Network Working Group
Request for Comments: 5381
Category: Informational

T. Iijima
Y. Atarashi
H. Kimura
M. Kitani
Alaxala Networks Corp.
H. Okita
Hitachi, Ltd.

October 2008

Experience of Implementing NETCONF over SOAP

Status of This Memo

IESG Note

This document discusses implementation experience of NETCONF over SOAP. Note that Section 2.4 of RFC 4741 states, "A NETCONF implementation MUST support the SSH transport protocol mapping". Therefore, a NETCONF implementation that only supports the SOAP transport described in this document and not (at least) also SSH is not in compliance with the NETCONF standards.

Abstract

This document describes how the authors developed a SOAP (Simple Object Access Protocol)-based NETCONF (Network Configuration Protocol) client and server. It describes an alternative SOAP binding for NETCONF that does not interoperate with an RFC 4743 conformant implementation making use of cookies on top of the persistent transport connections of HTTP. When SOAP is used as a transport protocol for NETCONF, various kinds of development tools are available. By making full use of these tools, developers can significantly reduce their workload. The authors developed an NMS (Network Management System) and network equipment that can deal with NETCONF messages sent over SOAP. This document aims to provide NETCONF development guidelines gained from the experience of implementing a SOAP-based NETCONF client and server.

Table of Contents

1. Introduction

1.1. NETCONF over SOAP

This document is not a product from the NETCONF WG but a report on the experience acquired by individual developers.

SOAP (Simple Object Access Protocol) was specified in [RFC4743] as one of the transport protocols for NETCONF. It is designed to use XML (eXtensible Markup Language) as its description language, which is a fundamental messaging technology for Web Services. For this reason, SOAP is well suited to the NETCONF protocol and can be deployed widely.

To develop a SOAP-based NETCONF client and server, several development tools are available as open-source software. The authors developed a SOAP-based NETCONF client and server by using available development tools. The SOAP-based NETCONF client was developed by utilizing Java APIs (Application Programming Interfaces) that are automatically generated from the XSD (XML Schema Definition) file and WSDL (Web Services Description Language) file obtained from [RFC4741] and [RFC4743], respectively. The SOAP-based NETCONF client that the authors developed acts as an NMS (Network Management System). The SOAP-based NETCONF server that the authors developed runs on network equipment and accepts NETCONF messages sent from the NETCONF client.

1.2. Motivation

The aim of this document is to describe why the authors believe SOAP is practical as a transport protocol for NETCONF when an NMS is developed. When developing an NMS that uses SOAP as its transport protocol, development tools and procedures can be used according to the Web Services framework. This document also describes the experience of implementing NETCONF over SOAP so that even those who have little knowledge of SOAP can start developing a SOAP-based NETCONF client and server.

This document describes an alternative SOAP binding for NETCONF that does not interoperate with an RFC 4743 conformant implementation as it relies on cookies used on top of the persistent transport connections of HTTP. This is provided for information purposes only based on the implementation experience of the authors.

2. NETCONF Development on Web Services Framework

SOAP is a fundamental messaging technology for Web Services. Therefore, if SOAP is used as a transport protocol for NETCONF, a network configuration performed by NETCONF is achieved on the Web Services framework. In this section, the overall architecture of Web Services is described.

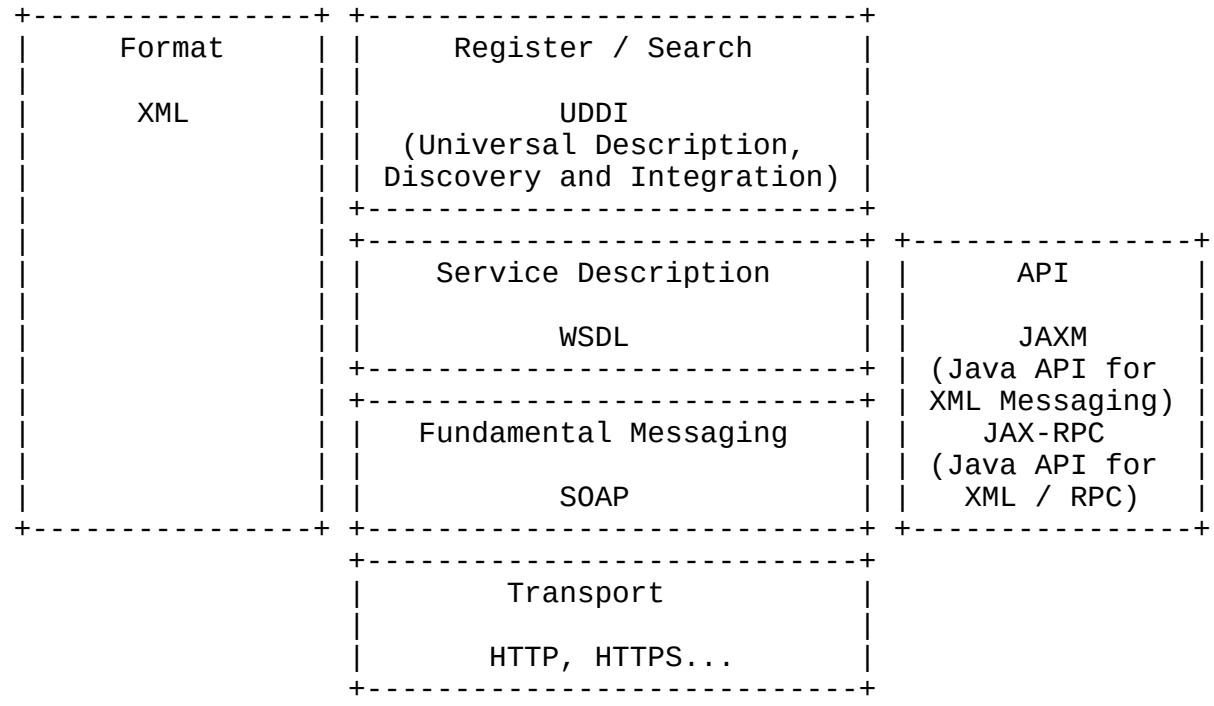

Figure 1: Overall Architecture of Web Services

As depicted in Figure 1, peripheral technologies around SOAP/HTTP are well developed. Therefore, if SOAP/HTTP is chosen as a transport layer for the NETCONF protocol, the NMS development based on the Web Services framework can choose from different optional services and might be less expensive based on the use of already available services.

2.1. WSDL as an Interface Description Language

WSDL [WSDL] defines how SOAP messages are exchanged between Web Services entities. Interfaces of Web Services entities are automatically generated by development tools when importing a WSDL file. Interfaces generated in this manner act as APIs. For the development of an NMS, only these APIs are necessary; there is no need to use SOAP directly.

Useful tools that can import a WSDL file are available with SOAP. For instance, Apache Axis [Axis] generates an interface from a WSDL file and is a widely used SOAP implementation middleware.

2.2. Generation of APIs

As described in the previous section, APIs are generated from a WSDL file by development tools such as Apache Axis. Such APIs are in the form of a Java library and act as programming interfaces for an NMS. By using these APIs, an NMS can send SOAP messages to Web Services entities.

3. Architecture of the NETCONF over SOAP Implementation

The architecture of the NETCONF over SOAP implementation is shown in Figure 2. A NETCONF implementation residing in an NMS works as a NETCONF client while network equipment acts as a NETCONF server. In this document, we call NETCONF-client and NETCONF-server implementations a NETCONF application and a NETCONF service provider, respectively. A SOAP implementation may be installed on both the NMS and the network equipment. Each instance of the SOAP implementations exchanges SOAP messages based on WSDL, as described in [RFC4743]. If Java libraries generated from the WSDL are provided in the NMS, engineers can develop a NETCONF application, which configures network equipment via the NETCONF protocol, by utilizing the Java library. There is no need for engineers to use XML or SOAP directly.

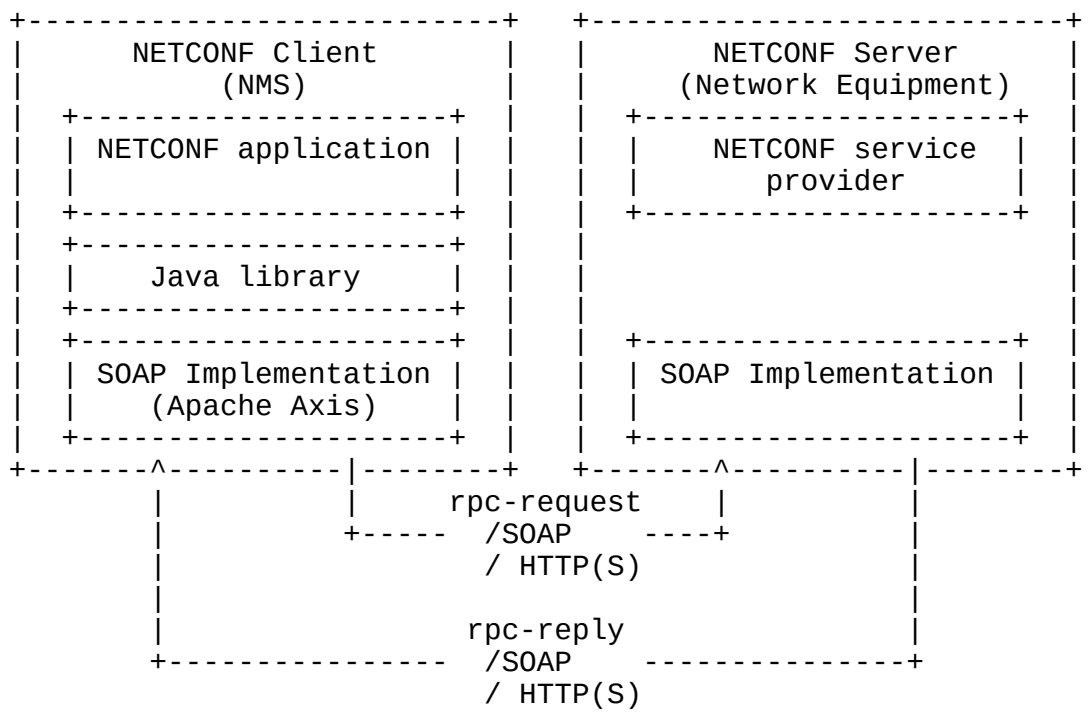

Figure 2: Architecture of NETCONF Implementation Using SOAP

The SOAP implementation in both the NMS and network equipment is explained in detail in the following sections.

3.1. SOAP Implementation in NMS

Several SOAP implementations appropriate for use in an NMS are available today. Apache Axis is one such widely used implementation.

Axis works as a SOAP implementation and an NMS-development tool. For instance, WSDL2Java, one of Axis' tools, generates Java-class files from a WSDL file. Another tool called Java2WSDL does the opposite: it generates a WSDL file from Java-class files. Consequently, various benefits can be obtained if Axis is introduced as a SOAP implementation.

To develop a NETCONF application that is capable of various functions such as releasing log messages, Java-class files generated by the Axis tool may be extended by adding more functions. By utilizing these Java libraries, engineers can easily develop NETCONF applications.

3.1.1. SOAP Parser in NMS

The SOAP Parser function is performed entirely by a SOAP implementation such as Apache Axis.

3.1.2. Session Maintenance in NMS

When exchanging NETCONF messages between an NMS and network equipment, a NETCONF session has to be maintained on both sides, as described in [RFC4741].

In [RFC4743], HTTP is specified as an option of an underlying protocol for NETCONF over SOAP. When HTTP is used for that purpose, it is also specified that a NETCONF session state is tied to the state of the underlying transport (TCP) connection (just like in NETCONF over SSH [RFC4742] and NETCONF over BEEP [RFC4744]). However, HTTP itself is a stateless protocol, and many server implementations process user requests independently of previous requests received over the same transport connection. To simplify implementation of the NETCONF service provider, we used the cookie field inside the HTTP header to map incoming requests to NETCONF sessions. Note that this means our implementation actually uses an alternative SOAP binding for NETCONF, which does not interoperate with RFC 4743 compliant implementations.

For example, the implemented NETCONF-session maintenance in the NMS works as follows. After the NMS sends a NETCONF hello message to the network equipment, the NETCONF service provider in the network equipment allocates a session identifier for the NETCONF application in the NMS and writes it inside the <session> element of a replying NETCONF hello message, as described in [RFC4741]. At the same time, the network equipment writes the same value in the cookie field inside an HTTP header. After that, a SOAP message encompassing the replying NETCONF hello message is added. When the NMS receives the newly allocated session identifier from the replying NETCONF hello message, the NETCONF application stores it and writes it inside a <session> element for subsequent NETCONF request messages and in a cookie field for subsequent HTTP headers. By recognizing the session identifier in NETCONF request messages and the cookie field in HTTP headers, the network equipment can maintain both a NETCONF session and the state of an HTTP connection. The NETCONF session is maintained over the maintained state of the HTTP connection. The stored session identifier is erased when the NMS sends a NETCONF close-session message and receives a NETCONF response message from the network equipment.

3.2. SOAP Implementation in the Network Equipment

To accept SOAP messages sent from the NMS, it is also necessary to provide SOAP in the network equipment. As in the case of NMS, some free SOAP implementations are available today for installation on network equipment. However, the memory capacity of the network equipment might be limited. Therefore, the SOAP implementation may be chosen taking memory capacity into consideration. In some cases, a memory-saving method will be required when implementing SOAP in the network equipment.

3.2.1. SOAP Parser in the Network Equipment

A SOAP header inside the SOAP envelope is defined as optional. Therefore, the module that processes the SOAP header can be omitted if the memory capacity in the network equipment is insufficient. In this case, a SOAP parser in the network equipment is allowed to parse only mandatory parts of a SOAP envelope.

3.2.2. Session Maintenance in the Network Equipment

To maintain NETCONF sessions with the NMS, the NETCONF service provider in the network equipment has to provide a session identifier to the NMS, as described in [RFC4741].

For example, the implemented NETCONF-session maintenance in the network equipment works as follows. When the network equipment receives a NETCONF hello message from the NMS, the NETCONF service provider in the network equipment sets a session identifier inside the <session> element of a replying NETCONF hello message, as described in [RFC4741]. At the same time, the network equipment also sets the same value in the cookie field inside an HTTP header. After that, a SOAP message encompassing the replying NETCONF hello message is added. The cookie field inside the HTTP header is used for maintaining the state of the HTTP connection over which the NETCONF-session maintenance is ensured. The network equipment then sends an HTTP response message to the NMS. When the network equipment receives a NETCONF close-session message from the NMS, it erases the stored session identifier.

4. Guidelines for Developing NETCONF Clients and Servers

In the case of SOAP transport mapping, sharing information on the kinds of development tools that are available would help developers start developing SOAP-based NETCONF clients and servers. That would contribute to the rapid deployment of SOAP-based NETCONF clients and servers.

4.1. Procedures of Development of NETCONF Clients

To develop a SOAP-based NETCONF client, a stub code may be generated. A stub is a library that is generated automatically from WSDL by a Web Services tool and that acts as a group of APIs. When using Apache Axis as a Web Services tool, a generated stub is in the form of Java APIs. These Java APIs display interfaces of a Web Service as if they are methods capable of configuring a local machine.

The WSDL file named "netconf-soap_1.0.wsdl", which is selected from [RFC4743], specifies NETCONF messages to be exchanged between the NETCONF client and server. These NETCONF messages are the "hello" message and "rpc" message. Therefore, stub codes for creating the "hello" message and "rpc" message are generated from "netconf-soap_1.0.wsdl". However, the file "netconf-soap_1.0.wsdl" is not sufficient because no service element is specified.

In "myNetconfService.wsdl", which is also selected from [RFC4743], a service element is specified and "netconf-soap_1.0.wsdl" is imported. Stub codes generated from those WSDL files are found in files such as "Netconf.java", "NetconfLocator.java", and "NetconfBindingStub.java".

When interfaces are used to operate the NETCONF protocol in the manner of "get-config" and "edit-config", for example, an XML schema file named "netconf.xsd", which is selected from [RFC4741], is used by being imported into "netconf-soap_1.0.wsdl". Using the XML schema, methods of operating the NETCONF protocol are generated in files such as "GetConfigType.java" and "EditConfigType.java".

When interfaces are used to configure network functions at the network equipment, a data model of each network function has to be defined in the style of an XML schema. The XML schema may be imported into "netconf-soap_1.0.wsdl" in the same manner as that of the XML schema in [RFC4741].

The connection between the NETCONF schema and a data model should be made by inserting the following attribute into elements of each data model. This attribute is defined in the XML schema in [RFC4741].

```
<xs:attribute name="operation" type="editOperationType"
default="merge"/>
```

Consequently, using "myNetconfService.wsdl" to import "netconf-soap_1.0.wsdl", NETCONF schema, and the data model makes it possible to generate stub files containing interfaces to configure network equipment.

When stub codes are generated, the development environment may be arranged as well. The development of a Java-based NETCONF client may use JDK (Java Development Kit) [JDK] and Apache Axis. In addition, using some IDE (Integrated Development Environment) such as Eclipse [Eclipse] with Apache Ant [Ant] and NetBeans [NetBeans] would reduce the developer workload significantly. When Eclipse is used as an IDE, first, the library (*.jar files) of Axis has to be added to the development project's build path as an external library. The library of Axis acts as a SOAP library, so there is no need to be concerned about SOAP messaging when programming a NETCONF client using the library of Axis.

4.1.1. Developing NETCONF Clients without Eclipse

Given that development of a NETCONF client is carried out in the environment of a Windows computer without Eclipse, and that "myNetconfService.wsdl" is placed in the "C:\NetconfClient" directory, a stub is generated by executing the following command in the command prompt.

```
C:\NetconfClient>java -classpath .;%AXIS_HOME%\lib\axis.jar;%
AXIS_HOME%\lib\jaxrpc.jar;%AXIS_HOME%\lib\saaj.jar;%AXIS_HOME%
\lib\commons-logging-1.0.4.jar;%AXIS_HOME%\lib\commons-discovery-
0.2.jar;%AXIS_HOME%\lib\wsdl4j-1.5.1.jar
org.apache.axis.wsdl.WSDL2Java -p stub myNetconfService.wsdl
```

In the directory where the WSDL file is located, the WSDL2Java command is executed. Locations of each Axis library have to be specified. The environment variable of "AXIS_HOME" is the directory where Axis is installed. By executing the above command, files with an extension of "*.java" are generated in the "stub" directory, which is specified by the above command. Inside the stub directory, we can find files such as "NetconfBindingStub.java", "Hello.java", and "GetConfigType.java".

Next, it is necessary to compile these files by executing the following command in the command prompt.

```
C:\NetconfClient>javac -classpath .;%AXIS_HOME%\lib\axis.jar;%
AXIS_HOME%\lib\jaxrpc.jar stub/*.java
```

After the compilation of those java files, "*.class" files are generated. After the compiling is done, the source code of the NETCONF client has to be written. Sample source code of the NETCONF client is shown in Figure 3. This NETCONF client is written by utilizing stub classes and interfaces, which are imported into the local package and referenced.

```
import org.apache.axis.types.UnsignedInt;
import org.apache.axis.types.*;

public class NetconfClient {
        /**
         * @param args
         */
        public static void main(String[] args) {
                // TODO Auto-generated method stub
                try{
                        NetconfClient client = new NetconfClient();
                        java.net.URL url = new java.net.URL(args[0]);
                        stub.Netconf netconf =
                                new stub.NetconfLocator();
                        stub.NetconfPortType stubNetconf =
                                netconf.getnetconfPort(url);

                        URI[] uri = new URI[1];
                        stub.holders.HelloCapabilitiesHolder
                        capability = new
                        stub.holders.HelloCapabilitiesHolder(uri);

                        UnsignedInt id = new UnsignedInt();
                        id.setValue(1);
                        org.apache.axis.holders.UnsignedIntHolder
                        holder = new
                        org.apache.axis.holders.UnsignedIntHolder(id)
                        ;
                        stubNetconf.hello(capability, holder);
                }catch(Exception e){
                        e.printStackTrace();
                }
        }
}
```

Figure 3: Sample Source Code of NETCONF Clients

To add functions such as the release of log messages, these functions have to be incorporated at this stage. Again, the NETCONF client is developed by compiling its source codes.

4.1.2. Developing NETCONF Clients Using Eclipse

When we use Eclipse and Apache Ant, the procedures described in the previous section are significantly simplified and executed at one time. In this case, files named "build.xml" and "build.properties" are required for Apache Ant.

The file named "build.xml" is written in XML and seen by Apache Ant when Apache Ant is running on Eclipse. The file specifies how Apache Ant behaves. According to the descriptions of the file, Apache Ant compiles source codes, generates JAR (Java ARchive) file, and so on. On the other hand, the file named "build.properties" specifies properties of the development environment where Apache Ant runs. This file is referred to by the "build.xml" file.

Examples of "build.xml" and "build.properties" are shown in Figure 4 and Figure 5, respectively.

```
<?xml version="1.0"?>
<project name="NetconfClient" default="all" basedir=".">
        <property file="build.properties"/>
        <path id="axis-classpath">
                <fileset dir="${axis.libdir}">
                        <include name="*.jar"/>
                </fileset>
        </path>
        <target name="prepare">
                <mkdir dir="${destdir}"/>
        </target>
        <target name="stub" depends="prepare">
                <java classname="org.apache.axis.wsdl.WSDL2Java" fork
                        ="Yes">
                        <arg value="-o"/>
                        <arg value="${srcdir}"/>
                        <arg value="-p"/>
                        <arg value="${stub.stubdir}"/>
                        <arg value="${stub.wsdlpath}"/>
                        <classpath refid="axis-classpath"/>
                </java>
        </target>
        <target name="compile" depends="stub">
                <javac srcdir="${srcdir}" destdir="${destdir}"
                        encoding="UTF-8">
                        <classpath refid="axis-classpath"/>
                </javac>
        </target>
        <target name="stub-jar" depends="compile">
                <jar jarfile="${stub.jar}" basedir="${destdir}"/>
        </target>
        <target name="all" depends="stub-jar"/>
</project>
```

Figure 4: build.xml of NETCONF Clients

```
axis.libdir=C:/axis-1_4/lib
srcdir=src
destdir=classes
stub.stubdir=stub
stub.wsdlpath=myNetconfService.wsdl
stub.jar=NETCONF.jar
```

Figure 5: build.properties of NETCONF Clients

The location of the WSDL file should be specified in the "build.properties" file. In the case shown in Figure 5, the location of the WSDL file is specified as being under the current directory.

By running Apache Ant on Eclipse, the steps specified in Figure 4 are taken. First, stub codes are generated. Then, compiling of those stub codes is executed. We were careful about the encoding style used for the compiling. After the compilation, Apache Ant will generate a JAR file, which is the output that compresses all stub files (*.class) and acts as a library. In this example, the name "NETCONF.jar" is specified in Figure 5. The "NETCONF.jar" file also has to be added to the build path of the development project as an external library.

After the "NETCONF.jar" file is added to the build path of the development project, source codes of the NETCONF client can be written by utilizing stub classes and interfaces. Source codes like the one shown in Figure 3 can be written. By running Apache Ant again, the source code of the NETCONF client is compiled. The NETCONF client is developed in this manner.

4.2. Procedures of Development of NETCONF Servers

In the Web Services framework, there are two approaches for developing a Web Services provider, namely a NETCONF server. One is called the top-down approach, and the other is called the bottom-up approach. The top-down approach is carried out by first designing a WSDL file. A skeleton source code from the WSDL file is then generated by using a Web Services tool such as Apache Axis. The generated skeleton code is just a template of the Web Services provider’s source code. Therefore, even though the Web Services provider’s skeleton code works on its own, if additional functions were necessary, the generated skeleton code would require additional source codes. This approach is superior to the bottom-up approach in terms of interoperability because the specification is already defined in the WSDL file. All vendors have to be in compliance with the WSDL file.

In contrast, the bottom-up approach is carried out by first creating Web Services from source code (e.g., Java bean) and then generating a WSDL file from the source code by using a Web Services tool such as Axis. This approach is faster and easier than the top-down approach. However, in the bottom-up approach, it is difficult to ensure interoperability since implementation of a Web Services becomes vendor-specific.

When developing a NETCONF server, the WSDL file is already defined in [RFC4743], so there is no choice but to develop the NETCONF server using the top-down approach. The remainder of this section describes the top-down approach for developing a NETCONF server.

To develop a SOAP-based NETCONF server using the top-down approach, a skeleton code is necessary. A skeleton is a library, which is also generated automatically from WSDL by a Web Services tool. When using Axis as a Web Services tool, the generated skeleton is in the form of a Java library. From the same WSDL file as that used for generating the stub code, skeleton codes are generated in files such as "NetconfBindingSkeleton.java", "Hello.java", and "GetConfigType.java".

When skeleton codes are being generated, the development environment may be arranged as well. Moreover, when a Java-based NETCONF server is being developed, in addition to JDK and Axis, a servlet container such as Apache Tomcat [Tomcat] is necessary. The "webapps\axis" directory under the Axis directory has to be copied to the "webapps" directory under the Tomcat directory.

4.2.1. Developing NETCONF Servers without Eclipse

Given that the development environment of a NETCONF server is created in the environment of a Windows computer without Eclipse and "myNetconfService.wsdl" is placed in the "C:\NetconfServer" directory, a skeleton is generated by executing the following command in the command prompt.

```
C:\NetconfServer>java -classpath .;%AXIS_HOME%\lib\axis.jar;%
AXIS_HOME%\lib\jaxrpc.jar;%AXIS_HOME%\lib\saaj.jar;%AXIS_HOME%
\lib\commons-logging-1.0.4.jar;%AXIS_HOME%\lib\commons-discovery-
0.2.jar;%AXIS_HOME%\lib\wsdl4j-1.5.1.jar
org.apache.axis.wsdl.WSDL2Java -p skeleton -s -S true -d Session
myNetconfService.wsdl
```

In the directory where the WSDL file is located, a WSDL2Java command is executed. Locations of each Axis library should be specified. The environment variable of "AXIS_HOME" is a directory where Axis is installed. By executing the above command, files with an extension

of "*.java" are generated in the "skeleton" directory, which is specified in the above command. Inside the skeleton directory, files such as "NetconfBindingSkeleton.java", "Hello.java", and "GetConfigType.java" exist. Furthermore, files named "deploy.wsdd" and "undeploy.wsdd" are found. "Deploy.wsdd" and "undeploy.wsdd" are used when deploying a NETCONF server in a servlet container and when undeploying a NETCONF server from a servlet container, respectively.

Adding source codes of NETCONF server functions to skeleton codes such as "NetconfBindingImpl.java" is required as the need arises. Functions such as the release of log messages have to be added at this stage. After that, by executing the following command in the command prompt, compilation of java files is carried out. Doing so will generate "*.class" files.

```
C:\NetconfServer>javac -classpath .;%AXIS_HOME%\lib\axis.jar;%
AXIS_HOME%\lib\jaxrpc.jar skeleton/*.java
```

A NETCONF server can be developed by following the above-described procedures. These class files should be copied into the directory "webapps\axis\WEB-INFO\classes" of the Apache Tomcat directory. Finally, the NETCONF server is deployed by executing the following command.

```
C:\NetconfServer>java -classpath .;%AXIS_HOME%\lib\axis.jar;%
AXIS_HOME%\lib\jaxrpc.jar;%AXIS_HOME%\lib\saaj.jar;%AXIS_HOME%
\lib\commons-logging-1.0.4.jar;%AXIS_HOME%\lib\commons-discovery-
0.2.jar org.apache.axis.client.AdminClient -p 832 depoy.wsdd
```

The command is executed in the directory where "deploy.wsdd" is located. The file "deploy.wsdd" is generated at the same time the skeleton code is generated. After deployment, the NETCONF server receives NETCONF messages sent from the NETCONF client.

4.2.2. Developing NETCONF Servers Using Eclipse

When Eclipse and Apache Ant are used, the procedures described in the previous section are significantly simplified and executed at one time. Files named "build.xml" and "build.properties" are required for Apache Ant. Examples of "build.xml" and "build.properties" are shown in Figure 6 and Figure 7, respectively.

```
<?xml version="1.0"?>
<project name="NetconfService" default="all" basedir=".">
        <property file="build.properties"/>
        <path id="axis-classpath">
                <fileset dir="${axis.libdir}">
                        <include name="*.jar"/>
                </fileset>
        </path>
        <target name="prepare">
                <mkdir dir="${srcdir}"/>
                <mkdir dir="${destdir}"/>
        </target>
        <target name="skeleton" depends="prepare">
                <java classname="org.apache.axis.wsdl.WSDL2Java" fork
                        ="Yes">
                        <arg value="-p"/>
                        <arg value="${skeletondir}"/>
                        <arg value="-o"/>
                        <arg value="${srcdir}"/>
                        <arg value="-s"/>
                        <arg value="-S"/>
                        <arg value="true"/>
                        <arg value="-d"/>
                        <arg value="Session"/>
                        <arg value="${wsdlpath}"/>
                        <classpath refid="axis-classpath"/>
                </java>
        </target>
        <target name="compile" depends="skeleton">
                <javac srcdir="${srcdir}" destdir="${destdir}"
                        encoding="UTF-8">
                        <classpath refid="axis-classpath"/>
                </javac>
        </target>
        <target name="copy2axis" depends="compile">
                <copy todir="${tomcat.axis.classesdir}" overwrite=
                        "true">
                        <fileset dir="${destdir}">
                                <include name="*.class"/>
                                <include name="*/*.class"/>
                                <include name="*/*/*.class"/>
                        </fileset>
                </copy>
        </target>
        <target name="deploy" depends="copy2axis">
                <java classname="org.apache.axis.client.AdminClient"
                        fork="Yes">
                        <arg value="-p"/>
```

```
                        <arg value="${deploy.port}"/>
                        <arg value="${deploy.ddname}"/>
                        <classpath refid="axis-classpath"/>
                    </java>
            </target>
            <target name="all" depends="deploy"/>
    </project>
```

Figure 6: build.xml of NETCONF Servers

```
axis.libdir=C:/axis-1_4/lib
tomcat.axis.classesdir=
C:/Program Files/Apache Software Foundation/Tomcat 6.0/
webapps/axis/WEB-INF/classes
srcdir=src
destdir=classes
skeletondir=skeleton
wsdlpath=myNetconfService.wsdl
deploy.port=832
deploy.ddname=src/skeleton/deploy.wsdd
```

Figure 7: build.properties of NETCONF Servers

The locations of the WSDL file and "deploy.wsdd" file have to be specified in the "build.properties" file. In Figure 7, the location of the WSDL file and "deploy.wsdd" file are specified as being under the current directory.

By running Apache Ant on Eclipse, the steps shown in Figure 6 are followed. First, skeleton codes have to be generated. After the skeleton codes are generated, source codes of the NETCONF server functions may be added to the skeleton codes according to the functions that developers intend to add.

Then, by running Apache Ant again, compiling of the skeleton codes is executed. As a result, class files of the NETCONF server are generated. Apache Ant copies these class files to the directory of Tomcat and deploys the NETCONF server. After that, the NETCONF server becomes accessible by the NETCONF client.

4.2.3. Developing NETCONF Servers with C Programming Language

When the NETCONF server for network equipment is being implemented, memory capacity might be limited, so it might not be possible to install a Java environment on the network equipment. The network-equipment platform might not support a Web Services tool. In that case, it may be necessary to implement SOAP as well as the NETCONF server by using C programming language on the network equipment.

To develop a NETCONF server capable of receiving NETCONF messages sent over SOAP/HTTP, the network equipment may have an HTTP daemon and a NETCONF service provider. A commonly used HTTP daemon can be used. A SOAP module may be added to the HTTP daemon as a connector between the HTTP daemon and the NETCONF service provider. The NETCONF service provider for parsing NETCONF messages sent from the NETCONF client and sending reply NETCONF messages toward the NETCONF client may be developed.

When an HTTP daemon receives a SOAP message that is sent over HTTP, the message is handed over to the SOAP module incorporated in the HTTP daemon. Then, the SOAP module removes the SOAP header and passes NETCONF messages to the NETCONF service provider. After that, the NETCONF service provider parses the NETCONF messages and configures the network equipment accordingly.

5. Security Considerations

The security considerations of [RFC4741] and [RFC4743] are applicable in this document. Implementers or users of SOAP-based NETCONF clients and servers should take these considerations into account.

As specified in the security considerations section of [RFC4743], transport-level security, such as authentication of users and encryption of transport protocol, has to be ensured by TLS (Transport Layer Security) in the case of NETCONF SOAP binding. That is, security has to be provided in the form of NETCONF/SOAP/HTTPS.

6. Acknowledgements

Extensive input was received from the members of the NETCONF design team, including: Andy Bierman, Simon Leinen, Bert Wijnen, Mehmet Ersue, Ted Goddard, Ray Atarashi, Ron Bonica, and Dan Romascanu. The following people have also reviewed this document and provided valuable input: Jari Arkko, Pasi Eronen, Chris Newman, Tim Polk, David Ward, Magnus Westerlund, and Christian Vogt.

7. References

7.1. Normative References

[RFC4741] Enns, R., "NETCONF Configuration Protocol", RFC 4741, December 2006.

[RFC4743] Goddard, T., "Using NETCONF over the Simple Object Access Protocol (SOAP)", RFC 4743, December 2006.

7.2. Informative References

[Ant] "Apache Ant".
<http://ant.apache.org/>

[Axis] "Web Services - Axis".
<http://ws.apache.org/axis/>

[Eclipse] "Eclipse".
<http://www.eclipse.org/>

[JDK] "Java SE".
<http://java.sun.com/javase/index.jsp>

[NetBeans] "NetBeans".
<http://www.netbeans.org/index.html>

[RFC4742] Wasserman, M. and T. Goddard, "Using the NETCONF Configuration Protocol over Secure SHell (SSH)", RFC 4742, December 2006.

[RFC4744] Lear, E. and K. Crozier, "Using the NETCONF Protocol over the Blocks Extensible Exchange Protocol (BEEP)", RFC 4744, December 2006.

[Tomcat] "Apache Tomcat".
<http://tomcat.apache.org/>

[WSDL] "Web Service Description Language (WSDL) 1.1".
<http://www.w3.org/TR/wsdl>

Authors' Addresses

Iijima Tomoyuki
Alaxala Networks Corp.
Shin-Kawasaki Mitsui Bldg.
890 Saiwai-ku Kashimada
Kawasaki, Kanagawa 212-0058
Japan

Phone: +81-44-549-1735
Fax: +81-44-549-1272
EMail: tomoyuki.iijima@alaxala.com

Yoshifumi Atarashi
Alaxala Networks Corp.
Shin-Kawasaki Mitsui Bldg.
890 Saiwai-ku Kashimada
Kawasaki, Kanagawa 212-0058
Japan

Phone: +81-44-549-1735
Fax: +81-44-549-1272
EMail: atarashi@alaxala.net

Hiroyasu Kimura
Alaxala Networks Corp.
Shin-Kawasaki Mitsui Bldg.
890 Saiwai-ku Kashimada
Kawasaki, Kanagawa 212-0058
Japan

Phone: +81-44-549-1735
Fax: +81-44-549-1272
EMail: h-kimura@alaxala.net

Makoto Kitani
Alaxala Networks Corp.
Shin-Kawasaki Mitsui Bldg.
890 Saiwai-ku Kashimada
Kawasaki, Kanagawa 212-0058
Japan

Phone: +81-44-549-1735
Fax: +81-44-549-1272
EMail: makoto.kitani@alaxala.com

Hideki Okita
Hitachi, Ltd.
1-280 Higashi-Koigakubo
Kokubunji, Tokyo 185-8601
Japan

Phone: +81-42-323-1111
Fax: +81-42-327-7868
EMail: hideki.okita.pf@hitachi.com

Full Copyright Statement

Intellectual Property