Dynamic Approach for Web Services Selection

and

Analysis of Security Protocols

A thesis submitted in partial fulfillment

of the requirements for the degree of

Master of Technology

in

Computer Science and Engineering

Specialization:Information Security

by

Abhishek Pandey

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela, Orissa, 769 008, India
May 2009

Dynamic Approach for Web Services Selection and Analysis of Security Protocols

A thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Technology

in

Computer Science and Engineering

Specialization:Information Security

by

Abhishek Pandey

under the guidance of

Prof.Sanjay Kumar Jena

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, Orissa, India
May 2009

To my parents

&

loving daughter

Aryana

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, Orissa, India.

Certificate

This is to certify that the work in the thesis entitled ***Dynamic Approach for Web Services Selection and Analysis of Security Protocols*** submitted by ***Mr.Abhishek Pandey*** in partial fulfillment of the requirements for the award of the degree of Master of Technology in Computer Science and Engineering during the session 2008–2009 in the department of Computer Science and Engineering, National Institute of Technology Rourkela is an authentic work carried out by him under my supervision and guidance.

To the best of my knowledge, the matter embodied in the thesis has not been submitted to any other University/Institute for the award of any Degree or Diploma.

Dr. Sanjay Kumar Jena
Professor
Dept. of Computer Science & Engineering
National Institute of Technology
Rourkela-769008 Orissa (India)

Place: NIT Rourkela
Date: 12 May 2009

Acknowledgment

Many people have contributed, directly or indirectly, to the successful completion of this thesis. They will all be remembered in my heart. First, I would like to thank my advisor, **Professor (Dr.) SanjayKumar Jena** for his guidance from conducting the research to writing the thesis and support that he has extended to me.

I am extremely grateful to **Professor (Dr.) B. Majhi**, Head of the Department, Computer Science and Engineering, for being a great source of advice. I would also like to thank Prof.S.K.Rath,Prof A.K.Turuk, Prof.B.Sahoo and Prof.Pankaj Sa for their support.

My special gratitude is due to my parents for their loving support. I owe my loving thanks to my wife and daughter.

I would also like to extend my sincere thanks to **Dr.Jayendra Narang** for his motivation and continious help.

I would like to thank all the faculty members, Department of Computer Science and Engineering who gave all possible help to bring my thesis work to the present shape and also like to thank Mr. Pushpendra Kumar Chandra for his kind support in completing the documentation of this thesis.

Abhishek Pandey

Abstract

In the domain of Web Services, it is not uncommon to find redundant services that provide functionalities to the clients. Services with the same functionality can be clustered into a group of redundant services. Respectively, if a service offers different functionalities, it belongs to more than one group. Having various Web Services that are able to handle the client's request suggests the necessity of a mechanism that selects the most appropriate Web Service at a given moment of time. This thesis presents an approach, Repository Based Web Services Selection, for dynamic service selection based on virtualization on the server side. It helps managing redundant services in a transparent manner as well as allows adding services to the system at run-time. In addition, the model assures a level of security since the consumers do not have direct access to the Web Services.

This work describes different security aspects of Web Services and technologies they use and a framework to introduce a message level security to SOAP (Simple Object Access Protocol). The purpose of the session protocol is explained along with the approach to authenticate two Web Services with each other and how to establish a shared secret session key with which they can encrypt their messages to ensure confidentiality. Various security issues that became relevant during the design of the system and at the time of setting up the SOAP session are being addressed in this work. The analysis of the session setup process proves that an adversary cannot break the protocol by interception, alteration or by resending of messages.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Introduction

Web Services Security

Problem Definition

Chapter 1

Introduction

1.1 Introduction

> *Web Services is an integrated solution for realizing the vision of the next generation of the Web*

Web Service is a software component invoked over the Web via an XML [1] message that follows the SOAP [2] Simple Object Access Protocol: is a simple XML based protocol to let applications exchange information over HTTP to transport it using open protocols) standard and also it is a communication protocol. Web Services are based on distributed technology and provide standard means of interoperating between different software applications across and within organizational boundaries with the use of XML. The basic Web Service platform is combination of HTTP and XML. The HTTP protocol is the most used Internet protocol. XML provides a language which can be used between different platforms and programming languages.

Each SOAP message is presented as an envelope with two sections - a header and a body. The header contains information about the message itself, and the body consists of data that has to be transferred to the recipient. These standards make the Web Services independent of any programming language, hardware and software platform.

Web services use XML to code and to decode data, and SOAP. Web Services are web applications whose interfaces are exposed over protocols like HTML and XML [3] .Web Services are described by Web Service Definition Language (WSDL) in XML format. WSDL [1] is a major language that provides a model and an

XML format to describe the syntax about Web services. It acts as a vocabulary, associated with UDDI.

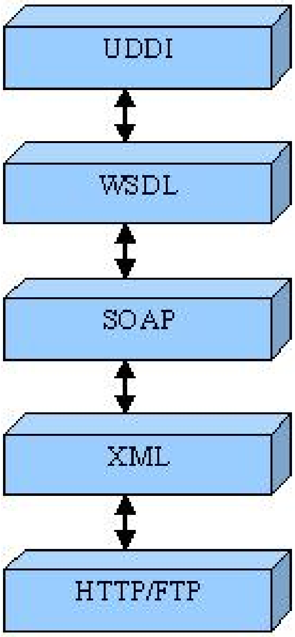


Figure 1.1: Web Service Structure

1.1.1 Web Services Selection Process

The Web Service description hides the implementation details, but at the same time, gives enough information that is necessary for the service interaction Figure 1.2 shows the web service design paradigm.

SOAP, UDDI and WSDL are used in different phases, called publishing, finding, and binding, in the Web Services development cycle. The Web Service Design paradigm is shown in Figure 1.2.The model begins with the publish phase, when an organization decides to offer a Web Service (1). The Web Service can be an existing application with a new Web Service front end, or it can be a totally new application. Once an enterprise has developed the application and made it available as a Web Service, the enterprise describes the interface to the application so that potential users interested in subscribing to it can understand how to access it. This description can be oral, in some human language such as English, or it

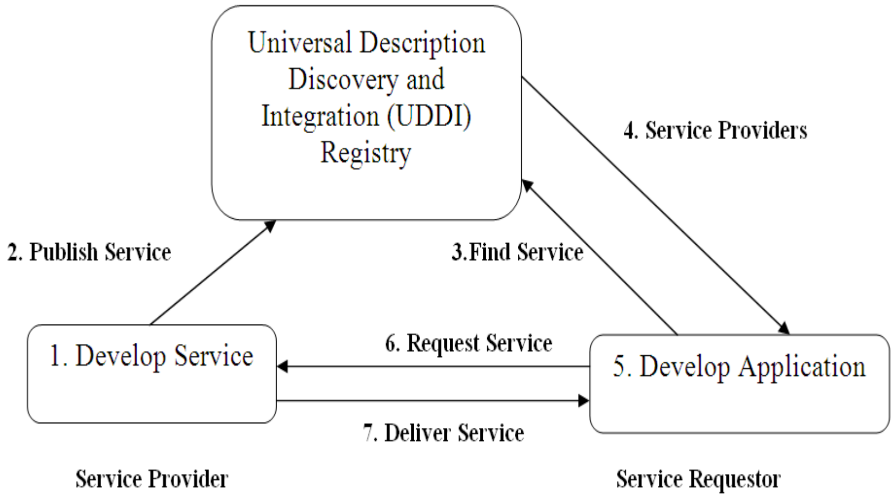


Figure 1.2: Web Service Design Paradigm

can be in a form, such as WSDL, that can be understood by Web Services development tools. To facilitate automated lookups, the service provider advertises the existence of the service by publishing it in a registry (2).

The next step of the model is the find phase. Once the service is advertised in a UDDI registry, potential subscribers can search for possible providers (3 and 4) and implement applications that utilize the service (5). Potential subscribers use the entries in the registry to learn about the company offering the service, the service being offered, and the interface to the service. The final phase of the model is the bind phase. When a subscriber decides to use a published service, it must implement the service interface, also called binding to the service, and negotiate with the service provider for the use of the service. When the application has been implemented and the business relationships resolved, the Web Service is utilized operationally. The only participants at this point are the service subscriber, who requests the service (6), and the service provider, who delivers the service (7). WSDL and UDDI [4] registries are generally only used during the initial discovery of the service and the design of the application.

Web Services can encapsulate a specific task or can be designed as a composition of other services, representing a complex aggregation. The Web Services conceptual model describes the process of discovery, request and response, as in figure1.3.Discovery is the process of finding the service that provides the functionality that is required. A request provides the input to the service. The response yields the output from the service. Service providers describe their Web Services and advertise them in a universal registry called UDDI. This enables service requestors to search the registry and find services. UDDI allows for the creation of registries that are accessible over the web.

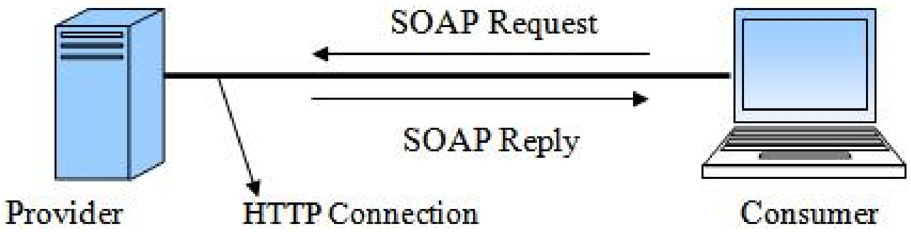


Figure 1.3: Web Service Invocation

IBM, Microsoft, and Oracle all have public UDDI servers running for commercial purposes. There are also many organizations developing third party servers for users to establish their private UDDI servers. The UDDI client offers two approaches to access the UDDI server: either through a standalone application providing an easy-to-use interface for developers; or through a software library working with the WS consumer or provider [5]. UDDI Browser is an open-source UDDI client following the first approach. A developer can use the application to browse, search, and even change information in the UDDI server. The WS consumer sends a SOAP request to the WS provider through an HTTP connection. The provider processes the request and returns a SOAP message as the reply to the consumer. Then, the provider closes the HTTP connection to finalize the invocation.

1.1.2 State of the art review

This section introduces Web service technologies and explains how to invoke, publish and provide Web services in a distributed environment. Section 2.7 also gives a short overview of trust-related projects and their significance for the development of behavior aware computer systems.

- **The Notion of the Web Service**

  In [3] the definition of a Web service is given as: "any process that can be integrated into external systems through valid XML documents over Internet protocols". This definition outlines the general idea Web services are built for Unlike services in general, Web services are based on specifications for data transfer, method invocation and publishing. This is often misunderstood and when a Web service is mentioned it sometimes refers to a general service provided on the Web, like the weather forecast on a Web page for example. The weather forecast is a service and provides its functionality for a variety of users but unless it comprises an interface to communicate with other applications via SOAP, it is not a Web service by definition. Web services can be seen as software components with an interface to communicate with other software components. They have certain functionality that is available through a special kind of Remote Procedure Call. In fact they even evolved from traditional Remote Procedure Calls. The difference lies in the interface and the method for transportation. Furthermore Web services cannot be viewed or used with an ordinary browser. They require a unified form of messaging embedded in a XML document. This communication architecture contains three subcomponents.

  - **Consumer:** This denotes the entity utilizing the Web service. This is another application in most cases.
  - **Transport:** It defines the means for the communication the Consumer uses while interacting with a service.
  - **Provider:** The service provider.

  In order to keep the whole system truly platform-independent, transport in both direction uses XML. This includes the description of an operation to execute and the data payload as well. Although transportation is not restricted to a specific protocol or method, HTTP became the most popular way to pass on XML documents between Web services. The following section will start with the first step in our course to understand Web services: Transportation.

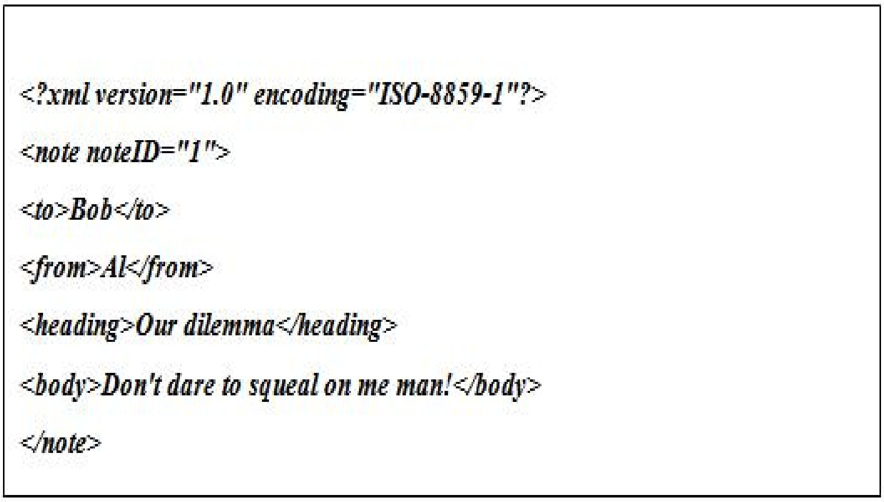
- **HTTP**

  Found everywhere on the Internet, HTTP (Hyper Text Transfer Protocol) is a ubiquitous protocol for data connections between Web browsers and servers. This protocol is the current standard for transferring HTML documents, although it is designed to be extensible to almost any document format like XML for example. HTTP Version 1.1 is documented in RFC 2068 [6]. It operates over TCP connections, usually to port 80, though any other port can be used. After a successful connection, the client transmits a request message to the server, which sends a reply message back. The simplest HTTP message is ”GET url”, to which the server replies by sending the named document. If the document doesn't exist, the server may send an HTML-encoded message stating this. This form of communication represents a typical request/response mechanism. A client sends a request for a specific document to the server and waits for a response. If the server does not respond with the requested document it is up to the client to wait for the timeout and request the same document again. This loosely coupled type of communication is very common in client-server architectures.

  In addition to GET requests, clients can also send HEAD and POST requests, of which POSTs are the most important. POSTs are used for HTML forms and other operations that require the client to transmit a block of data to the server. After sending the header and the blank line, the client transmits the data. This way Web services utilize the HTTP protocol to transmit both Data payload and service request to a Web service. Now it is time to explain how the transmitted data looks like.

- **XML**

  XML is an abbreviation for Extensible Markup Language [6] .It is designed to describe data and improve the functionality of the Web by providing more flexible and adaptable ways of information representation. It is called extensible because its format is not fixed like HTML. Instead, XML is a meta language which lets you design your own customized markup languages. A markup is a mechanism to specify structures within a document, whereas the way to add markup to a document is defined by the XML specification. But unlike HTML, XML does not specify semantics or a set of tags. There is no prescribed method for rendering XML documents, so semantics will be defined by the application using it or by style sheets. The following example will show the structure of an XML document and how data is represented:



```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note noteID="1">
<to>Bob</to>
<from>Al</from>
<heading>Our dilemma</heading>
<body>Don't dare to squeal on me man!</body>
</note>
```

Figure 1.4: XML Document and Data Representation

This basic XML document starts with the XML declaration in the first line. It defines the XML version and the used character encoding. In this case the document conforms to the 1.0 specification of XML and uses the ISO-8859-1 (Latin-1/West European) character set. It is important to specify the character set to avoid misinterpretation of the provided data.

The next line describes the root element of the document. Elements are one way to store data in an XML document. The following 4 lines describe the child elements of root (to, from, heading and body). By looking at the elements it is easy to see that the XML document represents a message. The last line finally describes the end of the root element, completing the note from Al to Bob. Along with the root element in the second line comes an attribute called noteID. Attributes are another way to store data and used to provide additional information about elements, also called meta-data. In this case it may be used to count the messages sent from Al to Bob. A list of legal elements that defines the document structure is the Document Type Definition (DTD). A document with correct XML syntax is called ”Well Formed” while a ”Valid” XML document also conforms to a DTD. More and more applications make use of XML to store information because of its benefits. Some of them are:

- The structure is well-defined and can be passed between different computer systems which would otherwise be unable to communicate.
- Data payload is encapsulated in tags and therefore readable by human viewers.
- Due to their textual nature, XML-Files are platform-independent.

These advantages made XML the perfect format to communicate between Web services. To ensure a platform and language independent use for every Web service, SOAP was developed. It is an XML application with defined elements and a redefined structure. The following section will treat SOAP in detail.

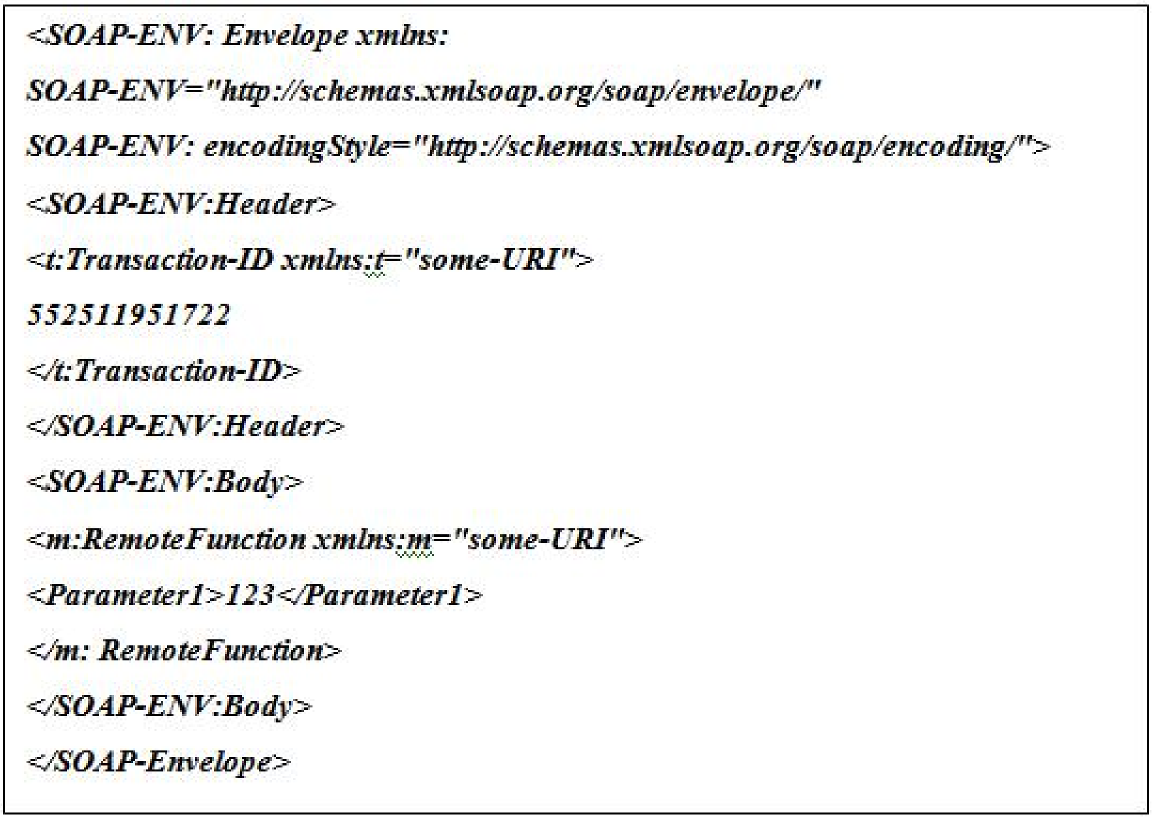
- **SOAP**

SOAP, the Simple Object Access Protocol was developed to enable a communication between Web services. It was designed as a lightweight protocol for exchange of information in a decentralized, distributed environment. SOAP is an extensible, text-based framework for enabling communication between diverse parties that have no prior knowledge of each other. This is the requirement a transport protocol for Web services has to fulfill. SOAP specifies a mechanism to perform remote procedure calls and therefore removes the requirement that two systems must run on the same platform or be written in the same programming language. SOAP also defines data encoding rules, called base level encodings or Section 5 encodings. It is important to note that these Section 5 encodings are not mandatory in any way, so clients and servers are free to use different conventions for encoding data as long as they agree on format. All this is done in the context of a standardized message format. The primary part of this message has a MIME type of text/xml and contains the SOAP envelope which is an XML document.

The envelope consists of a an optional header which may target the nodes that perform intermediate processing, and a mandatory body which is intended for the final recipient of the message. This way a firewall can be adjusted to filter SOAP Messages with an inappropriate header for example. The Header may also hold digital signatures for a request contained in the body. The body contains the serialized payload. For a request this is the method argument where the surrounding XML tag must have the same name as the called method. The response body contains the return value if it exists. Data types are not delineated in the SOAP envelope explicitly so the type of a result parameter cannot be discovered just by looking at the SOAP message.The anatomy of a SOAP Envelope is shown in fig 1.5.

In this example, the header contains some additional information enclosed by the Transaction-ID tag. This ID can be processed by any node before the final service node to ensure the request's correctness for example. The body contains but one method call in the request. The called method's name is RemoteFunction whereas the methods parameter Parameter1 is 123.

The parameters type may be of integer type but could be a String as well. The client application must decide how to handle it. SOAP messages are fundamentally one-way transmissions from a sender to a receiver, but they are often combined to implement a request/response mechanism. Summing up, SOAP is an XML-based protocol for sending messages and making re-



```
<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-ID xmlns:t="some-URI">
552511951722
</t:Transaction-ID>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
<m:RemoteFunction xmlns:m="some-URI">
<Parameter1>123</Parameter1>
</m: RemoteFunction>
</SOAP-ENV:Body>
</SOAP-Envelope>
```

Figure 1.5: Anatomy of SOAP Envelope

mote procedure calls in a distributed environment. Using SOAP, data can be serialized without regard to any transport protocol, although HTTP is typically the protocol of choice.

1.1.3 Publishing and Finding Web Services

With SOAP, a communication between Web services is possible and structured and each participant knows how to send or receive the corresponding SOAP Message. The final step to complete the communication architecture of Web services is to define how to access a service once it is implemented. This is where the Web Service Description Language (WSDL, [4]) steps in. WSDL describes services as collections of network endpoints, or ports. Again it is an XML document with a defined grammar where the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. WSDL documents use the following elements to describe a Web service:

- **Types:** A container for data type definitions
- **Message:** A definition of the data being passed in a single RPC.
- **Operation:** A description of an action (method) supported by the service.
- **Port Type:**A set of operations supported by one or more endpoints.
- **Binding:** A concrete data format specification for a particular port type.
- **Port:** A single endpoint defined as a combination of a binding and the network address where it can be found.
- **Service:** A collection of related endpoints.

Now that a Web service can be described completely, the only remaining problem is how a potential user can find the corresponding description (WSDL document). The following section deals with this last problem.

1.1.4 Using Web Services

To finally use a Web service, several steps have to be performed. Figure 1.6 shows the order of the events, followed by a description of how to execute each step.

1. **Locating the Web service:** This can either be done by browsing a public UDDI registry or by means of an existing WSDL document. It is possible to build a private UDDI registry as well. Private registries are easier to maintain due to their size but it can be hard to discover the UDDI registry's position. Sometimes, a company's main Web page is linked to WSDL documents, too.
2. **Creating the SOAP Message:** This is done by the development tool in most cases. Tools like Weblogic Workshop from BEA or Web service Development Kit from Microsoft will create valid SOAP messages for the methods described in the WSDL document or UDDI registry.

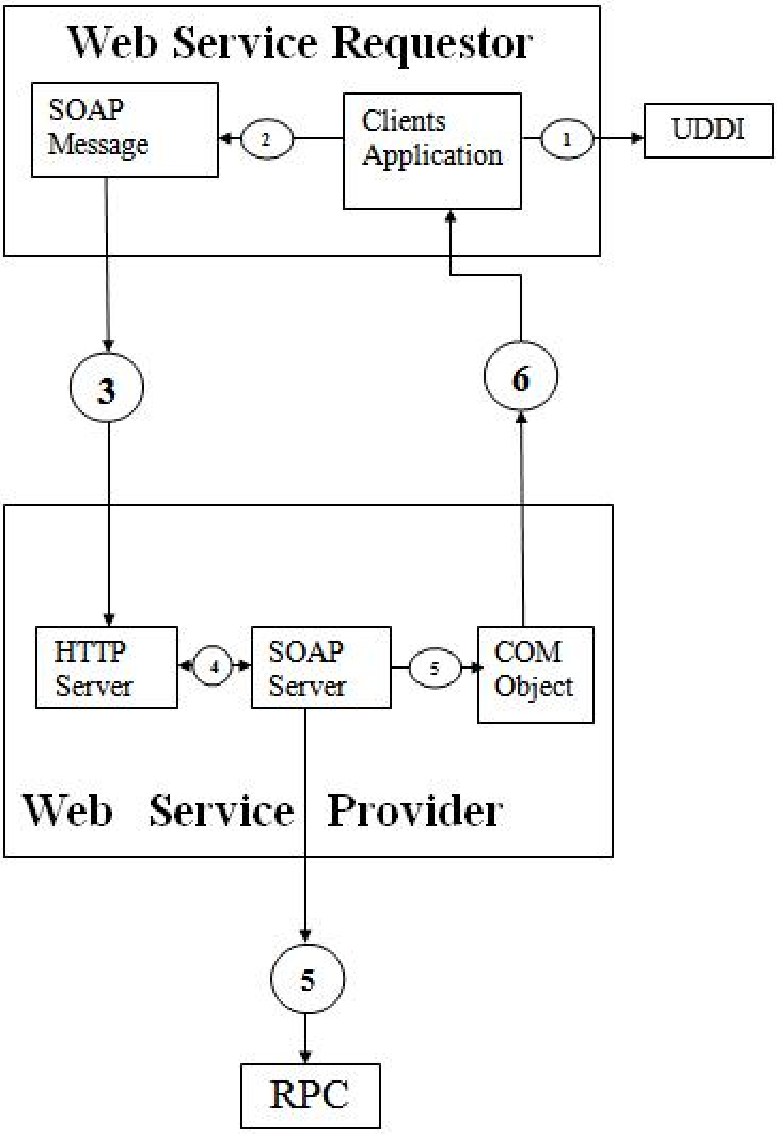


Figure 1.6: Invoking a Web Service

3. **Transmission:** Another advantage of message transport via HTTP is the service providers firewall setting. If the firewall permits Port 80 (HTTP POST/GET) connections, a SOAP message is able to pass through as well. If the firewall is unable to filter and process SOAP requests on the other hand, it leaves the system vulnerable to attackers who use the Web service's functionality for a potential attack.

4. **Parsing the SOAP message :** This is done by the provider's Application Server. The parser decides if the request is valid and decides which procedure

to call.

5. **Processing:** The service provider calls all necessary procedures, or even other Web services, to complete the requested task.

6. **Return the result:** The result is wrapped in a SOAP reply and returned to the requestor where the client application can parse the message and evaluate the included data.

1.2 Web Services Security

The security requirements of Web Services are similar to that of any other Internet based application. This report mainly deals with issues in providing basic security services, authentication, authorization, confidentiality, integrity and non-repudiation, to Web Services. Web Services enable the exchange of data and the remote invocation of application logic using XML messaging to move data through firewalls and between heterogeneous systems, after all the primary purpose of the Web Services is to enable many different applications to share data across a heterogeneous environment. Web Services are expressly targeted at distributed applications that cross-corporate boundaries, and consequently are likely to have challenging security requirements.

- **Authentication**

  To maintain a secure Web Service you need to know the identities of the parties who are establishing a Web Service connection. Authentication is a security requirement that ensures each entity involved in the usage of a Web Service- the requestor, the provider, and the broker (if there is one) are who they actually claim to be. Authentication is usually implemented using passwords, digital signatures etc. These authentication techniques work well for point-to-point authentication like client-server architecture. Web Service request and response need to go through many intermediaries before the request is processed. The design of Web Services requires support for end-to-end authentication. SSL does not support end-to-end authentication of chains of entities moreover SSL works at the Transport Layer to authenticate the incoming requests whereas Web Services require authentication at the Application layer. Let us consider the following example as shown in Figure 1, where a browser sends a request to the web site which forwards to a web service. The identity of the initiator or the sender which is the browser is not known to the web service as it sees only the web site which is an intermediary.

- **Confidentiality**

  Confidentiality in Web Services can be either Session-level or end-to-end. Session-level confidentiality ensures the consumer and provider that their communications cannot be overheard but Web service application topologies include all sorts of devices, PCs, proxies, demilitarized zones, gateways etc. Consequently, many intermediaries come between two communicating parties. SSL/TLS may secure the path between any two, but not from one end to the other.

  SSL can provide confidentiality between client and the website and between website and web service, but not between client and Web Service. The user credentials are encrypted and sent from browser to website in a secured channel. The web server then needs to decrypt the message and re-encrypt and send it to the web service. During this gap, the information could be inspected or modified. End to end confidentiality cannot be achieved using SSL. XML encryption facilitates encryption of SOAP message in part or as a whole to ensure end-to-end confidentiality. This technique is discussed in detail in next section.

- **Integrity**

  Integrity is the assurance that information can only be modified by authorized entities. Integrity can be achieved using hash functions and Message Authentication Codes (MAC). Session-level integrity can be provided by SSL whereas Web Services need end-to-end integrity. SSL may ensure integrity between any two points in the web service application topology but not from one end to the other. This is because at each intermediary point there is a possibility of message getting modified. End-to-end integrity can be provided using digital signatures. A digital signature is an application of XML signature, which is developed by W3C/IEFT XML Signature working group.

- **Non-Repudiation**

  Non-repudiation provides the capability to prove to a third party that a particular transaction occurred. Presently SSL is the most widely used protocol that provides non-repudiation in client-server architecture. Since Web Services involves passing of messages over a chain of entities where each entity both decrypts and re-encrypts the message using a protocol like SSL can provide non-repudiation between any two entities but not over the chain. XML encryption, XML Signature, and SAML are new technologies addressing this requirement. A detailed description of these technologies is provided later.

1.3 Problem Definition

Web Service Selection: The purpose of web service selection is to select optimal web service for a particular task.

When dynamic discovery is used in Web Services, it is common that the result of the discovery contains more than one provider. Unlike the file sharing P2P system in which a file download can be split into many small tasks running in multiple peers, a service invocation occurs between a provider and a consumer. As shown in Figure 1.7, the WS consumer must pick only one from all candidate providers to perform the invocation. Even for a composite Web Service consisting of many atomic Web Services, the selection issue still needs to be addressed when there are multiple providers available for an atomic service.

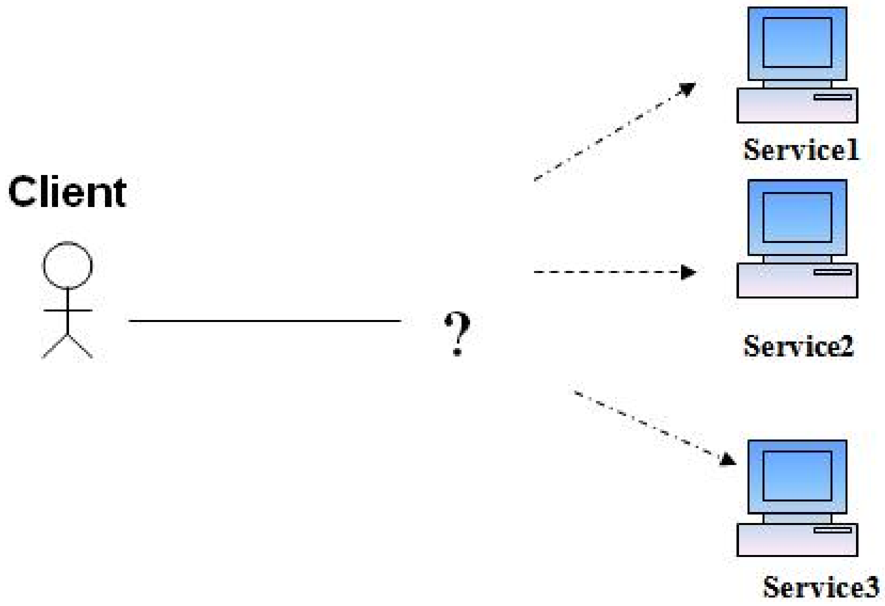


Figure 1.7: Client to select from Similar Services

In order to make a distinction between the services which provide the same functionality, selection criteria should be used. They help evaluate the Web Services within a group and choose the component that matches the needs and the preferences of the consumers, while taking into account the abilities of the providers.

Web Services can be ranked by the Quality of Service (QoS) they offer. QoS is a means to enable selection and filter out unqualified providers. QoS can be seen as an aggregated measure of generic criteria such as availability, reliability , failure rate, trust and reputation, response time, price, and network load and domain specific features .The reasoning mechanism is responsible for the selection of a Web Service at a particular moment of time. In order to distinguish one service from another using the specified criteria, this unit requires a set of instructions that help evaluate each component and choose the most appropriate one respectively. A set of instructions can be seen as a selection technique.XML Web services enable the exchange of data and the remote invocation of application logic using XML messaging to move data through firewalls and between heterogeneous systems.

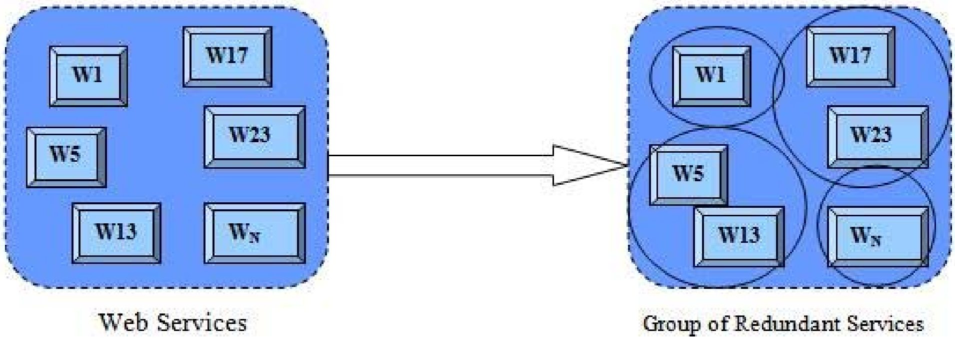


Figure 1.8: Redundant Web Services

Although remote access of data and application logic is not a new concept, but doing so in a loosely coupled fashion is. Hence it poses new challenges. In Web Services, the interface hides the implementation details of the service, allowing it to be used independently of the hardware and software platform on which it is implemented and also of the programming language in which it is written. This allows and encourages Web Services based applications to be loosely coupled, component oriented, cross technology implementations.

Security is the single biggest concern to deploy Web Services, SSL provides good point to point security, but fails to provide end - to - end security, which is needed to provide security for the transaction in the Web Services. The purpose of this work is to develop an approach for dynamic and transparent service selection and to evaluate the proposed architecture in terms of what selection techniques should be applied.

This work also aims at describing the concerns related to Web Service security and analyzing a few tools and techniques, which could be used to secure Web Services. The remainder of the thesis is organized as follows: Chapter 2 is an overview of the literature survey. Chapter 3 is the proposed approach and the evaluation of the model is discussed. The comparison between the various technologies and protocols, which could be used to address security concerns of Web Services are presented in chapter 4 and the implementation of encryption of SOAP messages are also discussed in chapter 4. Chapter 5 presents the conclusions of the thesis and some potential future directions.

Chapter 2

Literature Survey

Chapter 2

Literature Survey

2.1 Introduction

Researchers have proposed various approaches for dynamic web service selection. Maximilien and Singh [7] propose a multi-agent based architecture to select the best service according to the consumers' preferences. Maximilien and Singh describe a system in which proxy agents gather information on services, and also interact with other proxy agents to maximize their information and the conceptual model they use to interact with the services is detailed elsewhere [8]. The proxy agents lie between the service consumer and the service providers. The agents contact a service broker, which contains information about all known services, as well as ratings about its observed QoS. From there, the information is combined with its own historical usage, and the combined knowledge is used to select a service, though the authors do not detail how. The agencies contain data about the interactions between the clients and the services which is used during the Web Services selection process.

In his work, trust and reputation are taken into account during the decision process. Their approach divide the QoS attributes into objective and subjective. The former include QoS features such as availability, reliability, and response time. Their approach is shown in figure 2.1. Liu, Ngu, and Zeng [9] consider these features in their proposed approach as well but their major selection criteria is based on the QoS based service selection. They have considered three quality criteria namely execution time, execution duration and reputation for the selection.

In addition, execution price, duration, transactions support, compensation and penalty rate are the other criteria. The authors of [9] suggest an open, fair, and dynamic framework that evaluates the QoS of the available Web Services by using clients' feedback and monitoring.

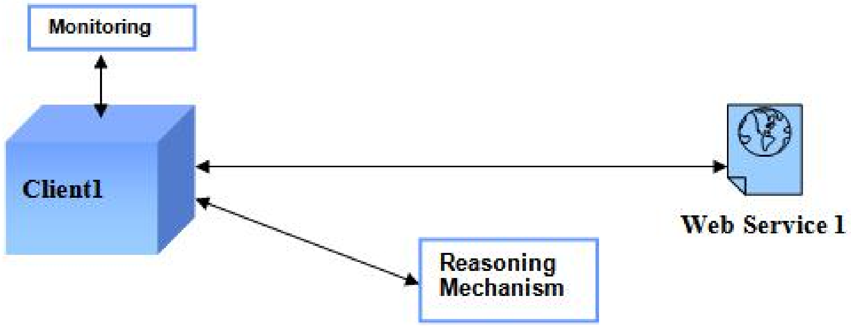


Figure 2.1: Model proposed by Liu, Ngu, and Zeng

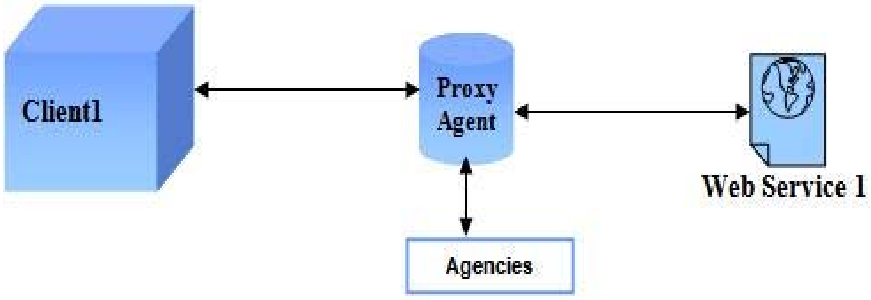


Figure 2.2: Model proposed by Maximilien and Singh

The reasoning mechanism is responsible for the selection of a Web Service at a particular moment of time. In order to distinguish one service from another using the specified criteria, this unit requires a set of instructions that help evaluate each component and choose the most appropriate one respectively. A set of instructions can be seen as a selection technique. The major components of a reasoning mechanism are criteria, model, and selection technique. The model collects information about the participants of the client-server interaction as well as represents

	Run-Time Selection	**Reasoning Mechanism**	**Transparency during selection**	**Clients Involved in Web Service Selection**	**Clients Feedback**
Maximilien and Singh	**Yes**	**Agents**	**Yes**	**No**	**No**
Liu, Ngu, and Zeng	**Yes**	**QoS computation** **QoS Ranking**	**No**	**No**	**Yes**

Table 2.1: Comparison of Approaches for Web Service Selection.

it as aggregated measures. Different selection techniques can implement various business logics in order to make a decision.

The reasoning mechanism in the approach proposed by Liu, Ngu, and Zeng [9] computes the QoS of the Web Services, ranks them, and selects the most appropriate one. To perform the selection, the QoS registry in their system takes in data collected from the clients, stores it in a matrix of web service data in which each row represents a web service and each column a QoS parameter [10] , and then performs a number of computations on the data, such as normalization. Clients can then access the registry, and are given a service based on the parameters that the client prefers. The bottleneck of the approach is the dependency on the consumers to give regular feedback about their past experience with the Web Services. An overview of their approach is shown in figure 2.2.The Success of this model is based on the clients or the end users and their will to provide the necessary feedback on QoS.

2.2 Web Services Security Protocols

- **XML Encryption**

  Encryption provides confidentiality. It does this by preventing the data from being understood except by the intended recipient. The XML Encryption

Syntax and processing standard defines a process for encrypting digital data and how the resulting encrypted data should be represented in XML. The smallest unit of information that can be encrypted is an element [11].

- **To encrypt XML elements:**

  1. Select the encryption algorithm and parameters.
  2. Obtain the key. If the key is going to be identified, construct a KeyInfo element. Encrypt the key, if it will be sent with the encrypted data, and construct an EncryptedKey element. Place it in KeyInfo or in some other portion of the document.
  3. Encrypt the data. For XML data, this can involve a transformation to UTF-8 encoding and serialization. The result is an octet string.
  4. Build the EncryptedType structure. Where the encrypted data is actually stored in the structure, instead of being referenced, the encrypted data must be base64 encoded.
  5. Replace the unencrypted element in the XML document with the EncryptedType structure .

- **To decrypt XML elements:**

  1. Process the element. Unspecified parameters must be supplied by the application.
  2. Obtain the decryption key. This may require using a private key to decrypt a symmetric key or to retrieve the key from a local store.
  3. Decrypt the data in CipherData.
  4. Process the decrypted data. This requires that the application restore the decrypted data, which is in UTF-8, to its original form. It must be able to replace the CipherData structure in the XML document with the results of the decryption. In some cases, additional processing is required.

- **Issues**

  The primary issues with using XML Encryption for Web Services are: **Out-of-band agreements between the sender and the receiver:** XML Encryption is very flexible and allows many parameters to be omitted from the CipherData structure. For instance, KeyInfo is optional. For the most part, we consider this flexibility a positive feature. If the data is decrypted immediately and does not have to persist, this is not a problem. However, if the encrypted data must be stored to protect confidentiality or if signatures have been applied to encrypted data and it is important to preserve a record of the signatures, leaving information out of the structure can lead to decryption problems at a later time. In general, including the encryption parameters in the structure is preferable. **Choice of algorithms and key lengths:**XML Encryption does not mandate the use of particular algorithms or key lengths. It is the user's responsibility to ensure that the right choices are made. The system implementer should carefully consider how long the encrypted data must be retained, how much use the keys will have, and the preferred algorithm, and then decide on the appropriate key length. **Application in SOAP:** XML Encryption specifies encryption for XML documents. It does not describe how XML Encryption data and structures are implemented within the SOAP message structure.

- **XML Signature**

  The XML Signature recommendation [11] defines how digital data is signed and how the resulting signature should be represented in XML. While the data to be signed is intended to be more general than XML, XML data is the principal application for XML Signature. With XML Signature, all or selected portions of an XML document can be signed.

  The recommendation defines the process for creating and representing an XML signature and then verifying the signature. It relies on existing algorithms for the signature, message digests, and message authentication codes.

It offers several established alternatives for certificates, including X.509. It can also be used without certificates. This represents a departure from established thinking about public key cryptosystems, but it can be justified under certain circumstances. The recommendation references other standards for transformation such as canonicalization, rendering the data in a standard way that eliminates inconsequential differences in representation, and encoding/ decoding.

Digital signatures are much more complex to implement than encryption. Because signatures are tied to the representation of the data being signed, caution must be exercised to ensure that the representation of the signed data and the verified data are consistent. Signature processing is much more subtle than encryption and is very sensitive to changes in data representation and processing order. Even if the signature was valid at the time it was created, it may not be verifiable because of changes that occurred as the message was routed.

Format/Structure An XML signature consists of two required elements, SignedInfo and SignatureValue. There are also two optional elements, KeyInfo and Object. SignedInfo. This includes the CanonicalizationMethod, which is discussed in the next section, for the SignedInfo element itself, the algorithms (usually a digest algorithm and a signature algorithm) used to produce the signature, and one or more references to the data being signed. Each Reference element includes a URI identifying the data being signed, the transforms that process the data (we will describe some transforms in the next section), an identifier of the digest algorithm used with the referenced data, and the value of the message digest.

SignatureValue: This is the value of the digital signature. It is base64 encoded.

KeyInfo: This provides the information needed by the receiving application to validate the signature. If it is omitted, the receiving application is expected to know how to validate the signature. For instance, two business partners may have previously exchanged public keys through some other means, thereby eliminating the need to include the public key as a child element of KeyInfo. If this hasn't been done, KeyInfo can contain a key identifier, the signer's public key, a reference to where the public key is available, or the signer's public key certificate. Several public key certificate formats are supported.

Object: This is a structure that carries any other information needed to support the signature.

- **WS Security Tokens**

  The WS-Security specification specifies an abstract message security model in terms of security tokens [12] combined with digital signatures as proof of possession of the security token referred to as a key. Security tokens assert claims, and signatures provide a mechanism for authenticating the sender's knowledge of the key. This signature can also be used to bind with the claims in the security token. This assumes that the token is trusted. It may be interesting to note that we do not specify a particular method for authentication. The specification only indicates that security tokens may be bound to messages. This is where the power and extensibility of WS-Security lies.

- **HTTP Authentication**

  HTTP basic authentication is orthogonal to the security support provided by WS-Security or HTTP Secure Sockets Layer (SSL) configuration. A simple way to provide authentication data for the service client is to authenticate to the protected service endpoint using HTTP basic authentication. The basic authentication is encoded in the HTTP request that carries the SOAP message. When the application server receives the HTTP request, the user name and password are retrieved and verified using the authentication mechanism specific to the server. Although the basic authentication data is send over HTTP, which is recommended. The integrity and confidentiality of the data can be protected by the SSL protocol.

2.3 Signature Creation/Verification Process

To create a digital signature:

1. Apply the transform or transforms to the data object to be signed. Transforms are applied in the order they are specified.

2. Calculate the message digest of the output of the transforms.

3. Create a reference element that includes the URI of the data object (optional),the transforms used, the digest algorithm, and the digest value. As many reference elements as needed may be created. This occurs if one signature covers several nodes of the document.

4. Create the SignedInfo element. Include the SignatureMethod, the CanonicalizationMethod, and the references previously generated.

5. Apply this method to SignedInfo.

6. Use the algorithms specified by SignatureMethod to create the signature. This usually means applying a message digest algorithm to the canonicalized SignedInfo and then signing the resulting digest.

7. Create the Signature element that contains the SignedInfo, the SignatureValue,KeyInfo (if needed), and Object (if needed).

8. Note that a different canonicalization algorithm or message digest algorithm can be applied to each referenced element.

To verify a signature:

1. Process the SignedInfo element according to the SignatureMethod specified in SignedInfo.

2. For each reference element, obtain the data object referenced.

3. Process each data object according to the specified transforms.

4. Digest the result according to the digest algorithm specified for the referenced element. Compare the result with the value stored in the corresponding reference element. If the two are not equal, the verification fails.

5. Obtain the necessary keying information. It may be available in KeyInfo, or it may have been preplaced.

6. Apply the signature method using the previously obtained key to confirm the SignatureValue over the canonicalized SignedInfo element.

Issues: There are several topics for consideration when implementing a digital signature system for Web Services.

Signature syntax vs. semantics: The XML Signature Recommendation deals with the syntax and technical process for creating an XML digital signature. Signatures have a meaning from a legal and business point of view. It is important to consider what need the signature meets and then ensure that the signature is being applied to the appropriate parts of the document to satisfy the need.

Out-of-band agreements between the signer and the verifier: The XML Signature Recommendation is very flexible and allows many parameters to be omitted from the signature. For instance, KeyInfo is optional. For the most part, we consider this flexibility a positive feature. However, leaving information out of the signature means that there can be problems with signature verification at a later time or if the verifier changes. In general, including signature parameters in the signature element is preferable.

Choice of algorithms and key lengths: XML Signature does not mandate the use of particular algorithms or key lengths. It is the user's responsibility to ensure that the right choices are made. The system implementer should carefully consider how long the signature must be retained and the preferred algorithm, and then decide on the appropriate key length.

	Message Alteration	**Loss of Confidentiality**	**Spoofing**	**Man in the middle**	**Replay of Message**
XML Encryption		**X**		**X**	**X**
XML Signature	**X**				**X**
WS-Security Tokens			**X**		
HTTP Authentication			**X**		

Table 2.2: Threats addressed by Web Services Standards

Application in SOAP: XML Signature specifies encryption for XML documents. It does not describe how XML Signature data and structures are implemented within the SOAP message structure.

Comparison: The table 2.2 shows the comparison of various Web Services Security protocols with respect to various threats addressed by Web Services standards.

2.4 Conclusion

In this chapter, we have seen the approaches for Web Services Selection and made a comparison between the existing models. We have also seen the various cryptographic techniques used to protect XML and SOAP messages. We have also viewed the process of encrypting and decrypting XML documents and how to sign and verify signatures on XML data. We also went over supporting techniques such as XPath, and Canonicalization, and security-related issues that must be addressed when using these techniques.

We have also seen the various Web Services security protocols and their applications in SOAP. We have addressed to issues like key length of the message and Out-of-band agreements between the signer and the verifier. Along with these the signing and verifying process has been discussed A comparison between the various Web Services Security protocols with respect to various threats addressed by Web Services standards.

Chapter 3

Proposed Approach: Web Services Selection

Chapter 3

Proposed Approach: Web Services Selection

3.1 Introduction

In the domain of Web Services, it is not uncommon to find redundant services that provide functionalities to the clients. According to the Web Services conceptual model (discussed in chapter 1), the client receives a list of services from the UDDI, selects one, and starts an interaction with the service to process the request .As seen in chapter 2, service selection is an important process and various techniques have been proposed. In this chapter, another approach for dynamic service selection and invocation is introduced, which has the following advantages in comparison with previous approaches:

- It provides location and replication transparency of the Web Services;
- It hides the system's complexity from the clients;
- It provides a transparent service selection from the client's point of view;
- It assures a level of security, since the clients do not have direct access to the Web Services.

The development of the proposed model in a real-world application and the evaluation of such a system is a complex procedure. It requires resources in terms of machines that run a set of experiments and time that should be devoted to each experiment setup, run, and analysis. In addition, it is very likely that the results of the experiments depend on the particular machine specifications and environment settings, such as web server, communication style (for example, Tomcat and Axis framework, if the system is implemented in Java). Fluctuations, due to network, memory, CPU, caching, and garbage collection, might appear as well. All this could reflect on the correct analysis of the obtained data and the validity of the conclusions. Furthermore, such an implementation is based on particular standards, protocols, and programming languages.

3.2 Repository Based Web Services Selection (RB-WSS)

We propose a technique for dynamic selection of Web Services which will also handle the problem of redundant Web Services. In this work, we introduce a model with a Web Service repository, as shown in figure 3.1 will act as an independent unit possessing a definite functionality. This repository will be used to redirect the client's request. This will also provide a level of security since it will not be allowed to invoke directly by the clients. This technique will prevent unauthorized access to the real services. This provision will also help to hide the systems complexity from the clients.

The repository will perform three functions namely, storing, collecting and reasoning. In storing operation a QoS feedback report is generated by the client and is saved in the repository. The QoS feedback report provides a historical reference for the consumer to assess the provider. Each provider only keeps the feedback information relevant to it. The collecting operation retrieves all necessary data from providers for the reasoning operation. The reasoning operation manages to select the best service provider for the consumer according to the collected data. Consider an example where clients needs the services (S1,S2) as in fig.3.1, it sends a request .The collecting, storing and reasoning mechanism interacts with the web services to find the most appropriate of the services and the results are stored in the repository for future reference. Web Services here interacts with the reasoning mechanism to find out the appropriate services. Once the service is selected, the

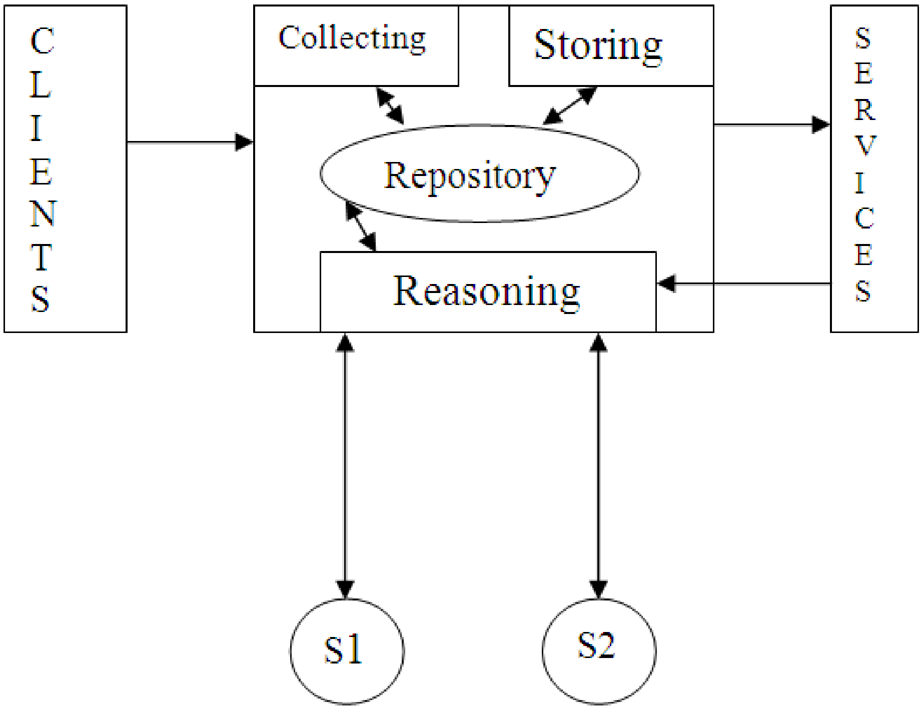


Figure 3.1: Repository based Web Service Selection

request is forwarded to it. Finally, when the result is generated, it is passed to the repository which sends it back to the client.

3.2.1 Algorithm: Selection of Service

This algorithm shows the necessary steps to choose a service and get the maximum quality results.

1. For finding a service for a specified task, perform a search on service descriptions.
2. Arrange all discovered services by their signature parameter and discard all other services.
3. Get the desired Service Parameters.
4. Collect the services result and order by their utility.
5. If no results are found, let the client reconsider the constraints, go to step 2.

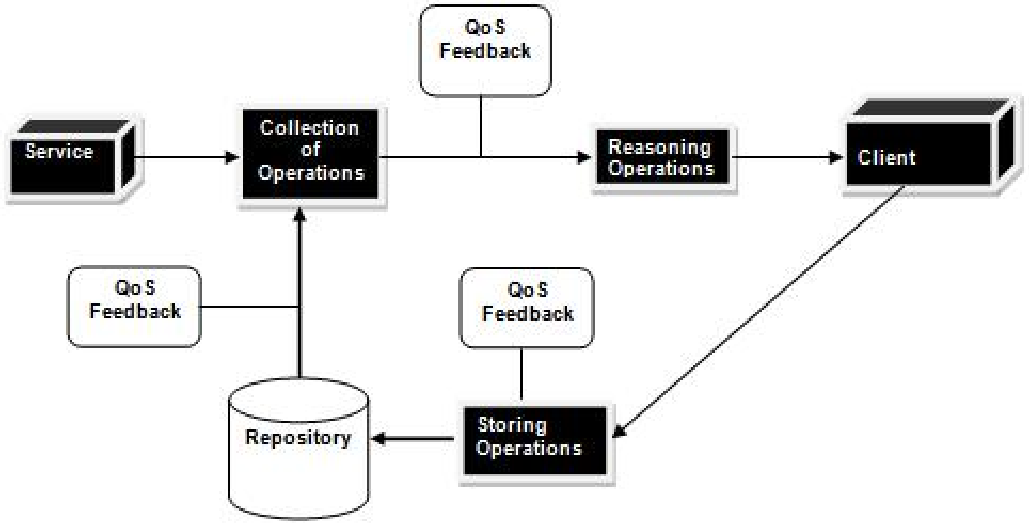


Figure 3.2: Selection Process

This algorithm provides an approach for selecting a specified service. Services, not matching the profile are discarded on the fly .It also helps in taking an alternative services.

3.2.2 Design of the Proposed Architecture

This section provides an overview of the design of the simulation model that is used to evaluate the RBWSS(Repository Based Web Services Selection) approach. The architecture, proposed here consists of three main components - Clients, Repository Layer, and Web Services, forming the key compound objects of the high-level view of the simulation design. Object LG (load generator) generates clients' requests (entities); object represents the behavior of the model, and each object WS corresponds to a Web Service. Object LS (load sink) is used as the end point of the entity flow. It accepts the incoming from the RBWSS entities and disposes of them.

The simulation design of the model is presented in figure 3.4. Component Manager is not included, since the settings of the reasoning mechanism are defined before the simulation runs. There is a Virtual Web Service corresponding to each group of redundant services with particular functionality. The reasoning mechanism is presented by a queue and a server.

3.2.3 AnyLogic Enterprise Library

AnyLogic provides libraries for simulating systems in various domains. The enterprise Library can be applied in discrete systems, such as manufacturing, services, business processes, etc. [13]. It is able to "create flexible models, collect basic and advanced statistics, and effectively visualize the process", in order to represent the system. At the same time, it "provides a higher-level interface for fast creation of discrete event models in the style of flowcharts", using objects like source, sink, queue, and delay server, and others, in a drag-and-drop manner [14] .Entity is a basic concept in the Enterprise Library. The entities represent individual units that are evaluated in the simulation. They can enter and leave objects through one directional port. The connections between the ports are established by connectors.

3.2.4 Evaluation

This section describes the evaluation of the proposed Repository Based Web Service Selection (RBWSS) approach. Firstly, a feasibility check is done to determine whether it is possible to build such architecture. Secondly, the behavior of the Web Services is observed and analyzed in different environments. Finally, to observe the behavior of the prototype with different selection techniques, the simulation method is used, since the study of the optimal strategies is difficult in a real environment where there are many uncontrollable parameters.

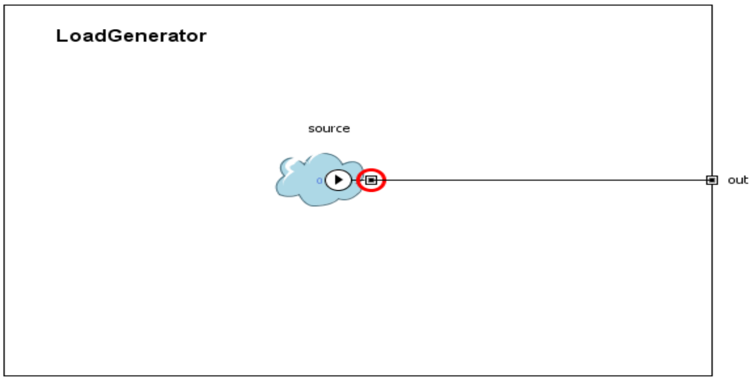


Figure 3.3: Load Generator

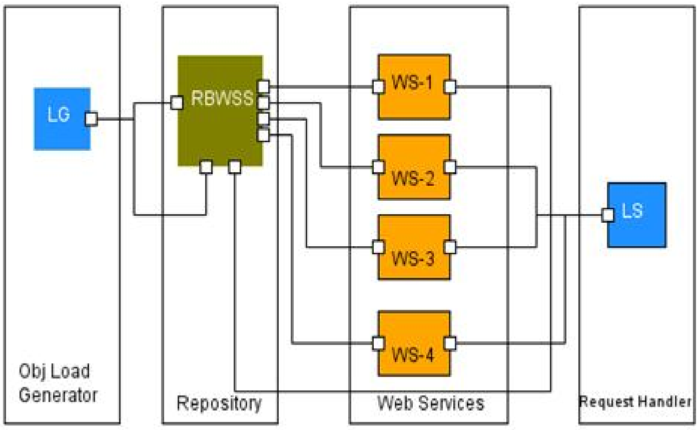


Figure 3.4: Selection Process

The evaluation process carried out by analyzing the RBWSS prototype in order to test whether or not the architecture is a feasible technique for managing redundant Web Services on the server side in a dynamic and transparent manner. A simplified prototype of the proposed architecture is developed, in order to observe if the model is an applicable approach for dynamic selection of redundant Web Services. The dynamic service selection is realized by reflection that allows us to obtain information at run-time about methods, constructors, and instance fields of classes, as well as to invoke them dynamically [15], [16].

The reasoning mechanism is based on a random selection technique that does not require any information about the services. Since the decision is done in a random manner, there is no need for selection criteria, neither the data about the services must be collected and aggregated by the model. The model of the reasoning mechanism contains only the WSDL descriptions of the redundant Web Services.

3.3 Results

The results of the developed framework are as follows:

- The model has a feasible technique for dynamic service selection on the server side. The layer is able to manage the redundant services in a transparent manner. All the necessary information, which should be available to the

		Processed Requests by the Web Services			
Total Request Processed	**Execution Time (in Milliseconds)**	**WS1**	**WS2**	**WS3**	**WS4**
60	72.5	15	15	15	15
120	144.5	30	30	30	30
60	100.5	10	18	15	17
120	158.5	20	36	30	34

Table 3.1: Simulation results

client, is the service description (WSDL) of the Web Services. The model hides the reasoning details during the decision-making process. From the consumer's point of view, the prototype is the real Web Service that handles the request.

- The scalability of the system that implements the described layer is expected to be the same as the scalability of a system that does not consist of redundant components. The decentralization of the Web Services assures that there is no single point of control and respectively of failure. There is a separate component that represents and manages each group of services and does not influence the proper work of the whole system.

- The architecture can be used as a layer that assures a level of security. The Web Services are called by the virtual layer and are never invoked directly by the clients. This technique can prevent unauthorized users from having access to the real services.

- It is possible for the response time of the proposed architecture to increase due to the reasoning mechanism. This is an expected result since the decision is taken at run-time. Furthermore, it implies a trade-off between the appropriate service selection and the execution time of the system.

The obtained data shows that the execution times of the simulation runs are higher for the load balancing technique compared to the fastest service selection but lower compared to the random selection technique. In terms of Web Services overloading, both the fastest and the load balancing techniques present similar results.

3.4 Conclusion

We propose an approach for dynamic service selection and, which has the following advantages in comparison with previous approaches:

- It hides the system's complexity from the clients.
- It provides a transparent service selection from the client's point of view.
- It assures a level of security, since the clients do not have direct access to the Web Services.

In future, other technology that can be applied in the Repository based system is Semantic Web technology. By describing the data in a machine-understandable manner and creating semantics of QoS criteria, the decision-making process would be based on more features as well as their relationships would be represented in a better and more flexible way. This chapter also describes the simulation environment AnyLogic and presents the design of the simulation that corresponds to the proposed Virtual Web Services Layer architecture. The clients are represented by object Load Generator; the behavior of the model is simulated by object RBWSS; and the Web Services are represented by objects WS.

Chapter 4

SOAP Level Security: Implementation and Analysis

Setting up the session

Analysis of SOAP session

Web Services Development Kit (WSDK)

Performance Analysis

Conclusion

Chapter 4

SOAP Level Security: Implementation and Analysis

The preliminary security papers by Microsoft and IBM [17] proposes various architectures for deploying a security service or token service which provides the necessary tokens in the network of Web Services to manage access and authentication. In the architecture shown in figure 4.1.Each web service in the system have a public and a private key. Of course, the private key is kept secret. The public key of each web service is stored in the Security web service. Also, every web service that provides a service to others has a list of the public keys of the web services that have access to it. Whenever the list of a web service in the Security web service is updated by the administrator, the SWS contacts the web service by establishing a session with it and updates its list. This is done by encoding the latest list inside the body of the SOAP envelope.

The web services that make requests to other web services do not contact the Security Web Service (SWS). This architecture has been chosen for scalability and availability reasons: when a lot of web services in the network need to communicate with each other, the SWS may be coded with requests for security tokens. In this architecture, the web services only need to be updated when something changes in the setup of the web services or in the way they communicate with each other.

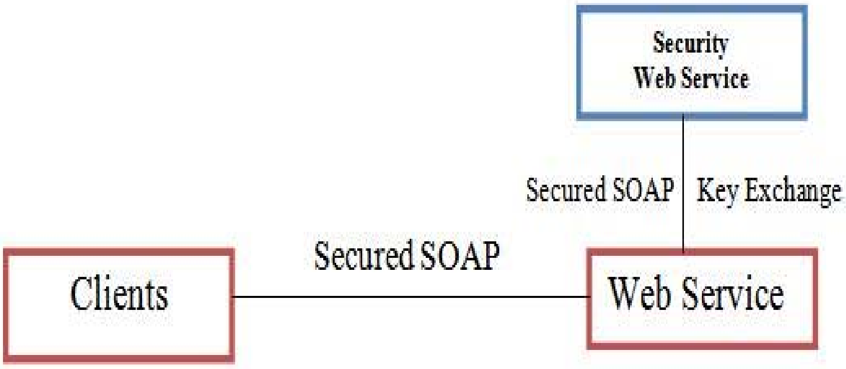


Figure 4.1: Architecture of Web Service Setup

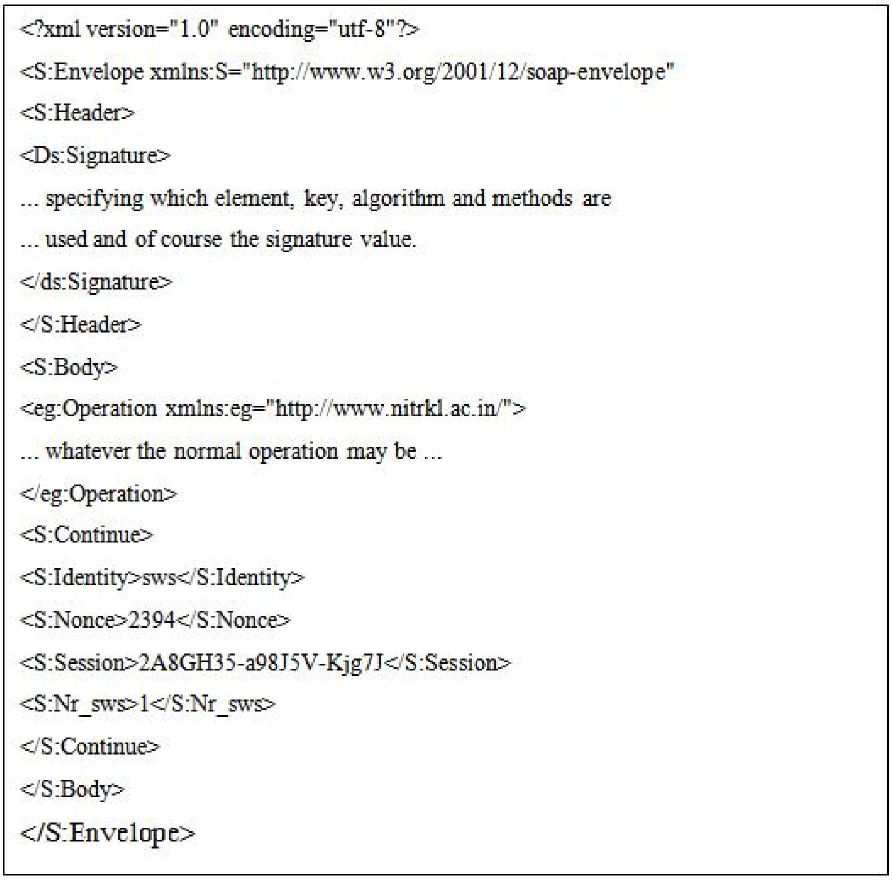
4.1 Setting Up The Session

The latest version of SOAP [7] version 1.2, the working draft from the W3C, does not have a notion of a session context, or a security context for that matter. This means that each and every SOAP message must have an elaborate < Security > element to address all the security issues at the message level. The problem of making SOAP messaging confidential between two computer systems can be addressed by Transport Level Security (TLS) for example Secure Sockets Layer (SSL). SSL appends Message Authentication Codes (MAC's) to the transmitted messages to ensure message integrity, which would preserve the integrity of the SOAP messages transmitted with SSL as well [18].

However, when various intermediaries or adversaries have to read the SOAP message [19] to determine who's next in the communication chain, message integrity and confidentiality must be preserved. SSL does not provide for non-repudiation to start with. You only have hop-to-hop security when you are using SSL to encrypt communications instead of end-to-end security; because it is possible that security has been breached on one of the web services along the way. Also, the idea behind SOAP messaging is to provide loosely coupled systems with a way that they can communicate with each other in a connectionless way.

For example, there could be a message queue that a receiver has to process first. This way the sender of a message must wait for the receiver to finish the queue; here SSL is not a good solution to this problem.

The following code here elaborates the how the response message is sent for setting up a session.



```
<?xml version="1.0" encoding="utf-8"?>
<S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
<S:Header>
<Ds:Signature>
... specifying which element, key, algorithm and methods are
... used and of course the signature value.
</ds:Signature>
</S:Header>
<S:Body>
<eg:Operation xmlns:eg="http://www.nitrkl.ac.in/">
... whatever the normal operation may be ...
</eg:Operation>
<S:Continue>
<S:Identity>sws</S:Identity>
<S:Nonce>2394</S:Nonce>
<S:Session>2A8GH35-a98J5V-Kjg7J</S:Session>
<S:Nr_sws>1</S:Nr_sws>
</S:Continue>
</S:Body>
</S:Envelope>
```

Figure 4.2: Setting up the Session

Now it is clear how the SOAP messages are constructed, the entire session is a sequence of three SOAP messages sent back and forth. The purpose of this session is to ensure that the two web services are indeed communicating with each other and that no adversary can compromise security. The following diagram is a schematic representation of the SOAP messages that are exchanged between web services and it illustrate which SOAP messages are sent and in which order. The session protocol initiates the session as shown in figure 4.2.between Web Services α and β . After a complete run of this protocol, both and agree upon the session id and they know how many messages each web service has sent (now and in the future). Using this session setup they can call each other's operations and exchange information securely, meaning no adversary can come in between them, at least in theory. In practice there might always be implementation errors or other factors involved that can compromise security. Any web service can end a session by sending a SOAP envelope back in which a < SessionEnd > element is the last element of the < Continue > element.

If the Serving Web Service (sws) agreed upon the security methods, it generates a session key to be used with a symmetric cipher (for example 3DES or AES) and encrypts this session key using the public key of the Client Web Service (cws) . This encrypted key is added to the subelement <Encryption > of the < Continue > element, inside the < EncryptedKey >element which is specified in [20] . The first element of the < Continue > element is the identity of the sws. The second element is the < Nonce > element as explained before, followed by the < Session > element and the < N_r > elements. The < Encryption > element contains the session key which the sws used to encrypt the body of the SOAP envelope. The session key is put in a < EncryptedKey > element with an "Id" attribute linking the session key to the session.

This works as follows: the cws first sends a SOAP message with a digital signature. The < Continue > element contains a < Nonce > element and empty < Session > and < N_r > elements. In this case another element, the < Encryption > element, is added that indicates it wants to communicate with encrypted content and to establish a session key. Upon receiving this SOAP message, the serving web service (sws) determines if it agrees with the way communication will take place. It determines if it agrees using the same canonicalization method, the signature method, the transform algorithm and the digest method. If it does, it remembers these methods and adopts them. If the sws do not agree on these methods, it sends a SOAP message back using its own preferences. For simplicity reasons, the cws must then conform to the preferences set by the sws, otherwise the session is aborted. The session key is encoded and sent back to the cws.

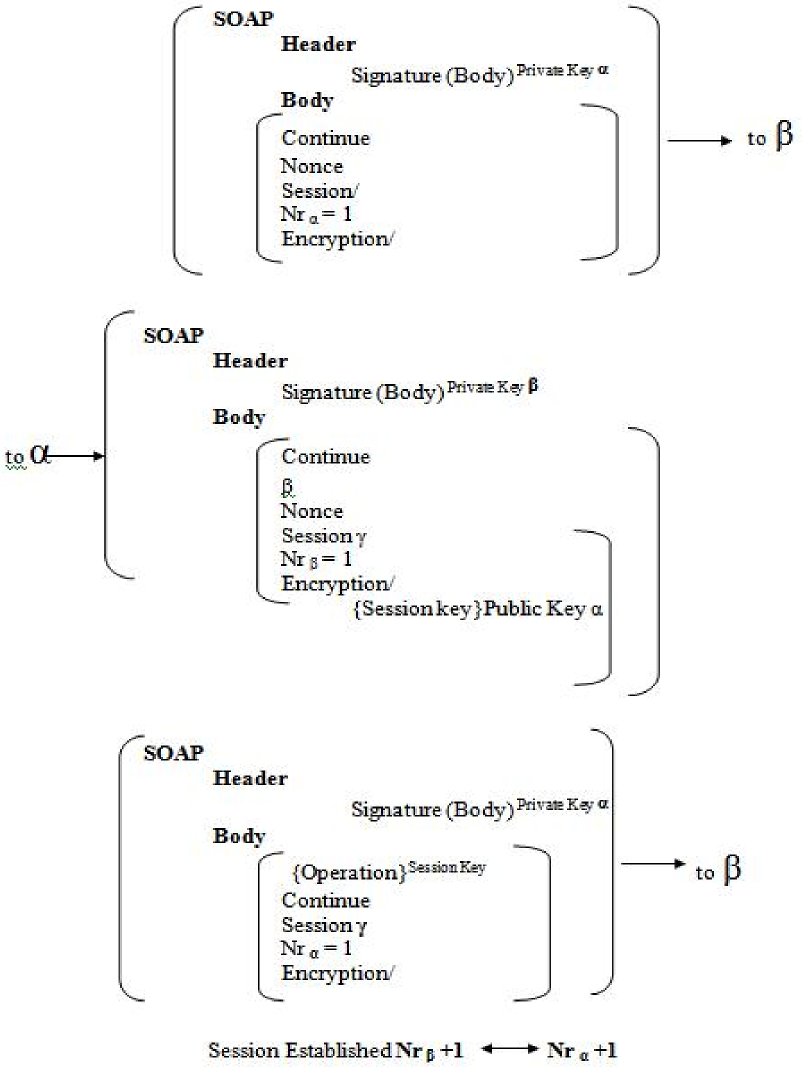


Figure 4.3: Establishing SOAP session

4.2 Analysis of SOAP session

There is always been a possibility that due to flaws in the security protocols an intruder can attack it.

Msg α_1 A ➠B : Header , { Body , Continue , Nonce , Session , N_r α (1), Encryption} Signed {PrivateKey α}

Msg β_1 A ➠B : Header , { Body , Continue , Nonce , Session (γ) , N_r β (1),N_r β (1),{SessionKey γ } PublicKey α} Signed PrivateKey β

Msg α $_2$ A ➠B : Header ,{ Body ,Continue ,{Operation}SessionKeyγ , Continue , N_r α (2). Session(γ)} Signed PrivateKey α

The purpose of the session protocol as explained in previous sections is to authenticate two web services to each other and to establish a shared secret session key with which they can encrypt their messages to ensure confidentiality. The validity of the protocols can be verified using Casper [21] , which is a compiler for analysis of security protocols. The security model can be made using Casper and this will be helpful in verifying it. Casper converts the modeled protocol into CSP [22] and FDR [23] is used as a model checker for CSP which in turn is helpful in verifying the security properties. These security properties can be statements such as "authenticated correctly" or "shared session key is secret". If there is a reachable state (while executing the protocol) where such a statement is not true, FDR will find a trace. If there is a trace, then there is an attack upon on the protocol. The setup of a session can be modeled as follows. This first example was the first attempt of finding a working protocol. After modeling it in Casper, the resulting Casper script is below

Msg α $_1$ A ➠I : Header , { Body , Continue , Nonce , Encryption} Signed PrivateKey α

Msg α $_1$ I_A ➠B : Header , { Body , Continue , Nonce , Encryption} Signed PrivateKey α

Msg β $_1$ B ➠I_A : Header , { Body , Continue , Nonce , Session (γ), Nrβ(1) {SessionKey γ} PublicKey α } Signed PrivateKey β

Msg β $_1$ I ➠A : Header , { Body , Continue , Nonce , Session (γ), Nrβ(1)

{SessionKey γ} PublicKey α } Signed PrivateKey β

Msg α $_2$ A ➠I : Header , { Body , {Operation} SessionKey γ , Continue, Session(γ)} Signed PrivateKey α

Msg α $_2$ I_A ➠B : Header, {Body, {Operation} SessionKey γ , Continue, Session(γ)} Signed PrivateKey α

The assertions that had to be verified were, whether or not web service and were correctly authenticated to each other and whether or not they agreed upon the session id and the session key. The session key also had to be a shared secret between the two web services. Unfortunately, running the protocol through Casper resulted in an attack upon the protocol itself, where the intruder acts as if it was alpha. This is a very common attack to a security protocol. The following trace consists of three actors where "I" stands for the intruder and the subscript to the intruder means that it is posing as another actor.

The trace found by Casper means that after a complete run of the protocol,thinks it has established a session with the intruder, however the intruder establishes a session with and thinks it has established a session with .This was just one trace that Casper found during the design of a session. After thorough analysis of the protocol, the correct definition of the protocol in Casper is

Msg α_1 : A ➠B : Header , { Body , Continue , Nonce , Session , N_r α (1), Encryption} Signed {PrivateKey α}

Msg β_1 A ➠B : Header , { Body , Continue ,B, Nonce , Session (γ) , N_r β (1), N_r β (1),{SessionKey γ } PublicKey α} Signed PrivateKey β

Msg α_2 A ➠B : Header , { Body , Continue , {Operation} SessionKey γ , Continue , N_r α (2) Session(γ)} Signed PrivateKey α

Casper did not find an attack to this protocol. The difference here is that can determine the identity of, because in the message is receives back from either or the intruder, it is stated that the identity of the sender of the second message must be . The question is though whether or not this is a good representation of a session setup with SOAP. It must be certain that all properties of the SOAP session are modeled in Casper. This means that every aspect of SOAP, relevant to the session setup, must be modeled in Casper. For instance, the fact that headers may always be removed by intermediaries may influence the level of security achieved. It is the task of the application to make sure that digital signatures are always verified and abnormalities dealt with.

4.3 Web Services Development Kit (WSDK)

WSDK exposes two programming models, Low-level API for direct access, higher-level integration with ASP.NET, exposed through ASMX. WSDK currently supports WS-Security, WS-Routing, WS-Referral, DIME, and WS-Attachments .WSDK currently does not support WS-Inspection, WS-Coordination, or WS-Transaction.

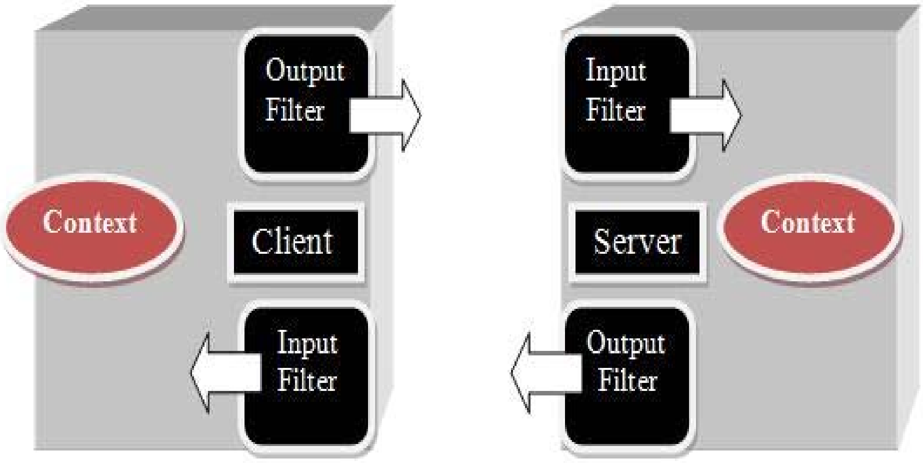


Figure 4.4: WSDK Process Model

The WSDK [24] is an engine for applying advanced Web service protocols to SOAP messages. This entails writing headers to outbound SOAP messages and reading headers from inbound SOAP messages. It may also require transforming the SOAP message body; for instance, encrypting an outbound message's body and decrypting an inbound message's body, as defined by the WS-Security specification. This functionality is encapsulated by two sets of filters: one for outbound messages and one for inbound messages.

All messages leaving a process request messages from a client or response messages from server are processed using the outbound message filters. All messages arriving in a process request messages to a server or response messages to a client are processed using the inbound message filters. SOAP messages are processed as they cross application boundaries utilizing a pipeline of filters.

Filters are responsible for processing SOAP headers. WSDK has the ability to help secure XML Web services across platforms and trust domains, including digital signing and encryption of SOAP messages that are compliant with the WS-Security specification. It has the ability to route an XML Web service through intermediaries using the WS-Routing specification, which describes how to place message addresses in the SOAP message header and enables SOAP messages to travel serially to multiple destinations along a message path. The route a SOAP message takes to an XML Web service can be transparently delegated among Web servers. The core features included in the technology preview of the Microsoft WSDK include:

1. The ability to help secure XML Web services across platforms and trust domains, including digital signing and encryption of SOAP messages those are compliant with the WS-Security specification;

2. The ability to route an XML Web service through intermediaries using the WS-Routing specification;

3. Communication between XML Web services can contain attachments that are not serialized into XML.

4.4 Performance Analysis

The aim here is to find out how performance is affected by introducing cryptography in SOAP messages. We conducted the experiments on a low performance host. This was the case because generally the servers or hosts of Web Services are targets of many requests. We calculated the response times before introducing cryptography into SOAP messages. SOAP payload may be XML document or any other content such as image, audio or video data.

The first part contains a SOAP message which includes the Header block created by the Message Handler. The second and subsequent parts contain payload(s)

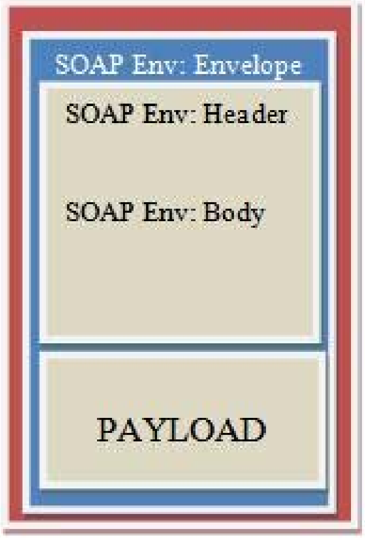


Figure 4.5: SOAP envelope and payload

which may be XML documents or any other content type such as image, audio or video data. The SOAP manifest header can contain elements that reference the separate parts using their content identifiers. This may be achieved using XLink references as shown in the following example. The XLink role attribute may be used to further qualify the type of data contained within the payload [25].

The results obtained after encapsulating cryptographic techniques with the SOAP messages and without cryptographic techniques are shown in table 4.1 and 4.2

Payload (in KB)	Response time (in Milliseconds)
0	0
1	380
78	410
156	580
312	700

Table 4.1: SOAP Analysis without Encryption

Payload (in KB)	Response time (in Milliseconds)
0	0
1	450
78	550
156	700
312	930

Table 4.2: SOAP Analysis with Encryption

The graph plotted shows the behavior of response time as the Payload of SOAP body increases after encapsulating cryptographic techniques.

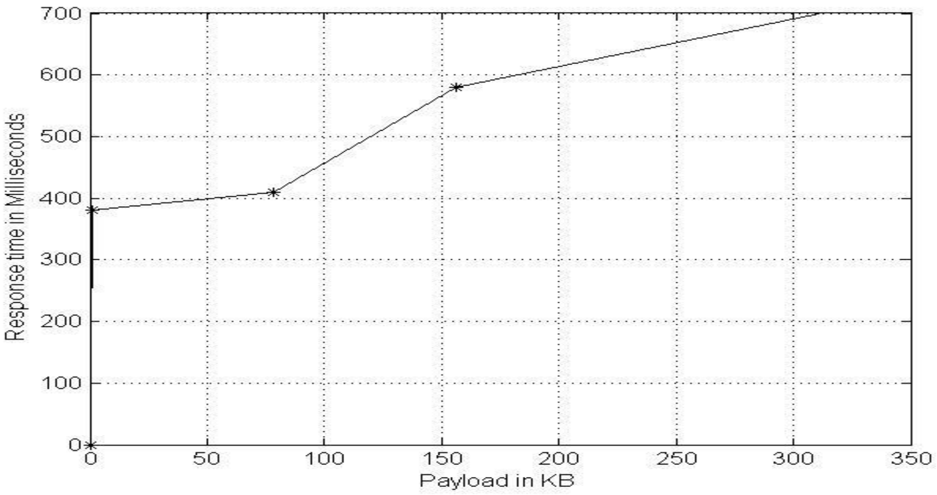


Figure 4.6: Response time v/s Payload-I

The test conducted includes a client encrypting the SOAP message with a symmetric algorithm (Triple DES). The service at the other end would decrypt the message using the same key. To have the same key at both sides for experiment, we are inputting the same number of key bytes at both sides for the generation of the key. Once the message is decrypted, the service performs the operation and then encrypts the response with the same key. The client decrypts the message and uses the response.

The analysis was conducted using Microsoft's WSDK (Web Services Development Kit) preview kit. The response time is measured in milliseconds. The

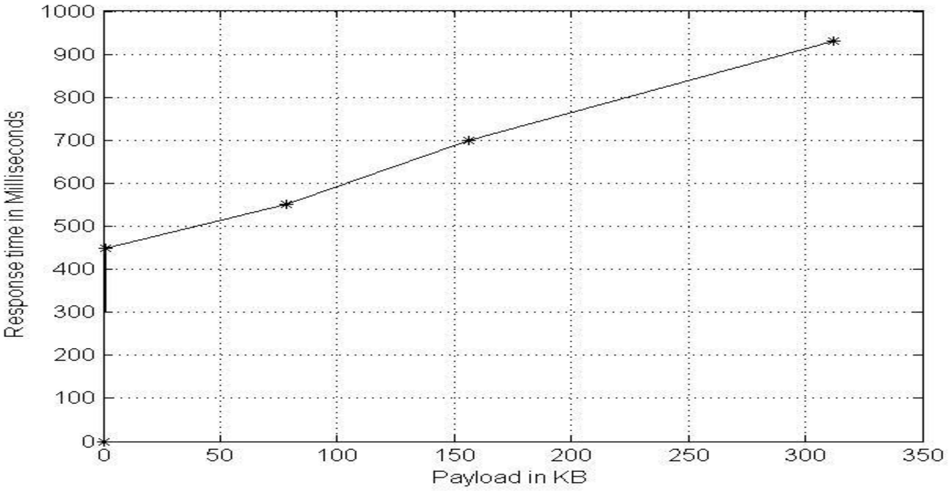


Figure 4.7: Response time v/s Payload-II

sequence of operations performed at the client side is depicted in the following figure. As we have seen from the graphs, the time taken for the asymmetric cryptography is very high affecting the overall performance. Here we can use both symmetric key cryptography and Asymmetric Key cryptography. The approach mentioned i.e. signing of the whole SOAP body message is of no use as this would be very much similar to having an SSL connection. As the previous sections dealt with problems in SSL, this mechanism of signing has the same problems.

We need to be able to encrypt parts of SOAP body. This is well documented in the standards of XML encryption. By observing the Graphs 1 and 2, we can conclude that there is little overhead in the response time using symmetric key cryptography. So we can asymmetrically encrypt a secret key, which was used to partially encrypt and send it to the other party in the tag <Encrypted Key>.The other party would decrypt the encrypted key and start decrypting the parts of messages encrypted using this key. Interestingly the overhead would be very little. Assuming that asymmetric algorithm RSA is used, the key size is 1024 bits which is equivalent to 1KB. The time for encryption and decryption on a heavily loaded server would be approximately equal to 45 seconds. Also the parsing time could affect the performance. But almost every day we are getting faster parsers. So this should not be a problem.

4.5 Conclusion

It is not uncommon for the security protocols to have flaws so that an intruder can attack it. The purpose of the session protocol as explained in this chapter is to authenticate two web services to each other and to establish a shared secret session key with which they can encrypt their messages to ensure confidentiality.

Web Services require end-to-end security since the requests might have to traverse over a chain of entities. The most popular protocol now in use, SSL, works well for point-to-point security services. SSL does not completely address the security issues of Web Services. By itself, WS-Security does not ensure security nor does it provide a complete security solution. WS-Security is a building block that is used in conjunction with other Web service specific protocols which include, XML encryption, XML signatures etc. to accommodate a wide variety of security models and encryption technologies to address the security concerns.

Partial encryption, signing and super encryption are a few of the proposed techniques which could be used to ensure end-to-end security. But the discussed techniques come with an overhead which is due to parsing large SOAP documents, among other things. As there are not many standard implementations for these protocols it is difficult to estimate the overhead. So the effect of these techniques on performance of Web Services transactions could not be assessed. Unless Web Services can be made can be made to work with reasonable response time, it is difficult to see them together solving the problem.

The communication between the service requestor and provider is not very hard to setup, but while using extensions to SOAP, it show that the implementations of the extensions is difficult. While building a network of secure web services that used the proposed security extensions is also hard. However, it was possible to implement the setup of SOAP session as explained in this chapter.

Digitally signing and encrypting SOAP elements consume resources. The main idea behind SOAP messaging is to provide a loosely coupled system with a way that they can communicate with each other in a connectionless way. We have addressed to the security issues that became relevant during the design of the system and at the time of setting up the SOAP session. The research involved the analysis of available standards and ways to use and develop them to create such a session by only using parts of those standards. The analysis of the session setup process proves that an adversary cannot break the protocol by interception, alteration or by resending of messages.

Chapter 5

Conclusion and Future Work

Chapter 5

Conclusion and Future Work

5.1 Conclusion

The first part of the research focuses on the communication between the service requestor and the service provider. Here we proposed a model for the dynamic selection of the web services which will facilitate the clients with the more appropriate solution in selecting the Web Services. The evaluation of the repository based prototype shows that such architecture can be built using a set of standard programming languages and protocols. However, in order to avoid fluctuations of the experimental results due to particular machine specifications, programming languages, and protocols, as well as to observe a large range of system parameters, the layer can be evaluated using an existing simulation tool. The accuracy of the predicted QoS criteria has a big impact on the selected Web Service. The repository helps improve the dependability of the system when a low level of Web Services availability is observed.

The second part of the research involved a thorough analysis of SOAP and the way it binds to the transport layer protocols. An important part of the research was to analyse SOAP and to setup a secure SOAP session. This can be done by using the proposed security extensions to SOAP. The research involved the analysis of the standards and ways to use and develop them to create such a session, by using the parts of those standards. The result of the research is a definition of a security protocol that has been analyzed using WSDK. More information on how a security context can be setup is reflected in this work. The analysis of the session setup proves that an adversary cannot break the protocol by the interception, alteration or the (re-)sending of messages.

However, implementation faults and the in-security of the servers running the web services, faults in the security considerations of each specification or the level of security might be compromised. Furthermore, security is most of all a social problem as well as a technical one.

5.2 Future Work

Firstly, there is still some more implementation work to be done. As explained, there are parts of the design document that have not been implemented. There are also beta versions or newer implementations of for example the SOAP D-Sig specification that are improvements upon the already implemented one.

Furthermore, there are a lot of specifications still under development that add extensions to SOAP and try to tackle the security issues mentioned in this thesis from another way. There is a lot of work to be done still in the area of securing web services and it might be very well possible that there are already specifications under development that try to setup a SOAP session and try to achieve message level security with web services. The stated solution in this thesis is just one of the ways to do this.

Bibliography

[1] C.M. Sperberg-McQueen Eve Maler Tim Bray, Jean Paoli and Fran cois Yergeau. Extensible markup language (xml) 1.0 (third edition). *http://www.w3.org/Protocols/rfc2616/rfc2616.html*.

[2] Nilo Mitra. Soap version 1.2 part 0: Primer. *http://www.w3.org/TR/soap12-part0/*, 2003.

[3] H. Kreger. Web services conceptual architecture,wsca 1.0. *http://www.cs.uoi.gr/zarras/mdw-ws/webServicesConceptualArchitectu2.pdf*.

[4] UDDI Browser. http://www.uddibrowser.org.

[5] Greg Meredith Erik Christensen, Francisco Curbera and Sanjiva Weerawarana. Web services description language (wsdl) 1.1.http://www.w3.org/tr/wsdl.

[6] http://webservices.xml.com/pub/a/ws/2001/04/04/webservices/index.html.

[7] E. Maximilien and M. Singh. Framework and ontology for dynamic web services selection. *IEEE Internet Computing*, pages 84–93, 2004.

[8] E.M. Maximilien and M.P Singh. Reputation and endorsement for web services. *SIGecom Exch*, 3.1:24–31, 2001.

[9] A. H. H. Ngu Y. Liu and L. Zeng. Qos computation and policing in dynamic web service selection. *In Proceedings of the International World Wide Web Conference*, pages 66–73, September 2004.

[10] Daniel A. Menasce. Composing web services–a qos view. *IEEE Internet Computing*, 8:88–90, November 2004.

[11] Marlon Dumas-Jayant Kalagnanam Liangzhao Zeng, Boualem Benatallah and Quan Z. Sheng. Quality driven web services composition. *In proceedings of the 12th International conference on World Wide Web (WWW)*, November 2003.

[12] March Hadley Martin Gudgin and Tony Rogers. Web services addressing 1.0 - core. *http://www.w3.org/TR/soap12-part1*, June 2002.

[13] XJ Technologies Company Ltd. Anylogic enterprise library tutorial. *http://www.xjtek.com/files/docs/en/EnterpriseLibraryTutorial.pdf*.

[14] XJ Technologies Company Ltd. Anylogic enterprise library tutorial. *http://www.xjtek.com/files/docs/en/EnterpriseLibraryReferenceguide.pdf*.

[15] Sun Microsystems. *Trail The Reflection API*.

[16] Dale Green. *Trail The Reflection API*.

[17] Giovanni Della-Libera Brandon Dixon and Joel. Securing web services world : A proposed architecture and roadmap. *http://msdn.microsoft.com/library/enus/html/securitywhitepaper.asp*.

[18] Traverso P. Dustdar S.-Leymann F. Papazoglou, M. Service- oriented computing: A research roadmap. *International Journal of Cooperative Information Systems*, 17:1–33, July 2008.

[19] Gombotz R Dustdar, S. Discovering web service workflows using web services interaction mining. *International Journal of Business Process Integration and Management (IJBPIM)*, 1:255–266, April 2006.

[20] Takeshi Imamura and Hiroshi Maruyama. Transform from xml signature. *http://www.w3.org/TR/xmlenc-decrypt*, December 2002.

[21] Philipa Broadfoot Gavin Lowe and Mei Lin Hui. Casper, a compiler for the analysis of security protocols. *http://web.comlab.ox.ac.uk/oucl/work/gavin.lowe/Security/Casper/*, December 2001.

[22] C.A.R. Hoare. Communicating sequential processes. 1985.

[23] Formal systems (Europe) Ltd. Failures-divergence refinement - fdr 2 user manual. *http://www.formal.demon.co.uk/FDR2.html*, 1997.

[24] http://www.xml.coverpages.org/MicrosoftWSDK200208.html.

[25] T. Nadalin. Web services security: Soap message security 1.0 (ws-security-2005) oasis web services security tc, oasis standard 200401. March 2005.

Dissemination of Work

Published

Abhishek Pandey and S.K.Jena "Dynamic Approach for Web Services Selection.", *In proceedings of the International MultiConference of Engineers and Computer Scientists 2009(IMECS),* pages 960-962, 18-20 March 2009, Hong Kong, China.

Communicated

Abhishek Pandey and S.K.Jena "Web Services: Analysis and Implementation of Security Protocols ", *Fourth International Conference on Risks and Security of Internet and Systems (CRiSIS 2009)*, August 2009,France.