8

A HORB-based Network Management System

A. Yamanaka, S. Nakajima, M. Tomono, and T. Tonouchi
C&C Research Laboratories, NEC Corporation,
4-1-1 Miyazaki, Miyamae-ku, Kawasaki, Kanagawa 216 Japan.
Telephone: +81-44-856-2259. Fax: +81-44-856-2233.
email: `yamanaka@swl.cl.nec.co.jp`

Abstract

The advent of Web technology motivates us to explore new software architecture on heterogeneous computer networks. Network management has been getting increasingly complex and challenging because of the unprecedented growth and heterogeneous nature of today's computer and telecommunication networks. Currently, the integration of the Telecommunication Management Network (TMN) and CORBA/ORB is being pursued. A next step would be an addition of Web technology to the integration. In this paper, we propose a new network management architecture and its implementation based on Java , HORB(a Java-based ORB system), and Web technologies. A network management system with this architecture provides a portable, multi-threaded, object-oriented, distributed, and architecture-neutral environment and can be easily customized by changing the functionalities of agents without stopping the whole system by using a new technique, MIBlet.

Keywords

Distributed system, Java, HORB, NMS, Management delegation

1 INTRODUCTION

The advent of Web technology motivates us to explore new software architecture on heterogeneous computer networks where a local and a Web-based networks coexist. On local networks, CORBA/ORB (OMG, 1993) is now a standard platform for distributed software in a variety of application areas (Kinane, 1995)(Horstmann, 1995). A question arises: how do we combine both Web technology and CORBA/ORB? Exploring some concrete application system on such architecture might be the best way to study an ideal division of labor between the two.

Telecommunication network management has been one of the primary concerns in the open distributed computing research community (Kinane, 1995). As network management systems (NMSs) become larger and provide advanced management services for a wide variety of networks, distributed software technology is adapted to implement NMSs easily.

J. Rolia et al. (eds.), *Open Distributed Processing and Distributed Platforms*

Actually, integration of the telecommunication management network and CORBA/ORB is being pursued in various institutions, including NMForum and X/Open (NMF xxx, 1996). A next step would be an addition of Web technology to the integration.

The present paper describes our experience in developing telecommunication network management software using HORB, a Java-based platform for distributed software (Hirano, 1996). The NMS is a client/server system where the clients are Java-enabled Web browsers and the servers consist of a gateway and agents. Since with the Web technology the client is located at a remote location, data traffic between the client and the servers is a primary concern in designing the labor division. Our solution to this traffic problem is similar to the delegation agent (Yemini, 1995). The agent that has Management Information Base (MIB) allows down-loading and executing a fragment of application programs (called MIBlet) operating on MIB data. Data traffic can therefore be greatly reduced by shifting management application functions from client sites to the agent.

This paper is organized as follows: section 2 introduces NMS architecture and summarizes issues related to the current NMS and our approach. Section 3 describes HORB. Section 4 discusses the implementation of the HORB-based NMS in detail, and section 5 briefly discusses system performance and some of future work.

2 NETWORK MANAGEMENT SOFTWARE

The increasing complexity and heterogeneity of modern networks and the advent of distributed computing are making network management more important and more complex. Many companies and research institutions have attempted to simplify the management scenario by defining a single and consistent way for managing heterogeneous networks based on either CMIP (ISO 9596, 1988) or SNMP (Case, 1990).

2.1 NMS Architecture and Issues

Figure 1 shows a typical conceptual architecture of a NMS.

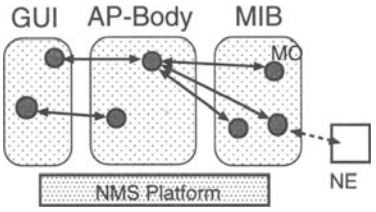

Figure 1 Conceptual Architecture

The NMS consists of a GUI, a MIB and an AP-body which mediates the former two components. These components are connected via the NMS platform. The GUI is responsible for interaction with users, and the MIB is a repository for information about

managed objects (MOs). Each MO is a software model for a particular aspect of managed network devices. The NMS AP-body is roughly a set of *functional objects*, each of which is responsible for a particular aspect of management application functions. The *functional objects* mediate between GUI objects and MOs (Nakajima, 1996).

Many NMSs have been implemented, but they have been based on a platform-centered paradigm. Each of them is organized as a two-layer hierarchy, with a platform manager executing at a workstation and agents embedded within network elements. Management agents are responsible for monitoring and controlling their devices and for collecting data in the MIB. In this paradigm, however, there are serious problems with scalability, flexibility, and efficiency.

- *Scalability*: Since a managed system becomes larger and more complex, management operations will wait longer for remote responses. Moreover, most management functions in the workstation are performed on data collected from physically separated agents. This paradigm concentrates the processing load on a single manager, which often interacts with a large number of agents.
- *Flexibility*: A manager can usually invoke only a fixed set of predefined services exported by each agent. These services remain unchanged until they are redesigned and the server is recompiled, reinstalled, and reinstantiated. If we want to add or change services, we must stop the system and reconfigure it.
- *Efficiency*: Because managers can invoke only agent services defined with general-purpose interfaces such as CMIS (ISO 9595, 1988), many management procedures require a large number of micro management interactions. As computers become faster, network latency becomes the most significant bottleneck for distributed applications.

This paper describes a new NMS architecture that is free from these problems. Its key solution is a combination of the Web technology and the Object Request Broker (ORB). In the present, Web technology consists of distributed hyperlink systems and the Java (Sun Microsystems, 1995) languages, and ORB is the Java-based ORB system HORB.

2.2 The approach

The NMS architecture we propose is based on Web technologies, Java, and HORB, ant it provides a portable, multi-threaded, object-oriented, distributed, and platform-independent environment. Since almost all the components of the system are implemented by Java programs, this NMS is very portable.

The problems with the platform-centered paradigm will be solved by submitting programs (called *MIBlets*) to the agent dynamically. *MIBlets* are Java programs and loaded dynamically to the agent, which is an example implementation of the "delegation" concept (Riecken, 1994)(Yemini, 1995). Management functions are moved from a client to an agent, and this is useful for distributed management systems. In our systems, MIBlets are passed to the agents dynamically without stopping the system. This makes the system flexible.

An ORB, HORB, is used for the data transportation in the implementation. With the HORB, Java objects can be passed between clients and servers freely.

Java-enabled Web browsers are used as the clients. This is in contrast to what is done in

the conventional NMSs, where a homemade GUI has been implemented for each system. These GUIs are not always portable.

3 HORB: A JAVA-BASED ORB

HORB is an object-oriented system that changes the style of Web programming, Internet programming, and distributed programming. It has been developed by S. Hirano at the Electrotechnical Laboratory of Japan. HORB is based on Java and written entirely by Java. so it is a very portable system. Its key features are the following (Hirano, 1996):

- True object-oriented network computing.
- Remote object creation, remote method invocation, object passing.
- 100% compatible with Sun Java and Java-enabled Web browsers.
- Architecture independent, portable, and interoperable.
- Supported Platforms: Any machine Java runs on.
- No IDL.
- Small runtime. Great for Internet PCs.

Unlike other Java-based CORBA systems, it does not require us to write IDL files. Of course, in programming heterogeneous systems, which are written with Java , C, and C++, CORBA/IDL system is useful. Our system, however, is just composed of Java programs, so "no IDL" feature helps us to program our system simply. In the CORBA systems, we have to write some information about objects as IDL: for example, a name of method and its type. In HORB, however, the `horbc` command checks Java programs and automatically generates adequate methods for the data transportation.

4 A HORB-BASED NMS

This section describes our new NMS, which is based on the conceptual architecture of shown in Figure 1 and on Java, HORB, and Web technology. In our implementation of NMS, HORB version 1.2.1. is used.

4.1 Basic Architecture and Clients

Figure 2 illustrates a basic architecture of the HORB-based NMS, which consists of four components: GUI clients, logic servers, a gateway and agents. These components are connected with HORB. The client is a Java-enabled Web browser and provides a GUI for end users. Actually, GUI objects are basically implemented as Java applets and are multi-threaded. One thread is for the interaction with users, and the others are for invoking MIBlet and for receiving events from the agents. As described in Section 4.4, events from the agents are notified asynchronously. One thread waits for these asynchronous events, receives them, and then transfers the results to the clients.

The logic server maintains a set of small Java programs called MIBlets, each of which implements a network management function that operates on managed objects stored in a MIB. MIBlets are the central components of the AP-body part shown in Figure 1. The

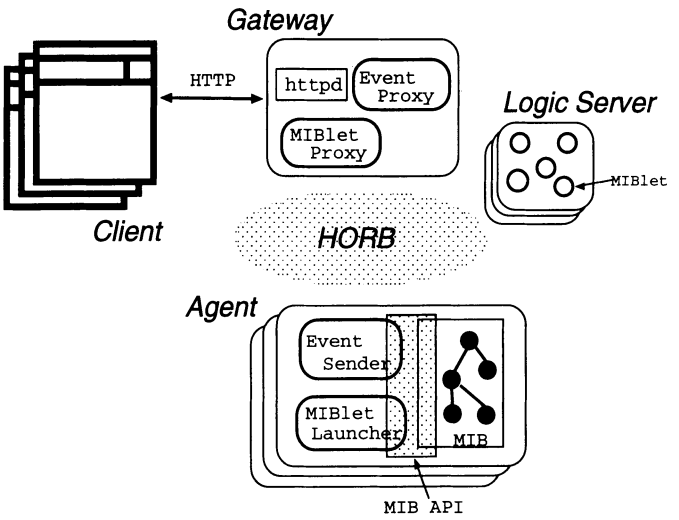

Figure 2 HORB-based NMS architecture.

agent has a MIB and provides interface functions to the distributed computing platform. The gateway mediates between the client and the MIB (actually, between Applets and MIBlets).

Roughly, the clients of Figure 2 corresponds to the GUI of Figure 1. The agent consists of the MIB and some auxiliary components that are a part of the NMS platform of Figure 1. The MIBlets maintained by the logic server constitute the AP-Body of Figure 1. Other components of Figure 2 can be considered to be the NMS platform of Figure 1.

4.2 Representation of Transported Data

In an architecture-neutral system, the sending and receiving data should be transparent. Since HORB provides a truly object-oriented programming style for distributed architectural components running on top of it, the sending of data requires invocation of a method that takes data value(s) as its argument(s), whereas the receiving of data consists of nothing more than obtaining the data value(s) returned by the method. Some Java objects should be designed to represent data sent and received.

Since Abstract Syntax Notation One(ASN.1) (ISO/IEC, 1990) is commonly used in NMSs for representing exchanged data, we have designed a set of Java objects that provide ASN.1 data values. According to the definition of ASN.1 data type, we have defined simple data types and complex data types. Simple data types are Integer, Octet String, Object Identifier, and NULL, etc. Complex ones are Sequence, Set, etc. In the following, we will show some examples of the Java objects.

The first example is class Asn, which is an abstract class for all the ASN.1-related Java objects.

```
public abstract class Asn {
  public byte id;
  public abstract String toString();
}
```

An instance variable, or a member variable id, represents the type of ASN.1 data. The Integer class is defined as shown below. Here, ID is a class that defines all the necessary constants representing data type. In particular, ID.Int is a constant that represents ASN.1 Integer.

```
public class Int extends Asn {
  public int value = 0;
  public Int() { }
  public Int(int i) { value = i; id = ID.Int; }
  public String toString() { return Integer.toString(value); }
}
```

The next example is a Sequence.

```
public class Seq extends Asn {
  public Asn value[];
  public int size;
  public Seq(){}
  public Seq(int size){this.size=size; value = new Asn[size]; id = ID.Seq;}
  public void add(int i, Asn a) {value[i]=a;}
  public String toString() { .... }
}
```

4.3 The Gateway

The gateway plays a key role in our NMS platform. It acts as a http server that establishes connections between clients and the servers. When a Java-enabled Web browser tries to connect with the servers, the gateway establishes the connection by sending necessary GUI applets to the browser, and it also provides a predefined port of socket communication to the browser. Since applets have only restricted access rights to most of the computation resources, the browser is not allowed to access various NMS functions directly. The gateway will take care of everything through the socket communication channel. As shown in Figure 3, the gateway consists of three main components: a Httpd server, a MIBletProxy, and an EventProxy.

The MIBletProxy and the EventProxy are daemon programs that are responsible for both downward and upward flow of information (downward is from clients to the agent, while upward is the other way around). Both proxies wait and listen to some connection trial by clients through the socket. When a connection is established, an execution thread is spawned in order to provide services to the clients. The MIBletProxy receives an ASN.1

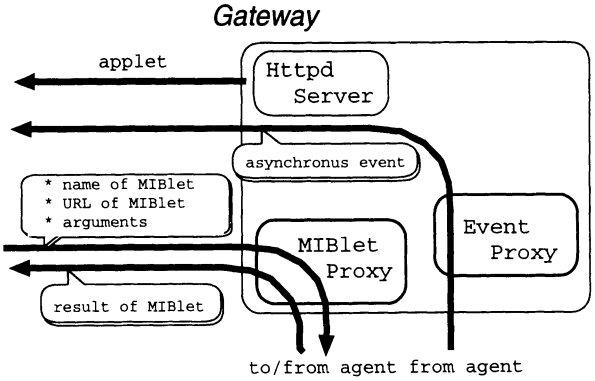

Figure 3 Architecture of the gateway.

Java Data, which contains a name of the MIBlet, the URL of the logic server where the MIBlet is stored, and arguments, and sends it to the `MIBletLauncher` in the agent. Upon completion of MIBlet execution, the `MIBletProxy` obtains the computation result from the agent and relays the result to the client. (See Section 4.4 for a detailed description of MIBlet execution.)

When the agent detects some change in MIB status, it is required to transmit the change to the client asynchronously. The `EventProxy` is responsible for this asynchronous event report to the client. (The details of this event report are also discussed fully in Section 4.4.)

4.4 The Agent

As Figure 4 shows, an agent consists of three components: a `MIBletLauncher`, a MIB, and an `EventSender`.

The `MIBletLauncher` waits for messages from clients via the `MIBletProxy`. A message is composed of three kinds of information: the MIBlet name, the URL of the MIBlet, and arguments for the MIBlet. Currently the URL is for security reasons restricted to be within the logic servers. Since MIBlet may update the contents of MIB, some restriction should be necessary. If any URL is permitted and MIBlet is downloaded from there, it is very insecure.

According to the message from a client, the `MIBletLauncher` tries to get the MIBlet from a logic server and execute it.

The interface for a MIBlet is defined as follows:

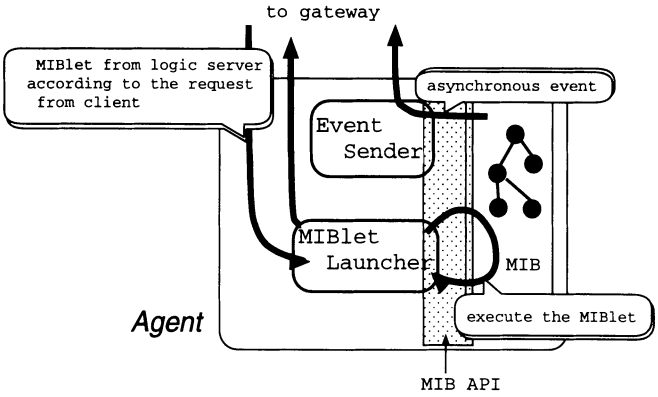

Figure 4 Architecture of the Agent

```
public interface MIBletInterface {
  public void Do(PrintStream o, Object arg[]);
}
```

Every MIBlet should implement this interface. A MIBlet has one method: `Do`. Its first argument is an output stream. If some objects are written to this stream, they are sent to the gateway automatically; the stream is connected to the gateway via HORB.

The idea of a MIBlet is similar to the recent proposal on delegation agents (Riecken, 1994)(Yemini, 1995). A MIBlet, however, has some advantages over other proposal on delegation agents. One is that we can reduce the network traffic by caching MIBlets. In the other delegation agents, programs are sent directly from a client. Some specific MIBlets are used frequently, so this caching is useful, because it means we don't have to get the MIBlet at each request. If some MIBlets are updated at a logic server, they must be reloaded. In this case, we can notify the `MIBletLauncher` to clear a part of the cache and an updated MIBlet will be reloaded from the logic server. This notification can be done by some special MIBlet, which manages the cache. Another advantage of the MIBlet is that it makes the security problems easier to solve. As mentioned earlier, the URL of the MIBlet is restricted to be within the logic servers, and the MIBlet has no access restriction in its execution time. This is a contrast to the applets. Applets are restricted with regard to the file I/O and the socket connection etc. In our system, the security problems can be concentrated in logic servers. So we only care for the access control to the MIBlets in logic servers; all MIBlets in logic servers are safe about execution. or combination of some.

The MIB is a database for managed objects: each managed object is a virtual model of network element hardware such as routers and ethernet cards. Since each MIB is usually developed without regard to the present NMS and may be implemented as C or C++ program, it is necessary to incorporate the whole MIB program in the Java execution environment in order to access contents of MIB from MIBlet. Actually, the MIB program

is incorporated by means of the native method technique of Java. MIBlet uses MIB API, which are actually to access and operate on the managed objects in the MIB. All the methods of the MIB API are implemented as native methods.

The `EventSender` is a standalone program waiting for the events from the MIB. In the MIB, unexpected problems may occur asynchronously in network elements and they must be announced to the client as soon as possible. These events will be caught by the `EventSender` and this server will send these events to the `EventProxy` of the gateway. The `EventProxy` will relay the events to the clients.

For the current NMS platform we have chosen OSI management MIB written in C++ language(Tonouchi, 1997). Therefore, our NMS provides CMIS (ISO 9595, 1988) like services to the clients. The CMIS defines many services; for examples, `M-GET`, `M-SET`,and `M-ACTION`. `M-GET` service is used to get a value from managed object in the MIB. `M-ACTION` service is used to perform some actions on a managed object in the MIB. In our NMS, these services are provided by the MIBlets and one MIBlet is presented for each service. For example, `MGet.class` is for the `M-GET` service. Of course, other special services (for example, searching some values from the whole MIB tree) are accomplished by some specific MIBlet. In the conventional platform-centered NMS, these services are provided by a combination of more than one `M-GET` service and are not efficient.

5 DISCUSSION

One might think that the Java system, (which is a bytecode interpreter system) would be the bottleneck limiting the system performance, but our preliminarily experiments have indicated that it is not and that MIBlets are very useful. At first, we show the environment of this preliminarily experiments. The gateway is located on an UltraSPARC of Sun Microsystems and the agent was located on another UltraSPARC of Sun Microsystems. The client was a Netscape Navigator on Windows 95, and these computers were connected by an Ethernet segment. In experiments evaluating two ways of getting 10 values from a MIB – by using `M-Get` services 10 times and by using one MIBlet that gets 10 values from the MIB – we found that the way using the MIBlet is about 8 times faster than the use of the `M-Get` services. If one gets 20 values, the use of the MIBlet is about 9 times faster than the use of the `M-Get` services. It seems that the advantage of the MIBlet increases as the quantity of transported data increases. This means that a bottleneck for the system performance is the network performance rather that the Java performance. Moreover, Just-In-Time compilers are being developed by many vendors, and their use would further improve system performance.

Future work should address the following issues:

- The security at various levels: client connection, MIBlet.
  As described in Section 4.4, we just care about MIBlet executions. Somebody could tamper with a MIBlet (Java bytecode) on the network transportation intentionally. We should be able to detect the change resulting from such an attack. Moreover, the authentication about connection between a client and the gateway may be necessary. These problems are outside the scope our current NMS, but they are interesting topics with regard to the distributed system. We plan to add a cryptography to our NMS.
- The efficient support for multiple agents.

Our current system can handle multiple agents at the same time. We, however, have to dispatch one MIBlets to each agent. If we want to dispatch one MIBlet to 10 agents, we have to make the request 10 times. To solve this inefficiency, we plan to introduce a "worm" feature to the MIBlet. With this new feature, the MIBlet will walk through the network devices and we don't have to make same requests to several agents.

- Flexible alarm analysis server.
  Most NMS is equipped with some specific alarm analysis server which helps to find the cause of events asynchronously issued by some of the agents and the events reflect some status changes in the managed network devices. In our implementation of NMS, it is easy to add such a server that is responsible for some specific application service because the servers can be connected by the infrastructure, HORB. Moreover, we could make the functionalities of the alarm analysis server flexible. We could send a small program fragment that provides some user-specific alarm analysis function to the alarm server. Alarmlet, instead of MIBlet would be a better name for such small program.

6 CONCLUSION

We have presented a new conceptual architecture of NMS and its implementation. This NMS enables us to manage the network devices at remote location from Java-enabled Web browsers. Since this NMS is implemented using Java and HORB, it is very portable. Since it provides a kind of delegation system based on MIBlet, it is scalable , flexible, and efficient. Flexibility is a very important property for a NMS and our NMS can easily be customized by changing the MIBlets without stopping the whole system. This is a great advantage over the conventional platform-centered NMS. As described in the last section, we plan to extend our NMS in three aspects in order to make it a practical system.

REFERENCES

Case, J., Fedor, M., Schoffstall, M., and Dabin, C. (1990) *The Simple Network Management Protocol (SNMP)* RFC 1157.

Geiger, D., Allen, W., Majtenyi, A., and Reder, P. (1994) IBM comipWorks: Technical Paper. IBM.

Hirano, S. (1996) *HORB Flyer's Guide* http://ring.etl.go.jp/openlab/horb/ .

Horstmann, T. and Wasserschaff, M. (1995) *Experiences with Groupware Development under CORBA,* ICODP'95.

Kinane, B. (1995) *Distributed Public Network Management Systems Using CORBA,* ICODP'95.

Nakajima, S., Tomobe, M., Miki, M., and Hayashi, H. (1996) *Object-Oriented Develpement Methodology for Telecommunication Network Management Software,* GLOBECOM '96.

Riecken, D. (1994) *Intelligent Agents,* Communications of the ACM, 37(7):18-21.

Tonouchi, T., Fukushima, T., Man'ki, A., and Nakajima, S. (1997) *An Implementation of OSI Management Q3 Agent Platform for Subscriber Network,* International Conf. of Communication '97. *submitted* .

Yemini, Y. and Goldszmidt, G. (1995) *NETWORK MANAGEMENT BY DELEGATION,* Proceedings of the 2nd Integrated Network Management, pp. 95-107.

ISO/IEC. (1990) *Information technology - Open Systems Interconnection - Specification of Abstract Syntex Notation One (ASN.1), second ed.*, Reference ISO/IEC.

ISO 9595. (1988) *Information Processing systems - Open Systems Interconnection - Management Information Service Definition - Part 2 : Common Management Information Service.*

ISO 9596. (1988) *Information Processing systems - Open Systems Interconnection - Management Information Service Definition - Part 2 : Common Management Information Protocol.*

NMF xxx. (1996) *TMN C++ Application Programming Interface* Issue 1.0 Draft 5 - For Public Comment, January 1996.

OMG. The Common Object Request Broker. (1993) *Architecture and Specification 1.2 edition.*

Sun Microsystems. (1995) *The Java Programming Language.* Addison-Wesley, 1995, ISBN-0-201-63455-4.

BIOGRAPHY

Atsuhiko Yamanaka received B.A. and M.S. in Computer Science from Tohoku University and now a PhD candidate at Tohoku University. Since 1996 he has been working at C&C Research Laboratories, NEC Corporation. His current research interests are Object-oriented frameworks, Software architecture, Type theory, and Application of formal method to software development. yamanaka@swl.cl.nec.co.jp.

Shin Nakajima received his B.A. and M.S. degrees in Physics from the University of Tokyo. He is leader of the Object-Oriented Software Architecture group at C&C Research Laboratories, NEC Corporation. His current research interests include distributed software architecture, object-oriented algebraic specification, and meta-level software architecture, whose primary application area is network and system management. He is visiting lecturer at Tokyo Metropolitan University. nakajima@swl.cl.nec.co.jp.

Masahiro Tomono received his B.A. and M.S. degrees in Electronics from the University of Tokyo. Since 1985 he has been working at C&C Research Laboratories, NEC Corporation. He has been engaged in CASE, Reverse engineering, Object-oriented frameworks and Software architecture. His current research interests include distributed-software architecture and design patterns, design methods and tools, and network and system management. tomono@swl.cl.nec.co.jp.

Toshio Tonouchi received B.A. and M.S. in Computer Science from the University of Tokyo. Since 1992 he has worked in C&C Research Laboratories, NEC Corporation. His current research interests include a high performance implementation of OSI management Q3 agent platorm and Application development support tools for Q3 agents. tonouchi@swl.cl.nec.co.jp.