Advantages of the Simple Object Access Protocol (SOAP) comparing with RPC Protocol

Pattara Kiatisevi
Adviser: Bernd Gloss
Seminar High Performance Network Architecture (HPNA), 2001
INFOTECH
Summer Semester 2001

15th June 2001

Abstract

A computer program can call a procedure or subprogram located on the same host and also on remote machines connected via a network. The mechanism to call a program on another machine is called Remote Procedure Call or RPC. There are a number of RPC technology available in the industry. In this report, we will introduce RPC technology and later focus on the original and popular implementation of RPC, ONC RPC and the new comer in the RPC market, SOAP (Simple Object Access Protocol). The details of each technology will be discussed and finally compared.
Keyword: RPC, XDR, Portmap, SOAP, XML, XML Namespace, XML Schema, Comparison, HTTP

Contents

List of Figures

List of Tables

1 Introduction

Almost all computer systems nowadays are connected to a network to communicate with other systems. Many techniques have been developed to support the interaction between these systems. One such technique is called Remote Procedure Call (RPC). Normally a program calls a procedure on the same host and gets the result back but it is possible also to do this with the remote host. RPC is a general term for the mechanism to run the procedure on the remote host and (optionally) take the result back. RPC works in the client-server fashion.There are many RPC specifications and implementations:

- **Open Network Computing (ONC) RPC** from Sun Microsystems[1], [4]. ONC RPC is one of the first commercial implementations of RPC. It is currently the most popular and is supported by most network software. We will focus at the implementation of this ONC RPC in this report.
- **Distributed Computing Environment (DCE)** from the Open Software Foundation (OSF)[2]. Although not as popular as ONC RPC, DCE RPC is industry-standard, vendor-neutral and has some technical advantages [5]. DCE is deployed in critical business environments by a large number of enterprises worldwide. More information about DCE can be found at http://www.opengroup.org/dce/.
- **The RPC specification from the International Organization for Standardization (ISO)**[3]. It has no current commercial implementation yet.
- **Common Object Request Broker Architecture (CORBA)**. CORBA was developed by a consortium of vendors through the Object Management Group (OMG), which has currently over 750 members including HP, IBM, Microsoft and also University of Stuttgart. Both International Organization for Standardization (ISO) and X/Open have accepted CORBA as the standard architecture for distributed objects (which are also known as components). CORBA is widely used and efficient but however it is complex and has steep learning curve. More information about CORBA can be found at http://www.omg.org.
- **Distributed Component Object Model (DCOM**) is generally equivalent to the CORBA in terms of providing a set of distributed services. But DCOM is invented and used by Microsoft while CORBA is the neutral standard developed by OMG and used by most of the rest companies.
- **RMI (Remote Method Invocation)** is the Java version of RPC developed by Sun Microsystems. RMI is supplied as part of the Java Development Kit (JDK). More information about RMI can be found at http://java.sun.com/.
- **XML-RPC,** from UserLand Software Inc.[6], is a simple XML-based RPC protocol using HTTP as transport protocol. It can be used with Perl, Java, Python, C, C++, PHP and many other programming languages. Implementations are available for Unix, Windows and the Macintosh in both commercial and open-source products.
- **Simple Object Access Protocol (SOAP)**, proposed to the World Wide Web Consortium (W3C) by a group of companies including Microsoft, IBM, and UserLand, is also an XML-based protocol and uses HTTP or SMTP as transport protocol. It has some interesting features which would be discussed further in this report.

This report will focus on **ONC RPC (Remark: later will be called only "RPC")** and **SOAP**. The first one is the original and one of the most popular implementation of RPC. The second one is the new comer in the RPC world that utilizes the power of XML and tends to be popular in the market in the future.

In Section 2, RPC (Sun RPC) will be introduced, followed by SOAP in Section 3. RPC and SOAP will then be compared in Section 4 and the Conclusion would summarize what we have done.

2 RPC

2.1 General idea of RPC

The RPC model describes how processes on different systems can communicate with each other. RPC is based on the concept of a procedure call in a programming language. The semantics of RPC are almost identical to the semantics of the traditional procedure call. The major difference is that while a normal procedure call takes place between procedures of a single process in the same memory space on a single system, RPC takes place between a client process on one system and a server process on another system via the network.

2.1.1 RPC addressing scheme

For a local procedure, specifying the name of procedure to be called is enough. However, for the remote one, ONC RPC Protocol defines a mechanism to identify a remote procedure to be called by three unsigned integer fields.

- Remote program number
- Remote program version number
- Remote procedure number

Program numbers are defined in a standard way:

- 0x00000000 - 0x1FFFFFFF: Defined by Sun
- 0x20000000 - 0x3FFFFFFF: User Defined
- 0x40000000 - 0x5FFFFFFF: Transient
- 0x60000000 - 0xFFFFFFFF: Reserved

Remote program version number can be used to determine if client and server are referring to the same version of code. It is possible that a server or a client is newer than another, they can negotiate the version to be used which both of them understand. Remote procedure number is the number for each procedures in the program (starting from 0, 1, 2 and so on). Server and Client **have to agree** with each other for these numbers to be used before making the connection.

According to [1], the RPC protocol can be implemented on any transport protocol. In the case of TCP/IP implementation, it can use either TCP or UDP as transport layer. Figure 1 shows the RPC protocol stack for the implementation of RPC over TCP/IP.

As described above, RPC uses the *program no., program version no.* and *remote procedure no.* to locate a remote procedure. However, in TCP/IP, a server program is located with *IP address* and *port number* that it binds to. Therefore at the server side, there will be an RPC portmap daemon running at a well-known TCP/IP port (usually port number 111 as assigned by RFC1700 and IANA) waiting for the connection from all RPC clients. The server program will first register itself with the portmap daemon and will get the port number that it should bind to. This port number assignment is not deterministic, the portmap daemon could assign it arbitrarily. The server program then listens to that port.

A client program from the client machine will connect to the portmap daemon, asking for the port number of a server program that it wants to connect to (of course client has to send appropriate parameters:

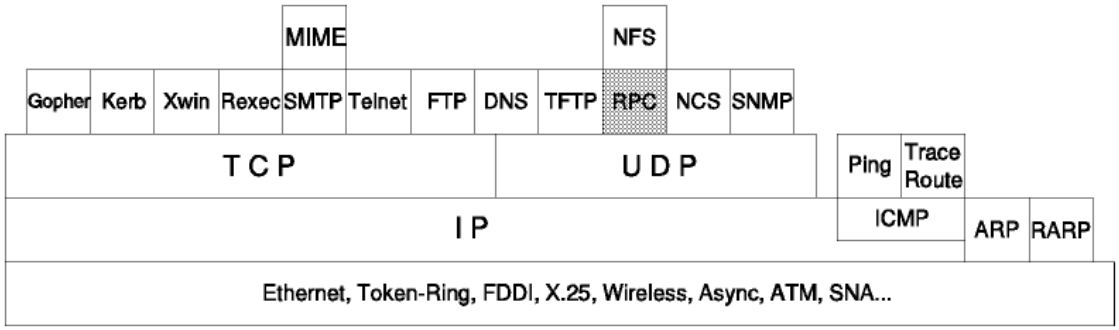

Figure 1: RPC Protocol Stack

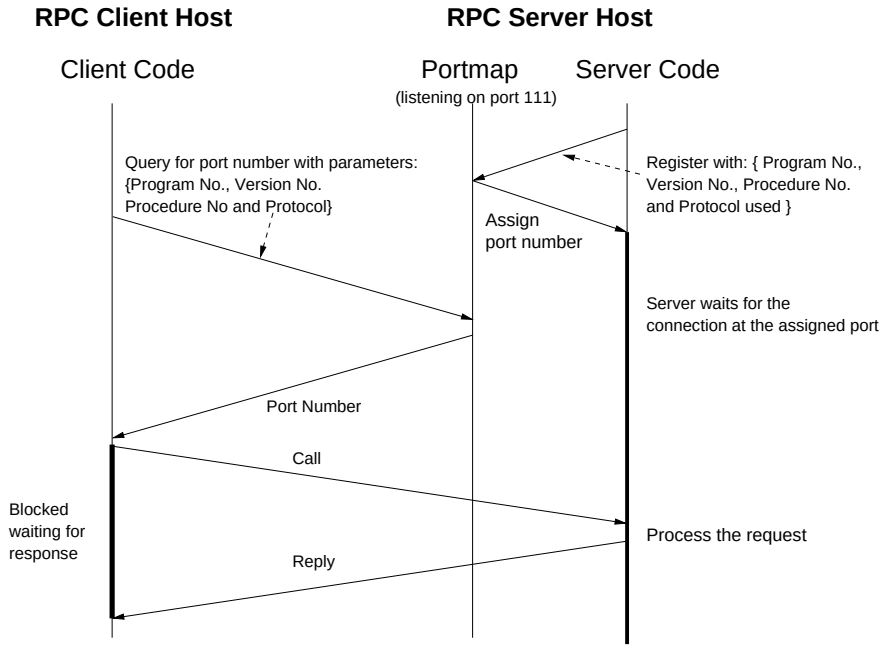

Figure 2: Signalling-time Diagram of an RPC call

program no., version no., procedure no. and *protocol*). The portmap daemon will then reply with the port that the server program is listening to. Then the direct connection between client and server can be established. This operation is illustrated using a signalling-time diagram in Figure 2.

2.1.2 RPC and encoding of data

Now we have the idea how the client and server can communicate. However, we can not simply inject the data from one machine to another machine via the network and hope that the other side would understand it correctly. Because different computer systems might have different architectures and might interpret the same stream of data differently. A good example would be the difference between *big endian* and *little endian* systems. The big endian system assumes that in a sequence of data, the most significant bit will be stored first (at the lowest storage address) and the least significant bit will be stored last (at the highest storage address), while in the little endian system it is reciprocal. In this case the data sent from one system to another system might be wrongly interpreted, therefore a special mechanism for handling this problem is needed.

RPC defines the *eXternal Data Representation* (XDR) as the data format to be sent over the network.

Data from sender will be converted to XDR format and transmitted to the receiver. When the receiver gets the data, it will convert the XDR-format data back to the format it understands. This process is called *marshalling* and *unmarshalling* respectively or generally *serialization*. With this mechanism, RPC can handle arbitrary data types, whether they are integer, floating-point, character string, or complex object. All data will be encoded into the XDR format and will be decoded at another end.

XDR is a binary format. So it saves the bandwidth compared to text-based data format. However it is more difficult to inspect by human than the text-based format which could be viewed by any text editor.

2.2 Features of RPC

In the last section we have learned the basic functions of RPC. Now we will take a look into more details of RPC in each interesting features.

Multiple interface levels

RPC offers multiple levels of Application Programming Interfaces (APIs). These levels provide different degrees of control balanced with different amounts of interface code to implement. In order of increasing control and complexity, the levels are the *simplified level*, *top level*, *intermediate level*, *expert level* and *bottom level*. Developers are free to determine the level according to the simplicity of programming interface or flexibility of control over RPC communications.

Broadcast RPC

Normally an RPC client sends request to one specific server and expects one reply. But in some cases, there are more than one server available and client is free to choose one of them. Broadcast RPC is designed to support this need. Client can send an RPC request in a broadcast manner, i.e. it does not specify the specific destination, and expects many answers (one or more answer from each responding machine). The client can be configured to select the first reply or all replies, or rebroadcast if there is no response.

Batching

RPC is designed so that clients send a call message and wait for servers to reply to the call. That implies that a client is blocked while the server processes the call. This is inefficient when the client does not need each message acknowledged. RPC batching lets clients process asynchronously. RPC messages can be placed in a pipeline of calls to a server. This can be done by the proper settings of return value of XDR routine in the calls and the call's time-out.

Authentication

In the default configuration of ONC RPC implementation, any arbitrary clients can call any remote procedures offered on the servers. This is fine as long as we are not offering any security-critical service on the server. Imagine a service like retrieving or modifying the password database, what would happen if we allow any clients to call the service. In such cases, some kinds of authentications might be needed to allow only authorized clients to use our services. Application programmers are also free to implement their own application-level authentication methods. But RPC also provides the basic authentication services which can be immediately used as following:

- **AUTH_SYS, AUTH_SHORT**: An authentication flavor based on UNIX operating system, process permissions authentication. It utilizes the existing user/group data already available on the UNIX operating systems.
- **AUTH_DES**: An authentication flavor based on DES encryption techniques.
- **AUTH_KERB**: Version 4 Kerboros authentication based on DES framework.

Multithreaded

As we can see from Figure 2, the client is blocked during the call operation to the server. This is not effective if client can do something else useful while waiting for response from the server. RPC supports multi-threaded mechanism which can solve this problem. In a multithreaded client program, a thread can be created to issue each RPC request, while other threads can proceed their tasks. As all system environments are shared among threads in the same process, the RPC call result can be used and further processed by all threads.

In the case of very busy server that has to handle large numbers of requests, it can also be configured to create a new thread for each incoming client request. The server will be ready for the new request immediately after creating the new thread to serve the current request. The new thread processes the assigned request on its own, sends a response back to the client and exits.

2.3 RPC deployment

This section will give a brief introduction on how to develop client and server programs that communicate with each other using RPC protocol. A client and server systems with operating systems that support RPC, and these software elements are needed:

- **RPC library** for both client and server
- **RPC portmap daemon** for server. It should be running and waiting for the connections.
- **Tools for developing RPC programs**, e.g. rpcgen, on the development machine

As ONC RPC has been in the industry for a long time, usually they are supported and included in the operating systems. Also ONC RPC is currently supported by GNU Library C, so it is probable that these needed elements are already available in your system.

Design the interface

The first step to develop the RPC applications is to design the interface of the server program that client will access to. It needs to be defined in *XDR data description language*, whose syntaxes are similar to C. Here is an example how the XDR data description language looks like:

```
program HELLO {
  version HELLO_VERS_1 {
    void HELLO_NULL(void) = 0;
    int HELLO_PUTS(string text<>) = 1;
  } = 1;
} = 400000;
```

In this case the program HELLO will have program number 400000 with protocol version number 1. It has two remote procedures: HELLO_NULL and HELLO_PUT with procedure number 0 and 1 respectively. After we have the interface, we can start to code the server and client programs.

Write server and client programs

In order to have a complete RPC application. These files are needed:

- Server program
- Client program
- Server stub [automatically generated]
- Client stub [automatically generated]

- Makefile and appropriate include files [automatically generated]

Server and client stubs are the codes that hide the underlying implementation of the RPC call. Makefile and include files are needed in the compilation process. To ease the programming task, RPC provides a tool *rpcgen*, which can automatically generate all these files and even the skeleton of the server and client program to be the starting point for the developers. The rpcgen tool needs the interface file (in XDR data description language as shown above) as input.

Developers could then focus only on programming the server and client programs where they can use the simple RPC call provided by the RPC library and the stubs without having to know the complexity of network programming. However, developers can also fine tune programs by adjusting these stubs and utilizing the low-level APIs of RPC.

Run the application

After all programs are written and compiled. The output executable files will be the server and client program which should be placed on the server and client machine respectively. After being executed, the server program will contact the portmap daemon to register itself and will get the port number that it will bind to and wait for the connection. By executing the client program, it will connect to the remote server and make a remote procedure call.

This Section gives a rough idea of RPC applications development process. With the help of the tool and the rich set of RPC APIs make it easy and efficient to develop an RPC application. More detailed information about RPC Application development can be found in [8], and in a lot of network programming books.

3 SOAP

As SOAP uses XML as the encoding format of the message, before jumping to the detail of the SOAP protocol, we will discuss briefly about XML and its benefits in Section 3.1. After that we will go into the detail of SOAP Protocol in section 3.2.

3.1 XML

3.1.1 Why XML?

There is usually a need to share data or information among multiple applications from different vendors, run on different kinds of machine. The data could be encoded in a language that all those applications could understand. The requirements of the language are as follows:

- Easy for developers to use and for computers to process
- Possible to store data in structured format
- Extensible for arbitrary data. The output document should be self-describing, not format-oriented
- Open standard

There are plenties of file formats existing in the world, for example, normal text, RTF (Rich Text Format), HTML, SGML, LaTeX, Microsoft Word's DOC format and etc. The question is that which format of data could we use? Binary format like DOC is hard to understand and to be inspected by human. HTML is text-based, easy to understand and already widely-used, but it is designed to convey the information "how the output of this WWW page should be?", not the structure of information inside. It is also not flexible enough to be used to create "structured document" which has nothing to do with the display output. SGML is capable of doing this task, but it is too complex and hence not effective to be used with simple tasks. LaTeX is the language that is mainly used for formatting and typesetting documents which focuses also on the output more than the structure of the document.

Since there was no such language that could fulfill this needs, XML (Extensible Markup Language) was developed.

3.1.2 What is XML?

XML (eXtensible Markup Language) is the universal format for structured documents and data on the Web. It has some interesting characteristics as following [10] :

XML is a method for putting structured data in a text file

XML is a set of rules, guidelines, conventions for designing text formats for such data, in a way that produces files that are easy to generate and read by computer and easy to inspect by human. XML is designed to be unambiguous, platform-independent, and supports extensibility, internationalization/localization.

XML looks like HTML but it is not HTML

Both XML and HTML use tag to identify elements, e.g. "** **data**)**" and use attribute to specify parameters inside the tag, e.g. "<font **color=red**>". But XML does not have specific meaning for each tag and attribute (e.g. **p** or **fon**t in this case). It leaves room for application to define its own way to interpret that data. One can not simply assume that "** **" in the XML file would mean the paragraph, as it could have other meanings, e.g. **price or percent** depending on how it is defined.

XML is text, but it is not meant to be interpreted by human

XML is like other programming language in which the content could be investigated and debugged by the programmers and developers. However the syntax has strict formats (even stricter than HTML) that a simple mistake could make the whole document unusable. So it is not intended to be read and edited like a letter or normal documents.

XML is a family of technologies

XML 1.0 Standard specifies what tags and attributes are. However there are a numbers of additional specifications that define also the set of tags and attributes for specific tasks. **Xlink** describes a standard way to add hyperlinks to an XML file. **XPointer & XFragments** are syntaxes for pointing to parts of an XML document. **XSL** (eXtensible Stylesheet Language) is the advanced language for expressing style sheets. It is based on **XSLT** (XSL Transformations), a transformation language that is often useful outside XSL as well, for rearranging, adding or deleting tags & attributes. The **DOM** (Document Object Model) is a standard set of function calls for manipulating XML (and HTML) files from a programming language. **Namespaces in XML** is a specification that describes how you can associate a URL with every single tag and attribute in an XML document. How that URL will be used is up to the application that reads the URL. **XML Schemas 1 and 2** help developers to precisely define their own XML-based formats.

XML is verbose

Since XML is text-based and uses the tag and attribute. The size of XML file is normally larger than binary format for the same purpose but a text format it is easier for human to inspect and debug than the binary one. The factors of disk space consuming and the readability of text-based format need to be weighted. This point will be mentioned later when we compare RPC and SOAP.

XML is new, but not too new

Development of XML started in 1996 and it is a W3C standard since February 1998, so one could see XML as a new and possibly immature technology. However, XML is based on SGML which is developed in early 80's. XML also utilizes the experience of HTML which is widely used on the Internet. Right now, XML is used in many applications by a large number of organizations all over the world.

XML is license-free, platform-independent and well-supported

XML is a neutral standard and supported by a large number of organizations. It is not platform- or implementation-specific.

XML is a huge topic and out of scope of this report. However, there are some terms that might be useful to know before discussing about SOAP further.

Document Type Definition (DTD)

DTD is a mechanism defined in XML 1.0 Specification to declare **constraints on the use of markup in XML**. An XML document that conforms to the specified DTD would be called the "**valid**" XML document. For example, DTD could specify that the element *email* should compose of *to, from, subject* and *body* attribute and these attributes should be of type CDATA (Character Data).

XML Schema

While DTD can be used to control the use of markup, automated processing of XML documents requires more rigorous and comprehensive facilities than what DTD can support. XML Schema is developed to be the next generation of DTD. XML Schema allows extensive constraints on the document structure, attributes, data-typing, and how these component parts fit together. Moreover, XML Schema has the syntax like XML itself, unlike DTD that has its own syntaxes, so it is easier for developers to learn and it is extensible. More information about XML Schema can be found in [11].

XML Namespace

An element and attribute in one context might have the same name as others but have different meanings. For example, element "P" in the Book element might be a sign for paragraph while "P" in the Food element might be the price. By adding prefix to the attribute name could let all the attributes be unique as long as the prefix is unique and the attributes under the same prefix are unique to each other. Then we will call the group *namespace*. Namespace is the collection of names that are unique and would have prefix as URI (Uniform Resource Identifier).

3.1.3 Examples of XML messages

Here is a simple example of XML. It shows how to encode an e-mail that has *to, from, subject,* and *body* data fields in XML.

```
<?xml version="1.0"?>
<memo>
  <to>nok@thai.com</to>
  <from>ott@thai.com</from>
  <subject>Report is in progress</subject>
  <body>Hello, the report is being written!</body>
</email>
```

Next example is the same message with the use of XML Schema. Usually XML Schema will be used in conjunction with the namespace. In this case, this schema is defined under namespace *em*. Notice that all elements are prefixed with *em*.

```
<?xml version="1.0">
<em:email xmlns:em="x-schema:myschema.xml" lan-
guage="Western" encrypted="128" priority="HIGH">
  <em:to>nok@thai.com"</em:to>
  <em:from>ott@thai.com"</em:from>
  <em:subject>My first schema</em:subject>
  <em:body>Hello, this is Pattara</em:body>
</em:email>
```

And here is the schema *myschema.xml*. The element *email* and its child attributes *to, from, subject* and *body* are defined. Notice how the attribute type and element type are declared.

```
<?xml version="1.0"?>
<Schema name="email" xmlns="urn:schemas-test-com:xml-
data" xmlns:dt="urn:schemas-test-com:datatypes">
  <Attribute-
Type name="language" dt:values="Western Greek Latin Univer-
sal"/>
  <AttributeType name="encrypted"/>
  <Attribute-
Type name="priority" dt:type="enumeration" dt:values="NORMAL LOW HIGH"/>
  <AttributeType name="hidden" default="true"/>
  <ElementType name="to" content="textOnly"/>
  <ElementType name="from" content="textOnly"/>
  <ElementType name="subject" content="textOnly"/>
  <ElementType name="body" content="textOnly"/>
  <ElementType name="email" content"eltOnly">
    <attribute type="language" default="Western"/>
    <attribute type="encrypted"/>
    <attribute type="priority" default="NORMAL"/>
    <element type="to" minOccurs="1" maxOccurs="*"/>
    <element type="from" minOccurs="1" maxOccurs="1">
    <element type="subject" minOccurs="0" maxOccurs="1"/>
    <element type="body" minOccurs="0" maxOccurs="1"/>
  </ElementType>
</Schema>
```

3.2 SOAP Protocol

3.2.1 General idea of SOAP

SOAP[13] provides a **simple** and **lightweight** mechanism for **exchanging structured and typed information** between peers in a **decentralized, distributed** environment using **XML**. SOAP does not itself define any application semantics or implementation specific semantics hence it could be used in a large variety of systems ranging from messaging systems to RPC. In this report, we will look at SOAP by focusing at the RPC application point of view.

In order to understand how SOAP looks like, we can start at the RPC (ONC RPC). In that case, we have an RPC client program running on the client and RPC Server program (and portmap daemon) are running on the server. SOAP is almost the same, except that the messages sent between client and server are encoded in XML. By using XML as the transport format, we have many benefits:

- XML itself is until now one of the best and most popular format to share documents and information between applications. It is trivial for developers to turn to SOAP as their applications might be using XML already.
- There are a lot of libraries to encode/decode XML data. It is also trivial to convert the data from the whatever legacy format into XML.

Moreover, instead of defining its own transport layer protocol, SOAP utilizes HTTP, which is already used on the Internet. SOAP Specification defines the use of SOAP over HTTP but however, the concept can be applied to other transport protocol, e.g. SMTP or whatever as long as the SOAP message could be transmitted and received correctly by the client and server.

Not only SOAP that uses XML as the transport format and HTTP as transport layer protocol, but also XML-RPC. XML-RPC has been in the industry even before SOAP. It has support libraries running

on almost all platforms, for almost all popular languages and environments (Perl, Python, Java, Frontier, C/C++, Lisp, PHP, Microsoft .NET, Rebol, Real Basic, Tcl, Delphi, WebObjects and Zope). XML-RPC is stable and mature. So why is SOAP being introduced? We can first look at the SOAP Specification.

SOAP Specification 1.1 consists of 3 parts:

- **The SOAP envelope** defines an overall framework for expressing what is in the message, who should deal with it, and whether it is optional or mandatory.
- **The SOAP encoding rules** defines a serialization mechanism that can be used to exchange instances of application-defined datatypes.
- **The SOAP RPC representation** defines a convention that can be used to represent remote procedure calls and responses.

So as we can see from the specification, SOAP has two functionalities in addition to basic RPC (that uses XML) features. The first is the envelope, which carries the information about the included message. The second is a set of rules for encoding application-specific datatypes. These two are the main points that make SOAP different from normal XML-RPC.

In the next section we will take a look into a SOAP message and followed by some examples.

3.2.2 SOAP message

A SOAP message is an XML document that contains:

- **SOAP Envelope** is the top element of the XML document representing the message. The envelope contains the body and optionally the header.
- **SOAP Header** is a generic mechanism for adding features to a SOAP message.
- **SOAP Body** is a container for mandatory information intended for the ultimate recipient of the message. By the word “ultimate”, it means that maybe there are intermediate recipients along the way which are not supposed to be involved in the content of the message.

SOAP envelope and body must exist in every SOAP messages, while SOAP header is optional. SOAP message must not contain a DTD. Instead, the proper Namespace (and Schema which would be defined in the Namespace) on all elements and attributes must be defined. The messages that have incorrect Namespaces will be discarded. See the last section about XML for more information about Schema and Namespace.

SOAP *encodingStyle* attribute can be used the indicate the serialization rules used in the SOAP message. It can be defined by the URI pointing to the appropriate XML Schema. The default specified in the specification is “http://schemas.xmlsoap.org/soap/encoding/”. Here is the example of open tag for SOAP envelope including encodingStyle attribute:

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/" SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
```

As we can define the encoding style with the XML Schema, SOAP has great support for application-specific types. There are simple types already available (e.g. int, float, string) and compound types like struct, array could also be constructed from elements of simple types. Developers could adjust the Schema to support their needs.

In the SOAP Header, we could define some extra elements with these attributes:

- **“mustUnderstand”** attribute specifies whether the header entry is mandatory or optional for the recipient to process. If a header element is tagged with a SOAP mustUnderstand attribute with a value of “1”, the recipient of that header entry must obey the semantics and process correctly to those semantics, or otherwise must fail processing the message.

- **"actor"** attribute specifies who should process the message. As a message can travel from the originator to the ultimate destination by possibly passing through a set of SOAP intermediaries. We can specify the actor attribute so that only the ultimate destination will process the message, other intermediaries will just forward the message further until it reaches destination without any processing.

In case of server can not process the request, can not understand the extra element in the header and the "mustUnderstand" attribute is set, or any faults occurs in the processing of request, SOAP Fault element will be used to carry error(s) back to client. In the SOAP Fault message, these sub-elements will be included: *faultcode, faultstring, faultactor* and *detail*. More information about SOAP Fault handling could be found in [13].

Examples of SOAP encoding scheme and SOAP transactions are shown in the Appendix.

3.3 SOAP deployment

There are 2 implementations of SOAP at the time of writing: *Microsoft SOAP SDK* and *Apache XML-SOAP* based on the code contributed by *IBM, SOAP4J*. [15] compared both and found that Apache XML-SOAP (IBM SOAP4J) appeared to be more standard-compliant, more stable and compatible with other platforms. Apache XML-SOAP is also freely available in the public domain so we will focus on this implementation in this report.

Apache XML-SOAP is an open-source implementation of the SOAP v1.1 and SOAP Messages with Attachments specifications in Java. In this Section we will give a brief introduction on how the SOAP client and server applications can be created. More detailed information can be found in [16].

We will need 2 systems, client and server. As Apache XML-SOAP code is in Java so the Java Development Kit (Sun JDK 1.2.2 is recommended) must be properly installed on both client and server. Other than that, we will need:

- **Web server software with Servlet functionality** on the server, e.g. Tomcat[17].
- **Apache XML-SOAP** on both server and client machines. On the server machine, the Apache XML-SOAP will need to be installed as a servlet application of the web server. The rpcrouter servlet will be up and running after the installation process.

The development steps are slightly different from those of ONC RPC. In RPC's case, works are done almost equally in coding both server and client program. Both of them require RPC-specific modification, i.e. RPC APIs will be used from both of them. In SOAP's case, the server program code has nothing different from normal Java code. One needs only to write a normal Java class, put it in the CLASSPATH of the web server so that it can be called from the server.

After creation of server program code, an additional step to tell the web server that this code is allowed to be called from client via SOAP protocol is needed. This step is called *deployment* and that piece of server program code will be called *service.* If the server and client will send the complex type of parameters, i.e. not the simple, pre-defined types e.g. integer, string, the serialization mechanism needs to be declared. Apache XML-SOAP provides a number of pre-built serializers and deserializers for many data types including Vectors, Enumerations, arrays, and JavaBeans.

In the deployment of a service, a unique *object ID* must be assigned. Two different services on the same server must not have the same object ID. This object ID will then be used by the client to identify which service it wants to call. After the service is deployed, web server is ready to serve the SOAP request for this deployed service.

In the client side, a Java client program needs to be written. Package *org.apache.soap* in Apache XML-SOAP provides support for performing RPC over SOAP and hides all low-level complexities from the developers. The client program shall create a *Call* object with the desired service object ID. Then client calls the *invoke* method of the Call object with parameters URL (to the location of rpcrouter servlet, e.g. http://server/apache-soap/rpcrouter) and SOAPAction header (the use is up to each application, could generally be a URI of the remote procedure). The request will be sent to the rpcservlet on the server side.

Rpcrouter servlet on the server receives the SOAP request, rebuilds a Call object using the object ID from the request, verifies the method name and invokes the server code. Server code is run and when

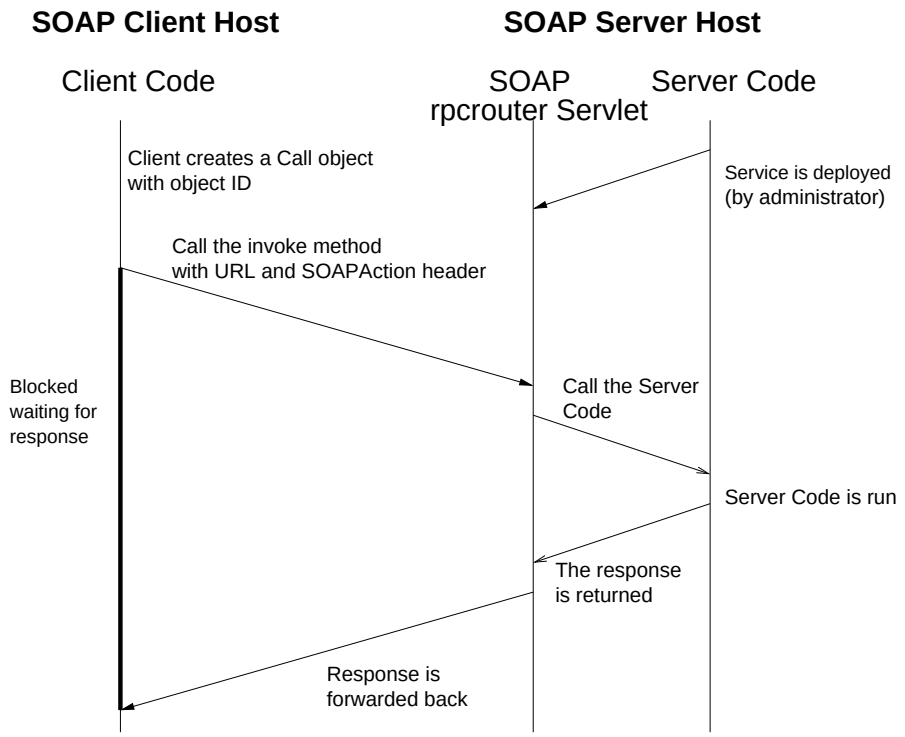

Figure 3: SOAP Transaction Signalling-time Diagram

finishes, return value is sent back to the rpcrouter servlet and forwarded back in the HTTP response to the client. Rpcrouter servlet also handles faults if occur and will send appropriate SOAP Fault messages to the client.

Signalling-time diagram in Figure 3 illustrates a SOAP transaction in the Apache XML-SOAP implementation.

4 Comparison of RPC and SOAP

In this Section we will compare RPC and SOAP in each aspect. First we look at the transport format of data, RPC uses the binary XDR format while SOAP uses text-based XML. Both formats can handle arbitrary data types. XDR has benefit as it is binary format that it saves storage space than the text-based one. However, it seems that the readability and interoperability of the format among various applications are more important than the disk space and network-bandwidth factors. XDR data format is specific to RPC. It is not known to be widely used by other applications so the conversion might be needed from the existing application-specific data format to XDR, while XML is nowadays accepted as a standard format of data interchanging. There are more and more applications that commit support for XML. SOAP seems to win at this point.

Both RPC and SOAP offer supports of primitive data types and also the customized data types. However, for complex and customized data types, in RPC one need to customize the XDR data description language which is RPC-specific and not widely used. While in SOAP, one can customize the XML Schema which is easier and extensible.

The next point is the transport protocol. ONC RPC implementation has its own transport protocol implementation based on TCP/UDP, while SOAP utilizes the existing popular transport protocols used in the Internet, HTTP and SMTP. It is hard to judge which method is better. RPC should be faster and save the network-bandwidth as its transport protocol is designed specially for this purpose while SOAP has to include the appropriate HTTP or SMTP headers and follow other specifications of these protocols.

Protocol	RPC (ONC RPC)	SOAP
Transport Format	XDR (binary, specific to this protocol)	XML(text-based, standard)
Simple Data Type	Available	Available
Customized Data Type	Achieved by XDR language	Achieved by XML Schema
Transport Protocol	Own implementation on top of TCP/UDP	HTTP/SMTP
Authentication	Built-in authentication support (NONE, SYS, DES, KERB)	- leverage to transport protocol -
Firewall Friendly	Difficult, complex to filter	Easy, because of the use of HTTP
Stability & Maturity	Yes	Need to be proved
Availability	Supported by most software	Fewer supported software

Table 1: Comparison of RPC and SOAP

However, by using HTTP and SMTP, it can utilize the extra features which are already available e.g. the encryption/authentication of data using public key technology offered by SSL/TLS.

RPC offers built-in authentication support (UNIX, DES, KERB) while SOAP leverages this to the underlying transport protocol (HTTP or SMTP) which has extensive supports for authentication also. This point none has advantage. However, if we look at the access control aspect, e.g. for the firewall to filter some services and allow some services, RPC is nightmare for network administrator because the server program will be assigned arbitrary port number from portmap daemon, one could never know which port will be assigned. For SOAP, situation is much better. All connections will be done to port 80 of server (HTTP port) and rpcrouter will serve as proxy for all requests, i.e. there will be no direct connection between client and server programs.

RPC has the clear advantage that it is mature and widely available. It has been used in the real production environments and proved to be successful. An example of important applications that utilize RPC is NFS (Network File System) which is supported by almost all operating systems. Also RPC has a rich set of APIs that is very flexible for developers to use. SOAP is new to the market and thus has fewer supported software and applications. SOAP still needs to prove itself if it will be popular in the future.

Table 1 summarizes the comparison of RPC and SOAP.

As we can see from the comparison, if you are looking for something stable and those disadvantages of RPC are not the problem for you, RPC would be the choice. But if you need the capability of XML and HTTP/SMTP, and would like to try the new technology, SOAP is clearly the choice. However, there are still many other RPC technologies as listed in Section 1, not only RPC and SOAP, to consider. XML-RPC is a very close choice to SOAP as it offers XML over HTTP protocol. Although XML-RPC does not have envelope and encoding as SOAP, it is more stable, not as complex as SOAP and has more supported libraries. RMI is an interesting choice of binary protocol (compared to RPC) if your applications are pure Java. CORBA is also binary protocol and supports variety of platforms.

5 Conclusion

RPC has advantages that it is stable, widely used, and bandwidth-save. However, as RPC uses XDR data format which is not common to other applications, conversions might be needed. Also RPC causes problems for network administrator to implement the access control mechanism. SOAP offers the use of XML, which is becoming the standard in data and information sharing nowadays, as the transport format. SOAP utilizes HTTP, which is extremely popular, feature-rich and firewall-friendly, as the transport protocol . SOAP also offers the feature of envelope and has rich support for application-specific data types. However, SOAP is new to the market and has fewer supported software. Apart from RPC and SOAP, other RPC implementations might also be considered.

References

[1] IETF: RFC 1057, Remote Procedure Call Protocol Specification Version 2

[2] OSF: DCE 1.0 Application Development Guide. Technical report, Open Software Foundation, December 1991.

[3] ISO: Remote Procedure Call Specification. ISO/IEC CD 11578 N6561, ISO/IEC, November 1991.

[4] Sun Microsystems Inc.: Network Programming Guide, Revision A March 27 1990.

[5] John Barkley: Comparing Remote Procedure Calls, http://hissa.nist.gov/rbac/5277/titlerpc.html

[6] XML-RPC Website, http://www.xmlrpc.org

[7] References TCP/IP Tutorial and Technical Overview, http://www4.ulpgc.es/tutoriales/tcpip/pru/3376fm.htm

[8] Francisco Moya Fernndez: RPC Tutorial, http://www.opensources.com/magazine/modern-network-programing/rpc

[9] Linux man page of “portmap”

[10] W3C: XML Page, http://www.w3.org/XML/

[11] W3C: XML Schema, http://www.w3.org/XML/Schema

[12] Chaidamrong Utirumn: XML Step by Step, Microsoft Press, Thailand, ISBN 974-87767-1-9

[13] W3C: SOAP 1.1 Specification, http://www.w3.org/TR/SOAP/

[14] Brett McLaughlin: A closer look at SOAP, http://www-106.ibm.com/developerworks/xml/library/x-soapbx2/

[15] James Snell: MS SOAP SDK vs IBM SOAP4J: Comparison & Review, http://windows.oreilly.com/news/soapreview_0600.html

[16] Tarak Modi: Clean up your wire protocol with SOAP, Part 1 & 2, http://www.javaworld.com/javaworld/jw-04-2001/jw-0427-soap.html

[17] The Jakarta Project: http://jakarta.apache.org/

[18] IT Specific Encyclopedia, http://www.whatis.com/

Appendix

SOAP Encoding

SOAP encoding rules define a serialization mechanism that can be used to exchange instances of application-defined data types. A type could be either a simple (scalar) type or is a compound type constructed as a composite of several parts, each with a type. According to the ability of XML and Schema, SOAP can have a very flexible way of defining application-specific type.

Examples of simple types that have built-in support in SOAP are: int, float , negativeInteger, string. We can use these types directly in the Schema. Here is the example of fragment of Schema.

```
<element name="Partnumber" type="int"/>
<element name="Price" type="float"/>
```

And we can construct enumeration type based on the simple type.

```
<element name="Color">
  <simpleType base="xsd:string">
    <enumeration value="Red"/>
    <enumeration value="Green"/>
    <enumeration value="Blue"/>
  </simpleType>
</element>
```

We could create the struct composing of those types.

```
<element name="Computercase"/>
  <complexType>
    <element name="Partnumber" type="int"/>
    <element name="Price" type="float"/>
    <element name="Color">
      <simpleType base="xsd:string">
        <enumeration value="Red"/>
        <enumeration value="Green"/>
        <enumeration value="Blue"/>
      </simpleType>
    </element>
  </complexType>
</element>
```

And here is the example of XML code.

```
<a:Computercase>
  <Partnumber>12345</Partnumber>
  <Price>100</Price>
  <Color>Green</Color>
</a:Computercase>
```

Examples of SOAP Transactions

Here is an example of the SOAP request. It composes of HTTP header part (from the line “POST /Stock-Quote HTTP/1.1” to “Content-Length: 472”), the SOAPAction line and SOAP Envelope. SOAPAction field can be used to indicate the intent of the SOAP HTTP request depending on the need of application. SOAP Specification places no restrictions on the format of it.

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: 472
SOAPAction: "Some-URI"
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/" SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  <SOAP-ENV:Header>
    <t:Transaction xmlns:t="some-URI" SOAP-
ENV:mustUnderstand="1"> 5 </t:Transaction>    </SOAP-
ENV:Header>
  <SOAP-ENV:Body>
    <m:GetLastTradePrice xmlns:m="Some-URI">
      <symbol>DEF</symbol>
    </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

And here is the example of the response. The response composes of HTTP header (from the line “HTTP /1.1 200 OK” to “Content-Length: 488”) and SOAP Envelope.

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: 488
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/" SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  <SOAP-ENV:Header>
    <t:Transaction xmlns:t="some-
URI" xsi:type="xsd:int" mustUnderstand="1">
      5
    </t:Transaction>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <m:GetLastTradePriceResponse xmlns:m="Some-URI">
      <Price>34.5</Price>
    </m:GetLastTradePriceResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```