

Fault Tolerance in Web Services

Goutam Kumar Saha

This paper demonstrates how to design a fault tolerant web application that relies on an affordable redundancy in data and function using Simple Object Access Protocol (SOAP). The proposed approach relies on a Single-Version fault tolerance model without using multiple versions or the N-Version fault tolerance model. The work aims to design a more dependable and reliable Web Service system at no extra cost on multiple versions.

Introduction

Web Services (e.g., latest traffic or weather report etc) are a means for requesting information over the Internet through encoding of both the request and response in XML using standard Internet protocols for transport in order to make the messages universally available. Web Services are the major component of many modern Business-to-Business (B2B) systems. Failures due to downtime or incorrect results in B2B systems might have considerable monetary penalties. That is why highly reliable B2B systems are must. Nowadays, WWW Consortium's (W3C) Simple Object Access Protocol (SOAP) has become the popular technology for developing web service application. SOAP is an Extensible Markup Language (XML) based protocol to let applications exchange information over Hyper Text Transfer Protocol (HTTP). In other words, SOAP is a protocol for accessing a Web Service. SOAP provides a way for two different systems to exchange information relating to a Remote Procedure Call (RPC) or other operation. It is important for application development to allow Internet communication between various programs. Conventionally, applications communicate using Remote Procedure Calls (RPC) between objects like Distributed Component Object Model (DCOM) and Common Object Request Broker Architecture (CORBA), but HTTP was not designed for this. RPC represents a compatibility and security problem, because firewalls and proxy servers normally block this kind of traffic. As all Internet browsers and servers support HTTP, it is better to communicate between various applications over HTTP, as a transport protocol. SOAP was created to accomplish this. SOAP provides a way to communicate between different applications running on different operating systems, with different technologies and programming languages. We need to use also Web Services Description Language (WSDL). WSDL is a document written in XML and it describes a Web service. WSDL provides a contract between a Web Service and the outside world. WSDL specifies the location of the service and the operations (or methods) the service exposes [1]. The Universal Discovery, Description and Integration protocol (UDDI) allows Web services to be registered so that they can be discovered by programmers and other Web services. We need to utilize both SOAP and WSDL because they are the most important Web services specifications, for better interoperability. In general, Web service is XML message that we send as the body of an HTTP POST request. The response is in XML that we then analyze. A typical Web Service is shown in Figure 1. XML, SOAP and Web Services give us a new paradigm for distributed computing [4], which includes XML as data, SOAP and HTTP as the protocols for moving data across the web, and Web Services protocols such as UDDI and WSDL for the discovery and connection those services. Web Services provider uses UDDI to register a web service with the web services repository. Web Services client uses UDDI to discover an appropriate web service and the way a client talks to a service provider depends on the WSDL.

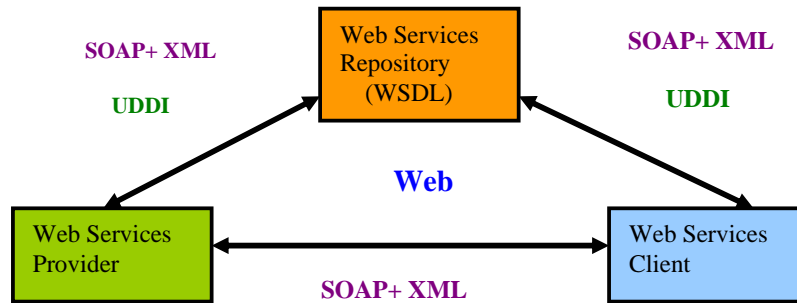


Figure 1. Typical Web Services

The Proposed Fault Tolerance Approach

In this paper, we concentrate on a simple web based application, as an example, for exchanging various process-parameters (for example, temperature and pressure values for a typical process) over Internet. We need to exchange correct values only. Any odd or wrong response value needs to be masked out. In this proposed fault tolerance approach, based on Single - Version fault tolerance model [2], we have used three redundant [3] XML documents (e.g., process_data1, process_data2 and process_data3) using individual XML Schemas (that is, document model). We have also used replicated SOAP functions (e.g., GetProcessdata1: to get data from process_data1, GetProcessdata2, GetProcessdata3 and GetProcessdata1Response etc). In WSDL, we have used three replicated operations. SOAP Requests (e.g., GetProcessdata1, GetProcessdata2 and GetProcessdata3 are sent to a HTTP server) and three sets of Request- Response (on temperature and pressure for the same process) are received from the Server. An application then, needs to vote upon the received three values on temperature and to accept the major value as the correct value of temperature.

processdata1.xml

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!-- process_data_1 -->
```

```
<process_data1 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\Documents and Settings\Administrator\My
Documents\w3c-gks\test\tft1.xsd">
```

```
    <temp/>
```

```
    <pressure/>
```

```
</process_data1>
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

```
<!-- Goutam Kumar Saha – Processdata1_Schema_1 -->
```

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
```

```

<xs:element name="pressure" type="xs:string"/>
<xs:element name="process_data1">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="temp"/>
      <xs:element ref="pressure"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="temp" type="xs:string"/>
</xs:schema>

```

processdata2.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- process_data_2 -->
<process_data2 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\Documents and Settings\Administrator\My
Documents\w3c-gks\test\tft2.xsd">
  <temp/>
  <pressure/>
</process_data2>
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- Goutam Kumar Saha – Processdata2_Schema_2 -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xs:element name="pressure" type="xs:string"/>
  <xs:element name="process_data2">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="temp"/>

```

```

        <xs:element ref="pressure"/>
    </xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="temp" type="xs:string"/>
</xs:schema>

```

processdata3.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- process_data_3 -->
<process_data3 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\Documents and Settings\Administrator\My
Documents\w3c-gks\test\tft3.xsd">
    <temp/>
    <pressure/>
</process_data3>
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- Goutam Kumar Saha – Processdata3_Schema_3 -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
    <xs:element name="pressure" type="xs:string"/>
    <xs:element name="process_data3">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="temp"/>
                <xs:element ref="pressure"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
<xs:element name="temp" type="xs:string"/>

```

</xs:schema>

The SOAP request:

The GetProcessdata1, GetProcessdata2 and GetProcessdata3 requests are sent to a http server. The request has a temp (i.e., temperature) parameter, and a pressure parameter whose values will be returned in the response. The namespace for the function is defined in "http://xyz.com/GetProcess" address.

POST /InProc HTTP/1.1

Host: www.xyz.com

Content-Type: application/soap+xml; charset=utf-8

Content-Length: nnn

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body xmlns:m="http://xyz.com/GetProcess">
    <m:GetProcessdata1>
      <m:ProcessTemp/>
      <m:ProcessPressure/>
    </m:GetProcessdata1>

    <m:GetProcessdata2>
      <m:ProcessTemp/>
      <m:ProcessPressure/>
    </m:GetProcessdata2>

    <m:GetProcessdata3>
      <m:ProcessTemp/>
      <m:ProcessPressure/>
    </m:GetProcessdata3>

  </soap:Body>
</soap:Envelope>
```

The SOAP response:

HTTP/1.1 200 OK

Content-Type: application/soap+xml; charset=utf-8

Content-Length: nnn

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
```

```

soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body xmlns:m="http://xyz.com/GetProcess">

    <m:GetProcessdata1Response>
      <m:ProcessTemp>54.0</m:ProcessTemp>
      <m:ProcessPressure>26.3</m:ProcessPressure>
    </m:GetProcessdata1Response>

    <m:GetProcessdata2Response>
      <m:ProcessTemp>54.5</m:ProcessTemp>
      <m:ProcessPressure>20.3</m:ProcessPressure>
    </m:GetProcessdata2Response>

    <m:GetProcessdata3Response>
      <m:ProcessTemp>54.5</m:ProcessTemp>
      <m:ProcessPressure>26.3</m:ProcessPressure>
    </m:GetProcessdata3Response>

  </soap:Body>
</soap:Envelope>

```

The WSDL

```

<message name="GetProcessdata1">
  <ProcessTemp type="xs:string"/>
  <ProcessPressure type="xs:string"/>
</message>
<message name="GetProcessdata2">
  <ProcessTemp type="xs:string"/>
  <ProcessPressure type="xs:string"/>
</message>
<message name="GetProcessdata3">
  <ProcessTemp type="xs:string"/>
  <ProcessPressure type="xs:string"/>
</message>

<message name="GetProcessdata1Response">
  <ProcessTemp type="xs:string"/>
  <ProcessPressure type="xs:string"/>
</message>
<message name="GetProcessdata2Response">
  <ProcessTemp type="xs:string"/>
  <ProcessPressure type="xs:string"/>
</message>
<message name="GetProcessdata3Response">
  <ProcessTemp type="xs:string"/>
  <ProcessPressure type="xs:string"/>
</message>

```

```
<portType name="ProcessData">
  <operation name="GetProcessdata">
    <input message="GetProcessdata1"/>
    <output message="GetProcessdata1Response"/>
    <input message="GetProcessdata2"/>
    <output message="GetProcessdata2Response"/>
    <input message="GetProcessdata3"/>
    <output message="GetProcessdata3Response"/>
  </operation>
</portType>
```

```
<binding type="ProcessData" name="gks1">
<soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http" />
<operation>
  <soap:operation
  soapAction="http://xyz.com/GetProcessdata"/>
  <input>
    <soap:body use="literal"/>
  </input>
  <output>
    <soap:body use="literal"/>
  </output>
</operation>
</binding>
```

Conclusion

We get reliable web services on using three folded XML documents and SOAP functions. However, such affordable overhead on time and memory space is not a problem for a web application in which we need to exchange sensitive data. We get a more dependable and reliable web services by using this single-version approach.

References

1. Simple Object Access Protocol (SOAP) 1.1, <http://www.w3.org/TR/SOAP/>
2. Goutam Kumar Saha, "Transient Software Fault Tolerance Using Single - Version Algorithm," ACM Ubiquity, Vol. 6(28), ACM Press, 2005.
3. Goutam Kumar Saha, "Software Implemented Fault Tolerance Through Data Error Recovery," ACM Ubiquity, Vol. 6(35), ACM Press, 2005.
4. Frank P. Coyle, "XML, Web Services and the Changing Face of Distributed Computing," ACM Ubiquity, Vol. 3(10), 2002.

Author's Biography

In his last seventeen years' research & development and teaching experience, he has worked as a scientist in LRDE, Defense Research & Development Organization, Bangalore, and at the Electronics Research & Development Centre of India, Calcutta. At present, he is with the Centre for Development of Advanced Computing, Kolkata, India, as a Scientist-F. He has authored one hundred research papers in various International and national Journals and Conference proceedings etc. He is a senior member in IEEE, Computer Society of India, a member in ACM, W3C ITS Working Group, and a Fellow member in IETE. He is an associate editor of ACM's publication Ubiquity. Saha has received various awards, scholarships and grants from national and international organizations, and is a referee for CSI Journal, AMSE Journal (France), IEEE Magazine and IJCPOL etc. His field of interest is on software based fault tolerance, dependable computing and NLP. He can be reached via sahagk@gmail.com or via gksaha@rediffmail.com.