JNMWare: Network Management Platform Using Java Technologies

Jae-Oh Lee and Jae-Yeol Kim
Dept. of Network Management, WarePlus Inc.
{jolee, kimjy}@wareplus.com

Abstract
As many Java applications are deployed in the real system, Java based network management is one of the emerging area for various technology parts. Many of network management platforms require the distributed platform for supporting service and business management areas. Many Java technologies related to network management have been developed; therefore it is desirable to create the development environment using Java technologies. Also, the increasing integration of telecommunications networks creates possibilities for interoperability among their respective management systems, using common management interfaces for CMIP, SNMP, CORBA, and Java RMI. In this paper, the network management platform (i.e., JNMWare) using Java technologies is introduced.

1. Introduction

The increasing integration of telecommunications networks creates possibilities for interoperability among their respective management systems, using common management interfaces such as CMIP, SNMP, CORBA, and Java RMI. Many Java-based applications related to network management (e.g., JMAPI, JNDI, Java IDL, etc.) have been developed; therefore it is desirable to create the development environment using Java technologies. In this paper, we describe the TMN-based management platform using Java technology that can support some certain candidate management interfaces as service components in a distributed computing environment. Due to the increasing complexity and heterogeneity of networks and services, many efforts have been made to develop distributed techniques for network management. These require the integration of service software components and the management components that manage services. This integration makes use of component reusability and results in the need to support multiple interfaces among components. From the perspectives of systems and network management, major interfaces include CMIP, SNMP, CORBA, and RMI [1,16]. CMIP and SNMP are dominant network management protocols, which have been used to exchange management messages to manage element and network resources, while object-oriented middlewares such as CORBA and RMI are considered to be valid candidates for service and business management.

Many organizations are currently developing standards for distributed management. The TINA-Consortium applies the principles of ODP and Object Management Group (OMG) standards to the needs of the telecommunication industry. Also, the work of the XOJIDM group specifies the translation of GDMO/ASN.1 to Interface Definition Language (IDL) to offer OSI standards-compliant management [12,13]. The TeleManagement Forum (TMF)'s SMART working group is addressing the standard related specifically to service management. The Web-Based Enterprise Management (WBEM) paradigm aims at specifying a new protocol, namely Hypermedia Transport Protocol (HMTP). All of these distributed management approaches demonstrate the variety of management interfaces which are being used, and will be used, for distributed systems management. Java, due to its strong connection to Web-based applications, and also due to its simplicity in Internet-oriented
programming, is currently very popular within the Internet community and is also supported by the majority of platforms. New Graphical User Interface (GUI) architectures can support the Java-based approach for service and business management.

In this paper, we describe the design and implementation of a management platform based on TMN using Java technologies that can support a variety of management interfaces. Some candidate Java technologies for network management are briefly presented. The translation tools for management information are described. The management platform that is used as a core facility for network management applications is identified. A management system implemented using the management platform is explained. Finally, we summarize the paper and explain considerations of future work.

2. Java Technologies

Objectives for JNMWare platforms require that management interfaces are standards-based, scalable and sufficiently robust for high availability systems. The management functions must be useful to provide the interoperability among them. Many devices and equipment provide SNMP/CMIP as management interface. Also, many service and business management systems want to access network management systems via their specific interfaces such as CORBA, RMI, CMIP and SNMP. The Java programming language environment provides a portable, interpreted, high-performance, object-oriented programming language, and includes a supporting run-time environment. The framework of the JNMWare platform using Java technology provides [17]:

- A TMN-based management framework for the adaptation of higher- and lower-level management systems conforming to standard interfaces.
- Integration of existing and future Java-based applications.
- Easier and simpler development of TMN-based management functions.

Management functions/services consist of management components that are closely related to the Fault, Configuration, Account, Performance, and Security (FCAPS). A management service is associated with multiple management functions. Fig. 1 shows management platform components and some candidate Java technologies.

Figure 1: Management Platform and Java Technologies
The Java programming language has evolved to provide not only better portability, but also improved productivity. While Java was initially used primarily for user presentation components, it is now being used in a wide variety of applications. From the development process and environment point of view, the impact of introducing Java will have to be taken into account all process levels. Java enables developers of protocol-independent network management systems to implement simple management applications and manipulate them in a dynamic way. There are some candidate Java technologies which are applicable to the management platform.

- **Object Serialization**: The capability to store and retrieve Java objects is essential to building all but the most transient applications. Object serialization may be extended to support persistence of Java objects. For the purposes of network management, persistence can be used for storing management information and interface between the management system and the GUI. All message objects must support object serialization [9].

- **Java Dynamic Management Kit (JDMK) and Java Management Extensions (JMX)**: Experimentation of Java Dynamic Management Kit (JDMK) [2] has demonstrated that this specification defines standard ways to make Java technology-based platforms, applications or devices manageable, and introduces Java technology-based dynamic management for the service age. JMX [3] provides a set of standard APIs at instrumentation and agent services levels, as well as APIs to facilitate integration with existing SNMP or WBEM-based systems.

- **Java Naming and Directory Interface (JNDI)**: Directory services play a vital role in Intranets and Internets by providing access to a variety of information about users, networks, services, and applications. JNDI can provide a variety of existing naming and directory services – existing ones such as DNS, NDS, NIS (YP), and X.500, and emerging ones such as LDAP. Use of LDAP would allow a directory-enabled MIT [5].

- **Remote Method Invocation (RMI)**: The Remote Method Invocation (RMI) [6] enables the programmer to create distributed Java-to-Java applications, in which the methods of remote Java objects can be invoked from other Java virtual machines, possibly on the different hosts. The distributed system interfaces can be designed in Java, and clients and servers can be implemented using those interfaces. The management applications having this mechanism can be in the scope of service and business areas.

- **Java Data Base Connectivity (JDBC)**: Java Data Base Connectivity (JDBC) [4] allows programmers to write to a single database interface, enables DBMS-independent Java application development tools and products, and allows database connectivity vendors to provide a variety of different connectivity solutions.

- **Java Servlets**: Servlets [11] are protocol-and platform-independent server-side components, written in Java, which dynamically extend Java-enabled servers. They provide a general framework for services built using the request-response paradigm. They can be also deployed into the world-wide web using the Java bindings by providing servlet classes.

- **Java Beans**: A Java Bean is a reusable software component that can be manipulated visually by means of a builder tool. The builder tool may include web page builders, visual application builders, GUI layout builders, or even server application builders. Beans are appropriate for software components that can be visually manipulated and customized to achieve some effect. For example, alarm-reporting functions can be defined using the Java Bean event feature [8]. A Java class supporting the manager system can be dynamically configured to manage the different agent systems, which have their own specific management information, interfaces and management services. Connecting managed object beans and services beans together in a visual application development environment can be used to rapidly develop management applications.

- **Java Native Interface (JNI)**: The Java Native Interface (JNI) [7] is a native programming interface. It allows Java code that runs inside a Java
Virtual Machine (VM) to interoperate with applications and libraries written in other programming languages, such as C, C++, and assembly. Using native function calls, a device-specific access method can be created by the developer.

- **Java Interface Definition Language (IDL):** The Java Interface Definition Language (IDL) [10] lets the developer create distributed Web-enabled Java applications that can transparently invoke operations on remote network services using the industry standard IDL and Internet Inter-ORB Protocol (IIOP) defined by the OMG. The idltojava stub generator produces stubs that are ORB independent, and which call into ORB specific protocol modules for all data marshalling or other ORB specific operations.

### 3. Management Information Tools and Management Communication

The semantics of the various components of Managed Object Class (MOC) is distinguished from the corresponding format. The syntax is defined by the ASN.1, which provides a wide variety of types ranging from simple bit strings to complex structures. The parameter template qualifies and further defines the structures in the syntax of attributes, action requests/responses and notifications.

#### 3.1 GDMO/ASN.1 Tools

All of ASN.1 Java objects are divided into an aggregate Java object (e.g., SEQUENCE, SET, CHOICE) and a non-aggregate Java object (e.g., INTEGER, OCTETSTRING, OBJECT IDENTIFIER), each having its constructors and methods. The generated ASN.1 Java objects are the aggregate Java objects in which the non-aggregate Java objects are provided as base object classes. In addition to the standard encode, decode, print and clone methods, they can also have special constructors and other routines that simplify their use. The inheritance hierarchy of the non-aggregate ASN.1 Java objects is defined as following.

![Figure 2: The Inheritance Hierarchy of Non-aggregate ASN.1 Java Objects](image-url)
They represent non-aggregate ASN.1 objects. Also, there are some support Java objects such as \texttt{AsnBuf} for buffer management, \texttt{AsnConfig} for the manipulation of ASN.1 configuration, \texttt{StringStack} for the utilities of bit string and octet string and \texttt{AsnAnyDefinedBy} objects. Unlike the objects generated for some of the aggregate objects such as SETs, and SEQUENCEs, the non-aggregate objects’ data members are typically protected and accessed via their methods.

The adoption of an object-oriented approach to model management information reduces the design of a managed object component to an “ad hoc” composition of basic modules. These modules are defined in the GDMO specification of the MOC. In this way, it is also possible to reuse modules as attributes in different classes, thus reducing the amount code to be written for each MOC but also making the integration of different modules easier. Defining an API for each basic piece, which in Java is a set of attributes and methods, allows modules to be integrated from the Java development kit. The GDMO/ASN.1 translators are used to define the Java objects needed to implement the MOC and the transfer syntax [12, 13]. They are classified into “ASN.1 To Java Object (A2J)” and “GDMO To Java Object (G2J)”. Fig. 3 shows the process flow of translation tools and the generated files to be compiled by the Java compiler [14,15].

“\texttt{A2J}” compiles ASN.1 modules into Java objects. The generated Java code contains equivalent objects and core routines to convert values between the internal (Java) representation and the corresponding Basic Encoding Rules (BERs) format. “\texttt{G2J}” also produces the Java objects from the GDMO templates which have their own defined Java objects imported from ASN.1 Java objects and their inheritance relationships. The number of the Java objects is equal to the number of the GDMO templates. The automatic creation of Java objects from the documents described by GDMO/ASN.1 as input files is necessary to create the association of management information with the management factory and application. The value Java object defines the static variables such as object identifiers and integers. Each ASN.1 Java object has some protected data members, constructors and methods. The basic methods perform assignment, encoding, decoding, and printing facilities. Each CHOICE type generates a Java object that has an anonymous union to hold the components of the CHOICE and a choice identifier field to indicate which component is present.

### 3.2 Management Communication

In order to follow the TMN architecture, it is very desirable to design the framework of the management platform using Java technologies which focus on the CMIP-based management interface and information model. This approach can therefore be easily embedded in a standards-conformant network and service management environment. TMN systems have three major requirements: extensibility, flexibility and openness. According to the M3010 standard, TMN focuses on object-oriented information technology, distribution of applications, and standard interfaces. As depicted in Fig. 4, CMIP/CMIS Java object has some methods.

![Figure 3: GDMO/ASN.1 Translators](image)
They include the service primitives for the basic management messages defined in CMIP. In addition, the Java objects and their related methods are used to support the associations which are made between manager system and agent system. As it includes some Java objects for CMIP operations and CMIP service errors, the developer who can use Java language might make the implementation of management functions and management services. Also, it can support the functionality of multiple reply (i.e., scoping, filtering, synchronization) which is performed by LinkedReply.

The following indicates some major Java APIs for CMIP/CMIS.

- **XMP\_(NA)\_Initialize**: The non-automatic/automatic initialization of ACSE Information.
- **XMP\_SetLocal/RemoteAddress**: The setting of the self/opposite OSI address.
- **XMP\_CreateContextWithAETitle**: The creation of a management context with AE-title (i.e., the opposite AE-title).
- **XMP\_CreateSessionWithAETitle**: The creation of wanted session with AE-title (i.e., the self AE-title).
- **XMP\_Connect**: The execution of ACSE connect operation.
- **XMP\_Disconnect**: The execution of ACSE disconnect operation.
- **XMP\_(Milli)\_TimeOut**: The time out value of management operation in a second/milli-second unit.
- **XMP\_WaitReceive**: The waiting and receiving the event report message and management response message.
- **XMP\_AssociationReq/Rsp**: The operation of M-ASSOCIATE request/response.
- **XMP\_ReleaseReq/Rsp**: The operation of M-RELEASE request/response.
- **XMP\_AbortReq**: The operation of M-ABORT request.
- **XMP\_xxx\_ArgumentReq**: The operation of M-xxx request in asynchronous mode.
- **XMP\_xxx\_ResultRsp**: The operation of M-xxx response.
- **XMP\_xxx\_Operation**: The operation of M-xxx request in synchronous mode.
- **XMP\_LinkedReplyRsp**: The operation of M-LINKED-REPLY response.

4. Architecture of JNMWare

4.1 Management Platform Framework and System

A management platform supporting multiple management interfaces is described in this section. The presumed interfaces are used for SNMP, CMIP, CORBA, and RMI. CMIP and SNMP have been predominately used as management communication interfaces and are directly adaptable to this platform. For the adaptation of CORBA-based management systems, the translation between IDL and Java can be used for the definition of the management interface and information. This allows a Java server to define objects that can be transparently invoked from remote CORBA clients. The RMI system also provides the capability to remotely invoke methods. This ability enables the development of distributed applications. Applications can execute and communicate so that they can run in parallel across multiple systems connected via a network. But the Java objects of management protocols and managed object definitions for CORBA and RMI need standardization. In current, we assume that
management protocols and managed object definitions for CORBA and RMI focus on TMN-based model. In order to follow the TMN architecture, it is very desirable to design the framework of the management platform using Java technologies which focus on the CMIP-based management interface and information model.

As depicted in above figure, the management system performs basic functions such as receiving and responding to management operations, sending of alarms, sending of performance data via management interfaces and providing management services via internal management operations such as create, delete, get, cancel-get, set, and action. Management services can be represented as Java thread objects and can be implemented by the manipulation of Managed Object Class Instance (MOCI) in the MIT via management interfaces, either remotely or locally. Remote monitoring can be polling-based, as is the case with SNMP-based systems, or event-driven. TMN management encourages an event-driven paradigm through facilities, such as object management, metric monitoring and summarization. In general, object management provides object creation, deletion and attribute value change notifications. Each MIT node must have service interfaces to access the service points provided by management interface mapping modules. Priority control is needed for system and performance tuning. This is easily accomplished by controlling the priority of every thread object. The management system may require configuration updates to control and alter the way in which it behaves or performs. Often, there is a requirement for more than one system to receive this configuration information and for ensuring that updates are performed automatically across all those systems. Some intelligence (e.g., the determination of VP and VC identifiers) is required and may be extended to support enhanced functions by exchanging management messages with other management systems and/or other intelligent systems. Each MOCI can access the management system via multiple management interfaces.

The management system is implemented as a
case of the framework of the Q-Adapter (QA) platform. To be compatibility with the existing standard conformant agent system, the QA system having CMIP-SNMP/CMIP-CMIP is implemented to manage ATM switches (e.g., ForeRunner ASX). When the QA receives CMIP messages from CMIP-based manager, it operates on the ATM switch via SNMP and maintains management information as MIT. Also, trap message is converted to CMIP event report message, which is forwarded to the registered destinations.

4.2 Management Function

In general, the processing capability of the system depends on the complexities of time and memory for processing information. Each node is composed of the MOC name in a string format and the Relative Distinguished Name (RDN) level for unique identification [16,17]. In addition, it has a list of information about the MOCIs. It has a linked list structure in a node. Each list consists of distinguished name, instance identifier, parent instance identifier, right instance, a set of attributes and next instance. This node information is the basic information applied to the management operations of create, delete, set, get, cancel-get, and action. Some of them can have their own scope and filter fields. Here, we must consider some key parameters for determining the processing capabilities required to carry out management operations involving other management systems. These are: 1) the encoding/decoding time of management messages; 2) the search time of the MIT; 3) the construction/deconstruction time of the MOCl; 4) the processing time for accessing the database; and 5) the response time from real resources or adaptable management interfaces. Traversing of the MIT for each management operation is done in pre-order method and started at the root node of the MIT. The containment relationship of the MIT is used for the naming service, which can be accomplished using JNDI.

The following figure indicates the situation in which the management services performing the periodic reporting function are carried out.
performance of management system. When a verified operator downloads a web page referring to an applet, the applet code is dynamically loaded and run within the operator’s Java virtual machine, which is typically part of the web browser.

4.3 Management Information Persistence

In this section, two methodologies for making the persistence of management information management platform are described. Fig. 7 describes the procedures about the persistence of management information and the execution of management application.

![Figure 7: Management Information Persistence](image)

The first one is accomplished by storing the management information into database. For this approach, some methods of Java object are provided by this platform. Each field of attributes of MOCI is stored in a stringed encoded format using these methods. The determination of attribute is accomplished by performing the decoding operation to its field data extracted from database. Basically, connectivity to the databases is established through the JDBC drivers, where each of those drivers is developed to provide access to proprietary database management systems. We use a simple client-server paradigm, where the client, through the JDBC driver, sends SQL queries to the server. When connecting to the database, each MOC has its own basic methods such as “connection with database server”, “query operation”, “connection close”, i.e., normal or abnormal, and so on. MOCs for accessing databases (i.e., in the current Mini SQL (MSQL) database) are imported from the “MSQL” Java object. The “MSQL” object class provides methods to perform manipulation of the database, such as Create, Drop, Insert, Delete, and Select. In this case, management application accesses the specified database when it performs some management operations. When the persistent management application wants to initialize its management service, it can reconstruct MIT and management service by recreation of the necessary MOCIs from the database.

The second one is using the object serialization interface of Java technology. This one is very useful when the management information is preserved. This situation can be occurred when the management system is corrupted by the normal or abnormal situation. In this case, management application serializes its MIT caused by some interruptions. The persistent management application can reconstruct its management service by deserializing the MIT from the specified file.

5. Conclusions

Nowadays the telecommunications market is characterised by fast evolving technology, a multiplicity of providers, and the need for value-added services (e.g., multimedia) services. Such trends promote rapid introduction of new and enhanced services and their management. Telecommunications services requirements for reusability, rapid provisioning, interoperability, reduced service interactions management, and distribution must be met. Because Java provides simplicity, an object-oriented approach, reliability, portability and multi-threading, Java-based distributed object computing has become increasingly popular as more complex products are written using a multi-tier architecture. Since many Java-based applications for network management have been developed, it is desirable to create the management platform using Java technologies. For
these reasons, we have developed a management platform (i.e., JNMWare) using Java technologies, which meets the network management requirements of the next generation. In the future, we will enhance the functionality of the management platform by integrating existing and new Java technologies. Also, we have been provisioning a distributed computing environment using CMIP/CMIS Java object which can be run on Java RMI. Here, it is referred to as CMIP Over RMI (COR) for a candidate management interface.

[References]