Wanjun HUANG, Uwe ROTH, Christoph MEINEL:
Improvement to the Smart Data Server with SOAP
in Proceedings of the 2nd WSEAS International Conference on Multimedia, Internet and Video Technologies (ICOMIV 2002), WSEAS Press, Skiathos, Greece, pp. 1141-1145, 9, 2002.

Improvement to the Smart Data Server with SOAP*

WANJUN HUANG, UWE ROTH, CHRISTOPH MEINEL
Institute of Telematics
Bahnhofstr. 30-32,D-54292, Trier
GERMANY
{huang,roth,meinel}@ti.fhg.de

Abstract: - As a distributed computing middleware, the Smart Data Server (SDS) provides a general framework for easy-to-build environment-independent modules with distributed functionality and a secure mechanism for invocation. But there are still some problems in the widespread cooperation with other distributed computing system and client terminal. Nevertheless, SOAP, as a new promising simple object access protocol, provides a mechanism based on XML for the structured message exchanging in the distributed environment. Here we take SOAP as the protocol for message transport between the remote client and the Smart Data Server instead of the its own "Information Package Transfer Protocol" (IPTP), and the SDS gains some new advantages: it can cooperate with other distributed computing solutions, such as CORBA, DCOM, RMI and so on; the SDS can be applied more widely for the SOAP message transferred through HTTP, a wide and most common used protocol in the Internet.

Key-Words: - SDS, Middleware, Distributed Computing, SOAP, Web Services

1 Introduction

With fast growth of the Internet, web service is not only limited to the web browsing, which can't satisfy people's increasing requirement any longer. Though the processor's computing ability has been increasing rapidly every several months, the application prospect will be more wide and fascinating if millions of computers around world can work together, which is the core idea of the distributed computing. Technologies involved in distributed computing can be commonly found in the following solutions: CORBA, DCOM, J2EE and so on. CORBA means "Common Object Request Broker Architecture" and belonged to OMG, which is an open and vendor-independent specification for an architecture and infrastructure that computer applications use to work together over networks [4]. CORBA has provides mapping from IDL to C, C++, ADA and Java, so it's language and platform independent. CORBA has provided a completive mechanism for distributed object computing, in which the Internet Inter-ORB Protocol (IIOP) has been adopted to support communication between different "Object Request Broker" (ORB) and platforms. The IIOP is a standard for facilitating communication between objects, defined by OMG. The CORBA with IIOP is suitable to construct a large, quick speed and stable applications, but too complex and not convenient to use for web services. DCOM is Microsoft's solution for supporting distributed computing with object, and it extends COM to the distributed system supporting remote object by running on a protocol called Object Remote Procedure Call (ORPC) [5]. The OPRC layer is based on RPC from DCE and interacts with COM's run-time services. The biggest problem of DCOM is the fact that it's only supported in Microsoft platforms. Though it's declared that some other operation system support this technology, it has not been popular and gained acceptance in Unix and other system today. Sun has two solutions for distributed computing. One is Java Remote Method Invocation (RMI) [9], and another is Enterprise Java Beans (EJB) based on RMI [9]. RMI is the extension to the core JDK, called Remote Method Call, which depend closely on the Java features, such as Java-Object serialization, Java Interface definition etc. So it can't leave the Java language, but it is easy to use and also inherits the advantage from Java, such as platform independence. In 1997 Sun announces its new solution, Enterprise Java Beans (EJB) [10], which is promised to unify the easy programmability of Java RMI with the cross-language support of CORBA. In EJB the remote objected message can be exchanged through the "RMI over IIOP", which is an integration of RMI and IIOP.

In [1] authors present a new server, called the Smart Data Server (SDS), which offers a framework for easy-to-build environment independent modules with multiple functionalities, and allows the client to call the procedure from remote client through the self-defined "Information

* Proceedings of the 2nd WSEAS International Conference on MULTIMEDIA, INTERNET and VIDEO TECHNOLOGIES (WSEAS ICOMIV 2002), Sep.25-29, 2002, Skiathos, Greece, pp.1141-1145

Package Transfer Protocol" (IPTP). But just like the above three solutions, they have a common problem: the application can't talk to each other among these solutions with the older protocols, such as IIOP, OPRC, RMI over IIOP and IPTP. Another problem is that these protocols are not friendly to firewall. But all these problems will disappear with the arising of SOAP [6], which has been proposed completely on the existed technologies, like HTTP and XML. In the following, first I will explain the architecture of the SDS, then introduce the original protocol, IPTP, which is used to transfer the client request and return result between clients and the SDS. After that, I will show you how the SOAP works in the Smart Data Server instead of IPTP. Finally the conclusion will be made to summarize and outline our future research.

2 Smart Data Server System

The Smart Data Server (SDS) is a middleware server, which acts as middle tier in a three-tier architecture (Client-SDS-Database). SDS not only makes the advance security communication possible through building SDS network [3], but also provides a general framework for distributed computing [2]. It is a pure Java implementation, so it inherits all the advantages of Java, namely writing once and running everywhere. The SDS can work with the client-server components and also some other middleware server. For example, when the SDS works with a Java servlet server, a servlet can also server as the client toward the SDS. When a client browser send a request to a servlet, the servlet-engine will interpret it and send a request to SDS if the servlet includes the code to call a distributed function of the SDS. The result returned from the SDS will then be transferred back to the browser via the servlet-engine. The SDS offers a general frame to build environment independent modules with distributed functions. The internal structure of SDS contains three layers: session layer, service layer and function layer, which are illustrated in the following Fig.1.

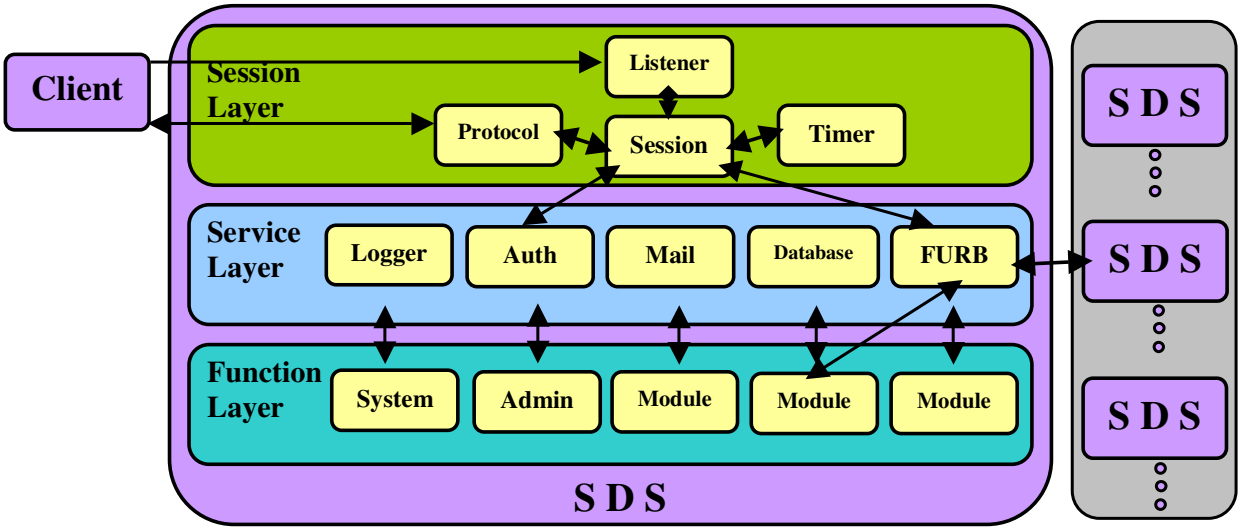

Fig.1 Internal Structure of the SDS

The Session Layer is responsible for handling client request, checking user authorization and creating timer based on requests. It contains the basic functionality for network-connections, session handling, protocol analysing and other request-related functionality. The protocol module establishes the connection with client using the socket from Listener module. In the original system the IPTP has been employed and the SOAP will be added to improve the SDS in this paper.

The Service Layer consists of a set of general services that is engaged by both the function layer and session layer modules. The logger module is used to record any error, warning and running state information, which are convenient for administrator to debug and check the SDS system. The database module provides many practical interfaces of database transaction for the function modules. And the FURB module which means Function Request Broker can gains the module, object name and the method to be called from

session layer, and invoke the local code if it exists in local server or ask for help from another SDS system.

Function Layer contains some application functionality, including system module, administrator module and some user-defined modules in which reside all the methods to be called from remote client.

3 Protocol Improvement to the SDS

As described previously, the protocol module of the SDS is engaged to transfer the request from the client and send back the result from the server. The original protocol is the Information Package Transfer Protocol, which is similar with HTTP for accessing Hyper Text document. In the following, a new promising protocol SOAP will be brought into the SDS instead of IPTP.

3.1 The Original Protocol-IPTP

Information Package Transfer Protocol (IPTP) includes a header and a body. The body holds the definition of the request message from client or of the result message from the SDS. The header holds some header elements including the information as following [2]:

- Which function-module has to answer the request?
- Who wants to access this function-module?
- Is this request part of a user-session (not a network - session), and which session is it?
- What type of the body will follow?
- How is the body encrypted?

The request and its results are stored in the body, which contain many elements constructed by the hashtable-type which include the hashtable-keys of "FUNCTION", "PARAMETERS", "DATA" and "RETURN". "FUNCTION" defines the function inside the module to be called, "PARAMETERS" defines the parameters of the function, "RETURN" defines which result is requested, and the "DATA" will be used if some data is requested. The definition and data type of each element in the body of IPTP will be clarified in Fig.2.

If a request or response is transferred between the SDS and the client, all the structural data types will be parsed into XML-like text structure. This text will be transferred via socket. After receiving this text, the information will be unparsed back to structural data type.

```
Struct          ::=   String | Vector | Hashtable
Vector          ::=   [Struct]
Hashtable       ::=   {(String, Struct)}

FunctionDef     ::=   ("FUNCTION"; String)
ParametersDef   ::=   ("PARAMETERS", Hashtable)
DataDef         ::=   ("DATA", Hashtable)
ReturnDef       ::=   ("RETURN": Hashtable)
ResultDef       ::=   ("RESULT": Hashtable)

RequestDef      ::=   {FunctionDef, ParametersDef,
DataDef, ReturnDef}
AnswerDef       ::=   {FunctionDef, ResultDef}
```

Fig.2 Body Elements Definition

3.2 SOAP Protocol in SDS

SOAP is an XML based protocol, which provides a simple mechanism for exchanging structured message in a distributed environment. Similar technologies appear in some other solutions, such as CORBA of OMG, DCOM from Microsoft, Sun's RMI and EJB and so on. But just like the SDS, all of them have some common problems: it's not easy to communicate among the different solutions and they are not friendly to the firewall. Though IIOP, ORPC and RMI are also standard protocols for object-oriented message transport, they are incompatible and not supported in most platforms. Another problem is that these older protocols are implemented through some specific socket ports, which are always forbidden in the most of firewalls. All of these difficulties will disappear immediately with the arising of SOAP. SOAP formats messages using XML that is a universal format for structural data in the Internet. It transfers its request and response through HTTP, which is a commonly used protocol for transportation of web content in the Internet. So when SOAP is added into the SDS as a transfer protocol instead of IPTP, the SDS has gained the following advantages immediately:

- It can collaborate with other distributed computing system because SOAP is increasing accepted by most vendors.
- The SDS can be applied widely for its SOAP request and response messages have been transferred through the common used transfer protocol-HTTP and HTTP's popular socket port 80 is seldom forbidden in firewalls.

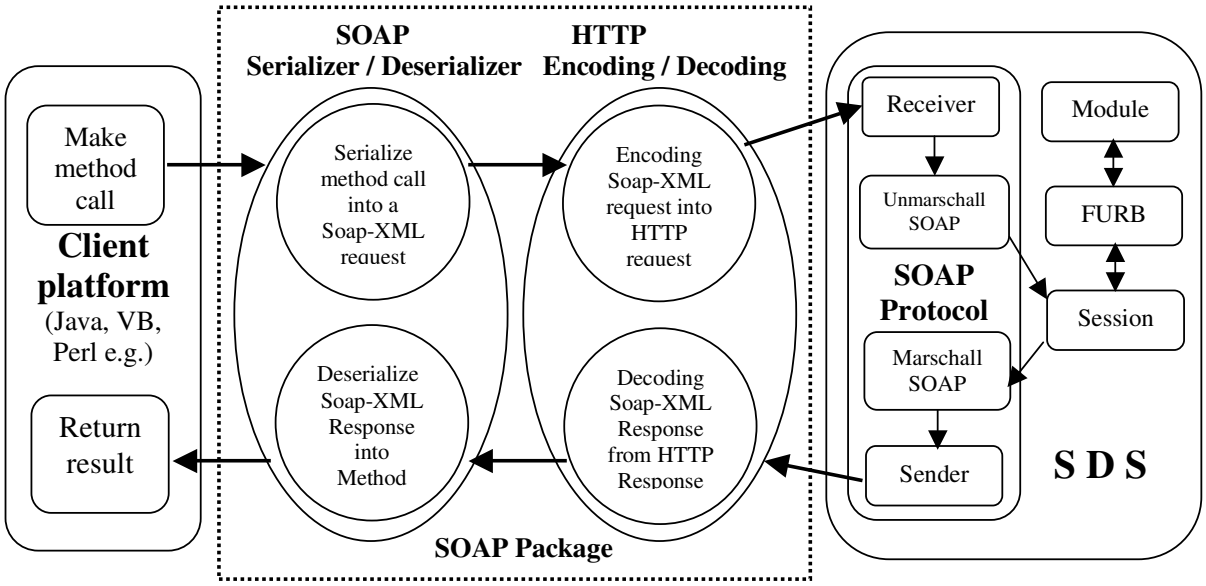

Fig.3 Workflow of the Soap Communication in the SDS

As illustrated in Fig.3, under any client platform circumstance, the request will be encoded into HTTP request using the SOAP package after the client makes the remote method call. This HTTP-SOAP request would then be sent to the SDS through HTTP protocol. When the receiver of SDS catches the HTTP-SOAP request, from which the module and called method name can be extracted, the distributed computing components of the SDS will be executed to generate the call result. The result will also be wrapt into HTTP response to be sent back, and in the client terminal the SOAP package will again be used to decode the SOAP response and get the return result.

In order to enhance the security of the remote invocation in the SDS, the module name, containing the objects and its relative methods, has to be provided in the client end. Normally, the SOAP only transfers the object, method and its parameters to the server. So how can we transfer this module name to the SDS? As we know, in the SOAP request message, the object and method name is only one, but its parameters can be many if it need. So we can take the module name as the method's first parameter, and its real parameters can be the second and the third parameter, as illustrated in following Fig.4 and Fig.5.

At last all of the necessary remote invocation information can be transferred to the server and we only need to separate the module name and its parameters after XML is parsed by the SOAP Protocol in the SDS. In our implement, Apache

```
<SOAP-ENV:Body>
  <ns1:computeAdd
   xmlns:ns1="urn:SdsSoapService"
   SOAP-ENV:encodingStyle=
   "http://schemas.xmlsoap.org/soap/encoding/">
       <module xsi:type="xsd:string">
            SimpleCompute
       </module>
       <param1 xsi:type="xsd:double">
            35.0
       </param1>
       <param2 xsi:type="xsd:double">
            64.0
       </param2>
 </ns1:computeAdd>
</SOAP-ENV:Body>
```

Fig.4. Client SOAP Request

```
<SOAP-ENV:Body>
  <ns1:computeAddResponse
   SOAP-ENV:encodingStyle=
   "http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:ns1="urn:SdsSoapService">
       <addResult xsi:type="xsd:double">
            99.0
       </addResult>
  </ns1:computeAddResponse>
</SOAP-ENV:Body>
```

Fig.5. SDS SOAP Response

SOAP toolkit has been adopted as the client implementation, and the Microsoft SOAP toolkit can also be a good substitute. The SOAP

serializing implementation of the SOAP protocol inside the SDS has been completed with the help of IBM's SOAP Envelope API [8].

4 Conclusion and Future work

SOAP, as a promising protocol for exchanging message based on XML, impels the distributed computing be applied more conveniently and more widely. As the result of its application in the Smart Data Server, the SDS can work with other distributed computing system, and can be invoked by the client under any platform using any programming language. These achievements are benefited to the popularity of SOAP and the support for SOAP in most of the distributed computing solutions. Now the SDS, as a middleware for distributed computing, has been achieved great success. But with the popularity of the mobile device, such as mobile phone, personal digital assistant (PDA) and palm computer, there appear some new requirements, for which the middleware of wired distributed computing can't be applied directly. So our next research interests focus on: how to add some new features to the SDS to adapt the temporary loss of network connectivity, frequent and unannounced changes happening in their executing environment and other problems with the arising of mobile devices.

References:

[1] U.Roth, E.G.Haffner, T.Engel, Ch.Meinel. *The Smart Data Server - A New Kind of Middle Tier*. Proceedings of the IASTED International Conference Internet and Multimedia Systems and Applications, 1999, pp361-365.

[2] U.Roth, E.G.Haffner, T.Engel, Ch.Meinel. *An Approach to Distributed Functionality: The Smart Data Server*. Proceedings of the WebNet International Conference 1999, pp931-935.

[3] E.G.Haffner, U.Roth, A.Heuer, T.Engel, Ch.Meinel. *Managing Distributed Personal Firewalls with Smart Data Servers*. Proceedings of the World Conference on WWW and Internet, AACE WebNet 2001, pp466-471.

[4] *The Common Object Request Broker: Architecture and Specification*. Object Management Group, 1995.

[5] N.Brown, Ch.Kindel. *Distributed Component Object Model Protocol – DCOM/1.0*, Microsoft Corporation. http://www.globecom.net/ietf/draft/draft-brown-dcom-v1-spec-03.html

[6] D.Box, D.Ehnebuske, G.Kakivaya, A.Layman, et.al *Simple Object Access Protocol (SOAP) 1.1*. Worl Wide Web Consortium, http://www.w3.org/TR/SOAP/

[7] Apache Soap, http://xml.apache.org/soap/index.html

[8] IBM SOAP Envelope API, http://www.trl.ibm.com/projects/xml/soap/env

[9] *Java Remote Method Invocation Specification*, Sun Microsystems Inc. http://java.sun.com/products/jdk/rmi/

[10] *Enterprise Java Beans Specification*, version 2.0, Sun Microsystems Inc. http://java.sun.com/products/ejb/docs.html

[11] Licia Capra, Cecilia Mascolo, Stefanos Zachariadis and Wolfgang Emmerich. *Towards a Mobile Computing Middleware: a Synergy of Reflection and Mobile Code Techniques*. Proc. of the 8th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'2001). October 2001.