An Enhanced SOAP Message Processing System for Mobile Web Services

Seok-Soo Kim, Gil-Cheol Park, *Member, KIMICS*

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 that performs SOAP message handling, chain and handling of server requests. 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 main contributions are to overcome the latency problem of current Web Services and to provide an easy mobile Web service 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.

Index Terms—WSDL, SOP, Mobile, Web, Servlet

I. INTRODUCTION

Mobile internet services enable users to access the internet from any location at any time providing flexible personalized information according to users' location and their information needs [1]. The mobile internet can provide various value added services in addition to basic communication services. As the internet capabilities are widely understood and wireless technologies advance, mobile internet services will soon be a major mediator in information delivery and in business transactions[2].

Mobile internet services, however, still have physical devices, network and content limitations. 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]. Lastly, mobile internet services also have content limitations 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 limitations 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 address these issues. Content limitations encourage researchers to find a method that can support reusing 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].

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. In fact, the former goal can be achieved by implementing Web Services. If the implementation is successful, interactions between applications are expected to be independent from the platform, programming language, middleware, and implementation of the applications involved. Nowadays Web Services become key applications in business-to-business, business-to-customer, and enterprise applications integration solutions[4]. In this paper, we focus on an automated content conversion from HTML/XML to Web Services capable data format, called WSDL (Web Services Description Language).

Web Services require specific messaging protocols known as SOAP for interaction. One main issue of SOAP implementation is the latency of SOAP execution [5]. We view that the latency problem is caused by the current SOAP message processing system architecture. The current SOAP processing system requires the Web Servlet container (e.g. Tomcat) to execute SOAP. Our hypothesis is that if a system processes the SOAP message directly, without help from Web Servlet container, the SOAP performance improves. To examine this hypothesis we implemented a SOAP message processing system, called SOAProc.

The paper is organized as follows: Section 2 summarises relevant research results, including HTML conversion and Web services technology. Section 3 explains our HTML conversion implementation.

Manuscript received August 29, 2005.

"This work was supported by a grant No. (R12-2003-004-03003-0) from Ministry of Commerce, Industry and Energy"

Seok-Soo Kim, Gil-Cheol Park are with Department of Multimedia, Hannam University, Daejeon, Korea(e-mail : sskim@hannam.ac.kr)

II. LITERATURE REVIEW

A. *Adapting Content for Wireless Internet Services*

Nowadays users can access internet services by employing various devices including PC, mobile phone and PDA. Therefore, internet service providers should cope with these diversities to satisfy users' ubiquitous internet access needs. In this case, the main problems are how the service providers can provide seamless service for wired or wireless internet service content without significant additional costs and how they can provide sufficient content to clients. Adapting content from wired Web content to wireless content is regarded as a promising solution.

Researchers usually focuses 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. (2000) and Dugas (2001) suggested an HTML/WML conversion proxy server, which converts HTML-based Web content automatically, and on-line, to WML. Saha et al. (2001) suggested a middleware that is seamless and transparently translates a Web site's existing contents to mobile devices. They also specified application integration, device independence, and optimal user interface as key challenges. Kurbel and Dabkowski (2002) proposed a dynamic user tailed WML content generation by using JSP (Java Server Pages) and JDBC-ODBC driver. Magnusson and Stenmark (2003) suggested a CMS-based approach to visualise Web information in a PDA. Pashtan et al. (2003) 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, including the mobile internet service. They are regarded as key applications in business-to-business, business-to-customer, and enterprise applications integration solutions[6]. 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.

B. *Application Integration with Web Services*

A Web Service, as defined by the W3C Web Services Architecture Working Group, is "a software application 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."

Figure 1 illustrates the Web Service protocol stack in terms of the internet reference model. This stack is a collection of standardized protocols and application programming interfaces (APIs) that allows individuals and applications to locate and utilize Web Services. The Web Service Layer could be placed between the Transport and Application Layer and is based on several standard internet protocols, whereby the lowest three of which (WSDL, SOAP, and typically HTTP) should be supported by all Web Services implementations for interoperability.

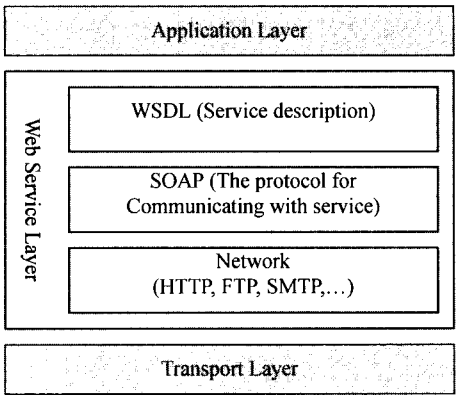

Fig. 1 Web service protocol stack

Each vendor, standard organization, or marketing research firm, defines Web Services in a slightly different way. For this reason, the architecture of a Web service stack varies from one organization to another. In spite of this fact, Web Services can be considered a universal client/server architecture that permits disparate systems to communicate with each other without using proprietary client libraries. Therefore, this architecture simplifies the development process by effectively eliminating code dependencies between client and server [7].

Gottschalk (Gottschalk, Graham et al. 2002) explains Web Services from the service-oriented architecture. The service provider creates Web Services and its service definitions and publishes the services with a service discovery agency. The interface of a Web service is described in an XML format called the Web Services Description Language (WSDL) (W3C 2005). A WSDL file contains descriptions of one or more interfaces and binding information for one or more services.

Typically the role of the discovery agency will be fulfilled by a registry, such as UDDI (Universal Description, Discovery and Integration). UDDI allows additional information describing the hosting business, and makes associations with the taxonomy to be published in association with the WSDL description so that others can access the service using a wide variety of search criteria, including category-based searches [4].

Once the Web Services are published, a requester may find them via the registry interface. The invocation of a service involves sending an XML message to the service provider and receiving an XML message in return. These XML interactions are governed by Simple Object Access Protocol (SOAP). A SOAP message is fundamentally a one-way transmission between SOAP nodes, from a SOAP sender to a SOAP receiver, but the SOAP messages are expected to be combined by applications to implement more complex interaction patterns. SOAP messages must

be carried on a communication layer, which is usually the HTTP, but any other transport protocol such as SMTP, MIME, and FTP for public domains as well as CORBA and Message Queuing protocols for private domains, could be used (W3C 2003).

A SOAP message consists of the following elements :

- Envelope : The Envelope element serves as a container for the other elements of the SOAP message. As it is the top element, the Envelope is the message.
- Header : The Header element is to encapsulate extensions to the message format without having to couple it to the payload or to modify the fundamental structure of SOAP. This allows extensions like transactions, encryption, object references, billing, and countless others to be added over time without breaking the specification. The Header element is optional therefore it may be eliminated.

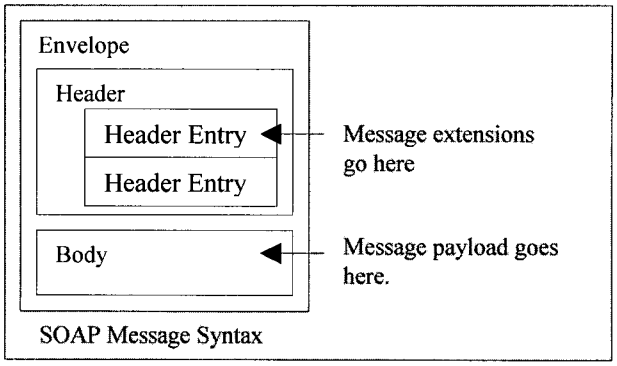

Fig. 2 SOAP Message Syntax

- Body : The Body element of a SOAP message is the location for application-specific data. It contains the payload of the message, carrying the data that represent the purpose of the message. It could be a remote procedure call, a purchase order, a style sheet, or any XML that needs to be exchanged using a message (Cauldwell, Chawla et al. 2001).

III. IMPLEMENTATION OF 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. In the standard Web Services implementation this is supported by Tomcat and AXIS. This architecture causes in efficiency as explained in Section 2.3. For this reason, we developed a SOAP message processing system, called SOAProc which 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 3) and our implementation (see Figure 4) 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.

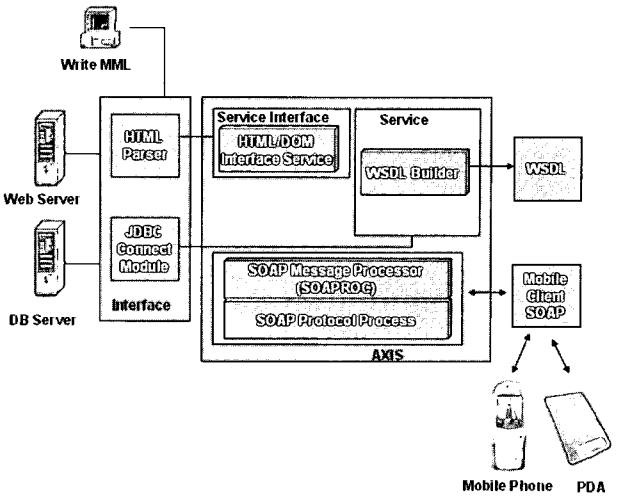

Fig. 3 A New Mobile Web Service Implementation by using the SOAProc and the WSDL builder

The next Section describes how the SOAProc system processes SOAP request and response messages in this implementation.

A. *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>
```

Fig. 4 SOAP Message Example

B. *Analysing SOAP Request Message with SOAPROC*

Algorithm that analyses the method and its parameters of SOAP request messages are presented in Figure 7. The algorithm is as follows:

Step1: Gets the SOAP messages
The system generates a FileInputStream of RequestSoap Message.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());
}
```

SoapMassageToServer.java

```
import org.apache.axis.message.SOAPEnvelope;
import org.apache.axis.message.SOAPBodyElement;
import org.apache.axis.message.MessageElement;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.ArrayList;
class SoapMassageToServer{
   public static void main(String[] args){
      try{
          FileInputStream      fis      =      new
          FileInputStream("RequestSoapMassage.xml")
          ;
          SOAPEnvelope env = new SOAPEnvelope(fis);
          system.out.println("soap Meassage");
          system.out.println(env);
          System.out.println();
          SOAPBodyElement sbe = env.getFirstBody();
          System.out.println("Service  Name  :  "  +
          sbe.getName());
          System.out.println("NamespaceURI  :  "  +
          sbe.getNamespaceURI());
          ArrayList al = sbe.getChildren();
          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());}}
      catch (IOException e){
          System.out.println(e);}
      catch (org.xml.sax.SAXException e){
         System.out.println(e);
      }
   }
}
```

Fig. 5 SOAP Message Processing Algorithm

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.

C. *Generating SOAP Response Message with SOAPROC*

Our system analyses the client SOAP request message and sends the analysing 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.

The response generation algorithm is illustrated in Figure 6 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/
envelope/
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.xmlsoap.
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 7 illustrates a complete SOAP response message that is generated by our system without Tomcat.

SoapMassageToClient.java

```
import org.apache.axis.message.SOAPEnvelope;
import org.apache.axis.message.SOAPBodyElement;
import org.apache.axis.message.RPCElement;
import org.apache.axis.message.RPCHeaderParam;
import org.apache.axis.message.RPCParam;
import javax.xml.soap.SOAPException;
import javax.xml.soap.SOAPFault;
class MakeServerSoap {
 public static void main(String[] args){
   try{
      String str = "test1Respons";
      String strElement = "test1Return";
      String elementValue = "aaa";
      String nameSpaceURI = "urn:stringtest";
      SOAPEnvelope env = new SOAPEnvelope();
      env.getBody(); //SOAPBody
      SOAPBodyElement body = new SOAPBodyElement();
      RPCParam rpcParam = new RPCParam(strElement,
      elementValue) ;
      RPCHeaderParam rpcHeaderParam = new
      RPCHeaderParam(rpcParam);
      RPCElement rpcElement = new RPCElement(str);
      rpcElement.addParam(rpcParam);
      rpcElement.setEncodingStyle("http://schemas.
      xmlsoap.org/soap/encoding/");
      rpcElement.setNamespaceURI(nameSpaceURI);
      env.addBodyElement(rpcElement);
      System.out.println(env);
      SOAPFault soapFault =
      env.getBody().addFault();
      soapFault.setFaultCode("code error\n");
      soapFault.setFaultActor("action error\n");
      soapFault.setFaultString("string error\n");
      System.out.println(soapFault);
   }
   catch(SOAPException e){
      System.out.println(e);
   }
 }
}
```

Fig. 6 SOAP Response Algorithm

```
<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: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>
```

Fig. 7 SOAP Response Algorithm

This Section demonstrates how our SOAP message processing system (SOAProc) analyses the SOAP request message and generates the SOAP response message without a Servlet engine. We suggest that the performance of SOAP processing is enhanced by applying this method. In the following Section, we provide a performance comparison result with a typical mobile Web Service system.

IV. CONCLUSION

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 experiment results demonstrate that the SOAP request processing performance of our approach is significantly better than that of the standard Web service implementation. Our system can process more service request about doubly efficient than that of typical Web service implantation with very small connection errors.

REFERENCES

[1] Siau, K., E. P. Lim, et al. (2001). "Mobile commerce: promises, challenges, and research agenda." Journal of Database Management vol.12, no.3: 4-13.
[2] Senn, J. A. (2000). "The emergence of m-commerce." Computer 33(12): 148-150.
[3] Kaasinen, E., M. Aaltonen, et al. (2000). "Two approaches to bringing Internet services to WAP devices." Computer Networks 33(1-6): 231-246.
[4] Farrell, J. A. and H. Kreger (2002). "Web services management approaches." IBM Systems Journal vol. 41, no. 2: 212-227.
[5] Chiu, K., M. Govindaraju, et al. (2002). Investigating the Limits of SOAP Performance for Scientific Computing. 11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11 '02).
[6] Ferris, C. and J. Farrell (2003). "What are Web services?" Communications of the ACM 46(6): 31.
[7] Myerson, J. M. (2002). Web Services Architectures: How they stack up.

Seok-Soo Kim

Received a B.S. degree in computer engineering from Kyungnam University 1989, and M.S. degree in Information engineering from Sungkyun-kwan University 1991 and Ph D. degree in Information engineering from Sungkyunkwan University 2002.

In 2003 he joined the faculty of Hannam University where he is currently a professor in Department of Computer & Multimedia Engineering. His research interests include Multimedia Communication systems, Distance learning, Multimedia Authoring, Telemedicine, Multimedia Programming, Computer Networking. Information Security. He is a Member of KICS, KIMICS, KIPS, KMS, and DCS.

Gil-Cheol Park

Received a M.S. degree in computer engineering from Soongsil University 1985, and Ph D. degree in Information engineering from Sungkyunkwan University 1994.

In 1998 he joined the faculty of Hannam University where he is currently a professor in Department of Computer & Multimedia Engineering. His research interests include Multimedia Communication systems, Distance learning, Multimedia Authoring, Multimedia Programming, Computer Networking. Information Security. He is a Member of KICS, KIMICS, KIPS, KMS, and DCS.