ISSN:0975-9646
R. Prabakaran et al, / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 6 (5) , 2015, 4344-4349

Network Management for New Era

Dr. R. Prabakaran, M.C.A., M.Phil., M.Tech,P.hD
Head, Department of Computer Applications,

J.Palanivel, M.C.A., M.Phil.,
Asst Prof, Department of Computer Applications,

Arignar Anna Institute of Management Studies & Computer Applications
Pennalur, Sri Perumbudur, Kanchipuram District.

Abstract -The increasing complexity and heterogeneity of modern networks has pushed industry and research towards a single and consistent way of managing networks. The effort to define a single industry-standard API for network management basically failed because it did not address aspects like complexity and ease of programming. Recently, a common approach is to map established network management standards into another object model, often based on the emerging CORBA standard. Unfortunately even this approach has shown many drawbacks mostly related to the significant amount of code that has to be linked with the final application and to the many limitations and imperfections of the mapping itself.

This paper describes a new approach to inter-domain management that attempts to overcome the limitations of current solutions. The goal is to allow people to write hybrid CMIP and SNMP based network management applications, using a single and simple object model. Relevant characteristics of this approach are: light, extensible, object-oriented, language-neutral, built upon software-components, string-syntax based, Internet-ready. This demonstrates that it is feasible to implement simple and light applications for inter-domain management without the need to use expensive or complex technologies.

Keywords: **Network Management, Object-Oriented Programming, Java, CORBA.**

1. INTRODUCTION

The increasing complexity and heterogeneity of modern networks and the advent of distributed computing are making network management both more important and complex. In this decade many companies and research institutions have attempted to simplify the scenario by defining a single and consistent way for managing heterogeneous networks based on both CMIP and SNMP. In this view, X/Open has defined an industry standard C-based API called XOM/XMP, able to unify these two dominant network management protocols. The idea was to allow people to write applications using a single API in order to simplify the integration of code written by different people.

Recently, the increasing popularity of the CORBA industry-standard pushed many people to write mappings between CMIP/SNMP and CORBA based on the assumption that CORBA will become the network management standard of the future and that everybody will use it instead of CMIP and SNMP. Despite their efforts, today there are many different mappings available that usually do not fully support CMIP/SNMP. Another drawback is related to the significant amount of code that must be generated for implementing these mappings and that has to be linked with the final application. Other than this, a network management expert that intends to write a management application must learn CORBA, IDL (the language used to specify the CORBA interfaces), how the mappings have been defined, and must have an ORB (Object Request Broker) installed somewhere. It is clear for instance, that the initial vision of SNMP to be simple and light has been jeopardized.

This is based on the idea that so far network management has been considered like a special software engineering problem where solutions must be built ad-hoc and cannot reuse widely established concepts. Today most of the network management people come from the "Vi, Unix and C" school and ignore new concepts and innovations like software components, and truly object-oriented software development (most of the code is object-based but not object-oriented). It is a common belief to pretend to solve a problem generating code for all the possible situations (for instance XOM/XMP and many CMIP/SNMP to CORBA mappings generate a class for each data type) instead of trying to define a way to simplify the problem. The advent of Java and TCL

demonstrated that the short reign of native-code-generating-object-oriented compiler is about to be over. Internet and the market demand light, machine-independent applications capable to roam from machine to machine.

The goal is to allow people to easily write light network management applications that fully support both CMIP and SNMP using a single and simple object model. These applications are Internet-ready and can be integrated with the world-wide web using the Java bindings here described. The guidelines and the code examples have been drawn from implementation experience and in the course of designing and implementing commercial products and research prototypes.

2. NETWORK MANAGEMENT STANDARDS

This shows how the process of integrating network management standard such as CMIP and SNMP is done. First of all the difference between the two standards is hidden, then the infrastructure is built. At last the integration is done.

Network programmers need a single way to manipulate instances of various object models. The main problem arises from the data types that have to be managed. In SNMP this is easy to handle because the different data types are about ten. CMIP is a lot more flexible in this respect and it allows the user to define new data types. Due to this, the number of data types that a network management application has to handle is not determined a priori. Therefore a way has to be defined to handle different data types of arbitrary complexity.

The solution proposed here is based on string notation. In this view, every data type is represented using strings. Aggregate data types like sequences or sets are a composition of basic data types like integer or boolean. The fact to have a single data type makes things simple and allows applications written in whatever language to use it even if this representation slows down the system code that uses data types to speed operation.

Despite the advantages of a string-based notation, some users may want to define information using a different object model. Programmers define data values using the string representation and then the encoder/decoder module converts this string to BER (Basic Encoding Rules) and back.

The conversion is based on meta-data information. In the stack, the ASN.1 and GDMO compilers compile input documents into a data file that is read by the encoder/decoder at startup time. These data files contain information about the data types and object-model dependent information. In the case of CMIP, data files contain information about managed object classes, name binding, actions and notification. In case of SNMP information concerning object identifiers and the textual description of the various attributes are stored in separate files. At runtime it is possible to access this information not only for encoding/decoding purposes but also for querying information about a particular attribute or action.

In SNMP there is no concept of connection and every message is sent independently usually over UDP. In CMIP every protocol request travels over an association that has to be established first and then closed when the communication is over. Users should not be concerned about associations and they should think only in terms of objects. Every time a request is sent to the stack, the object instance is analyzed and the correct agent managing that instance is identified and an association is opened. An association stays alive until it is closed either by one of the partners or when an error occurs (for instance if the connection goes down).

The string representation and to the automatic association handling, it is now possible to transparently manipulate remote instances using both SNMP and CMIP in a single and uniform way.

2.1 Application-side Bindings

Clients communicate with the Proxy over HTTP and because the data exchange type is based on strings, it is easy to write bindings in whatever programming language either object-oriented or not. For the sake of simplicity bindings described in this section are written using Java.

Similar considerations can be done for other languages such as C++ or TCL. The class hierarchy is quite simple. The class Proxy is responsible for handing communications with the Proxy application. It transparently sends the requests and receives the responses. The class Information contains the information relative to the request and to the response(s), stored in an

object of class java.util.Hashtable that are passed as input parameter to an instance of class Proxy. Subclasses SNMPObj and CMIPObj implement some high level manipulation functions for manipulating the input/output information and invoking Proxy methods whenever a request has to be issued. These subclasses have been provided to further simplify the access to the Information and Proxy classes and have to be considered like pure facilities.

The example below clarifies this situation.

```
Proxy p ;
CMIPObj cmip ;
try {
    p = new Proxy(HOSTNAME) ; //Where proxy host is running
    cmip = new CMIPO b j( p, "MIBCTL") ; // TITILE
    cmip.SetObjectClass ("system") ;
    cmip.SetO bjectInstance ("netowrkID") ; //Network ID ie. Telco
    cmip.SetAttribute ("systemTitle" , " ") ;
    cmip.CMIPGetAttributes () ; / / Issue t h e CMIP M-GET request
    System.out.println( "system T it le is :" +cmip.GetAttribute("systemTitle") );
} catch (Exception e ) {
     System.out.println ( "Error: " + e) ;
}
```

When the CMIPGetAttributes() method is called, the Proxy sends back the CMIP response containing objectClass, object Instance, currentTime and system Title. CMIPObj receives those values and puts them in the cmip instance itself. In case of system Title , the original empty value is replaced with the one returned by Proxy. Current Time, not present in the request, is added to the input object. This approach allows to easily getting and set attribute values other than allowing issuing operations in a few lines of code. If a request fails for whatever reason an exception of class Proxy Exception is raised: users should not deal with protocol errors but they should interact with remote objects only using programming constructs. The Information class and its subclasses SNMPObj and CMIPObj, greatly simplifies and reduces the code users have to write:

This solution allows saving bandwidth because only the needed attributes are exchanged between the Proxy and the Java application and because unmodified attributes, for instance object Class in a CMIP response, are not transmitted. Classes SNMPObj or CMIPObj other than issuing protocol requests, allow to retrieve metadata information and to convert object identifiers that can be expressed in both numeric or symbolic form.

```
public class Information extends java.lang.Object {
    public void SetAttribute (String name, Object value)
        throws IllegalArgumentException { }
    public Object GetAttribute(String name) {}
    public void RemoveAttribute(String name){}
    public Enumeration GetAttributeValues(){}
    public Enumeration GetAttributeNames(){}
    public void RemoveAllAttributes(){}
```

```
}
public class CMIPObj extends Information {
    public void SetObjectClass(String val) {}
    public String GetObjectClass() {}
    public void SetObjectInstance(String val){}
    public String GetObjectInstance(){}
    public Information GetActions() throws ProxyException{}
    public Information GetNameBindings() throws ProxyException{}
    public String GetSyntaxInfo(String syntax)throws ProxyException{}
    public String ConvertOID(String oid)throws ProxyException {}
            /* Management  Operations  * /
    public void CMIPCreateObject()throws ProxyException {}
    public void CMIPDeleteObject()throws ProxyException {}
    public Vector CMIPDeleteContainedInstances()throws ProxyException {}
    public void CMIPGetAttributes()throws ProxyException {}
    public Vector CMIPGetContainedInstances()throws ProxyException{}
    public  void CMIPSetAttributes()throws ProxyException{}
    public Vector CMIPSetContainedInstances()throws ProxyException{}
    public void CMIPPerformAction()throws ProxyException{}
    public Vector CMIPPerformActionContainedInst()throws ProxyException
    public int NotificationsAvailable()throws ProxyException {}
    public Information WaitForNotification(int timeout)throws ProxyException {}
    public void DeleteEFD()throws ProxyException {}
    public void CreateEFD(String inst,String fltr) throws ProxyException {}
}
    public class SNMPObj extends Information {
    public String SNMPGetAttributeInfo(String syntax)throws ProxyException {}
    public String ConvertOID(String oid)throws ProxyException{}

    /* Management Operations  * /
    public Vector SNMPWalk() throws ProxyException {}
    public void SNMPGetAttribute()throws ProxyException {}
    public void SNMPGetNextAttribute()throws ProxyException{}
    public void SNMPSetAttribute()throws ProxyException{}
}
```

Respectively:

- GetActions returns the CMIP actions that can be performed by the object
- GetNameBindings returns all the name bindings of the object, useful for creating managed objects.
- GetSyntaxInfo returns the requested ASN.1 syntax in HTML format
- ConvertOID is responsible for converting object identifiers
- SNMPGetAttributeInfo returns the attribute description specified in the RFC. Metadata information is either contained in the Proxy (the object identifier mapping information and the RFC information) or it is retrieved from the stack (ASN.1 information).

The class Proxy is responsible for handling the communications with the Proxy.

```
public class Proxy extends Object {
    public Proxy(String host)throws UnknownHostException, IOException {}
    public void dispose()throws IOException {}
    public Vector SendRequest(String oper, String context, Information input)
        throws ProxyException {}
   public String SEndOffLineRequest(String oper, String context,String inputStr)
        throws ProxyException {}
}
```

Respectively:

- SendRequest sends a protocol request to the Proxy using as input I nf or m at io n that contains information related to the target managed object
- SendOffLineRequest sends an off-line request to the Proxy (for instance OID mapping).

The context parameter contains protocol-related information. In case of CMIP it contains the agent AE-Title whereas for SNMP it contains the TCP/IP address of the SNMP agent. It is worth to remark that the operation parameter is a string (for instance "CMIPGet") used by Proxy to identify the droplet that implements such operation. This approach will allow in the future to support further protocols and object models such as CORBA without the need to modify the classes Proxy and Information hence to define a new object model. In fact it is sufficient to add some new droplets and define some new values for the operation parameter (for instance "CORBAGet").

The method SetObjectClass(String val) is defined like SetAttribute("objectClass", val) or the method CMIPGetAttributes() internally calls proxy.SendRequest("CMIPGet", agentAET, super /* Information * /).

The decision to base this work on the Proxy derives from the fact that, especially with the advent of Internet, applications have to be as light as possible. It does not make sense to duplicate part of the functionality of the Proxy on each network management application. Also, in case of CMIP, the Proxy should be installed by the ones who install the stack and the OSI agent, if any, and Proxy users should not be responsible for configuration or maintenance tasks.

This table shows that the proposed solution is preferable over the listed alternatives in many important aspects like application size and ease of use.

	Proposed Solution	**TCL-CMIS**	**Scotty**	**XMP**	**GOM**
Object-Oriented	Yes	No	No	No	Yes
Application Size	Light	Medium	Medium	Medium/Large	Large
Ease of Use	Easy	Easy	Easy	Difficult	Easy
Typing	Weak	Weak	Weak	Strong	Weak
Currently Supported Object Models	CMIP/SNMP	CMIP	CMIP/SNMP	CMIP/SNMP	CMIP/CORBA
Language Bindings	Java/C++	TCL	TCL	C	C++
Data Representation	String	String	String	XOM	GOM (11 types)
Metadata Access	Yes	No	No	Impl. Dependent	Yes
Pre-requisites	Java VM	TCL	TCL	XOM/XMP	Obj. Broker

Other solutions based on TCL, despite their simplicity and their similarity with the approach here described, have a bigger application size and hence cannot run unmodified on different platforms due to their use of C/C++ libraries that interface TCL with CMIP/SNMP resources. Finally, the proposed solution thanks to the Java application bindings and to its limited size enables the construction of a new class of network management applications that can be easily integrated with the world-wide web and Internet.

3. CONCLUSION

This paper shows a new approach to inter-domain management that overcomes limitations of many current solutions. Main characteristic are: ease of use, language neutral bindings, based on established technology like HTTP, small size, open to the integration of additional protocols. Bindings

REFERENCES :

1. SNMP Essentials, Second Edition, Douglas Mauro and Kevin Schmidt, O'Relly, 2005.
2. Java Programming with CORBA, 2nd Edition, Andreas Vogel and Keith Duddy, Wiley publishing.
3. Network Management: Principles and Practice, Mani Subramanian, Addison-Wesley, 2000
4. http://www.yolinux.com/TUTORIALS/CORBA.html
5. http://www.jacorb.org/
6. http://www.tcl.tk/man/tcl8.5/tutorial/tcltutorial.html