An Automated WSDL Generation and Enhanced SOAP Message Processing System for Mobile Web Services*

Gil Cheol Park[1], Seok Soo Kim[1], Gun Tae Bae[1], Yang Sok Kim[2] and Byeong Ho Kang[2]
[1]*School of Information & Multimedia, Hannam University*
133 Ojung-Dong, Daeduk-Gu, Daejeon 306-791, Korea
gcpark@mail.hannam.ac.kr
[2]*School of Computing, University of Tasmania*
Sandy Bay, Tasmania 7001, Australia
{yangsokk, bhkang}@utas.edu.au

Abstract

Web services are key applications in business-to-business, business-to-customer, and enterprise applications integration solutions. As the mobile Internet becomes one of the main methods for information delivery, mobile Web Services are regarded as a critical aspect of e-business architecture. In this paper, we proposed a mobile Web Services middleware that converts conventional Internet services into mobile Web services. We implemented a WSDL (Web Service Description Language) builder that converts HTML/XML into WSDL and a SAOP (Simple Object Access Protocol) message processor. The former minimizes the overhead cost of rebuilding mobile Web Services and enables seamless services between wired and wireless Internet services. The latter enhances SOAP processing performance by eliminating the Servlet container (Tomcat), a required component of typical Web services implementation. Our system can completely support standard Web Services protocol, minimizing communication overhead, message processing time, and server overload. Finally we compare our empirical results with those of typical Web Services.

1. Introduction

As the Internet potentials of the mobile Internet are widely understood, mobile Internet services become a major mediator in information delivery and in business transactions. Mobile Internet services, however, still have physical devices, network and content limitation. Firstly, mobile devices are limited by system resources such as smaller screens and less convenient input devices. Secondly, wireless networks have less bandwidth, less connection stability, less predictability and a lack of standardized and higher costs [1, 2]. Lastly, mobile Internet services also have content limitation because the amounts of available mobile content are still smaller than that of wired Internet services, and the consistency between wired and wireless Internet services is very critical. Physical device and network limitation make supporting common Internet standards, such as HTML, HTTP, and TCP/IP, difficult because they are inefficient over mobile networks. Therefore, new protocols such as WAP (Wireless Application Protocol) and WML (Wireless Markup Language) are proposed to overcome these limitations. Content limitation encourages researchers to find methods that support reuse of current wired Web information. Some researchers focus on the conversion of HTML documents to mobile Internet serviceable WML documents and direct access to databases, to provide efficient information delivery in the wireless environment [3-7]. However, these researchers do not focus on the capability that allows applications to interact over the Internet in an open and flexible way, but on the capability that provides dynamic wireless Internet service according to different network and device environments. The former goal can be achieved by Web Services, because interactions between Web Services applications are expected to be independent from the platform, programming language, middleware, and applications involved. For this reason, Web Services is regarded as key applications in business-to-business, business-to-customer, and enterprise applications integration solutions [8].

In this paper, we focus on the following two issues:

Automated HTML/XML conversion to WSDL: The goal system should dynamically generate WSDL files from existing HTML/XML files. A markup language converting system is implemented to convert HTML/XML to WSDL automatically.

Improve SOAP processing efficiency: One main limitation of Web Service is its inefficient performance

* This work is supported by the Asian Office of Aerospace Research and Development (AOARD) (Contract Number: FA5209-05-P-0253)

compared with other distributed computing approaches like Java RMI, CORBA, and DCOM (Distributed Component Object Model). The use of HTTP and XML represents a significant increase in run-time cost Web Services solutions [9-13]. We propose a method that enhances SOAP processing by changing service architecture. The typical SOAP processing system requires the Web Servlet container (e.g. Tomcat) to execute SOAP. It requires additional process and communication port. Our hypothesis is that if a system processes the SOAP message directly, without help from Web Servlet container, the SOAP performance improves.

The paper is organized as follows: Section 2 summarizes relevant research results, including HTML conversion and Web services technology. Section 3 explains our HTML conversion implementation, while Section 4 illustrates our SOAProc system implementation. In Section 5 we compare our system's performance with the typical Web Services implementation approach. Finally, conclusions and recommendations for further work are described in Section 6.

2. Literature Review

Researchers usually focus on HTML/WML conversion because the WAP is an alternative protocol for HTML in wireless Internet services using Wireless Markup Language (WML), a small subset of Extensible Markup Language (XML), to create and deliver content. Kaasinen et al. [3] and Dugas [14] suggested an HTML/WML conversion proxy server, which converts HTML-based Web content automatically, and on-line, to WML. Saha et al. [6] suggested a middleware that is seamless and transparently translates a Web site's existing contents to mobile devices. Kurbel and Dabkowski [4] proposed a dynamic user tailed WML content generation by using JSP (Java Server Pages) and JDBC-ODBC driver. Magnusson and Stenmark [15] suggested a CMS-based approach to visualise Web information in a PDA. Pashtan et al. [7] stressed context-aware wireless Web services, which can adapt their content to the user's dynamic content. Again, we wish to stress these researchers focus only on HTML/WML conversion, not Web Services compliable conversion. Therefore, in spite of their importance, application integration aspects inside and outside enterprises have not been seriously considered by the researchers. As the impotence of application integration over the Internet becomes more important, nowadays Web Services are critical to any Internet services. For this reason, we propose a method that converts HTML to WSDL. The WSDL files are used to provide Web Services with SOAP messaging protocols. More detailed explanation about the Web Services and its implementation issues are discussed in the following Section.

Web Services, as defined by the W3C Web Services Architecture Working Group, are "software applications identified by a URI, whose interfaces and bindings are capable of being defined, described, and discovered as XML artefacts. A Web service supports direct interactions with other software agents using XML-based messages exchanged via Internet-based protocols."[16]

There are several Web Services implementation methods, which differ in their support for class binding, ease of use, and performance [13]. Among them Apache Axis (Apache eXtensible Interaction System) with Tomcat is a popular implementation method. The Apache Axis project is a follow-on to the Apache SOAP project and currently has reached version 1.2.1, but it's not part of the Apache SOAP project. Axis is a completely new rewrite with emphasis on flexibility and performance. It supports HTTP SOAP request/response generation, SOAP message monitoring, dynamic invocation, Web service deployment, and automatic WSDL generation for Web services.

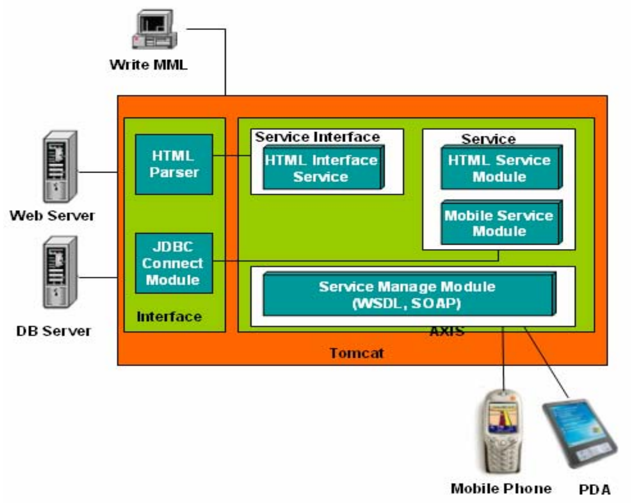

Figure 1- Typical Mobile Web Service Implementation with AXIS and Tomcat

Figure 1 illustrates this standard mobile Web service implementation. A Web Servlet container, like Tomcat, is required to provide mobile Web services with Axis. For wireless Internet service, the server administrator should write MML (Made Markup Language) to parse Web contents by using the administrative tool. A MML is used to generate service request forms or service results by dynamically parsing the existing Web contents and sending them to relevant model and clients. When a client, whether it is wireless or wired client, requests Web service via SOAP request, Apache Tomcat transfers it to Axis. Axis interfaces the SOAP request message into a relevant service by using the service management function. Service providing models are interfaced by using a WSDL module is provided by Axis. By implementing SOAP and distributed computing service, the system architecture can have a lightweight thin client structure and the service can be provided in a flexible way. However, this implementation is not efficient because it requires additional process for Web Servlet engine (Tomcat) and communication port. For this reason, we propose an alternative system that can process SOAP messages without using Web Servlet engine.

3. HTML/WSDL converter

Our HTML/WSDL content converter system consists of three sub-modules: the rule script, the script engine, and the markup language converter. The rule script stores rules for content reformatting rules, which are created by the user with the management program. The rules include personalization information and display structuring information of mobile devices. Secondly, the script engine reconstructs contents by using script rules and client (device) information. The markup language converter transforms markup language if the markup language that the server provides differs from what the client can process. Script rules are created as follows. If a Web site address is supplied, our system reads and parses the Web site information. The parsed information is then presented by using a DOM tree, in which the user can select and save node information to be served as wireless Internet content. The JML (Java Mark-up Language) editor defines XML tags and attributes of the saved items. TITLE, BASEURL, LINK, HREF, CONTENT, and ELEMENT are XML tag examples and many attributes are also available to customize mobile contents.

Figure 2 illustrates the operation of the converter. The user accesses the HTML/WSDL converter system via mobile devices and mobile networks. The converter gets the user's mobile device information such as display size and color, and URL information that the user requests by using the protocol detector. After getting this information, the converter requests URL information from the Web server. The Web server generates a HTML response message and sends it to protocol detector. The protocol detector then passes this HTTP response message to the selector with client information. The selector chooses WSDL information from the HTTP response message by using the script rules and returns this information to the protocol detector. The protocol detector in turn sends this information to the translator, which performs Mark-up language transformation, image transformation, paging and cashing. Lastly, the converter sends this processed result to the user.

4. SOAP Message Processor

In the Web Services, XML based SOAP messages are used when the clients request Web Services from the server or when the server sends Web Service response messages to the clients.

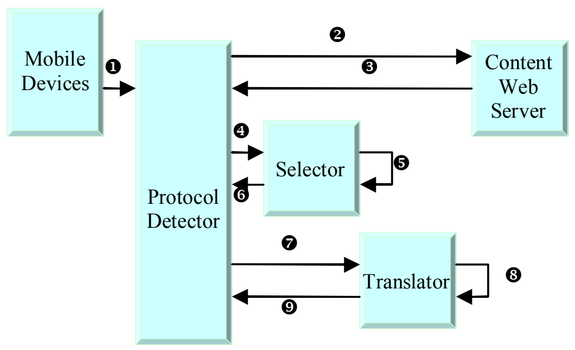

Figure 2- HTML/WSDL Content Converter Operation

In the standard Web Services implementation this is supported by Tomcat and AXIS. We developed a SOAP message processing system, called SOAProc, because typical architecture causes inefficiency by spawning new process and adding additional communication port. The SOAProc directly processes the SOAP request and response messages without using Servlet engine. Figure 3 illustrates our Web Services system implementation architecture, in which the SOAProc and the WSDL builder are used. The most significant difference between the standard system (see Figure 1) and our implementation (see Figure 3) is that our system does not include Tomcat. Instead of using Tomcat's WSDL and SOAP supporting function, WSDL files are directly generated by the WSDL builder and SOAP messages are processed by the SOAProc system.

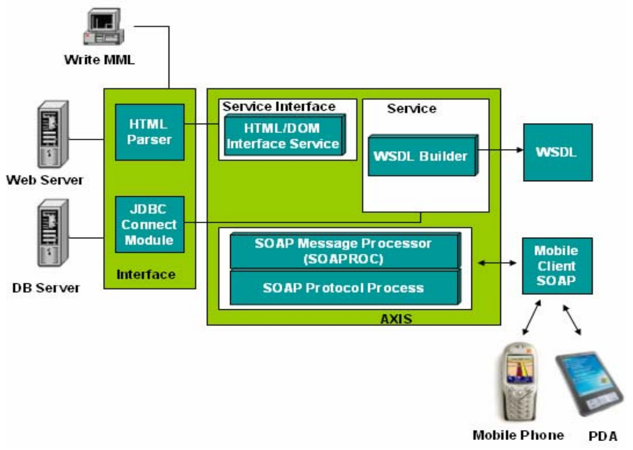

Figure 3 – Mobile Web Service using the SOAProc and the WSDL builder

4.1 SOAP Message Structure

Figure 4 illustrates an example of a SOAP message. The Header element is intentionally omitted in this example. <ns1: IntranetLogin …> indicates IntranetLogin method that will be called. The tags between

<ns1:IntranetLogin…> tag are parameters of method IntranetLogin, such as <userid> … </userid>, <pass> … </pass>, and <sessionidtag> … </sessionidtag>.

```
RequestSoapMassage.xml
```

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv =
"http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
<soapenv:Body>
    <ns1:IntranetLogin soapenv:encodingStyle
    =http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:ns1="urn:wireserver">
        <useridxsi:type="xsd:string">
            test
        </userid>
        <pass xsi:type = "xsd:string">
            pass
        </pass>
        <sessionidtag xsi:type="xsd:string">
            sessionidtag
        </sessionidtag>
    </ns1:IntranetLogin >
</soapenv:Body>
</soapenv:Envelope>
```

Figure 4 - SOAP Message Example

4.2 SOAP Request Message Analysis

The algorithm that is used to analyses the method and its parameters of SOAP request messages is as follows:

Step1: Gets the SOAP messages

The system generates a FileInputStream of RequestSoapMessage.xml (fis).

```
FileInputStream fis = new
FileInputStream("RequestSoapMassage.xml")
SOAPEnvelope env = new SOAPEnvelope(fis);
```

Then the system gets the SOAP message from the above FileInputStream.

```
SOAPBodyElement sbe = env.getFirstBody();
```

Step2: Gets the SOAP Body

The system extracts the SOAP Body from the SOAP message.

```
sbe.getName();
```

Step3: Analysing SOAP Body

The system finds IntranetLogin part of <ns:IntranetLogin …> from the Body of the SOAP message.

```
ArrayList al = sbe.getChildren();
```

The system creates array list of items between <ns:IntranetLogin…></ ns:IntranetLogin> in the Body of the SOAP message.

```
for(int i = 0 ; i < al.size() ; i++){
   MessageElement me = MessageElement)(al.get(i));
   System.out.println((i+1)+" th " +
   me.getName()+"'s value is " + me.getValue());
}
```

The system iteratively analyses the item list to get MessageElement like <userid> … </userid>, <pass> … </pass>, and <sessionidtag> … </sessionidtag>. In each iteration, the item's name and value are obtained by me.getName() and me.getValue()method. For example, if the system uses example in Figure 6, <userid … >test </userid> is in the first item of item list and 'userid' and 'test' are name and value, which can be get by using iterative analysis. The system can generate response message to the clients by using this result.

4.3 SOAP Response Message Generation

Our system analyses the client SOAP request message and sends the analyzing result to the Web server. When the Web server system generates a HTTP response message, our system generates a SOAP response message by using it. In this part, we describe a response generation algorithm, in which we assume the response result is a string type. The response results can be sent by a single or binary array. The result values and method namespace value are assumed as follows.

```
String str = "test1Respons";
String strElement = "test1Return";
String elementValue = "aaa";
String nameSpaceURI = "urn:stringtest";
```

The SOAP response message is generated as follows:

Step 1: Generate SOAP Basic Element

Our system generates the new SOAP response message by creating a new Envelope element and Body element.

```
SOAPEnvelope env = new SOAPEnvelope();
env.getBody(); //SOAPBody
SOAPBodyElement body = new SOAPBodyElement();
```

The SOAP message that is created until now is as follows:

```
<soapenv:Envelope
xmlns:soapenv=http://schemas.xmlsoap.org/soap/enve
lope/
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
<soapenv:Body />
</soapenv:Envelope>
```

Step 2: Add Content to SOAP Message

The SOAP contents are created by adding the above results values.

```
RPCParam rpcParam = new RPCParam(strElement,
elementValue) ;
RPCHeaderParam rpcHeaderParam = new
RPCHeaderParam(rpcParam);
RPCElement rpcElement = new RPCElement(str);
rpcElement.addParam(rpcParam);
rpcElement.setEncodingStyle("http://schemas.xmlsoa
p.org/soap/encoding/");
  rpcElement.setNamespaceURI(nameSpaceURI);
</soapenv:Envelope>
```

After adding the contents, the SOAP response message is as follows:

```
<ns1:test1Respons
soapenv:encodingStyle="http://schemas.xmlsoap.org/
soap/encoding/"
xmlns:ns1="urn:stringtest">
<test1Return
si:type="xsd:string">aaa</test1Return>
</ns1:test1Respons>
```

Step 3: Error Handling

If there are any errors in the Web server processing, the following code creates error messages.

```
SOAPFault soapFault = env.getBody().addFault();
soapFault.setFaultCode("code error\n");
soapFault.setFaultActor("action error\n");
soapFault.setFaultString("string error\n");
```

Step 4: Add SOAP Body Element

Lastly, the following code adds the SOAP Body element when there is no error. SOAP response generation is completed by doing this.

```
env.addBodyElement(rpcElement);
```

Figure 5 illustrates a complete SOAP response message that is generated by our system without Tomcat.

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/env
elope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
<soapenv:Body>
   <ns1:test1Respons
soapenv:encodingStyle="http://schemas.xmlsoap.org/
soap/encoding/" xmlns:ns1="urn:stringtest">
   <test1Returnxsi:type="xsd:string">
         aaa
   </test1Return>
   </ns1:test1Respons>
</soapenv:Body>
</soapenv:Envelope>
```

Figure 5 - SOAP Response Message

5. Experiment

5.1 Method

The experiment is focused on the performance evaluation of our mobile Web service system. Two sets of systems are prepared for our experiment. The first system is implemented with standard Web Services architecture as explained in Section 2. This implementation requires Tomcat Servlet container with AXIS. The second implementation is based on our approach. Where there is no Servelet container with the SOAP message processing performed by the SOAProc system and WSDL created by the WSDL builder. We conducted a simulated performance comparison experiment. Figure 6 illustrates the experiment process. If a client requests Web services by submitting a SOAP request, the experiment system analyses the SOAP message and sends a HTTP request to the content Web servers. If the experiment system receives a HTTP response message form the Web server, it generates WSDL and sends a SOAP response message to the clients' mobile device.

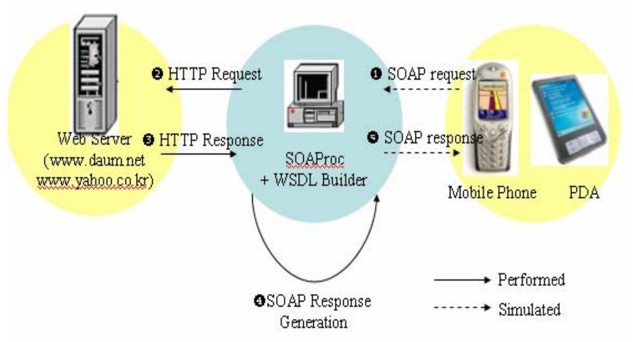

Figure 6 – Experiment System Procedure

SOAP requests are simulated by the mobile client simulation program, which connects to the experiment system and sends several SOAP request messages. There are time intervals, from 1 to 10 seconds between SOAP requests. If the connection is closed, the simulation program continually tries to connect to the experiment system. We assumed that there were 200 users at the same time. SOAP requests were created by four client programs and each program generated 50 threads at the same time. We chose two public Web sites [www.daum.net (dictionary) and www.yahoo.co.kr (stock)], which role as the content Web servers in our experiment. We assumed two kinds of specific information - dictionary and stock - are required by the user from these Web servers. Each service's timeout is 30 seconds.

The following results were collected to compare two experiment systems:

- Test time: how many seconds were consumed for the test.
- Number of Requests: how many requests were generated within test time.
- Connection Timeout: the connection numbers that were not connected within the request timeout.
- Connection Refuse: the request numbers that could not be connected because the server was busy.
- Connection Handshake Error: the number of session configuration failures after connection
- Connection Trial Time: How many times the client could not connect to the server.

- Request Timeout: how many times the timeout was exceeded.

5.2 Results

Table 1 summarizes the experiment as results, which illustrate an enhanced performance in all categories. Though the test time of our system is shorter than that of the standard system, the total number of requests is greater than that of standard system and the timeout number is less than that of the standard system. For example, whist the average request per second of our system is 17.94, that of standard system is 9.64. There are many connection errors in the standard system. Only some portion of 200 requests is successfully connected to the server while the others get a "refused" message from the server. However, those kinds of connection failures do not happen in our system.

Table 1 – Experiment Results

	SOAProc System	Standard System
Test time	29,400	46,200
Total Request	524,573	445,422
Connection Timeout	0	435
Connection Refused	0	18,960
Connection Handshake Error	0	513
Connection Trials	243	22,534
Request Timeout	756	119,891

6. Conclusions

Mobile Web services are critical solutions in the Internet service integration architecture. In this research we proposed a new Web Service architecture by implementing two significant systems. Firstly, the HTML/WSDL converter can support reusing current HTML based contents. This is essential for saving developing or maintenance costs and serving seamless Internet services both wired and wireless. Secondly, we proposed a new SOAP message processing system to diminish SOAP latency problems by eliminating the Tomcat Servelet container in the Web Services implementation. The SOAP request and response messages are directly processed by the SOAProc system. We can implement an alternative mobile Web Services system by using these two systems without violating standard Web Services protocols. Our system can process more service request about doubly efficient than that of typical Web service implantation with very small connection errors.

7. References

1. Siau, K., E.P. Lim, and Z. Shen, *Mobile commerce: promises, challenges, and research agenda.* Journal of Database Management, 2001. **vol.12, no.3**: p. 4-13.
2. Kim, H., et al. *An Empirical Study of the Use Contexts and Usability Problems in Mobile Internet*. in *35th Annual Hawaii International Conference on System Sciences (HICSS'02)*. 2002.
3. Kaasinen, E., et al., *Two approaches to bringing Internet services to WAP devices.* Computer Networks, 2000. **33**(1-6): p. 231-246.
4. Kurbel, K. and A. Dabkowski. *Dynamic WAP content Generation with the use of Java Server Pages*. in *Web Databases/Java and Databases: Persistence Options (Web&DB/JaDa)*. 2002. Erfurt, Germany.
5. Metter, M. and R. Colomb. *WAP Enabling Existing HTML Applications*. in *First Australasian User Interface Conference*. 2000.
6. Saha, S., M. Jamtgaard, and J. Villasenor, *Bringing the wireless Internet to mobile devices.* Computer, 2001. **vol.34, no.6**: p. 54-58.
7. Pashtan, A., S. Kollipara, and M. Pearce, *Adapting content for wireless Web services.* IEEE Internet Computing, 2003. **7**(5): p. 79-85.
8. Farrell, J.A. and H. Kreger, *Web services management approaches.* IBM Systems Journal, 2002. **vol.41, no.2**: p. 212-227.
9. Seshasayee, B., K. Schwan, and P. Widener, *SOAP-binQ: high-performance SOAP with continuous quality management.* Proceedings. The 2nd IEEE International Conference on Distributed Computing Systems, 2004: p. 158-165.
10. Kohlhoff, C. and R. Steele, *Evaluating SOAP for high performance applications in capital markets.* Computer Systems Science and Engineering, 2004. **19**(4): p. 241-251.
11. Chiu, K., M. Govindaraju, and R. Bramley, *SOAP for High Performance Computing*, in *11 th IEEE International Symposium on High Performance Distributed Computing HPDC-11 20002 (HPDC'02)*. 2002, Indiana University. p. 246.
12. Chiu, K., M. Govindaraju, and R. Bramley. *Investigating the Limits of SOAP Performance for Scientific Computing*. in *11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11 '02)*. 2002.
13. Davis, D. and M. Parashar. *Latency Performance of SOAP Implementations*. in *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*. 2002.
14. Dugas, R., *WWW Unplugged: An HTML to WML transcoding proxy.* 2001.
15. Magnusson, M. and D. Stenmark. *Mobile Access to the Intranet: Web Content Management for PDAs*. in *Americas Conference on Information Systems 2003*. 2003.
16. W3C, *Web Services Architecture Requirements.* Web Services Architecture Requirements, 2002.