STS Software Systems
Databases Languages Communication
Joachim W. Schmidt / Florian Matthes
Department of Information and Communication Technology
Technical University Hamburg-Harburg

Project Paper

Simple Object Access Protocol (SOAP)

by: Jack Koftikian

SuperVision from:
Prof. Dr. Joachim W. Schmidt

TUHH
Technische Universität Hamburg-Harburg

Table of Contents

Figures

1 - Abstract:

This paper focuses on the SOAP protocol which was developed concurrently by IBM and Microsoft to facilitate client / server communication using mainly the well-known HTTP protocol to invoke remote objects. SOAP Stands for: *"Simple Object Access Protocol"*. SOAP has gained popularity in this final year.

The task was to analyze and test SOAP for its functionalities and usage and try to develop a couple of simple demonstration programs to see SOAP live. We also tested the supplied sample applications, which proved to be useful in our research.

According to the research it is appropriate to give SOAP the following summarized definition: *"an XML / HTTP based protocol for accessing services, objects and servers in a platform independent manner".*

2 – Motivation:

The main motivation for this project is the fact that SOAP (Simple Object Access Protocol) is a new and promising protocol just released by IBM and Microsoft. At the start of this project there existed only little documentation on SOAP beyond the specification **[SOAP]**. A book dealing with SOAP was released from SAMS publishing. This book dealt mainly with Microsoft technology and it was not of that importance to the project, since we were concerned with Java and APACHE-SOAP for Java.

As it is announced, soon there will be available books for different topics on SOAP, especially dealing with the Java programming language and the web.

The main ***goal*** of this project is to further discover SOAP and to find the ways that we can exploit this technology and make use out of it. Also to test it and to see it functioning as promised and to find the possible practical applications that can stem from it.

We needed to explore the possibilities of SOAP on our own by examining sample implementations of the protocol [**APACHE**]. Also we wanted to test SOAP on a known system like Tamino[1] and to extend the capabilities of it by SOAP as:

- To create a Tamino Service that can be accessed with SOAP.
- Offer services that are not realized by Tamino (ex: update feature).
- Offer an interface to Tamino which clearly separated the invocation protocol from the communication protocol (which is not the case for the original Tamino release).

(1) Tamino: An XML based database created by Software-AG (www.tamino.com). It comes with a simple Internet browser interface to do the simple main tasks by using XQL (XML Query Language).

3 – Introduction:

In this paper, we will try to first introduce the SOAP (*Simple Object Access Protocol*) protocol, its origins, specifications and current versions, and then we will speak about the different ways that we can use SOAP to fulfill our demands on a heterogeneous network. We will also try to compare SOAP to other similar remote access protocols (like CORBA and COM).

We will show how the sample applications and our demos worked and functioned and what we expected from them and what was the outcome.

Finally we will be talking about the possible next steps that research should be done in, especially in the STS department.

This report is divided into sections. The descriptions of the main ones are: In section 4 we introduce SOAP generally and theoretically. We also compare it to other remote access techniques. In section 5 we go deeper into the SOAP protocol and try to find out how a SOAP client talks to a SOAP server and what is needed on both ends. Section 6 explains the practical part of SOAP. We demonstrate here how we tested SOAP and what kind of extra functionalities we could add to existing applications using SOAP. In section 7 we talk about our results and what can be done in the future to research more the SOAP protocol.

Note: Paragraphs in *italics* in this report indicate that they were copied exactly from the given reference or resource (citations).

4 -What is SOAP ?

SOAP stands for “Simple Object Access Protocol”, a relatively new protocol for distributed applications developed by Microsoft, IBM, DevelopMentor, and UserLand.

SOAP is highly “flexible“ and can support different applications; however, the most important one is to enable remote procedure calls (RPC) over HTTP using XML.

SOAP is an XML-based object invocation protocol and was originally developed for distributed applications to communicate over HTTP and through corporate firewalls and it was meant to access services, objects and servers in a platform-independent manner.
The original SOAP specification (1.0) outlines two major design goals

1- Provide a standard object invocation protocol built on Internet standards, using HTTP as the transport and XML for data encoding.
2- Create an extensible protocol and payload format that can evolve. **[SOAP]**

A later 1.1 specification states: “*A major design goal for SOAP is simplicity and extensibility*”. [MSFT]

4.1 – Idea Behind SOAP:

The world-wide-web is evolving from Web sites that simply serve up Web pages into dynamic Web services that interactively perform tasks involving multiple steps executed on a user's behalf. These tasks may require one Web service to call on other Web services, coordinating the steps much like a traditional software program executes commands. The problem today is that integrating with other services and touching different devices remains difficult, because tools (like CORBA or COM) and common conventions for their interconnection are relatively difficult to manage. What the Web needs is a common way to tie all these services together.

Historically, the solution for creating this kind of rich application-to-application communication has been to employ an object model such as Microsoft's DCOM or the Object Management Group's Internet Inter-ORB Protocol (IIOP) which is part of the Common Object Request Broker Architecture (CORBA). But these technologies have some limitations when it comes to creating Web services. In particular, DCOM and IIOP/CORBA are rich environments, which means that implementations and applications that use them tend to be complex and symmetrical. In other words, to build a distributed application using them, we typically need the same distributed object model running at both ends of the connection. But, the Internet doesn't guarantee what specific kind of client or server software is running at the other end of the connection; often the only common kind of communication channel available is just an HTTP-connection.

The problem with HTTP alone is that it is mainly a mechanism for passing files from server to client. To create more ambitious Web services, it is needed to extend HTTP. SOAP does exactly that: *it adds a set of HTTP headers and a rich XML payload to enable complex application-to-application communication over the Internet.* **[MSFT]**

The main idea behind SOAP was to:

- Improve Internet interoperability
- Integrate various business systems

DCOM and CORBA are considered as being too complex especially on the client side. SOAP will do better because it is simple, easy to use on top of existing communication protocols, based on XML and is implementation-independent.

SOAP is a very simple protocol that relegates most of the "real" work to an underlying service or component model. It is not a replacement for COM or

Enterprise JavaBeans or even CORBA components--it is simply a wrapper technology to make those services more accessible over the Internet. **[CNET]**

Since SOAP is based on XML, it's compatible with many programming models and allows businesses to easily exchange data with each other over the Internet. In practice, most companies build and run a mix of applications and distribution technologies built using COM, CORBA, Java RMI, and other technologies. A protocol that gives companies greater freedom to link systems both internally and across the Net with other companies is expected to be welcomed warmly. **[CNET]**

4.2 - Formal Specification:

(From IBM and Microsoft website) **[MSFT]**

SOAP doesn’t care about:

- Operating System
- Programming Language
- Object Model

SOAP extends HTTP with:

- Headers: to identify that it’s a SOAP message
- XML payload: that contains the actual data

This is to enable complex app-app communication over the net. The HTTP header describes the object to be activated.

- HTTP only supported in SOAP version 1.0.
- SMTP support in SOAP version 1.1.

4.2.1 Goals of SOAP:

- Provide a standard object invocation protocol built on Internet Standards using HTTP as the transport & XML for data encoding.
- Create an extensible protocol & payload format that can evolve over time.
- No distributed garbage collection, type safety or versioning.
- No bi-directional HTTP communication.

- No message box-caring or pipeline processing.
- No Object by reference.
- No Object activation.

(The last five goals or objectives are programmer specific that go unnoticed from most users. For more information check the reference [SOAP])

- SOAP works on existing Internet Infrastructure. (Routers, firewalls, proxy servers)
- A program needs only to know how to format a SOAP request.
- A programmer should be able to implement SOAP in a couple of days in any programming language as long as the client sends a valid SOAP request.
- Cost of implementing SOAP will be a small price to pay for universal interoperability.

4.2.2 SOAP Messages:

SOAP defines 2 kinds of messages: CALL & RESPONSE.

A typical scenario is shown in the following figure 1:

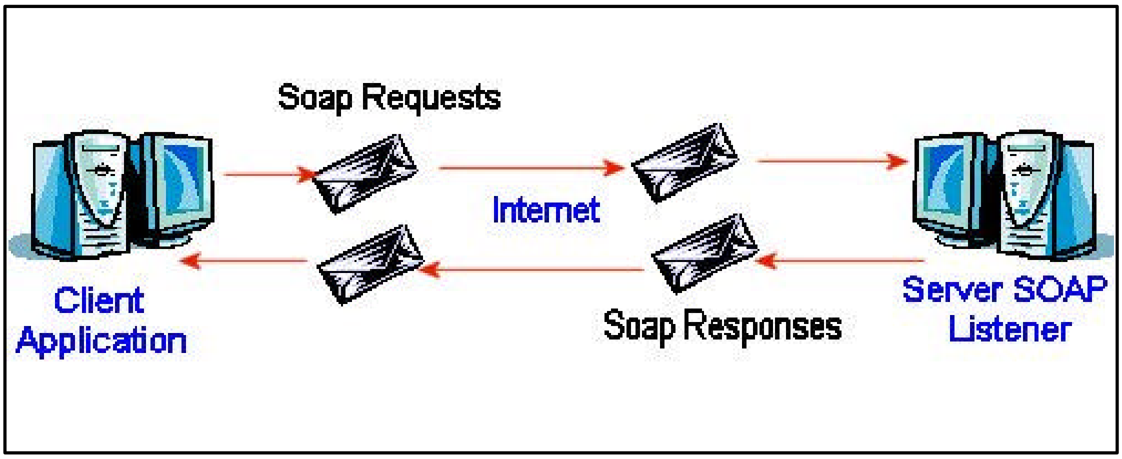


Figure 1: SOAP Messages

Because SOAP requires No Object Model → SOAP can be implemented in any language, as long as the client sends a valid SOAP request (that is, passing along the appropriate parameters to an actual remote endpoint on the server).

4.2.3 SOAP Messages with Attachments:

Summary: Binding a SOAP message to be carried within a MIME (multipart/related) message.

Idea: Send a SOAP message with attached binary data (images) using MIME.

Rules: -The primary SOAP message must be carried in the root body part of the Multipart/Related structure.

- MIME parts must contain either:
  - A content-ID
  - A content-Location

Objective: - Extending SOAP capabilities to be applied in new applications.

4.2.4 Two Worlds: IBM vs Microsoft:

The main creators and drivers of the SOAP specification are IBM and Microsoft, and as usual, each industry tries to focus on its tools to develop SOAP applications. The main tools that both companies created are:

- MS SOAP SDK and IBM's SOAP4J toolkit.

They are both compliant with the SOAP specification. When comparing both tools, users have found the following:

- On stability & compatibility IBM proved superior, but Microsoft’s tool is easier to use and to program with.

IBM donated its code to APACHE and hence it is public now and can be further improved more quickly as user support increases for it.

Some examples on the net for working interoperable systems include:

- Microsoft-based SOAP client talking to Apache-based SOAP server.
- Microsoft-based SOAP client talking to Perl-based SOAP server.
- Java clients talking to Microsoft-based SOAP servers.
- Java clients talking to Perl-based SOAP servers.

And they all worked, as claimed by the authors!! [DEVX]

In the project work we used the APACHE / IBM SOAP 4 JAVA libraries to test our Client and Server SOAP applications. We will talk about them more later in the report.

4.3 – Comparison Against Other Remote Access Techniques:

DCOM and CORBA provide rich application-to-application communication. This is usually complex and symmetrical (need same platform...etc.). SOAP promises to overcome these problems. The following (figure 2) shows the interaction between the server and a client for both DCOM and SOAP. The structures look similar, but SOAP gives us greater platform and location independence. This is due to the fact that SOAP uses HTTP as transport protocol and the data carried is in XML format, which is human-readable, whereas DCOM or CORBA use their own binary format to transport data which is much more complicated to debug or understand. This also means that they need special infrastructure: ***[DEVX]***

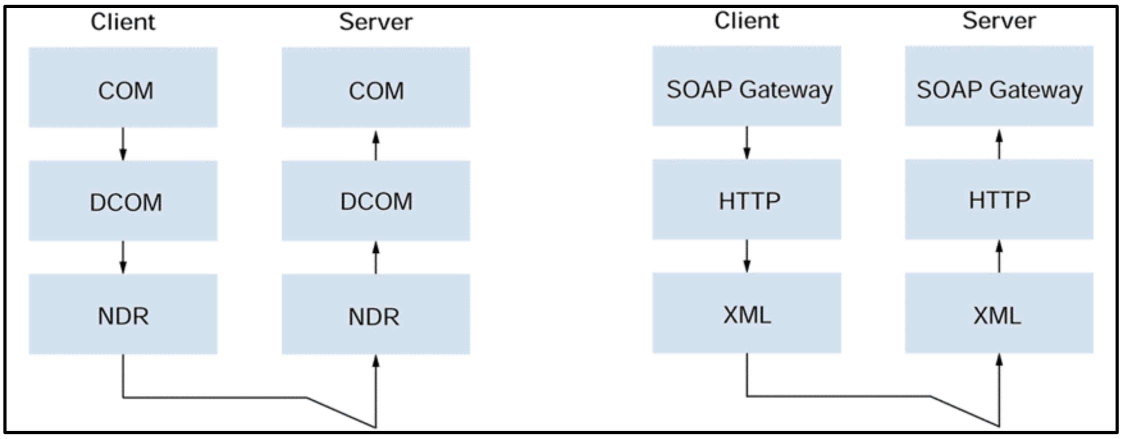


Figure 2: SOAP vs DCOM

In the above figure NDR stands for Network Data representation, which is a DCOM specific binary data exchange format.

The following table 1 shows the differences between SOAP and DCOM but also is a good source to understand more about SOAP and its advantages: ***[DEVX]***

Feature	SOAP	DCOM
Ease of use	HTTP and XML make for easy implementation and debugging. Text-based representation of information allows for easy deciphering of method calls and return results.	Creating COM objects for Windows using ATL, MFC, or VB is fairly easy using the Visual Studio tools. Enabling those objects to be called remotely through DCOM requires additional, difficult administration tasks.
Cross-platform support	Being based on Internet specifications allows SOAP to be supported on any computer platform. Interoperability between different SOAP-enabled computer systems is a definite reality.	Although DCOM has been ported to other platforms and has been submitted to the W3C for standards approval, it has been used only for Windows applications. Cross-platform communication using DCOM is not a common sight.
Binary Data	XML has not addressed embedding binary data within an XML document. Some proposed approaches: (1) encode binary data as text using something like binhex, (2) reference the binary data as an attachment, or (3) encode the data as an array of simple types. Option 1 would increase the size of transmitted data by more than 33 percent, option 2 would run into problems with cross-platform endian differences, and option 3 would be large to transmit.	DCOM allows for the transmission of binary data and handles the ordering of the bytes for endian correctness using standard marshaling. Custom marshaling adds an additional mechanism by which the DCOM object and its proxy can exchange data in a custom, proprietary format. Custom marshaling is advantageous when the object designer wishes to expose an interface that would be easy to use but would suffer performance penalties when remoted over a network.
Object Identity and lifetime	SOAP itself does not mandate any object identity other than a URL endpoint. This means the SOAP objects must be stateless or the SOAP server must maintain state for the client using cookies, special object identities within the SOAP calls, or identifiers in the URL string. Lifetime of SOAP objects on the server becomes an issue if the server is maintaining state. The server, with no means to determine if the client process has gone away, will need to timeout the object to reclaim its resources.	DCOM inherently employs object identity through the use of CLSIDs and server names using CoCreateInstanceEx. Object lifetime is also directly controlled by the client using AddRef and Release calls on the object, and indirectly by the server with a pinging and timeout mechanism.
Pointers to objects	SOAP has an identity mechanism to embedded subobjects of other objects. The downside of this approach: The entire contents of the subobject need to be passed instead of merely a pointer to the object. A SOAP method that wants to pass a pointer to an object would need to pass the URL of the object as a string. The receiving application would need to understand that the string actually represents a pointer to another SOAP object.	Interface methods for a DCOM object can return pointers to other interfaces. This pointer passing is fully supported by DCOM as long as the interface pointer being passed can also be remoted.
Protocol Transports	Although SOAP defines HTTP as the protocol for transmitting method calls, the SOAP specification hints that using other transports such as SMTP or MSMQ is not inconceivable. Using an asynchronous protocol such as SMTP or MSMQ would require the SOAP method calls to be unidirectional.	DCOM is inflexible in its transport and encoding rules.

Table 1

The main drawbacks of CORBA and DCOM are the configuration complexity and the relatively complicated security model that render them tough to get to work within a LAN, and nearly impossible to deploy over firewalls. This is what SOAP promises to solve and does it excellently. It mixes the power of remote procedure calls (RPCs) with the flexibility of XML, using HTTP as the underlying protocol that can be used from almost anywhere.

DCOM is highly efficient and flexible, but also highly complicated. One of the primary benefits of SOAP, on the other hand, is that its power does not come at the expense of simplicity.

The main first drawback of SOAP is the relative high footprint and parsing overhead involved in processing SOAP request compared to an equivalent DCOM or JAVA RMI method invocation. This is the result of using a hierarchical textual language as XML. This parsing overhead affects processor requirements and not call execution time, but the fact that SOAP was meant to be used on the internet brings the good news, since message transmission time over a WAN surpasses the amount of time it takes to parse the XML (sometimes this is a disadvantage also!).

5 - Typical Properties of a SOAP Application:

In the following we will be describing about how to implement a SOAP client/server system according to our project work, which was mainly using Apache SOAP for Java. So the clients and servers were written in JAVA.

5.1 - What happens on the Client side?

The main idea in SOAP for a client is to formulate the right SOAP message and to send this message in a CALL to the Server. Client needs to write an appropriate SOAP Envelope and put in it an optional Header and mandatory Body. The actual message is encoded in XML format. Figure 3 shows how a SOAP call is formatted:

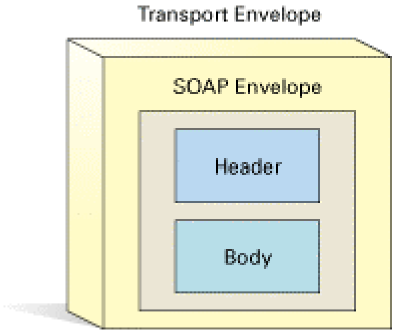


Figure 3: SOAP Envelope

The following is a sample HTTP CALL message with SOAP:

Example 1:

```
POST /StockPrice HTTP/1.1             // HTTP HEADER
Host: www.stockpriceserver.com        //This is the header part which
Content-Type: text/xml                //states the content type and
Content-Length: nnnn                  //length. Also the server URL

<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/1999/XMLSchema/instance/"
xmlns:xsd="http://www.w3.org/1999/XMLSchema/"> //Envelope & Headers
  <SOAP-ENV:Body>        //Start of SOAP Body
     <ns1:getAddressFromName xmlns:ns1="urn:AddressFetcher" SOAP-
     ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
     <nameToLookup xsi:type="xsd:string">John B. Good</nameToLookup>
     </ns1:getAddressFromName>
  </SOAP-ENV:Body>    //End of SOAP Body
</SOAP-ENV:Envelope>  //End of SOAP Envelope
```

The <symbol> element contains the actual parameter that is being passed.

And here is the response to it:

```
HTTP/1.1 200 OK               //
Content-Type: text/xml        // HEADER
Content-Length: nnnn          //

<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/1999/XMLSchema/instance/"
xmlns:xsd="http://www.w3.org/1999/XMLSchema/">
  <SOAP-ENV:Body>  //Start of SOAP Body
    <ns1:getAddressFromNameResponse xmlns:ns1="urn:AddressFetcher"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <return xmlns:ns2="urn:ibm-soap-address-demo"
xsi:type="ns2:address">    //Return Element
        <city xsi:type="xsd:string">Anytown</city>
        <state xsi:type="xsd:string">NY</state>
        <phoneNumber xsi:type="ns2:phone">
          <areaCode xsi:type="xsd:int">123</areaCode>
          <number xsi:type="xsd:string">7890</number>
          <exchange xsi:type="xsd:string">456</exchange>
        </phoneNumber>
        <streetName xsi:type="xsd:string">Main Street</streetName>
        <zip xsi:type="xsd:int">12345</zip>
        <streetNum xsi:type="xsd:int">123</streetNum>
      </return>
    </ns1:getAddressFromNameResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

An example using Apache Java for SOAP is the following:

1. Build the CALL:

Example 2:

```
        Call call = new Call();
        call.setTargetObjectURI("urn:test");
        call.setMethodName("calcTest");
        call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);
        URL url = new URL
("http://localhost:8080/soap/rpcrouter.jsp");

        //send the parameters
        Vector params = new Vector();
        params.addElement(new Parameter("value1", String.class,
first, null));         //first contains a string ("Hello")
        call.setParams (params);
```

So basically we create a new instance from the CALL class and then specify our target method name, the target object URI (which acts like an identifier, which we will talk about more later in the report). We also set the Encoding Style URI, here it is set to: “NS_URI_SOAP_ENC” which is the default for common simple types (here the parameter is: the Java Type STRING). We add all the parameters to a vector and invoke the Call. xxx

This Java interface hides away the complexities of writing the actual Call although one can still make mistakes that can only be found out during the actual call invocation. How the call is routed and reaches its destination is another question that we will deal in the “Data exchange between Client and Server” section later on. The Apache SOAP API provides lots of other useful classes for SOAP calls and requests which are discussed in the “Functionalities Provided by SOAP Libraries” section.

2. Send the Call and get the Response:

```
//invoke the call by specifying the url of the router
//server(rpcrouter in our case).
 Response resp = call.invoke( url, "");
 Parameter result = resp.getReturnValue ();

//print the response to standard output
 System.out.println (result.getValue ());
```

Of course this code is stripped of Try and Catch keywords and other checking statements for simplicity reasons here.

5.2 -What happens on the Server side?

When we send our Call it is “somehow” routed to the server, which is in our case, basically a Java object that is waiting for its method to be invoked. It doesn’t care if this invocation comes locally or remotely, it will function in the same way. For the above Example 2 where we made our call, an appropriate Test Server class would be the following:

```
class addTest
{
private String result;

      //method that returns the result

      public String calcTest(String a)
            {
            result = a + " SOAPworked";
            return(result);
            }
}
```

As we notice that the target method name is calcTest as we indicated in our call. Also the return is of type String, which is a simple SOAP type. Our server class looks like a normal Java class, no special keywords or methods are added to show the fact that this is a SOAP server! But still a main question remains which is:

“How the Client Call finds the Server which is located somewhere on the internet and who is routing this method calls?” This will be answered in the next section.

5.3 - Data exchanged between Client and Server:

The actual data that is exchanged between the client and server in SOAP is in XML format, but as we are using Apache SOAP for Java, this means that Java raps up the data in an object to be able to forward it between its methods and this object is of type *"Parameter"*.

For the Apache/IBM implementation, every time we create a SOAP service that will act like a server for any SOAP requests, we have to deploy it as a service using the supplied Deployment Manager (which is also considered an XML-SOAP administrative tool). The Deployment Manager takes care of all the deployed services and it is installed in the web server. This is done to assure the proper routing of calls since the Depolyment Manager makes sure that each identifier uniquely identifies a service with its different parameters (ex: Java Class, Method names, encoding types…etc.)

With the IBM-SOAP Administration Tools it is possible to use a Web browser to deploy and un-deploy services and to review the list and the definitions of the services deployed on a given SOAP server.

The options are:

- **Deploy** to deploy a new service.
- **Un-deploy** to remove a deployed service.
- **List** shows the list of services currently deployed in the server.

5.3.1 Service Deployment Information:

We review here the information that defines a deployed service. This information must be provided when using the Deploy function, and can be browsed using the List function. It is refered to this information as "properties" of the service.

- ***ID.*** *An URN uniquely identifies the service to clients. It must be unique among the deployed services, and be encoded as a URI. We commonly use the format: urn:UniqueServiceID . It corresponds to the target object ID, in the terminology of the SOAP specification.*
- ***Scope.*** *Defines the lifetime of the object serving the invocation request. This corresponds scope attribute of the <jsp:useBean> tag in the JavaServer Pages. It may thus have the following possible values:*

    - ***page:*** *the object is available until the target JSP page (in this case the rpcrouter.jsp) sends a response back or the request is forwarded to another page*
    - ***request:*** *the object is available for the complete duration of the request, regardless of forwarding.*
    - ***session:*** *the object is available for the complete duration of the current Java session.*
    - ***application:*** *any page within the application may access the object. In particular, successive service invocations belonging to different sessions will share the same instance of the object.*

    It is important to observe that the value of this attribute can have important security implications. The page and request scopes assure the isolation of successive calls. On the other extreme, application scope implies that all service objects are shared among different users of the SOAP server.

- ***Method list.*** *Defines the names of the methods that can be invoked on this service object.*
- ***Provider type.*** *Indicates whether the service is implemented using Java or a scripting language.*
- ***For Java services, Provider class.*** *Fully specified class name of the target object servicing the request.*
- ***For Java services, Use static class.*** *If set to "Yes" the class method that is made available is a static method, and thus no object will be instantiated. When static invocation is used, the "scope" property is not applicable.*
- ***For script services, Script language.*** *Indicates the scripting language used to implement the service.*
- ***For script services, Script filename.*** *Name of file containing the script, or*
- ***For script services, Script.*** *The actual script to run.*
- ***Type mappings.*** *In order to control the serialization and deserialization of specific Java types to and from XML in a particular encoding style, it may be necessary to provide serialization and deserialization classes that know how to perform the correct conversions for those types. The XML-SOAP server already includes serialization classes for most basic types in the SOAP encoding style, as well as a Bean encoding class that can provide a generic serialization of a bean in terms of its properties. It also includes XMI serializer/deserializer classes to support the XMI encoding style. Since different types may require additional support for correct serialization, the XML-SOAP maintains a registry of Serializers and Deserializers. The registry is accessible to service administrators through the XML-SOAP administration tool, as well as through a program API. In order to register a (de)serializer class, the class must implement the Serializer or Deserializer interfaces.* ***[SOAP]***

After the service is deployed all its information is in place, then it is logical to store this information so that it can be retrieved next time the Server is initiated. Here Apache uses a JSP page that calls the appropriate SOAP methods to do the job. This is the rpcrouter.jsp file found in the Webapps directory under SOAP. It calls the Pluggable Configuration Manager that is responsible for saving the current list of deployed services so that when the SOAP server is restarted the services will not need to be redeployed again.

Rpcrouter creates an instance of the SOAP class ServiceManager. Then adds all the deployed services to an Array that can be retrieved using the List() method or can be used for undeployment using the undeploy() method. So when a client invokes a SOAP Call, this call is handeled by the Rpcrouter and its ID is checked to find the appropriate service in the ServiceManager, and when the Target Method exists then the Call is made and result is returned to the calling client.

5.4 - Building blocks of a SOAP application:

The following figure would illustrate what is the complete cycle of a SOAP message request, also, which SOAP building blocks it passes through (Figure 4):

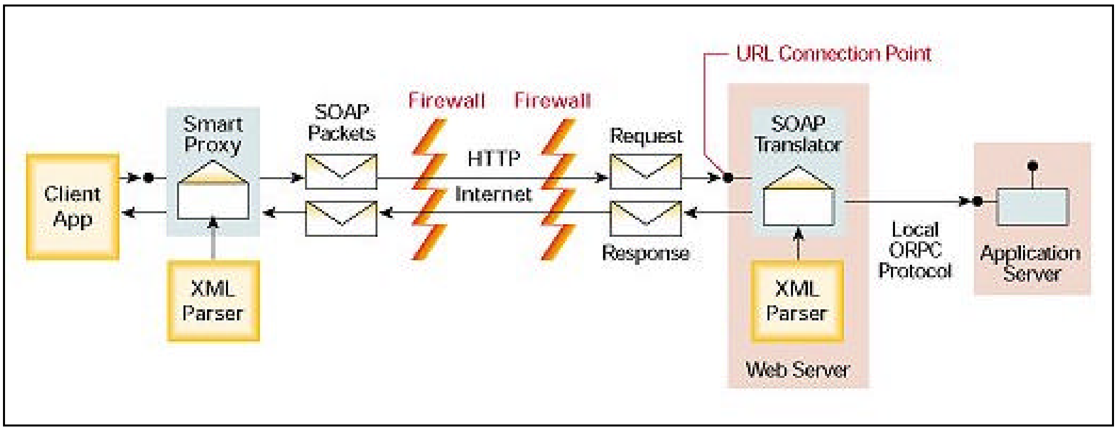


Figure 4: Building blocks of a SOAP applicaiton

In figure 4 some blocks are not necessary as the “FireWalls”. They are there just to demonstrate that SOAP can go through proxies and firewalls.

The “Smart Proxy” is actually the set of classes, which you use to create a request. In our sample application this is the CALL and PARAMETER classes as is described in 6.2.3.2.

The “URL connection point” is the RPC Router in our case, which we talked about previously.

Also there exists an XML parser on both ends of the communicating parties, which indicates that the SOAP envelope is in XML format and parsers are needed to translate it into the DOM tree.

As it is obvious from the figure that SOAP is using only HTTP and doesn’t need any other protocols to help the message go through.

5.5 - Functionalities provided by SOAP Libraries:

The actual functionalities that the SOAP Apache Libraries provide are numerous, we will just present some of them. In the following figure we show the dependencies of our Tamino Server and client on SOAP classes: [**SOAP DOC**]

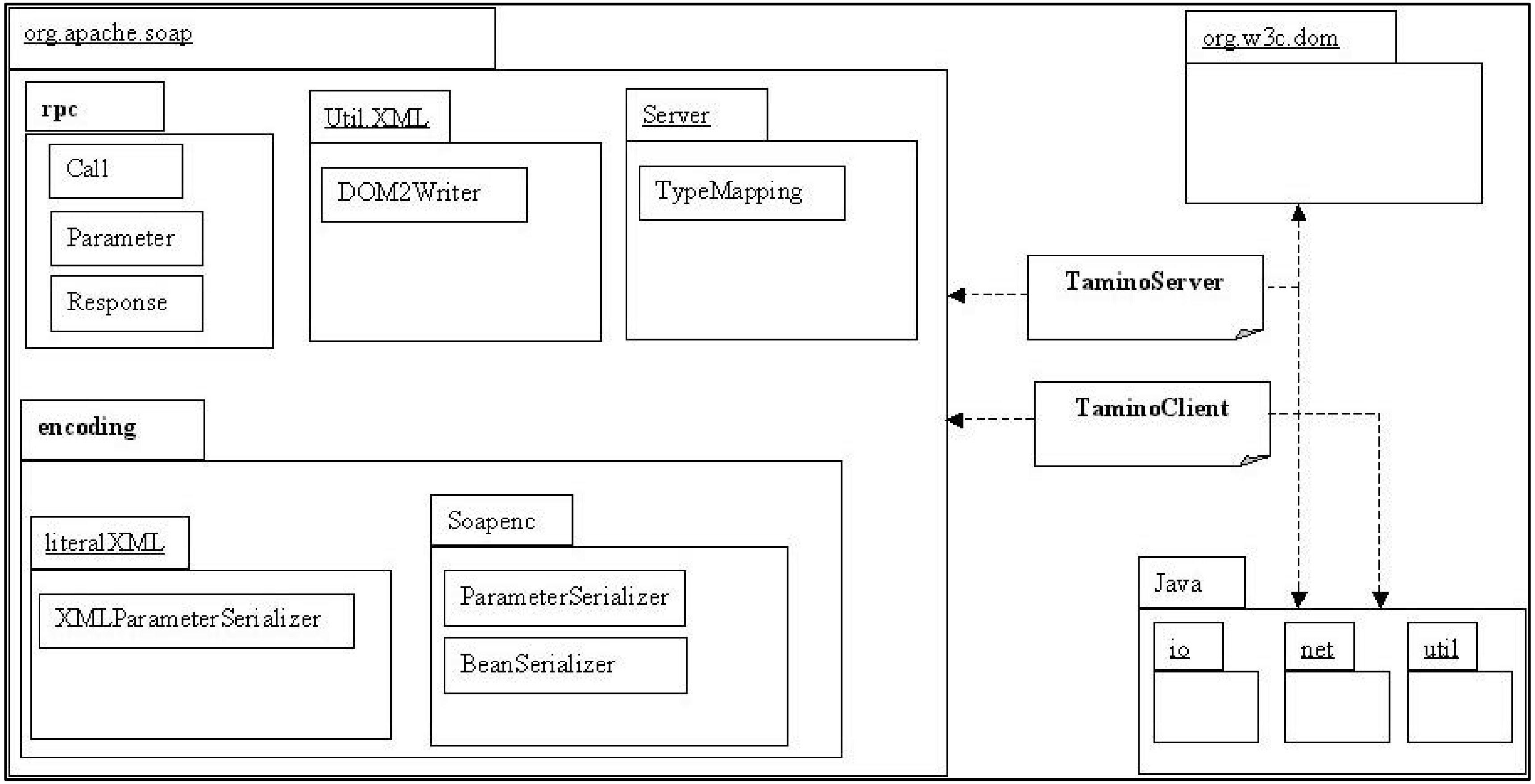


Figure 5: Package dependencies of Tamino Client and Server

CLASSES & FUNCTIONALITIES:

The most important ones, that one can use almost often, or the ones that introduce interesting functionality to the SOAP protocol are [**SOAP DOC**]:

Call: this class provides the main functionality for calling a remote method. It helps to construct the call, set the encoding style, set the SOAP mapping registry if need be. Finally it invokes the Call. A `Call` object represents an *RPC* call.

DeploymentDescriptor: This class represents the deployment information about a SOAP service.

DOM2Writer: This class is a utility to serialize a DOM node as XML-string.

HTTPUtils: Provides the HTTP post method.

Parameter: A `Parameter` represents an argument to an *RPC* call. `Parameter` objects are used by both the client and the server.

Response: A `Response` object represents an *RPC* response. Both the client and the server use `Response` objects to represent the result of a method invocation.

RPCMessage: An `RPCMessage` is the base class that `Call` and `Response` extend from. Any work that is common to both `Call` and `Response` is done here.

ServiceManager: This class takes care of deploying, listing and undeploying the SOAP services. Also it keeps the associating deployment descriptors.

SMTP2HTTPBridge: This class can be used as a bridge to relay SOAP messages received via email to an HTTP SOAP listener. This is basically a polling POP3 client that keeps looking for new messages to work on. When it gets one, it forwards it to a SOAP HTTP listener and forwards the response via SMTP to the original requestor (to either the ReplyTo: or From: address).

TCPTunnel: A `TcpTunnel` object listens on the given port, and once `Start` is pressed, will forward all bytes to the given host and port.

TypeConverter: A *TypeConverter* is used to convert an object of one type to one of another type. The converter is invoked with the class of the from object, the desired class, and the from object itself. The converter must return a new object of the desired class.

TypeMapping: This class keeps all the info about a type mapping: the encoding style, the XML element type, the Java type that's supposed to map to, and the names of the Java classes that implement the mapping between XML and Java.

XMLParserLiaison: An interface between a client and an XML-parser.

XMIParameterSerializer: A `ParameterSerializer` is used to serialize and deserialize parameters using the `XMI` encoding style.

XMLParameterSerializer: An `XMLParameterSerializer` is used to serialize and deserialize parameters using the `literal xml` encoding style. This class is only capable of serializing/deserializing parameters of type `org.w3c.dom.Element`

5.6 - SOAP Security:

SOAP is a new protocol that for cross-platform communication that can bypass firewall defenses and could leave companies open to what experts describe as a fresh class of security vulnerabilities.

Currently, developers struggle to make their distributed applications work across the Internet when firewalls get in the way. Since most firewalls block all but a few ports, such as the standard HTTP port 80, all of today's distributed object protocols like DCOM & CORBA suffer because they rely on dynamically assigned ports for remote method invocations.

SOAP uses HTTP so it can bypass all these firewalls and this is what causes the worry that SOAP may pose threat to companies, but SOAP traffic could be filtered even though firewalls are not Soap-aware, since SOAP messages have a unique HTTP Header field that can be used for this purpose.

The argument against this is that since SOAP doesn’t enforce any kind of security of its own, people will overlook this problem while programming and lots of security holes will be left uncovered. ***[VENU]***

A reply to all these claims came from Microsoft and IBM in the beginning of February 2001, when they submitted an extension to SOAP called: **SOAP Security Extensions: Digital Signature**. They proposed a standard way to use the XML Digital Signature syntax [XML-Signature] to sign SOAP 1.1 messages. They defined a SOAP header entry <SOAP-SEC:Signature> for this purpose. XML Encryption and Authentication are necessary also.

This specification defines the use of XML Signature in SOAP 1.1 headers. As one of building blocks for securing SOAP messages, it is intended to be used in conjunction with other security techniques. Digital signatures need to be understood in the context of other security mechanisms and the threats to an entity.

Digital signatures are, according to the IETF RFC 2828[DIGSIG]: “*A value computed with a cryptographic algorithm and appended to a data object* *in such a way that any recipient of the data can use the signature to verify the data's origin and integrity.*”

For example, digital signatures alone do not provide message authentication. One can record a signed message and resend it (replay attack). To prevent this type of attack, digital signatures must be combined with an appropriate means to ensure the uniqueness of the message, such as time stamps. One way to add this information is to place an extra <ds:Object> element that is a child of the <ds:Signature>.***[SOAP-Security]***

6 - Practical Demos:

During the project work we tried to find and test how SOAP will work with our sample test programs and needs. To start with the installation of APACHE-SOAP first you need a Web application server that supports **servlets** and **JSPs**. Some of the common choices for a web application server includes:

- Apache Tomcat v3.1
- BEA - WebLogic Application Server v5.1
- IBM WebSphere v3.5
- JRun 3.0
- Microsoft Internet Information Server

SOAP supports all of the above, but for convenience we used Apache Tomcat v3.1 (see technical details in the end of the report for more information).

6.1 - Samples provided by SOAP

The SOAP version that we tested had three different samples programs that show the capabilities of SOAP. It took lots of effort and time to make the first program run over our system, since this included the complete understanding of how the whole system works, and which parameters and server setups are needed, but after that, running the rest of the samples was relatively easy.

The main sample provided is called: "Address Book". This program consists of the server: Address Book, which provided different methods to access an address book.

Users can:

1- PutAddress( ): insert a new address into the address book. It takes as arguments: the complete address (`streetNum streetName city state zip areaCode`) with the name of a person. Then it instantiates an object of type Address and puts this info in it. It sends this Address instance with the SOAP Call. Another parameter that it sends is the name of the owner of this address (which is of type String).

2- PutListing( ): insert a complete XML document into the address book. It takes as an argument an XML file and it passes it to the XML parser liason which reads it and puts it as a parameter in the SOAP Call.

3- GetAddress( ): specify a query name and return the matching address. The name is sent as the SOAP Call parameter. It is of simple Type String.

4- GetAllListing( ): return the whole address book in XML format. No arguments sent. Just call the right method on the server.

The Address Book is maintained in XML format and saved in a file called "sample_listings.xml". All addresses should conform to a certain format, which is enforced by constructing these addresses from a class called Address. Also phone numbers in the address book have a separate class called “Phone Number”, that dictates how a phone number should look like.

Another sample is the “Get Quotes” sample, which can retrieve stock market quotes. By providing a ticker symbol, the SOAP call is constructed and sent to the SOAP server, which contacts a predefined server on the internet to retrieve the information back to the client.

An interesting variant of this Stock service is to send the request to an SMTP server, which will reply to the provided email address in the SOAP Call. This functionality of SOAP may open the opportunity into new applications that may take advantage of this. An example would be serving clients which are not always connected to a network and it is not critical to get a response instantly, so they can check the response whenever they want , from wherever they want !

A third sample is of the “Calculator”. The main idea here is to show how to make a server do all the calculations and return just a result to the client. In this case it is a simple task, but this can be extended to more laborious tasks that can be sent over the Internet to powerful servers to do the job for slow clients. Here the arguments are of Java type Double, they are 2 arguments that are sent with the Call, the server does the calculation (plus, minus, multiply or divide) and returns the result also as a Double value. The server-side object is a Jave Script file. The client is Java and the server Java script → this is an example of independence of programming language for SOAP services.

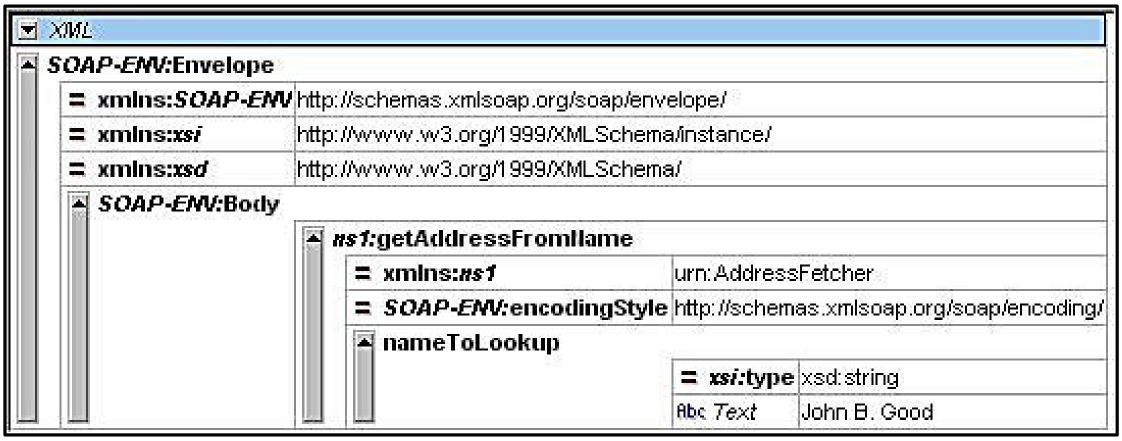
6.1.1 - **Actual SOAP requests and replies:**

To have a feeling about how the actual data looks like in a SOAP request and response and also to identify the building blocks of such activities we present here some sample requests and responses done on the sample data provided by SOAP1.1.

6.1.1.1. Address Book example:

REQUEST:

The following image shows the request sent to the Address Book server with a parameter of type String. The request is to “Look Up a Name”. Figure 6 shows the complete request in XML format.



```
XML
SOAP-ENV:Envelope
    = xmlns:SOAP-ENV   http://schemas.xmlsoap.org/soap/envelope/
    = xmlns:xsi        http://www.w3.org/1999/XMLSchema/instance/
    = xmlns:xsd        http://www.w3.org/1999/XMLSchema/
    SOAP-ENV:Body
                       ns1:getAddressFromName
                           = xmlns:ns1                 urn:AddressFetcher
                           = SOAP-ENV:encodingStyle    http://schemas.xmlsoap.org/soap/encoding/
                           nameToLookup
                                                       = xsi:type   xsd:string
                                                       Abc Text     John B. Good
```

Figure 6: Request to Address Book (*XMLSPY*)

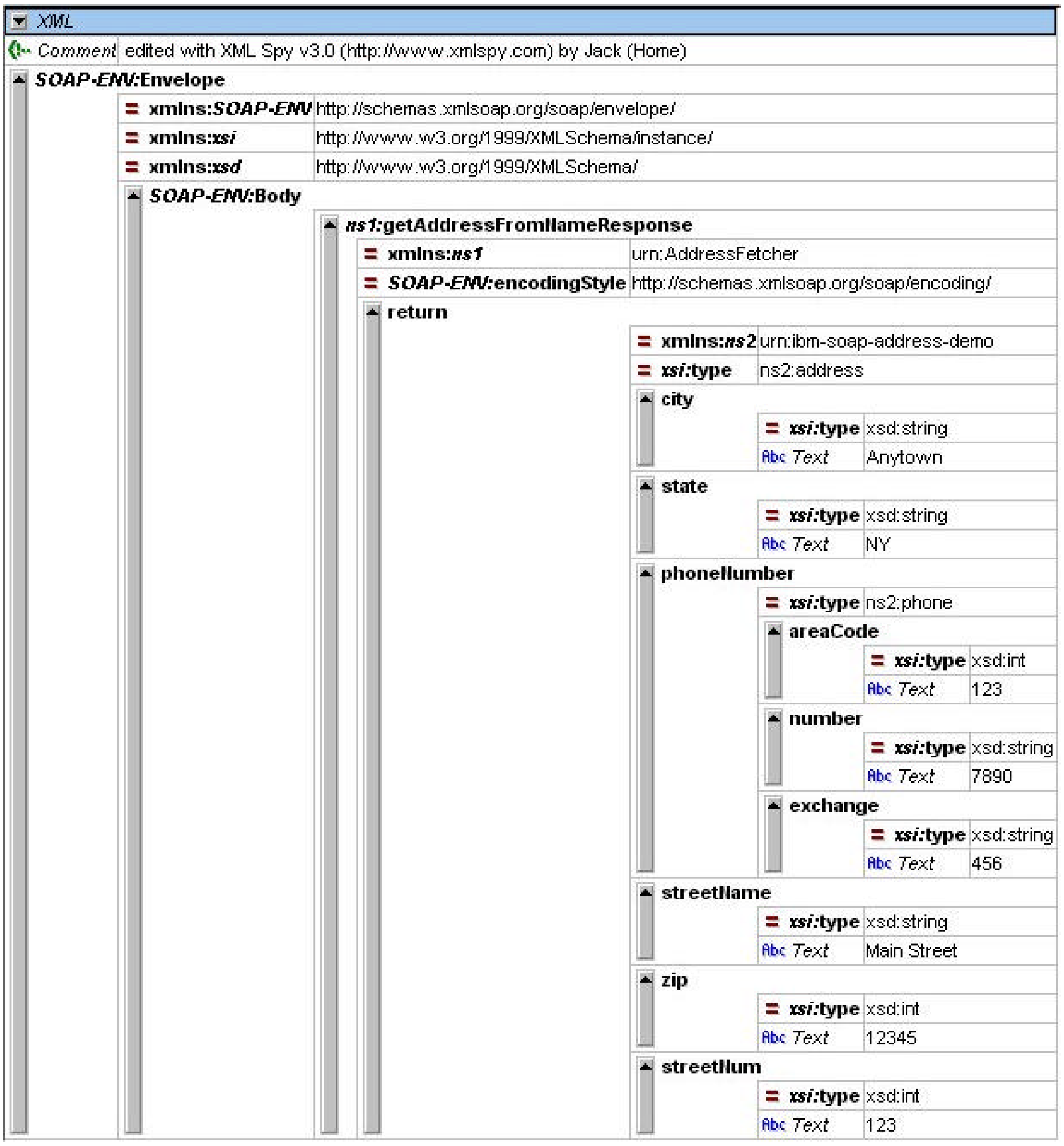
The request starts with a declaration that it is of type XML, and then followed by the SOAP-ENV element to indicate that what follows is an envelope of a SOAP request. This is followed by declaration of 3 namespaces: One for the envelope, one for the xsi type and one for the xsd (xsi and xsd are for XML schemas).

Then comes the second block which is the SOAP Body. It contains the name of the remote method (getAddressFromName). The namespace: urn:AddressFetcher is for the use of RPCRouter. It uses this namespace as a unique ID to know where to route this request and where the target server resides also it contains the info for type-mapping. The encodingStyle indicates how to encode the actual parameters.

Here the parameter is of type String and the name to lookup its address is “John B. Good”.

REPLY:

Figure 7 shows the reply to the above request from the address book:



```
XML
Comment  edited with XML Spy v3.0 (http://www.xmlspy.com) by Jack (Home)
SOAP-ENV:Envelope
    = xmlns:SOAP-ENV  http://schemas.xmlsoap.org/soap/envelope/
    = xmlns:xsi       http://www.w3.org/1999/XMLSchema/instance/
    = xmlns:xsd       http://www.w3.org/1999/XMLSchema/
    SOAP-ENV:Body
        ns1:getAddressFromNameResponse
            = xmlns:ns1                      urn:AddressFetcher
            = SOAP-ENV:encodingStyle         http://schemas.xmlsoap.org/soap/encoding/
            return
                = xmlns:ns2  urn:ibm-soap-address-demo
                = xsi:type   ns2:address
                city
                    = xsi:type  xsd:string
                    Abc Text    Anytown
                state
                    = xsi:type  xsd:string
                    Abc Text    NY
                phoneNumber
                    = xsi:type  ns2:phone
                    areaCode
                        = xsi:type  xsd:int
                        Abc Text    123
                    number
                        = xsi:type  xsd:string
                        Abc Text    7890
                    exchange
                        = xsi:type  xsd:string
                        Abc Text    456
                streetName
                    = xsi:type  xsd:string
                    Abc Text    Main Street
                zip
                    = xsi:type  xsd:int
                    Abc Text    12345
                streetNum
                    = xsi:type  xsd:int
                    Abc Text    123
```

Figure 7: Reply from Address Book SOAP server (*xmlspy*)

As shown in the figure, the SOAP reply is very much similar to the request. It also contains the same 3 namespace declarations. It is followed by the Body where it is obvious that this SOAP message is a response since now the ns1 field has a value of “getAddressFromNameResponse” and the SOAP body has a return element.

Then there is the return element, which contains the whole returned values; in this case it is a complete address composed from (City, State, PhoneNumber (areaCode, number, exchange), streetName, zip, StreetNum).

6.2- **Our Test Samples:**

First we tried to make a simple SOAP call, which was to send a simple Java type, namely a String and on the server side to append another String and return it back to the Client. The code is the Example 2 in section 5.2. The main functionalities and building blocks that we had to fix were:

1- Client class: where we had to construct the Call and put the parameters in it and invoke the right method on the Server side. xxxx

2- Server class: where the proper class & method had to be written so that they would return the expected result (in String format for our case).

3- Run Tomcat.

4- Deploy the service, using the provided server side tool from Apache. This will enlist the service in the Service-Manager along with its Deployment Description. By pointing the browser to http://localhost:8080/soap one can deploy services, list current ones, or undeploy old ones.

5- Run the Client program and wait for the response…

This trial worked successfully and we got the returned String in the correct format.

6.2.1 TaminoServer:

We tried to test SOAP with the Tamino database which is an XML based database developed by Software AG (www.tamino.com). We used SOAP to find out if we can query this XML based database, delete some entries, add entries, or implement new features into it, like updating the database.

Note: It should be noted that we use the term TaminoServer for our SOAP service. Technically there exists a default Tamino Server in the Tamino database that has nothing to do with SOAP and shouldn't be confused with our TaminoServer(SOAP service) mentioned in this paper.

The following figure 8 shows all the clients and server that we needed for the job to be done.

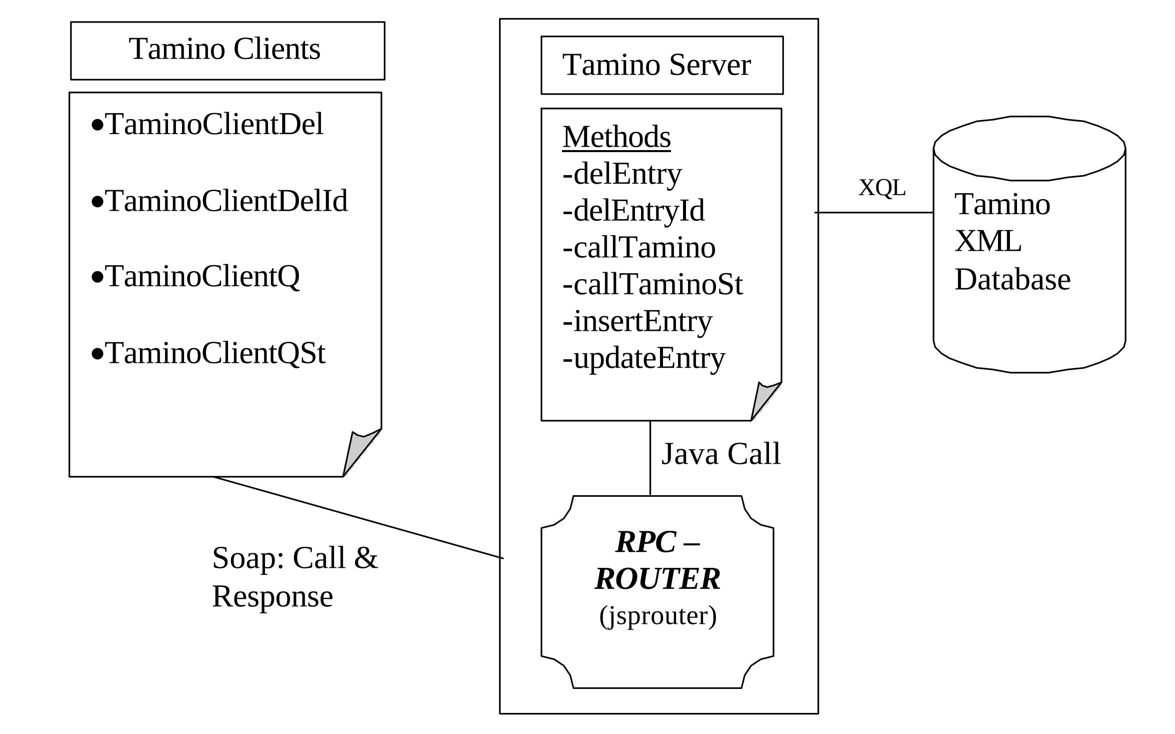


Figure 8: General Diagram of the system

As it is shown in the above figure, data exchanged between the main 2 parties is in XML. Also the TaminoServer uses native XQL to communicate to the Tamino Database.

The most important concept above is that, the 4 main entities can reside anywhere on the Internet and still they should be able to communicate to each other since they are using SOAP as a protocol.

6.2.1.1 - Deployment Diagram of our system:

In the STS department we deployed the system as showed in Figure 9:

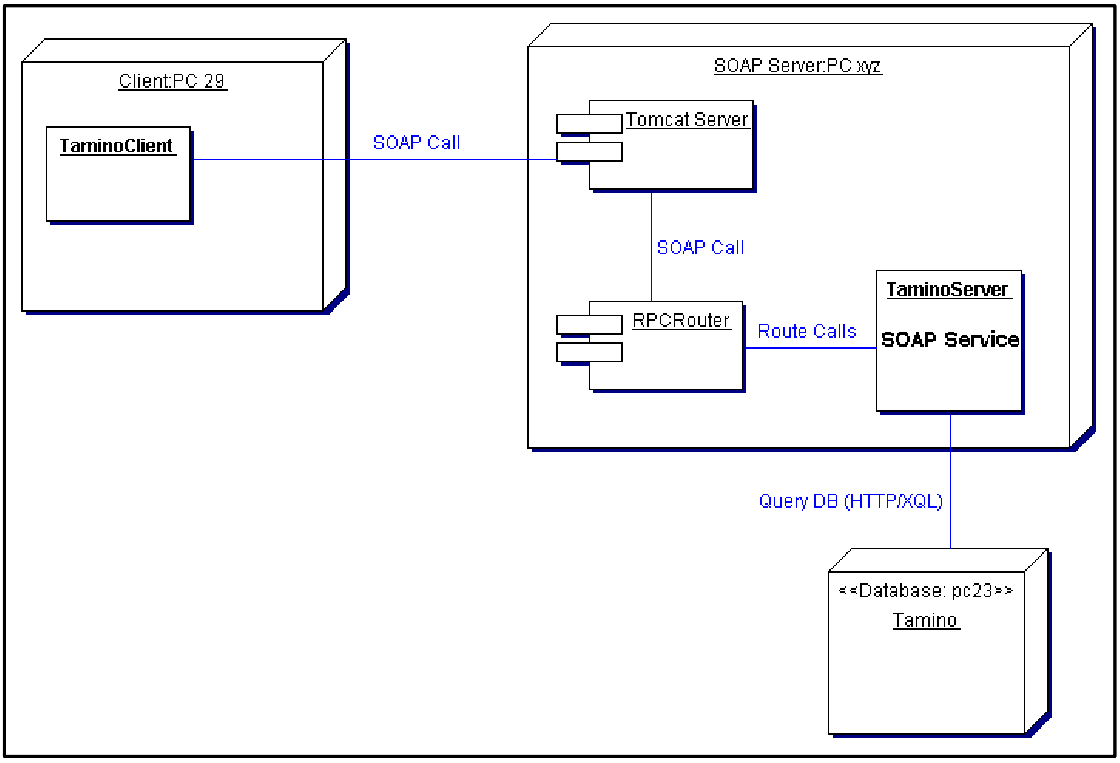
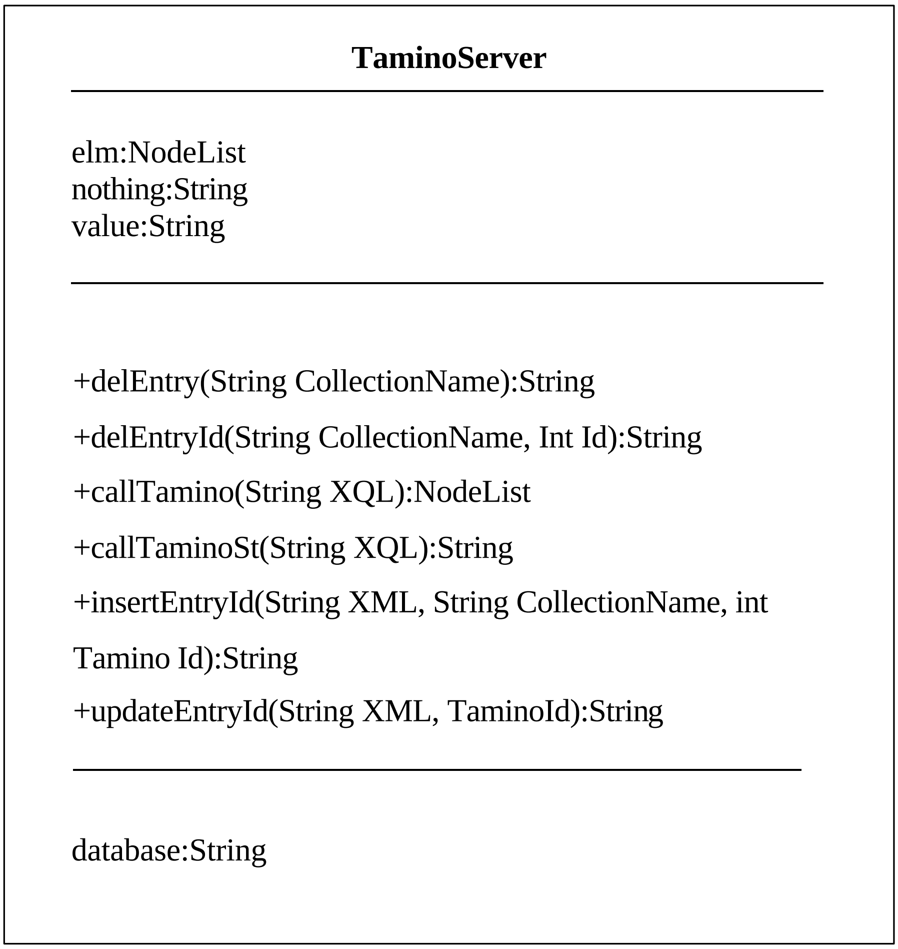


Figure 9: Deployment PCs in STS

6.2.1.2 - Class Diagram of TaminoServer:

The following is a Class Diagram (Figure 10) for the TaminoServer:



TaminoServer

elm:NodeList
nothing:String
value:String

+delEntry(String CollectionName):String
+delEntryId(String CollectionName, Int Id):String
+callTamino(String XQL):NodeList
+callTaminoSt(String XQL):String
+insertEntryId(String XML, String CollectionName, int Tamino Id):String
+updateEntryId(String XML, TaminoId):String

database:String

Figure 10: Server Class

As we can see from the diagram, that the Server has six methods that can be performed on Tamino. UpdateEntryId is the new functionality that Tamino doesn’t have originally. It uses insertEntry and delEntryId to complete the functionality.

6.2.2 - Tamino Server Methods:

- delEntry (String collectionName): This method takes the String argument collectionName, then opens an URLconnection with the Tamino database and tries to delete the whole collection. It returns to the client the Tamino message returned by the database itself, which shows if the deletion was successful or not.
- delEntryId (String CollectionName, int TaminoId): This method does the same thing as the above one, but this time it takes the TaminoId as an argument too. So it tries to find a specific entry in a specific collection in the Tamino Database. It also returns to the client the Tamino message in a String format.
- CallTamino (String XQL): this method takes a String argument which is the XQL query, then forms the appropriate Tamino query and sends it to Tamino using a URL connection. It gets the result in XML format from Tamino. It creates a NodeList from this result and sends it back to the client.
- CallTaminoSt (String XQL): this method does the same as the above one, but instead of creating a NodeList to return to the client, it appends all the result in a StringBuffer, converts it to string and sends it back to the client. We only implemented this method for testing purposes.
- InsertEntry(String XML): This method would send an XML string to be entered into the Tamino database. It would return a success string to let the client know of the result
- UpdateEntryId(String XML, TaminoId): This method uses InsertEntry and delEntryId to fulfil its task. First it retrieves and deletes the entry in question, then inserts the new and updated one back into Tamino. This functionality was not available without using SOAP.

Note: The Tamino Database URL is a local variable in the server that can be accessed using the methods setDatabase(string DB) and getDatabase. It is by default set to the Test database that we have. This setting will stay for the life of the current session.

6.2.3 - Tamino Client:

To call the TaminoServer methods using SOAP, we write the following clients:

- TaminoClientDel: this client needs a CollectionName as a parameter to incorporate it in the SOAP call and try to delete the whole collection. It calls delEntry, its counterpart, on the server side to do the job.
- TaminoClientDelId: This client requires from the user 2 arguments, the Collection name and the tamino id. It calls delEntryID on the server side.
- TaminoClientQ: takes an XQL string as the input from the user and calls callTamino on the server side. It accepts return of type NodeList. So the return is completely in XML format to be processed by the client later for other uses.
- TaminoClientQSt: Same as TaminoClientQ. It calls callTaminoSt that returns the result in a whole string that can be displayed easily. This client was constructed for testing reasons to see Tamino working. An XML return is of more use than String one.

Note: Actually we can combine the whole of the above clients into one client that can have a nice user-interface which can make type checking and prevent errors and be user friendly. The unified client would enable us to: Query, Delete, Update or insert elements directly into Tamino.

The current Tamino Interface supplied by Software AG ™ can't do all those operations. Users must type directly XQL queries into it and should know how to format their queries to perform right. Using Java and SOAP enables us to hide away the complexities from the user and ensure a correct working of the system.

6.2.3.1 - Comparison of calls to Tamino:

1- Direct Call: if we use the Tamino Interactive Interface then one can invoke a query on Tamino from a normal browser using the following XQL(our Tamino server is on PC 23 in STS): http://pc23.sts.tu-harburg.de/tamino/test?_XQL=test.

2- Local Call: If a user that reside on the same machine as the TaminoServer class then he would invoke it as a normal Java call. Ex: callTamino(test).

3- SOAP Call: For SOAP first we would need to deploy the service as we described in section 5.3.1. Then from a SOAP client we need to format the Call as in section 5.1. Then we make a Java Remote method invocation and get the result back. An Activity diagram for constructing the call is in Figure 11 and the whole system in the sequence diagram in figure 12.

The main idea in the SOAP call is that it separates the invocation procedure (Call) from the communication protocol. With the SOAP server, we can access Tamino from any point on the network. We don't need to install special Tamino client software on our side. All we need is to know how to format the SOAP call and to invoke the right methods, which should be published from the server side anyway to be usable.

6.2.3.2 - The SOAP Call:

The following activity diagram shows the steps to construct a Call for a SOAP service:

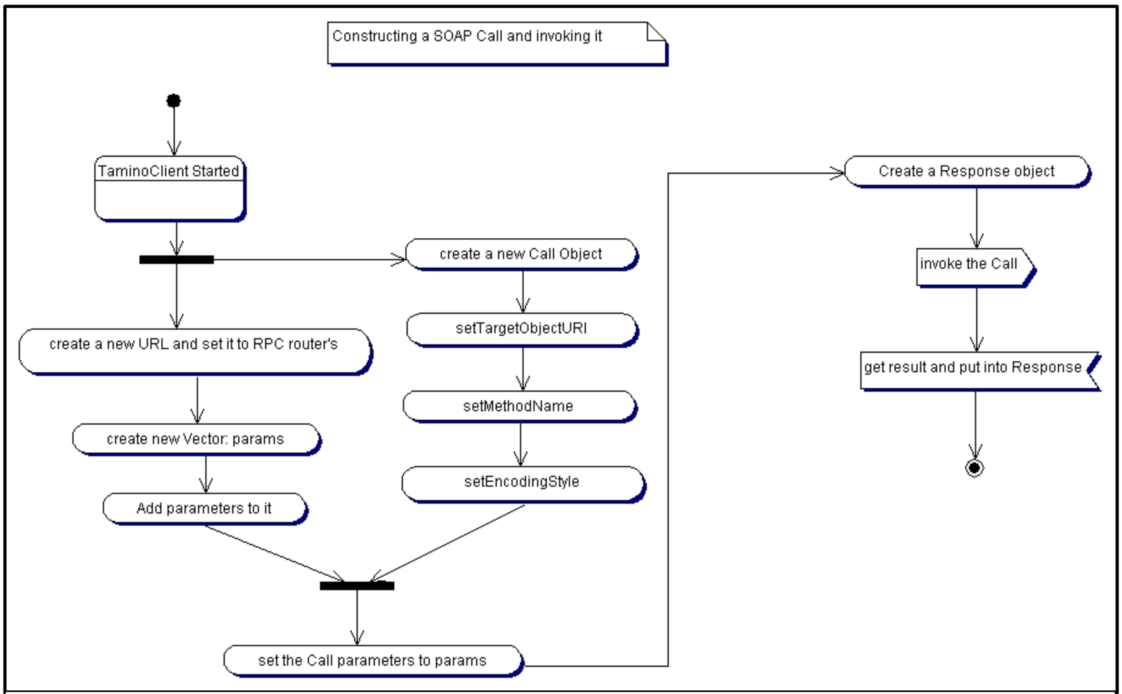

Constructing a SOAP Call and invoking it
TaminoClient Started
create a new Call Object
create a new URL and set it to RPC router's
setTargetObjectURI
setMethodName
create new Vector: params
Add parameters to it
setEncodingStyle
set the Call parameters to params
Create a Response object
invoke the Call
get result and put into Response

Figure 11: Activity diagram for creating a SOAP Call (Tamino Client case)

6.2.3.2 - Sequence Diagram for the system:

The following Sequence diagram shows another view for how a message is passed from a client to the SOAP server, especially in our TaminoClient case.

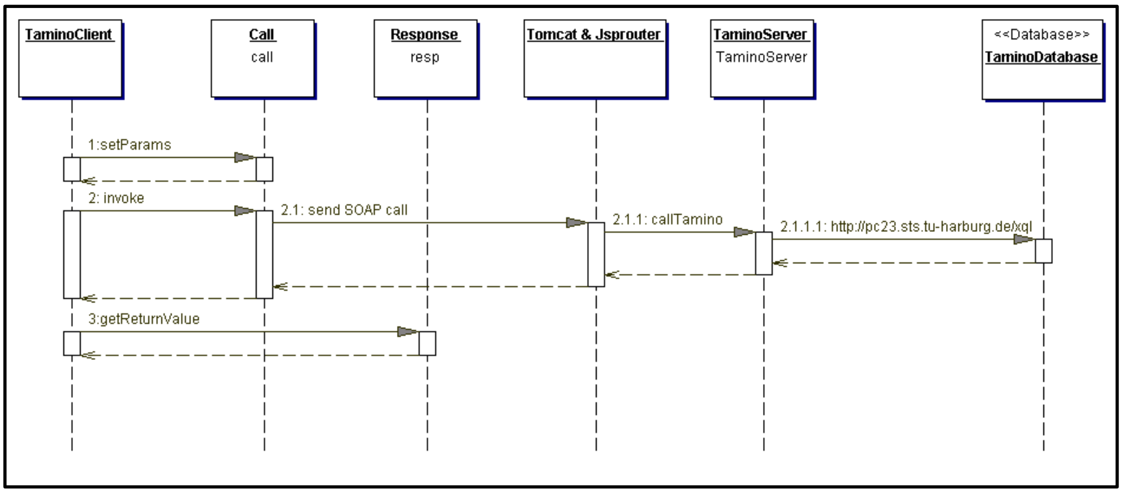


Figure 12: Sequence Diagram for a typical SOAP call

The JSP page or the RPCRouter routes the SOAP calls to the appropriate SOAP service using the namespace identifier that was assigned during deployment.

7- **Results:**

7.1 - Where can SOAP be used:

As it is obvious, SOAP should be used when interoperability is of importance. Currently the execution time of SOAP requests takes longer than of a DCOM or a CORBA one. This is due to the fact that parsing of the XML data requests with current parsers take time. This is overshadowed by the fact that transmission times on the Internet in particular and networks in general take time.

SOAP is the key to connecting different type of platforms and networks together. This will prove important in lots of fields like:

- Databases: where different databases will be able to talk to each other and exchange information in a known and common format: namely XML.

- E-Commerce systems: currently there are lots of different vendors for e-commerce packages, specially the B2B ones. Each use their own programming ideology to implement the system. SOAP will certainly help these different systems and companies to exchange information more easily.

- InterNetworking: Connecting different networks, which are behind firewalls will be much more easier than before, since SOAP uses HTTP or SMTP as the transport protocol, which both can bypass firewalls. Most firewalls have their HTTP port open which is the one that enables a network to access the Internet, but still for security reasons they may filter out SOAP packets if needed by scanning for special header fields.

- Operating Systems: SOAP will prove important in this field also, since an application written in one OS will be able to invoke methods in another application in another OS, which may be implemented totally in a different programming language. As long as both systems talk HTTP and use XML this is feasible.

7.2 - Future Research:

From our research in STS and from what have been already the trend in the programming world we would expect SOAP to be researched and used in the following fields.

1- ebXML: Lots of e-commerce companies are already integrating SOAP into the next Messaging Services Specification, ebXML. This development by ebXML will result in an open, widely adopted global standard for reliably transporting electronic business messages over the Internet.

The ebXML Messaging Specification encompasses a set of services and protocols that allow an electronic business client to request services from electronic business servers over any application-level transport protocol, including SMTP, HTTP and others. ebXML defines a general-purpose message, with a header that supports multiple payloads, while allowing digital signatures within and among related messages. Although the header is XML, the body of the message may be XML, MIME or virtually anything digital.

It would be of interest to see how this will be done by the industry and how one can use it and take advantage of it.
[http://lists.w3.org/Archives/Public/xml-dist-app/]

2- pocketSOAP: This is the specification for SOAP to be used on pocket PC’s. This is a SOAP client COM component for the Windows family. This would be an intresting field if it develops to be used with the other OS’s also, like PALM OS or LINUX for hand-held PC’s.[MSFT]

3- SOAP/EJB/CORBA interoperability: This should be an interesting field to research in. Since CORBA and EJBs are the current hype and they are very interesting technologies themselves and it would be of use to know how SOAP will interoperate and function with them.

7- Summary & Conclusion:

From our research we would conclude that SOAP will be a major player on the networking field. Lots of companies are already adopting it and trying to create tools for it and integrating it with current systems.

Still some work needs to be done concerning security issues and compatibility issues. SOAP will allow different and foreign machines to talk to each other in their own native language. This will increase interoperability between different vendors and systems.

As a result SOAP seems to be a promising protocol that is currently growing up and soon will be the focus of lots of people.

This research project allowed us to further gain understanding of how this protocol works and how it can be used for our different projects and needs. It may be the important link or bridge we were searching for most of our software system problems. It may prove to be an important tool to narrow the gap between different database systems or e-commerce applications.

Still a lot needs to be done to research it more in detail and see how it will develop more and which technologies will be supporting it next.

8- Technical details:

An Important comment for configuring Tomcat is the following:

Tomcat comes with an XML parser (lib/xml.jar) which has the DOM level 1 interfaces. Even if you put Xerces 1.1.2's xerces.jar in your classpath, the wrong interfaces are found by any Java code running in Tomcat because the shell script / batch file that runs Tomcat puts the user's classpath at the end. So, you must edit tomcat.sh or tomcat.bin in the bin/ directory and put xerces.jar at the BEGINNING of the classpath the script builds.

NOTE: For more information consult online documentations at: http://jakarta.apache.org/tomcat/index.html

10 - References:

1- CNET: http://news.cnet.com/news/

2- SOAP-WRC: http://www.soap-wrc.com/webservices

3- SOAP-Security: http://www.w3.org/TR/SOAP-dsig/

4- MSFT: http://msdn.microsoft.com/soap/default.asp

5- DEVX: http://www.devx.com

6- SOAP: http://www.w3.org/TR/SOAP/ (Submission to W3C)

7- SAMS: Understanding SOAP, The Authoritative Solution, by Kennard Scribner and Mark C. Stiver. 2000, by SAMS publishing.

8- VENU: http://www.vnunet.com/News

9- Apache: http://xml.apache.org/soap

10- SOAP DOC: http://localhost/soap/docs/apiDocs/index.html