Enforcing Quality of Service in .NET based Web Services

Dr. L. Arockiam*
N. Sasikaladevi**

Online access at www.publishingindia.com

Abstract

Web services play a vital role in today's business environments. Different vendors are available to design the web services. Dominant industry players in the design of web services are J2EE and .NET. Web services are used by different business environments. Enforcing quality of service in web service is needed today. Reliability, Availability, scalability, and security are service- level characteristics that determine Quality of Service requirements. These characteristics are highly desirable for web services. In this paper, we have given solution for reliability issue for .NET based web service.

Keywords - EventLog, SOAP, Web service, XMLSignature.

1. Introduction

A high-level definition of a Web service would be a programmable application component accessible through standard Internet protocols. Many more detailed definitions of Web services exist, and they all seem to contain the following elements in the description:

- Web services expose useful functionality through standard Internet protocols. In most cases, this protocol is SOAP over HTTP.
- Web services describe their interfaces in enough detail to allow a user to build a client application to talk to them. An XML document named a Web Services Description Language (WSDL) document contains this description.
- Users can search for available Web services in some form of registration database. Universal Description, Discovery and Integration (UDDI) is the most common way to implement this.

Because Web services are standards-based and platform independent, they provide a natural fit when it comes to getting applications in different platforms to interoperate with each other. This partially explains the alacrity with which so many vendors have endorsed the Web services standards. [1]

*Lecturer(SG), St.Joseph's College, Trichy
**Lecturer, Bishop Heber College, Trichy

Layered Approach :
Web service is a five layer model. Fig.1 shows the web service stack.

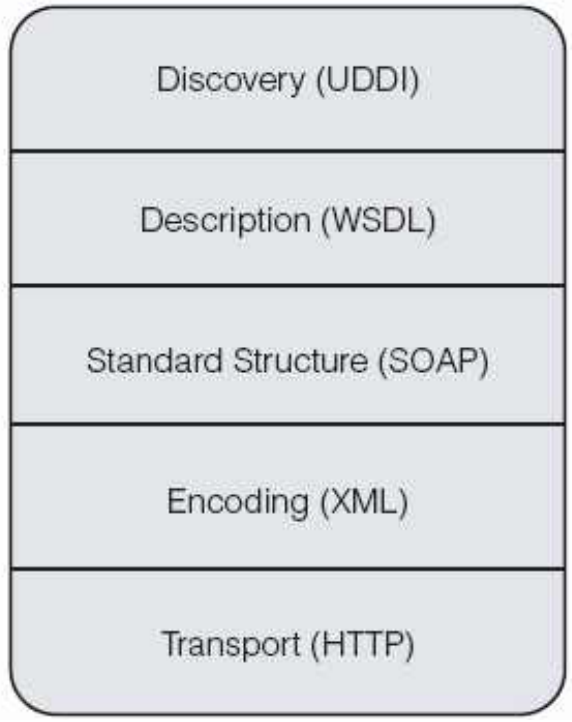

Fig. 1 : Web service protocol stack

Transport (HTTP) : At the lowest level, two components in a distributed architecture must agree on a common transport mechanism. Because of the near universal acceptance of port 80 as a less risky route through a firewall, HTTP became the standard for the transport layer. However, Web services implementations can run on other transport protocols such as FTP and SMTP, or even other network stacks, such as Sequenced Packet Exchange (SPX) or non-routable protocols such as NetBEUI. Changing from the dependence on HTTP or HTTPS (for encrypted connections) is possible within the bounds of the current specification.
Encoding (XML) : After agreeing on the transport, components must deliver messages as correctly formatted XML documents. This XML dependence ensures the success of the transfer, because both provider and consumer know to parse and interpret the XML standard.
Standard Structure (SOAP) : Although XML defines message encoding, it does not cover the structure and format of the document itself. To guarantee interoperability, both provider and consumer must know what to send and what to expect. SOAP is a lightweight, message-based protocol built on XML (XSD version 2) and standard Internet protocols, such as HTTP and SMTP. The SOAP protocol specification defines an XML structure for messages (the SOAP envelope), data type definitions, and a set of conventions that implement remote procedure calls and the format of any returned data (the SOAP body).
Description (WSDL) : The description layer provides a mechanism for informing interested parties of the particular bill of fare that a Web service offers. Web Services Description Language (WSDL) provides this contract, setting out for each exposed component :

- Component Name
- Data Types
- Methods
- Parameters

This WDSL description enables a developer for a remote component to query your Web service and find out what the service can do and how to get it to do it. The WSDL file is an XSD-based XML document that defines the details of your Web service. It also stores your Web service contract. The WSDL file is usually the first point of entry for any client attaching to your Web service so that the client knows how to use it.
Discovery (UDDI) : Discovery attempts to answer the question "Where." If you want to connect to a Web service at an Internet location you can enter the URL manually. Service provider publishes their service on a Universal Description, Discovery and Integration (UDDI) server. Consumer or client can find the service by connecting to the UDDI server using an agreed message format to locate the URL for the service. [2]

Fig. 2 shows the working principle of web service and usage of all the five layers.

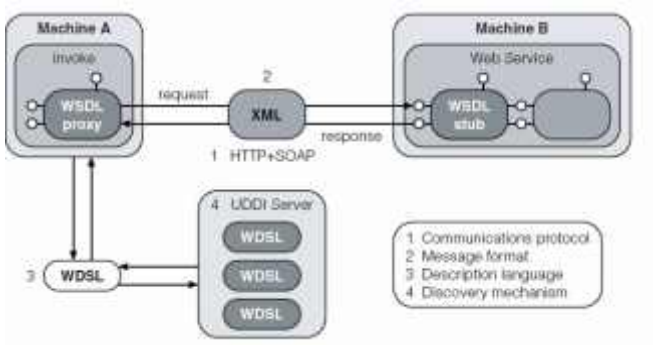

Fig. 2 : Web service working strategy

2. .net Web Service Architecture

In this paper, we have given the solution for reliability issue in .NET based web service. Consumer of this web service may either be .NET client or any other. Before getting into the detailed reliable .NET web service model, existing .NET web service architecture is shown in fig. 3.

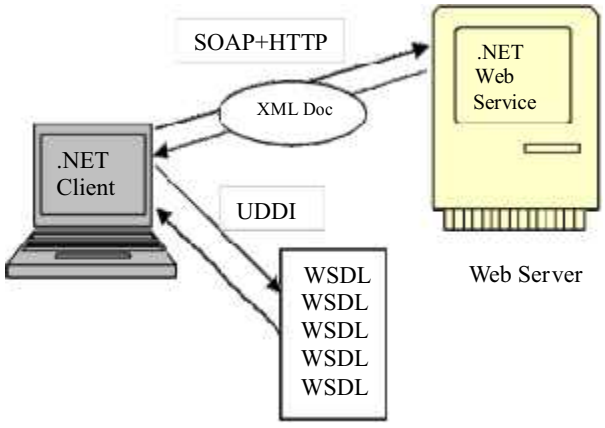

Fig. 3 : .NET Web Service

NET provides a solution with a dedicated component called a proxy class, which performs the heavy lifting for your application. The proxy class wraps the calls to the web service's methods. It takes care of generating the correct SOAP message format and managing the transmission of the messages over the network (using HTTP). When it receives the response message, it also converts the results back to the corresponding .NET data types.

3. Quality of Service

Quality of Services is extremely important in managing services. Reliability, Availability, scalability, and security are service-level characteristics that determine Quality of Service requirements. These characteristics are highly desirable for web services. With Quality of Services, businesses can provide differential business services and capacity on-demand. This is also one of the key objectives in Utility Computing.

The term "Quality of Services" (QoS) has been widely used in the telecommunications and data center communities to refer to treating different network packets or infrastructure services differently and not with the same best-effort service. Applying the QoS concept to software engineering, QoS usually refers to the systemic quality of reliability, availability, scalability, manageability and security for developing and deploying web services. [2]

It is important to design and deploy .NET web services with this systemic quality.

This paper elaborates one of the metric of QoS called reliability in .NET bases web services.

Reliability for .NET web services requires service requests or business data from a sender to be transmitted to the target recipient successfully and accurately. The target recipient should be able to acknowledge receipt if necessary. The business data needs to be accurate and without errors. Successful transmission can be achieved by resending the service requests and business data until there is a receipt acknowledgement from the recipient or by persisting them in a reliable data store so that the target recipient can pick them up.

Reliability can be producer-centric or consumer-centric. In the producer-centric scenario, the sender (producer) needs to ensure that the target recipient (consumer) receives the business data. Otherwise, the sender resends the business data until the target recipient sends an acknowledgement. If the target recipient is offline or unavailable, the sender is responsible for resending the business data whenever the target recipient service becomes available. In the consumer-centric scenario, the target recipient (consumer) is responsible for retrieving the business data from the sender (producer). This requires the sender be highly available for information retrieval. [2]

In essence, reliability of web services is expected in different layers or tiers in the application architecture, which includes the following :

- The underlying communication mechanism or data transport layer for both Java and .NET applications needs to be reliable.
- When a sender exchanges some business data with a recipient, the contents of the business data should be reliably transferred. For example, if the business data contain a data type of a large scientific quantity or a complex data type, the recipient should receive them verbatim without any data type conversion or XML encoding errors between web service client and web service server.
- There should be a receipt acknowledgement capability web services. One of the common design criteria is to enable the timeout on the message or acknowledgment of delivery to avoid applications waiting for an extended period of time.
- There should be a logging mechanism at both ends of the Java and .NET applications for audit trail and for compliance reporting purpose.
- Guaranteed delivery is not the only characteristic of reliability. Once-and-only-once (also known as idempotence) requirement may also apply to the QoS requirements, meaning that the message can only be delivered one time to its destination.
- Reliable messaging, in cases such as financial data transfer, also requires a high level of security.

Therefore, it is important to perform a comprehensive QoS check to ensure that the web service architecture adequately addresses individual QoS requirements.

4. Enforcing Reliability in .Net Web Service

The .NET framework provides various classes to support reliability. All the requirements of the reliability are satisfied in this paper. Reliability is enforced by using the following mechanisms.

- **Event Logging :** During the execution of the web service, all the events cause by the web service are traced by Event logger and it is send to the Windows event viewer. This event log file is used to auditing and system restore if the web service causes any damages. To implement this strategy SOAP extensions and System. Diagnostics. Event Log class are used.
- **Message Passing :** Acknowledgement has to be received from Web service to ensure reliable delivery of data. This is achieved by using Microsoft Messaging Queue (MSMQ).
- **Digital Signature :** Web services for banking sector and other financial sectors had a high risk of money. There is high demand for the authentication of web services to be use. This is accomplished by the use of the XML signature.

A) Message Passing with Timestamp

Primary requirement to enforce reliability is Guaranteed delivery and to avoid duplications. This is accomplished by using the message passing mechanism by .NET called MSMQ. The .NET client sends the data to the web service in XML document. This is acknowledged by the web server by sending SOAP message back to client.

A queue is a persistent structure that exists on a particular machine. You need to create this persistent structure before you can write to it or read from it. The sender first has to open the queue for writing. This is done by instantiating an object of class System. Messaging. Message Queue. static method, Message Queue. Create, does that. Instantiating the object opens the queue for reading or writing, which is conceptually similar to opening a file rather than creating one. Once you've made the connection to the queue, you can create an object of any .NET class that supports XML serialization and pass it to the Send method of the queuing connection. The .NET wrapper now serializes the object into an XML stream and transmits that stream as the body of an MSMQ message. he object will be automatically reconstituted on the recipient side. the property Message. Time To Reach Queue specifies the amount of time allowed between a message being sent and the message reaching its final destination queue. If it doesn't reach the queue by that time, it is destroyed. The property Message. Time To Be Received specifies the amount of time allowed between a message being sent and that message being retrieved from the queue by the recipient. The message is now in the queue and ready for reading. If the sender's and recipient's machines are not the same (as they are here for ease of use), the message would be buffered in an outgoing queue on the sender's machine and sent to the recipient when a connection between the two machines became available. When you click Enumerate, I open the queue and call the method Message Queue. Get All Messages. This returns an array of objects of class System. Messaging. Message, providing a static snapshot of all messages in the queue. Each message contains MSMQ properties such as priority and time sent. If you double-click the message in the listbox, my code pops up a form showing some interesting properties of the message including the body of the message containing the serialized Point structure.he .NET Framework provides you with a choice of three formatters: XML, binary, and ActiveX. I used XML Formatter for messages. The sender-side object gets serialized into an XML packet. When the object gets deserialized, the recipient program has to provide the System. Type object describing the class to which the recipient wants the message reconstituted. This strategy is called loose coupling because the message itself does not specify the type of object to which it maps on the recipient. The recipient can deserialize the message into any class that can handle an XML packet of that layout. The sender and recipient do not have to share any code; they need only agree on the XML schema of the serialized object. The .NET provides this facility through the namespace called System.Xml.Serialization.[7]

B) Event Logging

One of the powerful features of .NET is, it allows the applications to maintain the event log in order to ease the auditing. The .NET framework has System. Diagnostics. Event Log class to support logging. we can monitor the logs from windows event viewer utility. Windows XP, NT and other upgraded version of windows support this utility.

SOAP extension is used to logs SOAP messages to the Windows event log. All SOAP extensions consist of two ingredients : a custom class that derives from System. Web. Services. Protocols. Soap Extension and a custom attribute that apply to a web method to indicate that the SOAP extension should be used. The custom attribute is the simpler of the two ingredients.

The Soap Extension Attribute : The Soap Extension attribute allows to link specific SOAP extensions to the methods in a web class. When we create Soap Extension attribute, derive from the System. Web. Services. Protocols. Soap Extension Attribute, as shown in Listing 1 :

Listing 1 :

[Attribute Usage (Attribute Targets. Method)] public class Soap Log Attribute: System. Web. Services. Protocols. Soap Extension Attribute { ... }

The Attribute Usage attribute : It indicates where we use custom attribute. SOAP extension attributes are always applied to individual method declarations, much like the Soap Header and Web Method attributes. Thus, we should use Attribute Targets. Method to prevent the user from applying it to some other code construct (such as a class declaration). You should use Attribute Usage anytime you need to create a custom attribute—it isn't limited to web service scenarios.

Every Soap Extension attribute needs to override two abstract properties : Priority and Extension Type. Priority sets the order that SOAP extensions work if you have multiple extensions configured. However, it's not needed in simpler extensions such as the one in this example. The Extension Type property returns a Type object that represents your custom Soap Extension class, and it allows .NET to attach your SOAP extension to the method. In this example, the class name of the SOAP Extension is Soap Log,

Listing 2 :

```
private int priority;
public override int Priority{
get { return priority; }
set { priority = value; }
}
public override Type ExtensionType
{
get { return typeof(SoapLog); }
}
```

In addition, we can add properties that will supply extra bits of initialization information to your SOAP extension. The following example adds a Name property, which stores the source string that will be used when writing event log entries, and a Level property, which configures what types of messages will be logged. If the level is 1, the SoapLog extension will log only error messages. If the level is 2 or greater, the Soap Log extension will write all types of messages. If the level is 3 or greater, the Soap Log extension will add an extra piece of information to each message that records the stage when the log entry was written.

Listing 3 :

```
private string name = "Soap Log" ;
public string Name
{
get { return name;}
set { name = value; }
}
```

```
private int level = 1;
public int Level
{
get { return level;}
set { level = value; }
}
```

Now apply this custom attribute to a web method and set the Name and Level properties. Listing 4 uses the log source name My Service. do Credit and a log level of 3 :

Listing 4 :

```
[Soap Log (Name="MyService.doCredit", Level=3)]
[Web Method()]
public int do Credit()
{ ... }
```

Now, whenever the do Credit method is called with a SOAP message, the Soap Log class is created, initialized, and executed. It has the chance to process the SOAP request message before the do Credit method receives it, and it has the chance to process the SOAP response message after the do Credit method returns a result.

The Soap Extension class provides many of the methods we need to override, including the following :

- GetInitializer () and Initialize () : These methods pass initial information to the SOAP extension when it's first created.
- Process Message () : This is where the actual processing takes place, allowing your extension to take a look at (and modify) the raw SOAP.
- Chain Stream () : This method is a basic piece of infrastructure that every web service should provide. It allows you to gain access to the SOAP stream without disrupting other extensions.

Here's the class definition :

Listing 5 :

```
public class SoapLog : System. Web. Services. Protocols. Soap Extension { ... }
```

ASP.NET calls the GetInitializer () method the first time extension is used for a particular web method. It gives the chance to initialize and store some data that will be used when processing SOAP messages. Store this information by passing it back as the return value from the GetInitializer () method.

When the GetInitializer () method is called, receive one important piece of information the custom attribute that was applied to the corresponding web method. In the case of the Soap Log, this is an instance of the Soap Log Attribute class, which provides the Name and Level property. To store this information for future use, we can return this attribute from the GetInitializer () method, as shown here :

Listing 6 :

```
public override object Get Initializer (Logical Method Info methodInfo, Soap Extension Attribute attribute)
{
return attribute;
}
```

Actually, the GetInitializer () method has two versions. Only one is invoked, and it depends on whether the SOAP extension is configured through an attribute (as in this example) or through a configuration file. If applied through a configuration file, the SOAP extension automatically runs for every method of every web service. Even if we don't plan to use the configuration file to initialize a SOAP extension, we still need to implement the other version of GetInitializer (). In this case, it makes sense to return a new Soap Log Attribute instance so that the default Name and Level settings are available later :

Listing 7 :

```
public override object GetInitializer (Type obj)
{
return new Soap Log Attribute ();
}
```

GetInitializer () is called only the first time SOAP extension is executed for a method. However, every time the method is invoked, the Initialize () method is triggered. If we returned an object from the Get Initializer() method, ASP.NET provides this object to the Initialize () method every time it's called. In the Soap Log extension, this is a good place to extract the Name and Level information and store it in member variables so it will be available for the remainder of the SOAP processing work.

Listing 8 :

```
private int level;
private string name;
public override void Initialize(object initializer)
{
name = ((SoapLogAttribute)initializer).Name;
level = ((SoapLogAttribute)initializer).Level;
}
```

The workhorse of the extension is the Process Message () method, which ASP.NET calls at various stages of the serialization process. A Soap Message object is passed to the Process Message () method, and you can examine this method to retrieve information about the message, such as its stage and the message text. The Soap Log extension reads the full message only in the After Serialize and Before Deserialize stages, because these are the only stages when you can retrieve the full XML of the SOAP message. However, if the level is 3 or greater, a basic log entry will be created in the Before Serialize and After Deserialize stages that simply records the name of the stage.

Here's the full Process Message () code :

Listing 9:

```
public override void ProcessMessage(SoapMessage message)
{
switch (message.Stage)
{
case System. Web. Services. Protocols. Soap Message Stage. Before Serialize :
if (level > 2 )
Write To Log (message. Stage. To String (),
Event Log Entry Type. Information);
break;
case System. Web. Services. Protocols. Soap Message Stage.
```

```
After Serialize :
Log Output Message (message);
break;
case System. Web. Services. Protocols. Soap Message Stage.
Before Deserialize : Log Input Message (message);
break;
case System. Web. Services. Protocols. Soap Message Stage.
After Deserialize :
if (level > 2 )
Write To Log (message. Stage. To String(),
Event Log Entry Type. Information);
break;
}
}
```

The Process Message () method doesn't contain the actual logging code. Instead, it calls other private methods such as Write Log Log (), Log Output Message (), and Log Input Message(). The Write To Log () is the final point at which the log entry is created using the System. Diagnostics. Event Log class. If needed, this code creates a new event log and a new log source using the name that was set in the Name property of the custom extension attribute. Here's the complete code for the Write To Log () method :

Listing 10 :

```
private void Write To Log (string message, Event Log Entry
Type type)
{
// Create a new log named Web Service Log, with the event
source
// specified in the attribute.
EventLog log;
if (!EventLog.SourceExists(name))
EventLog.CreateEventSource(name, "Web Service Log");
log = new EventLog();
log.Source = name;
log.WriteEntry(message, type);
}
```

When the SOAP message is in the Before Serialize or After Deserialize stage, the Write To Log () method is called directly, and the name of the stage is written. When the SOAP message is in the After Serialize or Before Deserialize stage, you need to perform a little more work to retrieve the SOAP message. Before you can build these methods, you need another ingredient—the Copy Stream () method. That's because the XML in the SOAP message is contained in a stream. The stream has a pointer that indicates the current position in the stream. The problem is that as you read the message data from the stream (for example, to log it), you move the pointer. This means that if the log extension reads a stream that is about to be deserialized, it will move the pointer to the end of the stream. For ASP.NET to properly deserialize the SOAP message, the pointer must be set back to the beginning of the stream. If you don't take this step, a deserialization error will occur.

To make this process easier, we can use a private Copy Stream () method. This method copies the contents of one stream to another stream. After this method is executed, both streams will be positioned at the end.

Listing 11 :

```
private void Copy Stream (Stream from stream, Stream to
stream)
{
Stream Reader reader = new Stream Reader (from stream);
Stream Writer writer = new Stream Writer (to stream);
writer. Write Line (reader. Read To End());
writer. Flush ();
}
```

Another ingredient needed is the Chain Stream() method, which the ASP.NET plumbing calls before serialization or deserialization takes place. Your SOAP extension can override the Chain Stream () method to insert itself into the processing pipeline. At this point, the extension can cache a reference to the original stream and create a new in-memory stream, which is then returned to the next extension in the chain.

Listing 12 :

```
private Stream old Stream;
private Stream new Stream;
public override Stream Chain Stream (Stream stream)
{
old Stream = stream;
new Stream = new Memory Stream();
return new Stream;
}
```

Of course, this is only part of the story. It's up to the other methods to either read data out of the old stream or write data into the new stream, depending on what stage the message is in. You do this by calling the Copy Stream () method. Once you've implemented this somewhat confusing design, the end result is that every SOAP extension has a chance to modify the SOAP stream without overwriting each other's changes. For the most part, the Chain Stream () and Copy Stream () methods are basic pieces of SOAP extension architecture that are identical in every SOAP extension you'll see. The Log Input Message() and Log Output Message () methods have the task of extracting the message information and logging it. Both methods use the Copy Stream() method. When deserializing, the input stream contains the XML to deserialize, and the pointer is at the beginning of the stream. The Log Input Message() method copies the input stream into the memory stream buffer and logs the contents of the stream. It sets the pointer to the beginning of the memory stream buffer so that the next extension can get access to the stream.

Listing 13 :

```
private void Log Input Message (Soap Message message)
{
Copy Stream(old Stream, new Stream);
message. Stream. Seek(0, Seek Origin. Begin);
Log Message (message, new Stream);
message. Stream. Seek(0, Seek Origin. Begin);
}
```

When serializing, the serializer writes to the memory stream created in Chain Stream(). When the Log Output Message() function is called after serializing, the pointer is at the end of the stream. The Log Output Message () function sets the pointer to the beginning of the stream so that the extension can log the

contents of the stream. Before returning, the content of the memory stream is copied to the outgoing stream, and the pointer is then back at the end of both streams.

Listing 14 :

```
private void Log Output Message (Soap Message message)
{
message. Stream. Seek (0, Seek Origin. Begin);
Log Message (message, new Stream);
message. Stream. Seek(0, Seek Origin. Begin);
Copy Stream (new Stream, old Stream);
}
```

Once they've moved the stream to the right position, both Log Input Message () and Log Output Message () extract the message data from the SOAP stream and write a log message entry with that information. The function also checks whether the SOAP message contains a fault. In that case, the message is logged in the event log as an error.

Listing 15 :

```
private void Log Message (Soap Message message, Stream
stream)
{
Stream Reader reader = new Stream Reader (stream);
event Message = reader. Read To End();
string event Message;
if (level > 2)
event Message = message. Stage. To String () +"\n" + event
Message;
if (event Message. Index Of("<soap:Fault>") > 0)
{
// The SOAP body contains a fault.
if (level > 0)
Write To Log (event Message, Event Log Entry Type. Error);
}
else
{
// The SOAP body contains a message.
if (level > 1)
Write To Log (event Message, Event Log Entry Type.
Information);
}
}
```

This completes the code for the Soap Log extension. Using the Soap Log Extension to test the Soap Log extension, you need to apply the Soap Log Attribute to a web method, as shown here :

Listing 16 :

```
[Soap Log (Name="Employees Service. Get Employees
Logged", Level=3)]
[Web Method()]
public int Get Employees Logged ()
{...}
```

You then need to create a client application that calls that method. When you run the client and call the method, the SoapLog extension will run and create the event log entries.

To verify that the entries appear, run the Event Viewer (choose Programs ➤ Administrative Tools ➤ Event Viewer from the Start menu). Look for the log named Web Service Log. It will show the event log entries that you'll see after calling do Credit twice with a log level of 3.

The Soap Log extension is a useful tool when developing or monitoring web services. As the event log fills, old messages are automatically discarded. You can configure these properties by right-clicking an event log in the Event Viewer and choosing Properties.[4]

B) Digital Signature

"Digital signatures alone do not provide message authentication. One can record a signed message and resend it (replay attack). To prevent this type of attack, digital signatures must be combined with an appropriate means to ensure the uniqueness of the message, such as time stamps."

XML digital signature is used to ensure high level of security. XML Signature may also be used for integrity and no repudiation of WSDL files also, so that a definition of a Web Service can be published and later trusted to not have been tampered with, or been forged by a third party. XML Signature provides a useful means of expressing a digital signature over XML data. XML Signature finds multiple uses for Web Services security. When used alone, it provides data integrity. When linked to the signer's identity, it provides no repudiation of data content. In addition, it can be used for authentication. When SOAP routing occurs, it can be used for workflow.

.NET support XMLSignature by providing classes in the namespace System.Security.Cryptography.Xml. The SignedXML class in the .NET Framework includes a number of high-level methods used for XML Signature, such as the following :

- Signing Key Specifies the key that signs the XML data
- Add Reference Adds a signed resource as a Reference to the XML Signature
- Signing Key Specifies the key used to perform the signature
- Compute Signature Calculates the signature, using the signing key
- Check Signature Verifies a signature

1. Creating an XML Signature by producer :

Flowchart for the XML Signature is given below. XML Signature specification is standardized by w3c organization.

2. Validating an XML Signature by consumer :

The validation of an XML Signature follows many of the same steps as the creation of the signature. The reason for this is simple: in order to ensure that the signed data hasn't changed, the digest of the data must be recalculated and compared with the original. If a signature contained only a digest, then it would be equivalent to a checksum. If that were the case, an imposter could simply change the digest as well as changing the signed data, to conceal the change. That is why the SignatureValue contains a version of the digest which has been encrypted using the private key of the signer. The corresponding public key is used to obtain the digest. If this matches the digest of the signed data that means that the data has not changed since it was signed.

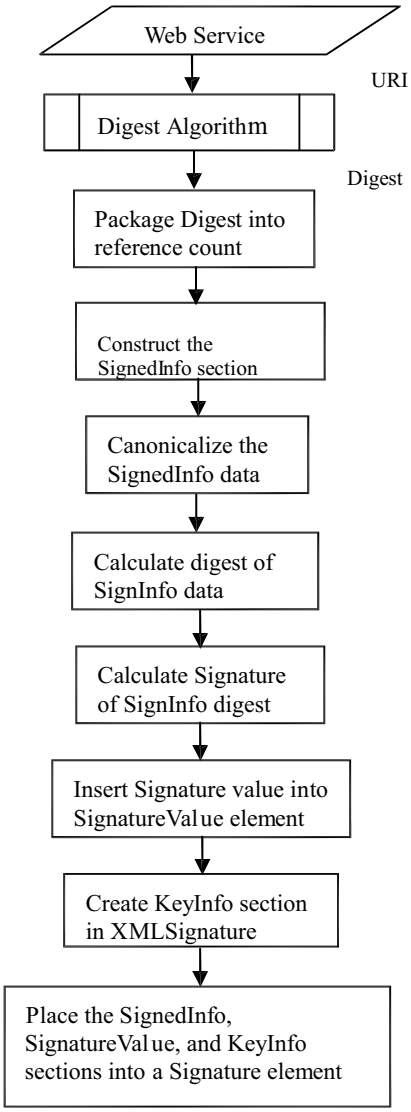

Fig. 4 : Creating XML Signature

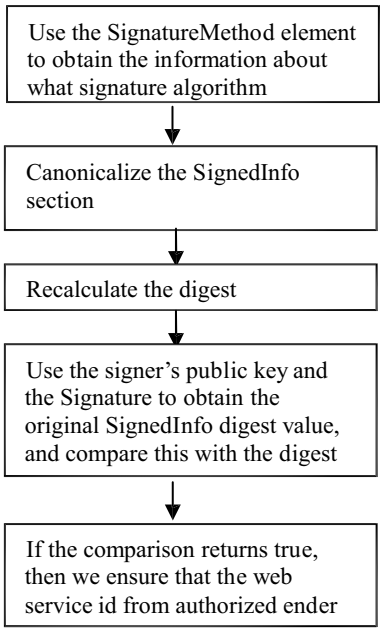

Fig. 5 : Checking XML Signature

5. Conclusion

The reliable architecture for web service is designed to enforce the quality of service. Reliability, availability, scalability, and security are service-level characteristics that determine Quality of Service requirements. In this paper, we have given solution for reliability. In future, the remaining metrics for QoS have to be solved. This particular architecture is designed with .NET framework. But different vendors for web services are available. In future, neutral architecture for web services with QoS metric will be developed.

6. References

1. *MS developers team, "Application Interoperability" by Micro Soft Corporation, 2003.*
2. *Marina Fisher, Ray Lai, Sonu Sharma, Laurence Moroney, "Java EE and .Net Interoperability: Integration Strategies, Patterns, and Best Practices" by SunMicrosystems, 2006.*
3. *Bruce Bukovics, ".NET 2.0 Interoperability recipes" by Apress, 2006.*
4. *Mathew MacDonald, Mario Szpuszta, ".Professional ASP.NET 2.0" by Apress, 2005.*
5. *"Security in .NET: The Security Infrastructure of the CLR Provides Evidence, Policy, Permissions, in MSDN Magazine*
6. *"Security and Enfocement Services" in MSDN Magazine at http://msdn.microsoft.com*
7. *"Security in .NET: Enforce Code Access Rights with the Common Language Runtime" in MSDN Magazine at http://msdn.microsoft.com*
8. *LaMacchia, Lange, Lyons, Martin, and Price. .NET Framework Security. Addison Wesley Professional, 2002.*
9. *Bipin Joshi, Manupulating windows event log published at www.dotnetbips.com*