Accessing Web Services in iPhone/iOS Applications
Venkata Naga Sunil Kumar Mukkavilli
Department of Engineering
Arizona State University
vsmukkav@asu.edu

Abstract - Web application clients that use Web services, typically, require an intermediate virtual (proxy) class to access the web services. These proxy classes act as the mediator between the client and the web service thereby bridging the communication between them. In other words, it is a class functioning as an interface to the web service, hiding the implementation of the web service, and only exposing the web methods. This also allows the developer to completely concentrate on the client application logic, spending less or no effort on the intricacies of the communication with the Web service. Java and .NET have frameworks to generate a proxy class for a deployed web service, and use it to access the web service.

This project focuses on generating an Objective-C proxy class for a web service to be used in iPhone, iOS web application clients. The project includes a parser to generate the proxy, the proxy class, and an underlying proxy framework, which is used for the communication between the service and the client. The proxy class is generated by parsing the WSDL of the deployed web service and the framework is used to generate the request using the method parameters, send the request, and parse the response received from the web service for the result.

Index Terms – Web service, iOS, iphone, Web application client, proxy class.

I. INTRODUCTION

Apache Axis takes care of creating proxy classes to access web services in Java, and the wsdl.exe tool in .NET generates proxy classes for use in C#. Except for some third party applications, there is no standard tool to generate a proxy class in Objective-C (cocoa frameworks), and a base class framework to generate the request xml, and subsequently receive the response. Web applications running on iOS need to manually generate a web service request to invoke the web service, and to get the response.

This project provides a generalized solution to access any kind of synchronous web service (Web services built in Java, and deployed using Apache Axis 1.4), without having the burden of developing logic to communicate with the web service. First we will present a view of what Web services are, and how they are accessed.

A. Web Service[1]

Web Services are typical web based applications that provide some resources, and these resources can be accessed by any remote system/application over the networks. Its interface is described in a machine understandable format (Web Service Definition Language, or WSDL), and can be used irrespective of the programming language in which it is written, or the platform on which it is implemented. Web services fulfill a specific task or a set of tasks requested by a client.

A Web service is mainly described using a standard, XML format, called its service description that provides all of the details necessary to interact with the service, including message formats (that detail the operations), transport protocols used, and location of the Web Service. Its main advantage lies in its ability of not exposing the implementation details to the user. “This allows and encourages Web service based applications to be loosely coupled, component oriented, cross-technology implementations” [3]. Web services can be used alone or in combination with other Web services to carry out a complex module or a business transaction. In other words, it is a distributed
service which can be accessed over the internet and useful to several users or applications across multiple platforms and languages.

Figure 1: Accessing a Web Service[1]

B. Different Types of Web Services

Web Services are mainly distinguished based on the mode of communication (protocol) used by the service consumer and provider. Although there are different types of Web Service protocols, the three main important types are discussed below.

• SOAP

SOAP (Simple Object Access Protocol) is a method for exchanging XML based messages over the Network for providing and consuming web services. "It is a lightweight protocol for exchange of information in a decentralized, distributed environment" [4]. It is primarily based on XML, and consists of three main parts: an envelope that defines a framework which describes what is in a message and how to process it, a set of encoding rules for expressing instances of service-defined data types, and a convention for representing remote procedure calls and responses. It is widely criticized for its design complexity. Google, Amazon and ebay Web services widely use SOAP.

• REST

Representational State Transfer (REST) means "each unique url is a representation of some object". The contents of that object can be retrieved using an http get, we can use a post, put, or delete to modify the object (in practice most of the services use a post for this). XML is not the only format that can be used for data interchange in REST. The format of information (representation) returned can be either in xml, json, plain text or even in html format.

REST architecture is mainly focused on two things: resources and interface. Resources are defined by the application’s state and functionality that is represented by a unique url. These resources share a uniform interface to transfer the state of the service between the client and server. It is used in all of yahoo’s web services including Flickr, some of ebay and Amazon web services.

• XML-RPC

It is defined by a set of rules and implementations that allow applications running on different platforms, and different operating systems to make Remote Procedure Calls (RPC) over the network.

It uses HTTP as the transport protocol and XML as the language for encoding. XML-RPC is designed to be as simple as possible, while at the same time allowing complex data structures to be transmitted, processed and returned. It uses XML for encoding and decoding the remote procedure calls along with its parameters. It was replaced by SOAP that has the advantage of being a simpler, settled standard. It is mainly used in Blogger from Google.

In this project, we are mainly focusing on accessing SOAP based web services, and would be discussing the protocol in detail, and so, the remaining two are beyond the scope of this project.

C. Proxy Class

A proxy class is created as a protective layer for the client to get away from the complexities involved in invoking a Web service. A proxy class is a class containing all of the methods and objects exposed by the deployed Web service. These methods handle the marshalling of the parameters into SOAP, sending the SOAP request over HTTP, receiving the response from the Web service,
and unmarshalling the return value. A proxy class may be generated from a service description as long as it conforms to the Web Services Description Language (WSDL) standard. Because a proxy class communicates with the Web service across the Network, it is a good idea to verify that the url property of the proxy class references a trusted destination.

A proxy class is placed in the same virtual machine (normally same directory) as the client. The proxy object of this class communicates with the web service using the internet. Its interface provides the same method signatures as the web service, but hides the implementation and communication details from the client.

![Figure 2: Communication through a proxy class](image)

A Web service client may invoke proxy class methods, which communicate with a Web service over the network by processing the SOAP messages sent to and from the Web service. This helps in the client application developer to mainly focus on the application logic, leaving aside the burden of communicating with web service to the proxy class.

## II. PROBLEM STATEMENT

Currently, there is no built-in tool in Objective-C that automatically generates a proxy class which compiles and executes on iOS application programming interface, and there is no framework for this proxy to carry out all the communication with the remote web service. So, this results in the client programmer spending effort to develop logic for the communication with the web service, in addition to the application logic.

The project reported in this paper was to develop an economic as well as a generic solution to generate a wrapper class for a web service containing all the web methods, as well as a base class which takes care of generating and communicating the request to the web service, and receiving the response. So, my project provides a solution for all the client applications running on an iPhone/iOS that need to access a web service. They can use my tool to generate a proxy class to invoke the web methods, and use my base class to communicate with the web service.

### A. Scope

Currently, my project can be used to invoke web methods, with parameters and return types which includes all the basic data types such as int, float, double, boolean, string. It can be used to pass one-dimensional arrays as parameters, and receive arrays as return values. I have included all the primitive data types as well as empty and null values as parameters and return types for the web methods.

However, right now it cannot accommodate multi-dimensional arrays, user defined and complex data types, and class objects as parameters and return types. It can be extended to include all the different types in the future, as its flexible and robust design allows its extension and improvement in all the possible ways.

### B. Need for Proxy Class

The increased development of web services in different languages and technologies also increased their usage through the network. Since, a web service can only be accessed using a SOAP request xml, a proxy class can be used as wrapper class for the web service, and the proxy framework can be used to carry out the communication with the service. These two combined together can be used in the client application, thus saving the time and effort in developing the communication logic for the programmer. In iOS programming interface for
Objective-C, there is no standardized tool to generate this proxy class to invoke the web service. In light of the above mentioned reasons, there is significant motivation in generating Proxybase class and an underlying ProxyBase framework to access these web services in iPhone/iPad or iOS applications.

C. Proxy generator Tools in other languages

In Java, apache axis, which acts as the container for deploying the web services, is also used to generate a proxy class for the web service. The Apache Axis WSDL2Java tool generates Java proxy files that contain methods that can be invoked by a client application to send SOAP requests to a service. SOAP requests received by a service are decoded by the same Axis-generated libraries and turned back into the methods and arguments they represent. The command used is

```
java -cp $jwscp org.apache.axis.wsdl.WSDL2Java [Web Service WSDLurl]
```

where $jwscp is an environment variable set to the path to the Axis-libraries [5].

The Microsoft .NET Framework comes with a command-line utility called wsdl.exe that generates classes from WSDL definitions. Used in Visual Studio integrated development environment, wsdl.exe can generate a proxy class for consuming the service. It is very similar and has the same functionality as that of WSDL2Java. The command used is

```
wsdl /out:myProxyClass.cs [Web Service WSDLurl]
```

D. Basis for Proxy Generation-WSDL

The main base for determining the methods in a web service and creating a proxy class is the WSDL document.

WSDL is an XML-based language used to describe a web service programmatically. It contains five layers, each one built on top of the previous one: types, message, portType, binding and service. The document is marked by a root element: definitions. The WSDL specification defines three sets of extension elements for three protocols: SOAP, HTTP (GET/POST) and MIME. A WSDL definition is an XML document with a root definition element from the namespace: http://schemas.xmlsoap.org/wsdl/. The entire WSDL schema is publicly available [6].

The types, message and portType layers describe the web service with all necessary details and serve as the interface definition of the web service. But they do not bind the web service to any protocol. All the elements used in these three layers are defined within the WSDL namespace [6]. The binding and service layers then binds the interface definition to a particular protocol and provide the location and address of the web service, using elements defined for this protocol. These elements are called extension elements. By using extension elements, WSDL can describe a web service bound to virtually any protocol. The elements for these layers are described below.

- **types** element

  There is only one types element in a WSDL document. It contains an XML schema element, which contains the definition of all elements (and all the custom data types that are used to define these elements) that are going to appear in the web service method calls. In the example below, the types element defines array of integers.

  ```xml
  <wsdl:types>
  <schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://localhost:8080/axis/Calculator WS.jws">
  ```

![Figure 3: Generating a Proxy Class using WSDL definition of the Web Service](image-url)
<import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>

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

• message element

The message element defines an abstract message that can serve as the input or output of an operation. This element describes the data being exchanged between the service providers and consumers. Each Web Service has two messages: input and output. The input describes the input parameters for the Web Service and the output describes the return data from the Web Service. Each message contains zero or more part parameters, one for each parameter of the Web Service's method. Each part parameter associates with a concrete type defined in the types container element. Here is a simple message element, which contains a request and a response. The addRequest has two integer parameters, and its response is an integer as well.

<wsdl:message name="addRequest">
  <wsdl:part name="a1" type="xsd:int"/>
  <wsdl:part name="a2" type="xsd:int"/>
</wsdl:message>

<wsdl:message name="addResponse">
  <wsdl:part name="addReturn" type="xsd:int"/>
</wsdl:message>

• portType element

There is only one portType element in a WSDL document. It contains one or more operation elements, each representing one web method. Each operation can have 1–3 elements, each being input, output and fault, which reference the request, response and fault messages defined in the lower message layer. The portType element combines multiple message elements to form a complete request or request-response operation. For example, a portType can combine one request and one response message into a single request/response operation. This is most commonly used in SOAP services. A portType can define multiple operations. The portType element below gives details about two operations in the web service, nullParams and add, and their respective request and response types.

<wsdl:portType name="CalculatorWS">
  <wsdl:operation name="nullParams" parameterOrder="s s1">
    <wsdl:input message="impl:nullParamsRequest" name="nullParamsRequest"/>
    <wsdl:output message="impl:nullParamsResponse" name="nullParamsResponse"/>
  </wsdl:operation>
  <wsdl:operation name="add" parameterOrder="a1 a2">
    <wsdl:input message="impl:addRequest" name="addRequest"/>
    <wsdl:output message="impl:addResponse" name="addResponse"/>
  </wsdl:operation>
</wsdl:portType>

• binding element

The binding element binds the interface definition of the web service represented by the portType element to a particular protocol. It provides concrete information on what protocol is being used to transfer portType operations and information where the service is located. For SOAP protocol, the binding is <soap:binding>, and the transport is SOAP messages on top of HTTP protocol. The binding element below, defines the name of the binding along with the type which points to the port for this binding. It gives the protocol details of add operation, detailing about its request and response operations.

<wsdl:binding name="CalculatorWSSoapBinding" type="impl:CalculatorWS">
  <wsdl:operation name="add">
    <wsdl:input name="addRequest"/>
    <wsdl:output name="addResponse"/>
  </wsdl:operation>
</wsdl:binding>
• **service** element

The service element provides the location of the web service in the term of the used protocol. It contains one or more port elements. Each port element represents one location. If there is more than one port, it means that there are multiple locations of the same web service. For each of the supported protocols, there is one port element. Web service clients can learn from the service element where to access the Web service, through which port to access the service, and how the communication messages are defined. This element also includes an optional documentation element to provide human-readable documentation. Here, the service element gives details of the CalculatorWSWSService, along with its name, binding, and its location.

```xml
<wsdl:service name="CalculatorWSWSService">
  <wsdl:port binding="impl:CalculatorWSSoapBinding" name="CalculatorWS">
    <wsdlsoap:address location="http://localhost:8080/axis/CalculatorWS.jws"/>
  </wsdl:port>
</wsdl:service>
```

Figure 4: Class diagram for Proxy Generator
So, these five elements combined together provide the exact interface of the web service. All the methods that are available in the Web service along with their parameter and return types can be inferred by parsing the wsdl document. Using these logistics, the signature of all the web methods is re-created in the proxy class. The main difference between the actual web methods and the methods in the proxy class is that the proxy class methods only contain the signature, and are unaware of the exact functionality of the method.

The figure above (Figure 4) gives the class diagram of Proxy Generator on how exactly the wsdl is parsed in java to generate a Java proxy class as well as an Objective-C proxy class with the signature of the web methods. Part of it is used from Kranthi Devineni’s project [2], which deals with generating a java proxy class which is used to access web services from an android device. Though Apache Ant is a built-in tool to generate a java proxy class, it cannot be used to consume web services in android applications. So, Kranthi developed a proxy generator, as well as an underlying base framework for the proxy class to be used in android devices. Since the logic remains same for parsing the wsdl, I used Kranthi’s wsdl parser implemented in Java to generate an Objective-C class.
III. PROJECT IMPLEMENTATION AND DESIGN

A. The ProxyBase Interface in Objective-C

By manipulating the wsdl document of the Web Service, the proxy class can be generated, but for this proxy class to work, there must be an underlying base class. This base class invokes the web methods with the request, and receives the response. The main objective of the base class is to act as the medium between the proxy class and actual web service. While the proxy generator of Figure 4 can be run on any computer, the proxy class and the base classes are loaded onto an iOS device to the client application to provide it access to the web service.

The generated proxy class inherits the ProxyBase interface (the header file) to use all the methods designed in the base class and invoke the web service. Because the ProxyBase communicates with the XML Web service across the Internet, the ProxyGenerator class initializes the url property of the base class with the address of the web service in the generated Objective-C proxy class.

The ProxyBase interface in Objective-C is designed and developed as the base class for the proxy class, and as an underlying class for carrying out all the communication with the web service over the internet. The ProxyBase (shown in Figure 5) consists of mainly four methods, which form the core of the Project.

- **invoke** method
  
  This is the main method or the entry point to the base class. All the method stubs in the proxy class call this method with two parameters: an NSString object containing the name of the web method to be invoked, and an NSArray object containing the parameters to invoke the web method. Its return type is an NSObject, which contains the response output value of the invoked web method. This method calls the next three methods in the same order, before returning NSObject back to the called method in the proxy.

- **generateXML** method
  
  This method is used to generate the SOAP request xml or the SOAP request to be used in invoking the web service. It wraps the method name to be invoked along with the parameters to be passed to the web service. Given below is a sample SOAP request xml.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <emptyReturnFunction
      soapenv:encodingStyle="http://schemas.xmlsoap.org/soap
      /encoding/>
    <op2 xsi:type="xsd:string">sunil</op2>
  </emptyReturnFunction>
</soapenv:Body>
</soapenv:Envelope>
```

- **callrequest** method
  
  The proxybase class imports another class viz., RequestResponse class, which is used to establish a connection with the Web service at the specified url, invoke the connection with the request xml, and receive the response. This method instances the RequestResponse class with the generated SOAP xml.

- **parseResponse** method
  
  After the RequestResponse class receives the response from the web service, the response xml is parsed for the return value to the invoked method. This method focuses on parsing the response, and wrapping the return value into an NSObject. It returns this object to the called method. A DOM parser proves effective compared to a SAX parser for parsing the response xml, as it needs to look for particular tags in the xml. The NSXMLDocument class is the DOM parser provided by foundation frameworks, but it is not available in iOS SDK. So, I used an open source class in Objective-C called GDataXML[7], an NSXML styled DOM Parser provided by google. A sample SOAP Response xml is shown below.

```xml
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <emptyReturnFunction
      soapenv:encodingStyle="http://schemas.xmlsoap.org/soap
      /encoding/>
    <op2 xsi:type="xsd:string">sunil</op2>
  </emptyReturnFunction>
</soapenv:Body>
</soapenv:Envelope>
```
Apart from these methods, there are two more methods, used to convert primitive data types to NSObjects, and vice-versa. Since, boxing and unboxing are not automatic in Objective-C, wrapper classes are used to convert back and forth for passing objects between proxy class methods, and invoking the web service with the primitive values.

B. Sequence of Execution

The ProxyGenerator application contains classes to be used for parsing the wsdl of the web service, and generate the proxy class. Half of the classes used in the ProxyGenerator are already used in Kranthi’s project[2], and I appended some functionality into those classes to generate an Objective-C class as well as a Java class. So, in effect, I re-used the functionality[2] Kranthi already developed for parsing the wsdl document in Java and used it to generate an Objective-C class with all the web methods. The advantage of this approach is that a single proxy generator can be used to generate a proxy object for an Android application (the part of the project completed by Kranthi) or for an iOS application.

When the ProxyGenerator application is started, providing the wsdl url and (android or iphone) as its arguments, the main method in WSDL2Proxy class is invoked. This class invokes either the WSDL2Java or WSDL2ObjC class based on the argument provided. The next class that is invoked is WsdIParser that has a function parseWsdl which takes wsdl document url as argument, retrieves the WSDL at the specified URL and parses the whole document using DOM parser, building the data objects-ProxyObject, Function, Parameter and Type during this process. Function object stores function name, array of parameters and function return type. Parameter object stores parameter name and its data type which is a Type object. Lastly, a Type object has attributes name that stores parameter and return value’s data type name and a boolean isArray that specifies if the type of the data element is an array. After all data objects are stored, parseWsdl function returns a ProxyObject object. ProxyObject is a structure which has array of functions, ProxyName, targetNamespace as its attributes.

Finally, the generate function of WSDL2Java class takes ProxyObject as an argument and generates a Java Proxy class, and generate method in WSDL2ObjC class generates an Objective-C proxy class (both header and objective-C files). The proxy class contains all the methods available in the web service. The generated header and class files should be copied to the Client application package. In addition to this, the ProxyBase framework (ProxyBase and RequestResponse classes, and their header files) should also be included in the client application.

So, now when the client application wants to use a Web Service, it should first import the proxy class to invoke a web service method, it invokes the same method in the proxy class, and the underlying base class carries out the communication and gets the response output to the client.

IV. VALIDATION & TESTING

To test the functionality of my application, I used the generated proxy class, and the ProxyBase in an iphone application. Since the main motive is to successfully transport the primitive data values as parameter (request) and return (response), I used a web service that has different methods based on all the possible primitive data types and 1D-arrays. I successfully tested my application with the following cases.
A. Test Cases
• Int type parameters and int return types
• Float type parameters and float return types
• Double type parameters and double return types
• Array of double parameters and double return types
• No parameters and string return types
• String parameters and void return type
• No parameters and empty string return type
• No parameters and empty int Array return types
• Int parameters and Array of String return types
• No Parameters and null String return types
• Int parameters and Array of int return type
• Bool parameters and bool return types
• Mixed data type parameters and bool return type
• Null double parameters and double return types

V. CONCLUSION & FUTURE WORK
Since Web Services are widely used, this project provides a tool that can be used to allow smartphone applications to access web services, for usage in iPhone/iPad applications in particular and iOS applications in general.

However, it has some limitations, and can be extended in future in a number of ways. Since, it is designed and developed in Object-oriented languages (Java for generating proxy and Objective-C for the underlying base class), it can be easily extended, without altering the current classes and methods. It can be extended in the future for:
• Usage with multi-dimensional arrays, user defined data types, and class Objects as parameters and return types
• Asynchronous Web method calls, to carry out another request before the previous request is completed, as the present application first waits for response before sending another request.

• Enhance the WSDL parser class, so it works with wsdl generated by multiple web service execution environments (.NET, for example).

VI. PROJECT DELIVERABLES
• Proxy Generator jar file, which generates the proxy class in Java/Objective-C
• Proxy Framework which contains the base classes for the proxy class
• Sample iPhone application to test the generated proxy class using the base classes
• Ant build file to generate the Proxy Generator jar file
• Source Code for Proxy Generator

VII. ACKNOWLEDGEMENTS
I would like to thank my advisor Dr. Timothy Lindquist, without whom, this idea wouldn’t have been materialized into a project and implemented, and I wouldn’t have worked on a wonderful concept. I would also like to thank Dr. Bruce Millard, and Prof. Richard Whitehouse for their acceptance to be in my Project evaluation team. Finally, I would also thank Kranthi Devineni, my co-student, who allowed me to use her WSDL parser implementation to generate the Objective-C proxy class.

VIII. REFERENCES