axis2/services/DirectoryService"/>
        </wsdl:port>
        <wsdl:port name="DirectoryServiceHttpport "
binding="ns2:DirectoryServiceHttpBinding">
        <http:address
location="http://localhost:8080/Axis2/services/DirectoryService"/>
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

Axis2 Data Type Handling

Data Types

```
<xs:complexType name="AbstractCollection">
  <xs:sequence>
    <xs:element minOccurs="0" name="empty" type="xs:boolean"/>
  </xs:sequence>
</xs:complexType>
```

```
<xs:complexType name="AbstractSet">
  <xs:complexContent>
    <xs:extension base="ax22:AbstractCollection">
      <xs:sequence/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

```
<xs:complexType name="HashSet">
  <xs:complexContent>
    <xs:extension base="ax22:AbstractSet">
      <xs:sequence>
        <xs:element minOccurs="0" name="empty"
type="xs:boolean"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

```
<xs:complexType name="Person">
  <xs:sequence>
    <xs:element minOccurs="0" name="emailAddresses"
nillable="true" type="xs:anyType"/>
    <xs:element minOccurs="0" name="firstName" nillable="true"
type="xs:string"/>
    <xs:element minOccurs="0" name="id" nillable="true"
type="xs:string"/>
    <xs:element minOccurs="0" name="lastName" nillable="true"
type="xs:string"/>
    <xs:element minOccurs="0" name="phone" nillable="true"
type="ax21:PhoneNumber"/>
  </xs:sequence>
</xs:complexType>
```

```
<xs:complexType name="PhoneNumber">
  <xs:sequence>
    <xs:element minOccurs="0" name="areaCode" nillable="true"
type="xs:int"/>
    <xs:element minOccurs="0" name="firstThree" nillable="true"
type="xs:int"/>
    <xs:element minOccurs="0" name="lastFour" nillable="true"
type="xs:int"/>
  </xs:sequence>
</xs:complexType>
```

SOAP Operations

HashSet<Person> getDirectory

WSDL

Request:

n/a

Response:

```
<xs:element name="getDirectoryResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="return"
nillable="true" type="ns0:HashSet"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Actual

Request:

n/a

Response:

```
<ns:getDirectoryResponse xmlns:ns="http://directoryservice"
xmlns:ax21="http://data.directoryservice/xsd"
xmlns:ax22="http://util.java/xsd">
  <ns:return type="directoryservice.data.Person">
    <ax21:emailAddresses>gcs@wpi.edu</ax21:emailAddresses>
<ax21:emailAddresses>gary.schorer@ll.mit.edu</ax21:emailAddresses>
  <ax21:firstName>Gary</ax21:firstName>
  <ax21:id>1</ax21:id>
  <ax21:lastName>Schorer</ax21:lastName>
  <ax21:phone type="directoryservice.data.PhoneNumber">
    <ax21:areaCode>781</ax21:areaCode>
    <ax21:firstThree>956</ax21:firstThree>
    <ax21:lastFour>2600</ax21:lastFour>
  </ax21:phone>
</ns:return>
  <ns:return type="directoryservice.data.Person">
    <ax21:emailAddresses>gcs@wpi.edu</ax21:emailAddresses>
<ax21:emailAddresses>gary.schorer@ll.mit.edu</ax21:emailAddresses>
  <ax21:firstName>Gary</ax21:firstName>
  <ax21:id>2</ax21:id>
  <ax21:lastName>Schorer</ax21:lastName>
  <ax21:phone type="directoryservice.data.PhoneNumber">
    <ax21:areaCode>781</ax21:areaCode>
    <ax21:firstThree>981</ax21:firstThree>
    <ax21:lastFour>5356</ax21:lastFour>
  </ax21:phone>
</ns:return>
</ns:getDirectoryResponse>
```

ArrayList<Person> getDirectoryList

WSDL

Request:

n/a

Response:

```
<xs:element name="getDirectoryListResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="return"
nillable="true" type="xs:anyType"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Actual

Request:

n/a

Response:

```
<ns:getDirectoryListResponse xmlns:ns="http://directoryservice"
xmlns:ax21="http://data.directoryservice/xsd"
xmlns:ax22="http://util.java/xsd">
  <ns:return type="directoryservice.data.Person">
    <ax21:emailAddresses>gcs@wpi.edu</ax21:emailAddresses>

<ax21:emailAddresses>gary.schorer@ll.mit.edu</ax21:emailAddresses>
  <ax21:firstName>Gary</ax21:firstName>
  <ax21:id>1</ax21:id>
  <ax21:lastName>Schorer</ax21:lastName>
  <ax21:phone type="directoryservice.data.PhoneNumber">
    <ax21:areaCode>781</ax21:areaCode>
    <ax21:firstThree>956</ax21:firstThree>
    <ax21:lastFour>2600</ax21:lastFour>
  </ax21:phone>
</ns:return>
  <ns:return type="directoryservice.data.Person">
    <ax21:emailAddresses>gcs@wpi.edu</ax21:emailAddresses>

<ax21:emailAddresses>gary.schorer@ll.mit.edu</ax21:emailAddresses>
  <ax21:firstName>Gary</ax21:firstName>
  <ax21:id>2</ax21:id>
  <ax21:lastName>Schorer</ax21:lastName>
  <ax21:phone type="directoryservice.data.PhoneNumber">
    <ax21:areaCode>781</ax21:areaCode>
    <ax21:firstThree>981</ax21:firstThree>
    <ax21:lastFour>5356</ax21:lastFour>
  </ax21:phone>
</ns:return>
</ns:getDirectoryListResponse>
```

Person[] getPeopleByName(String)

WSDL

Request:

```
<xs:element name="getPeopleByName">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="name"
nillable="true" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Response:

```
<xs:element name="getPeopleByNameResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element maxOccurs="unbounded" minOccurs="0"
name="return" nillable="true" type="ns1:Person"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Actual

Request:

```
<dir:getPeopleByName>
  <dir:name>a</dir:name>
</dir:getPeopleByName>
```

Response:

```
<ns:getPeopleByNameResponse xmlns:ns="http://directoryservice"
xmlns:ax21="http://data.directoryservice/xsd"
xmlns:ax22="http://util.java/xsd">
  <ns:return type="directoryservice.data.Person">
    <ax21:emailAddresses>gcs@wpi.edu</ax21:emailAddresses>
<ax21:emailAddresses>gary.schorer@ll.mit.edu</ax21:emailAddresses>
    <ax21:firstName>Gary</ax21:firstName>
    <ax21:id>1</ax21:id>
    <ax21:lastName>Schorer</ax21:lastName>
    <ax21:phone type="directoryservice.data.PhoneNumber">
      <ax21:areaCode>781</ax21:areaCode>
      <ax21:firstThree>956</ax21:firstThree>
      <ax21:lastFour>2600</ax21:lastFour>
    </ax21:phone>
  </ns:return>
  <ns:return type="directoryservice.data.Person">
    <ax21:emailAddresses>gcs@wpi.edu</ax21:emailAddresses>
<ax21:emailAddresses>gary.schorer@ll.mit.edu</ax21:emailAddresses>
```

```
<ax21:firstName>Gary</ax21:firstName>
<ax21:id>2</ax21:id>
<ax21:lastName>Schorer</ax21:lastName>
<ax21:phone type="directoryservice.data.PhoneNumber">
  <ax21:areaCode>781</ax21:areaCode>
  <ax21:firstThree>981</ax21:firstThree>
  <ax21:lastFour>5356</ax21:lastFour>
</ax21:phone>
</ns:return>
</ns:getPeopleByNameResponse>
```

Person getPersonById(String)

WSDL

Request:

```
<xs:element name="getPersonById">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="id"
nillable="true" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Response:

```
<xs:element name="getPersonByIdResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="return"
nillable="true" type="ns1:Person"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Actual

Request:

```
<dir:getPersonById>
  <dir:id>1</dir:id>
</dir:getPersonById>
```

Response:

```
<ns:getPersonByIdResponse xmlns:ns="http://directoryservice">
  <ns:return type="directoryservice.data.Person"
xmlns:ax21="http://data.directoryservice/xsd"
xmlns:ax22="http://util.java/xsd">
    <ax21:emailAddresses>gcs@wpi.edu</ax21:emailAddresses>

<ax21:emailAddresses>gary.schorer@ll.mit.edu</ax21:emailAddresses>
    <ax21:firstName>Gary</ax21:firstName>
    <ax21:id>1</ax21:id>
    <ax21:lastName>Schorer</ax21:lastName>
    <ax21:phone type="directoryservice.data.PhoneNumber">
      <ax21:areaCode>781</ax21:areaCode>
      <ax21:firstThree>956</ax21:firstThree>
      <ax21:lastFour>2600</ax21:lastFour>
    </ax21:phone>
  </ns:return>
</ns:getPersonByIdResponse>
```

String getNameById(String)

WSDL

Request:

```
<xs:element name="getNameById">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="id"
nillable="true" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Response:

```
<xs:element name="getNameByIdResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="return"
nillable="true" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Actual

Request:

```
<dir:getNameById>
  <dir:id>1</dir:id>
</dir:getNameById>
```

Response:

```
<ns:getNameByIdResponse xmlns:ns="http://directoryservice">
  <ns:return>Gary Schorer</ns:return>
</ns:getNameByIdResponse>
```

void addPerson(Person)

WSDL

Request:

```
<xs:element name="addPerson">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="toAdd"
nillable="true" type="ns1:Person"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Response:

n/a

Actual

Request:

```
<dir:addPerson>
  <dir:toAdd>
    <xsd:emailAddresses>gcs@wpi.edu</xsd:emailAddresses>
<xsd:emailAddresses>gary.schorer@ll.mit.edu</xsd:emailAddresses>
  <xsd:firstName>Gary</xsd:firstName>
  <xsd:id>0</xsd:id>
  <xsd:lastName>Schorer</xsd:lastName>
  <xsd:phone>
    <xsd:areaCode>781</xsd:areaCode>
    <xsd:firstThree>981</xsd:firstThree>
    <xsd:lastFour>5356</xsd:lastFour>
  </xsd:phone>
</dir:toAdd>
</dir:addPerson>
```

Response:

n/a

Notes:

Email addresses will not be stored properly. Axis2 fails to correctly pass Collection elements, resulting in all of their contents being lost.

void addPeople(ArrayList<Person>)

WSDL

Request:

```
<xs:element name="addPeople">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="toAdd"
nillable="true" type="xs:anyType"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Response:

n/a

Actual

Request:

```
<dir:addPeople>
  <dir:toAdd?></dir:toAdd>
</dir:addPeople>
```

Response:

n/a

Notes:

Due to the fact that WSDL defines the request element as being of type `xs:anyType`, it is impossible to know how it should be formatted. No format was found that could result in Axis2 being able to properly convert the XML into Java.

void addPeopleArray(Person[])

WSDL

Request:

```
<xs:element name="addPeopleArray">
  <xs:complexType>
    <xs:sequence>
      <xs:element maxOccurs="unbounded" minOccurs="0"
name="toAdd" nillable="true" type="ns1:Person"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Response:

n/a

Actual

Request:

```
<dir:toAdd>
  <xsd:emailAddresses>gcs@wpi.edu</xsd:emailAddresses>
<xsd:emailAddresses>gary.schorer@ll.mit.edu</xsd:emailAddresses>
  <xsd:firstName>Gary</xsd:firstName>
  <xsd:id>5</xsd:id>
  <xsd:lastName>Schorer</xsd:lastName>
  <xsd:phone>
    <xsd:areaCode>781</xsd:areaCode>
    <xsd:firstThree>981</xsd:firstThree>
    <xsd:lastFour>5356</xsd:lastFour>
  </xsd:phone>
</dir:toAdd>
<dir:toAdd>
  <xsd:emailAddresses>gcs@wpi.edu</xsd:emailAddresses>
<xsd:emailAddresses>gary.schorer@ll.mit.edu</xsd:emailAddresses>
  <xsd:firstName>Gary</xsd:firstName>
  <xsd:id>5</xsd:id>
  <xsd:lastName>Schorer</xsd:lastName>
  <xsd:phone>
    <xsd:areaCode>781</xsd:areaCode>
    <xsd:firstThree>981</xsd:firstThree>
    <xsd:lastFour>5356</xsd:lastFour>
  </xsd:phone>
</dir:toAdd>
```

Response:

n/a

Notes:

Email addresses will not be stored properly. Axis2 fails to correctly pass Collection elements, resulting in all of their contents being lost.

APPENDIX D: CXF ANALYSIS RESULTS

Setup Procedure

- 6) Annotate the service class using `@WebService` (above the class definition, optionally specifying a `serviceName`) and optionally, `@WebParam` (before each parameter in each method, specifying a name)
- 7) Drag CXF lib folder contents into `WebContent/WEB-INF/lib` in Eclipse.
- 8) Setup `web.xml` (see below)
- 9) Add `jaxws:endpoint` elements for each service class to `beans.xml` (see below)

Spring Support

Included. Enabled in the default servlet. Anything specified in the `contextConfigLocation` will then be scanned for beans with annotated classes to deploy as services. These beans are fully capable Spring beans.

REST Support

Present, but not practical to implement with POJO's. REST setup requires that an overloaded version of each function be created that takes in a complex type. This complex type needs to have a setter for each named parameter specified in the annotation `@HttpResource(location="")`. Additionally, separate `jaxws:endpoint` elements must be created for the SOAP and REST interfaces.

JSR181 Annotation Support

Full. Correctly responds to `@WebService` (`serviceName`, `endpointInterface`), `@WebParam` (`name`), `@WebMethod` (`operationName`, `exclude`) and `@WebResult` (`name`).

Fault Handling

Any exception thrown in the service will be translated into a SOAP fault. The `faultstring` of the fault will be set to whatever the message of the exception was and the `faultcode` will be set to `soap:Server`.

Required Configuration Files

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4"
xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
  <display-name>CXF</display-name>

  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>WEB-INF/applicationContext.xml</param-value>
  </context-param>

  <listener>
    <listener-class>
      org.springframework.web.context.ContextLoaderListener
    </listener-class>
  </listener>

  <servlet>
    <servlet-name>CXFServlet</servlet-name>
    <servlet-class>
      org.apache.cxf.transport.servlet.CXFServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>CXFServlet</servlet-name>
    <url-pattern>/services/*</url-pattern>
  </servlet-mapping>
</web-app>
```

applicationContext.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:jaxws="http://cxf.apache.org/jaxws"
xsi:schemaLocation="
  http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
  http://cxf.apache.org/jaxws
http://cxf.apache.org/schemas/jaxws.xsd">

  <import resource="classpath:META-INF/cxf/cxf.xml" />
  <import resource="classpath:META-INF/cxf/cxf-extension-soap.xml"
/>

  <import resource="classpath:META-INF/cxf/cxf-extension-http-
binding.xml"/>
  <import resource="classpath:META-INF/cxf/cxf-servlet.xml" />
```

```
<jaxws:endpoint id="directoryService"
implementor="directoryservice.Directory" address="/DirectoryService" />
</beans>
```

WSDL

```
<?xml version="1.0" encoding="utf-8"?><wsdl:definitions
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:ns1="http://directoryservice/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="DirectoryService"
targetNamespace="http://directoryservice/">
  <wsdl:types>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://directoryservice/" attributeFormDefault="unqualified"
elementFormDefault="unqualified"
targetNamespace="http://directoryservice/">
<xs:complexType name="phoneNumber">
<xs:sequence>
<xs:element minOccurs="0" name="areaCode" type="xs:int"/>
<xs:element minOccurs="0" name="firstThree" type="xs:int"/>
<xs:element minOccurs="0" name="lastFour" type="xs:int"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="person">
<xs:sequence>
<xs:element maxOccurs="unbounded" minOccurs="0" name="emailAddresses"
nillable="true" type="xs:string"/>
<xs:element minOccurs="0" name="firstName" type="xs:string"/>
<xs:element minOccurs="0" name="id" type="xs:string"/>
<xs:element minOccurs="0" name="lastName" type="xs:string"/>
<xs:element minOccurs="0" name="phone" type="tns:phoneNumber"/>
</xs:sequence>
</xs:complexType>
<xs:element name="getDirectoryList" type="tns:getDirectoryList"/>
<xs:complexType name="getDirectoryList">
<xs:sequence/>
</xs:complexType>
<xs:element name="getDirectoryListResponse"
type="tns:getDirectoryListResponse"/>
<xs:complexType name="getDirectoryListResponse">
<xs:sequence>
<xs:element maxOccurs="unbounded" minOccurs="0" name="person"
type="tns:person"/>
</xs:sequence>
</xs:complexType>
<xs:element name="getDirectory" type="tns:getDirectory"/>
<xs:complexType name="getDirectory">
<xs:sequence/>
</xs:complexType>
<xs:element name="getDirectoryResponse"
type="tns:getDirectoryResponse"/>
<xs:complexType name="getDirectoryResponse">
<xs:sequence>
<xs:element maxOccurs="unbounded" minOccurs="0" name="person"
type="tns:person"/>
</xs:sequence>
</xs:complexType>
<xs:element name="getNameById" type="tns:getNameById"/>
<xs:complexType name="getNameById">
<xs:sequence>
```

```

<xs:element minOccurs="0" name="id" type="xs:string"/>
</xs:sequence>
</xs:complexType>
<xs:element name="getNameByIdResponse" type="tns:getNameByIdResponse"/>
<xs:complexType name="getNameByIdResponse">
<xs:sequence>
<xs:element minOccurs="0" name="name" type="xs:string"/>
</xs:sequence>
</xs:complexType>
<xs:element name="getPersonById" type="tns:getPersonById"/>
<xs:complexType name="getPersonById">
<xs:sequence>
<xs:element minOccurs="0" name="id" type="xs:string"/>
</xs:sequence>
</xs:complexType>
<xs:element name="getPersonByIdResponse"
type="tns:getPersonByIdResponse"/>
<xs:complexType name="getPersonByIdResponse">
<xs:sequence>
<xs:element minOccurs="0" name="person" type="tns:person"/>
</xs:sequence>
</xs:complexType>
<xs:element name="getPeopleByName" type="tns:getPeopleByName"/>
<xs:complexType name="getPeopleByName">
<xs:sequence>
<xs:element minOccurs="0" name="name" type="xs:string"/>
</xs:sequence>
</xs:complexType>
<xs:element name="getPeopleByNameResponse"
type="tns:getPeopleByNameResponse"/>
<xs:complexType name="getPeopleByNameResponse">
<xs:sequence>
<xs:element maxOccurs="unbounded" minOccurs="0" name="person"
type="tns:person"/>
</xs:sequence>
</xs:complexType>
</xs:schema>
</wsdl:types>
<wsdl:message name="getPeopleByName">
<wsdl:part element="ns1:getPeopleByName" name="parameters">
</wsdl:part>
</wsdl:message>
<wsdl:message name="getDirectoryListResponse">
<wsdl:part element="ns1:getDirectoryListResponse" name="result">
</wsdl:part>
</wsdl:message>
<wsdl:message name="getPersonByIdResponse">
<wsdl:part element="ns1:getPersonByIdResponse" name="result">
</wsdl:part>
</wsdl:message>
<wsdl:message name="getPersonById">
<wsdl:part element="ns1:getPersonById" name="parameters">
</wsdl:part>
</wsdl:message>
<wsdl:message name="getDirectoryList">
<wsdl:part element="ns1:getDirectoryList" name="parameters">
</wsdl:part>

```

```

</wsdl:message>
<wsdl:message name="getDirectoryResponse">
  <wsdl:part element="ns1:getDirectoryResponse" name="result">
    </wsdl:part>
  </wsdl:message>
<wsdl:message name="getPeopleByNameResponse">
  <wsdl:part element="ns1:getPeopleByNameResponse" name="result">
    </wsdl:part>
  </wsdl:message>
<wsdl:message name="getNameByIdResponse">
  <wsdl:part element="ns1:getNameByIdResponse" name="result">
    </wsdl:part>
  </wsdl:message>
<wsdl:message name="getNameById">
  <wsdl:part element="ns1:getNameById" name="parameters">
    </wsdl:part>
  </wsdl:message>
<wsdl:message name="getDirectory">
  <wsdl:part element="ns1:getDirectory" name="parameters">
    </wsdl:part>
  </wsdl:message>
<wsdl:portType name="Directory">
  <wsdl:operation name="getDirectoryList">
    <wsdl:input message="ns1:getDirectoryList"
name="getDirectoryList">
    </wsdl:input>
    <wsdl:output message="ns1:getDirectoryListResponse"
name="getDirectoryListResponse">
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="getDirectory">
    <wsdl:input message="ns1:getDirectory" name="getDirectory">
    </wsdl:input>
    <wsdl:output message="ns1:getDirectoryResponse"
name="getDirectoryResponse">
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="getNameById">
    <wsdl:input message="ns1:getNameById" name="getNameById">
    </wsdl:input>
    <wsdl:output message="ns1:getNameByIdResponse"
name="getNameByIdResponse">
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="getPersonById">
    <wsdl:input message="ns1:getPersonById" name="getPersonById">
    </wsdl:input>
    <wsdl:output message="ns1:getPersonByIdResponse"
name="getPersonByIdResponse">
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="getPeopleByName">
    <wsdl:input message="ns1:getPeopleByName" name="getPeopleByName">
    </wsdl:input>
    <wsdl:output message="ns1:getPeopleByNameResponse"
name="getPeopleByNameResponse">
    </wsdl:output>
  </wsdl:operation>

```

```

    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="DirectoryServiceSoapBinding"
type="ns1:Directory">
    <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="getDirectoryList">
      <soap:operation soapAction="" style="document"/>
      <wsdl:input name="getDirectoryList">
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="getDirectoryListResponse">
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="getDirectory">
      <soap:operation soapAction="" style="document"/>
      <wsdl:input name="getDirectory">
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="getDirectoryResponse">
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="getNameById">
      <soap:operation soapAction="" style="document"/>
      <wsdl:input name="getNameById">
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="getNameByIdResponse">
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="getPeopleByName">
      <soap:operation soapAction="" style="document"/>
      <wsdl:input name="getPeopleByName">
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="getPeopleByNameResponse">
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="getPersonById">
      <soap:operation soapAction="" style="document"/>
      <wsdl:input name="getPersonById">
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="getPersonByIdResponse">
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="DirectoryService">
    <wsdl:port binding="ns1:DirectoryServiceSoapBinding"
name="DirectoryPort">
      <soap:address
location="http://localhost:8080/CXF/services/DirectoryService"/>

```

```
</wsdl:port>  
</wsdl:service>  
</wsdl:definitions>
```

CXF Data Type Handling

Data Types

```
<xs:complexType name="phoneNumber">
  <xs:sequence>
    <xs:element minOccurs="0" name="areaCode" type="xs:int"/>
    <xs:element minOccurs="0" name="firstThree" type="xs:int"/>
    <xs:element minOccurs="0" name="lastFour" type="xs:int"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="person">
  <xs:sequence>
    <xs:element maxOccurs="unbounded" minOccurs="0"
name="emailAddresses" nillable="true" type="xs:string"/>
    <xs:element minOccurs="0" name="firstName"
type="xs:string"/>
    <xs:element minOccurs="0" name="id" type="xs:string"/>
    <xs:element minOccurs="0" name="lastName"
type="xs:string"/>
    <xs:element minOccurs="0" name="phone"
type="tns:phoneNumber"/>
  </xs:sequence>
</xs:complexType>
```

SOAP Operations

HashSet<Person> getDirectory

WSDL

Request:

```
<xs:element name="getDirectory" type="tns:getDirectory"/>

<xs:complexType name="getDirectory">
  <xs:sequence/>
</xs:complexType>
```

Response:

```
<xs:element name="getDirectoryResponse"
type="tns:getDirectoryResponse"/>

<xs:complexType name="getDirectoryResponse">
  <xs:sequence>
    <xs:element maxOccurs="unbounded" minOccurs="0"
name="person" type="tns:person"/>
  </xs:sequence>
</xs:complexType>
```

Actual

Request:

```
<dir:getDirectory/>
```

Response:

```
<ns1:getDirectoryResponse xmlns:ns1="http://directoryservice/">
  <person>
    <emailAddresses>gcs@wpi.edu</emailAddresses>
    <emailAddresses>gary.schorer@ll.mit.edu</emailAddresses>
    <firstName>Gary</firstName>
    <id>2</id>
    <lastName>Schorer</lastName>
    <phone>
      <areaCode>781</areaCode>
      <firstThree>981</firstThree>
      <lastFour>5356</lastFour>
    </phone>
  </person>
  <person>
    <emailAddresses>gcs@wpi.edu</emailAddresses>
    <emailAddresses>gary.schorer@ll.mit.edu</emailAddresses>
    <firstName>Gary</firstName>
    <id>1</id>
    <lastName>Schorer</lastName>
    <phone>
      <areaCode>781</areaCode>
      <firstThree>956</firstThree>
      <lastFour>2600</lastFour>
    </phone>
  </person>
</ns1:getDirectoryResponse>
```

ArrayList<Person> getDirectoryList

WSDL

Request:

```
<xs:element name="getDirectoryList" type="tns:getDirectoryList"/>

<xs:complexType name="getDirectoryList">
  <xs:sequence/>
</xs:complexType>
```

Response:

```
<xs:element name="getDirectoryListResponse"
type="tns:getDirectoryListResponse"/>

<xs:complexType name="getDirectoryListResponse">
  <xs:sequence>
    <xs:element maxOccurs="unbounded" minOccurs="0"
name="person" type="tns:person"/>
  </xs:sequence>
</xs:complexType>
```

Actual

Request:

```
<dir:getDirectoryList/>
```

Response:

```
<ns1:getDirectoryListResponse
xmlns:ns1="http://directoryservice/">
  <person>
    <emailAddresses>gcs@wpi.edu</emailAddresses>

<emailAddresses>gary.schorer@ll.mit.edu</emailAddresses>
    <firstName>Gary</firstName>
    <id>2</id>
    <lastName>Schorer</lastName>
    <phone>
      <areaCode>781</areaCode>
      <firstThree>981</firstThree>
      <lastFour>5356</lastFour>
    </phone>
  </person>
  <person>
    <emailAddresses>gcs@wpi.edu</emailAddresses>

<emailAddresses>gary.schorer@ll.mit.edu</emailAddresses>
    <firstName>Gary</firstName>
    <id>1</id>
    <lastName>Schorer</lastName>
    <phone>
      <areaCode>781</areaCode>
      <firstThree>956</firstThree>
      <lastFour>2600</lastFour>
    </phone>
  </person>
</ns1:getDirectoryListResponse>
```

Person[] getPeopleByName(String)

WSDL

Request:

```
<xs:element name="getPeopleByName" type="tns:getPeopleByName"/>

<xs:complexType name="getPeopleByName">
  <xs:sequence>
    <xs:element minOccurs="0" name="name"
type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Response:

```
<xs:element name="getPeopleByNameResponse"
type="tns:getPeopleByNameResponse"/>

<xs:complexType name="getPeopleByNameResponse">
  <xs:sequence>
    <xs:element maxOccurs="unbounded" minOccurs="0"
name="person" type="tns:person"/>
  </xs:sequence>
</xs:complexType>
```

Actual

Request:

```
<dir:getPeopleByName>
  <name>a</name>
</dir:getPeopleByName>
```

Response:

```
<ns1:getPeopleByNameResponse
xmlns:ns1="http://directoryservice/">
  <person>
    <emailAddresses>gcs@wpi.edu</emailAddresses>
    <emailAddresses>gary.schorer@ll.mit.edu</emailAddresses>
    <firstName>Gary</firstName>
    <id>2</id>
    <lastName>Schorer</lastName>
    <phone>
      <areaCode>781</areaCode>
      <firstThree>981</firstThree>
      <lastFour>5356</lastFour>
    </phone>
  </person>
  <person>
    <emailAddresses>gcs@wpi.edu</emailAddresses>
    <emailAddresses>gary.schorer@ll.mit.edu</emailAddresses>
    <firstName>Gary</firstName>
    <id>1</id>
    <lastName>Schorer</lastName>
```

```
<phone>
  <areaCode>781</areaCode>
  <firstThree>956</firstThree>
  <lastFour>2600</lastFour>
</phone>
</person>
</ns1:getPeopleByNameResponse>
```

Person getPersonById(String)

WSDL

Request:

```
<xs:element name="getPersonById" type="tns:getPersonById"/>

<xs:complexType name="getPersonById">
  <xs:sequence>
    <xs:element minOccurs="0" name="id"
type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Response:

```
<xs:element name="getPersonByIdResponse"
type="tns:getPersonByIdResponse"/>

<xs:complexType name="getPersonByIdResponse">
  <xs:sequence>
    <xs:element minOccurs="0" name="person"
type="tns:person"/>
  </xs:sequence>
</xs:complexType>
```

Actual

Request:

```
<dir:getPersonById>
  <id>1</id>
</dir:getPersonById>
```

Response:

```
<ns1:getPersonByIdResponse xmlns:ns1="http://directoryservice/">
  <person>
    <emailAddresses>gcs@wpi.edu</emailAddresses>
    <emailAddresses>gary.schorer@ll.mit.edu</emailAddresses>
    <firstName>Gary</firstName>
    <id>1</id>
    <lastName>Schorer</lastName>
    <phone>
      <areaCode>781</areaCode>
      <firstThree>956</firstThree>
      <lastFour>2600</lastFour>
    </phone>
  </person>
</ns1:getPersonByIdResponse>
```

String getNameById(String)

WSDL

Request:

```
<xs:element name="getNameById" type="tns:getNameById"/>

<xs:complexType name="getNameById">
  <xs:sequence>
    <xs:element minOccurs="0" name="id"
type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Response:

```
<xs:element name="getNameByIdResponse"
type="tns:getNameByIdResponse"/>

<xs:complexType name="getNameByIdResponse">
  <xs:sequence>
    <xs:element minOccurs="0" name="name"
type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Actual

Request:

```
<dir:getNameById>
  <id>1</id>
</dir:getNameById>
```

Response:

```
<ns1:getNameByIdResponse xmlns:ns1="http://directoryservice/">
  <name>Gary Schorer</name>
</ns1:getNameByIdResponse>
```

void addPerson(Person)

WSDL

Request:

```
<xs:element name="addPerson" type="tns:addPerson"/>

<xs:complexType name="addPerson">
  <xs:sequence>
    <xs:element minOccurs="0" name="toAdd"
type="tns:person"/>
  </xs:sequence>
</xs:complexType>
```

Response:

```
<xs:element name="addPersonResponse"
type="tns:addPersonResponse"/>

<xs:complexType name="addPersonResponse">
  <xs:sequence/>
</xs:complexType>
```

Actual

Request:

```
<dir:addPerson>
  <toAdd>
    <emailAddresses>gcs@wpi.edu</emailAddresses>
    <emailAddresses>gary.schorer@ll.mit.edu</emailAddresses>
    <firstName>Gary</firstName>
    <id>0</id>
    <lastName>Schorer</lastName>
    <phone>
      <areaCode>781</areaCode>
      <firstThree>981</firstThree>
      <lastFour>5356</lastFour>
    </phone>
  </toAdd>
</dir:addPerson>
```

Response:

```
<ns1:addPersonResponse xmlns:ns1="http://directoryservice/">
```

void addPeople(ArrayList<Person>)

WSDL

Request:

```
<xs:element name="addPeople" type="tns:addPeople"/>

<xs:complexType name="addPeople">
  <xs:sequence>
    <xs:element maxOccurs="unbounded" minOccurs="0"
name="toAdd" type="tns:person"/>
  </xs:sequence>
</xs:complexType>
```

Response:

```
<xs:element name="addPeopleResponse"
type="tns:addPeopleResponse"/>

<xs:complexType name="addPeopleResponse">
  <xs:sequence/>
</xs:complexType>
```

Actual

Request:

```
<dir:addPeople>
  <toAdd>
    <emailAddresses>gcs@wpi.edu</emailAddresses>
    <emailAddresses>gary.schorer@ll.mit.edu</emailAddresses>
    <firstName>Gary</firstName>
    <id>0</id>
    <lastName>Schorer</lastName>
    <phone>
      <areaCode>781</areaCode>
      <firstThree>981</firstThree>
      <lastFour>5356</lastFour>
    </phone>
  </toAdd>
  <toAdd>
    <emailAddresses>gcs@wpi.edu</emailAddresses>
    <emailAddresses>gary.schorer@ll.mit.edu</emailAddresses>
    <firstName>Gary</firstName>
    <id>0</id>
    <lastName>Schorer</lastName>
    <phone>
      <areaCode>781</areaCode>
      <firstThree>981</firstThree>
      <lastFour>5356</lastFour>
    </phone>
  </toAdd>
</dir:addPeople>
```

Response:

```
<soap:Fault>
  <faultcode>soap:Server</faultcode>
  <faultstring>Unmarshalling Error : Current state not
START_ELEMENT or END_ELEMENT</faultstring>
</soap:Fault>
```

void addPeopleArray(Person[])

WSDL

Request:

```
<xs:element name="addPeopleArray" type="tns:addPeopleArray"/>

<xs:complexType name="addPeopleArray">
  <xs:sequence>
    <xs:element maxOccurs="unbounded" minOccurs="0"
name="toAdd" type="tns:person"/>
  </xs:sequence>
</xs:complexType>
```

Response:

```
<xs:element name="addPeopleArrayResponse"
type="tns:addPeopleArrayResponse"/>

<xs:complexType name="addPeopleArrayResponse">
  <xs:sequence/>
</xs:complexType>
```

Actual

Request:

```
<dir:addPeopleArray>
  <toAdd>
    <emailAddresses>gcs@wpi.edu</emailAddresses>
    <emailAddresses>gary.schorer@ll.mit.edu</emailAddresses>
    <firstName>Gary</firstName>
    <id>5</id>
    <lastName>Schorer</lastName>
    <phone>
      <areaCode>781</areaCode>
      <firstThree>981</firstThree>
      <lastFour>5356</lastFour>
    </phone>
  </toAdd>
  <toAdd>
    <emailAddresses>gcs@wpi.edu</emailAddresses>
    <emailAddresses>gary.schorer@ll.mit.edu</emailAddresses>
    <firstName>Gary</firstName>
    <id>0</id>
    <lastName>Schorer</lastName>
    <phone>
      <areaCode>781</areaCode>
      <firstThree>981</firstThree>
      <lastFour>5356</lastFour>
    </phone>
  </toAdd>
</dir:addPeopleArray>
```

Response:

```
<soap:Fault>
  <faultcode>soap:Server</faultcode>
  <faultstring>Unmarshalling Error : Current state not
START_ELEMENT or END_ELEMENT</faultstring>
</soap:Fault>
```