A Component-based Operations Support System Platform
Supporting Quality of Service Policy

Si-Ho Cha*, Jae-Oh Lee**, Kuk-Hyun Cho*
*Kwangwoon University
**Korea University of Technology and Education
({sihoc, khcho}@cs.kwangwoon.ac.kr, jolee@kut.ac.kr)

Abstract
Provisioning services with guaranteed Quality of Service (QoS) in current QoS-enabled networks is not possible without the network and service management support. QoS can only be effective and provide guaranteed services if QoS elements are adequately configured and monitored. However, today’s network management platforms are not QoS-aware and administrators are forced to investigate each QoS-enabled device to check QoS parameters. To address these problems, this paper introduces a Component-based Operations Support System (COSS) Platform with QoS policy support. The COSS conforms to Model-View-Controller (MVC) architecture, and bases on IETF Policy-Based Management (PBM) and Enterprise JavaBeans (EJB) technologies. And we show the policy-based QoS management system applying the COSS platform for DiffServ networks.

1. Introduction
As the Internet evolves towards the global multi-service network of the future, a key consideration is support for services with guaranteed QoS. QoS can only be effective and provide guaranteed services if QoS elements are adequately configured and monitored. Thus, in addition to the management of network elements and related services, an OSS must also manage QoS aspects [1, 2]. QoS management must take place within the same environment used to manage standard network elements and management platforms should be aware of QoS. A common environment is required to allow administrators to proceed with QoS and standard management-related tasks in an integrated fashion [2, 3].

The Internet has brought a new paradigm of communication and networked services, and it has also brought a set of new IP-based technologies, new technical standards, and new business styles [4]. The rapid deployment of IP-based communication systems requires an Operations Support System (OSS) that can handle various new IP-based services with guaranteed QoS and support the rapid deployment of new services.

It will be possible to accomplish by using a QoS-aware COSS platform. To address these issues, The QoS-aware COSS platform [5] conforms to MVC model and bases on Internet Engineering Task Force (IETF) PBM and EJB technologies. In the COSS platform, All QoS policies and service management objects are implemented as EJB beans in the EJB container and server of the COSS platform.

The IETF has been investigating policies as a means for managing IP-based multiservice networks. Policy-Based Management (PBM) has emerged as a promising paradigm for network operations and management. It is based on high-level management policies, that is, business rules that describe the desired behavior of the network in a way as independent as possible of network devices and topology.

A management framework based on EJB technology simplifies service management applications by handling many details of application behavior automatically, without complex programming. By employing an EJB-based management framework, application developers no longer need to develop applications against specific operating system Application Program Interfaces (APIs) or specific vendor’s middleware APIs [6].

Because the COSS employs EJB, It will reduce significantly development cost and time-to-market for management applications. It will also improve the maintainability, reliability and overall quality of applications. Because the COSS employs PBM, It will simplify the operations of network management. It will
also provide QoS management for administrators.

This paper is organized as follows. Section 2 investigates technologies related to COSS platform with QoS support and section 3 presents an architecture of the COSS platform. Section 4 shows the policy-based QoS management system applying the COSS platform for DiffServ networks and section 5 describes its implementation. Finally in section 6 we present conclusions and future works.

2. Background

2.1 Policy-Based Management

PBM means to manage the behavior of a network through high-level business rules or directives which describe the behavior of the network in a way as independent as possible of network devices and topology. The amount of network management task can be reduced by using policies because one policy can be used for policy targets that are of various types, i.e., network nodes or interfaces, and that have been developed by a variety of vendors [7]. Policies are modeled by the Policy Framework Working Group of the IETF. A protocol called Common Open Policy Services (COPS), its usage called COPS-RSVP and COPS-PR, and data formats conveyed by COPS-PR, witch are called Policy Information Bases (PIBs), are also standardized by the IETF.

![Policy Framework](image1)

Figure 1. The IETF Policy Framework

Figure 1 presents the IETF policy framework. The Policy Management Tool (PMT) takes as input the high-level policies that a user or administrator enters in the network and converts them to a much more detailed and precise low-level policy description that can apply to the various devices in the network. The Policy Repository (PR) is used to store the policies generated by the PMT, and can be used to store both high-level and low-level policies. The Policy Decision Point (PDP) is responsible for interpreting the policies stored in the PR and communicating them to the Policy Enforcement Point (PEP). PEP is a device that can apply and execute the different policies.

2.2 Network Management with EJB Technology

The EJB framework is a platform-independent, interoperable, and server-side component-based model, and provides low-level services, such as support for transactions, concurrency, persistence, security, multi-threaded execution and life cycle management for the enterprise beans. Figure 2 illustrates an overview of the EJB framework.

![EJB Framework](image2)

Figure 2. The EJB Framework

There are two important types of EJB beans: session beans and entity beans. A session bean is a non-persistent object that implements some business logic by running process for the client on the server. Session beans are usually implemented long-lived transaction style application logic to access several entity beans. Session beans can be stateless or can maintain conversational state across methods and transactions. An entity bean is a persistent object that represents data maintained in a data store and is a shared resource. The internal values of an entity bean are synchronized with the contents of persistent storage through the runtime service. Entity beans represent business entity objects that exist beyond the lifetime of the EJB container...
process. An entity bean is identified by a primary key. There are two types of entity beans by the way entity persistence is accomplished: container managed entity beans and bean managed entity beans. A business entity is implemented as an entity bean or a dependent object of an entity bean. A conversational business process (a business process with one actor) is implemented as a session bean. A collaborative business process (a business process with multiple actors) is implemented as an entity bean. If it is necessary for any reason to save the intermediate state of a business process in a database, the business process is implemented as an entity bean [8].

To implements an enterprise bean, the bean developer needs to define the bean’s home and remote interfaces that will be used by the EJB container to create the client view of the enterprise bean. The home interface defines the bean’s life cycle methods: methods for creating new beans, removing beans, and finding beans. Client uses a Java Naming and Directory Interface (JNDI) naming server to look up an object reference of a Home interface. Home interface is responsible for creating a new Bean instance, and returns an object reference of a remote interface. The remote interface for an enterprise bean defines the bean’s business methods: the methods a bean presents to the outside world to do its work. The remote interface accepts requests from the client and delegates them to the EJB Bean, which executes these requests under the control of the EJB Container [5, 6].

Bean instances are created and managed by the EJB container. Beans can be customized at deployment time by editing their environment properties. Development information is separated using eXtensible Markup Language (XML). Clients only access beans via the EJB container and server. Beans that use standard container APIs run everywhere. Bean integration possible without source code changes.

A management framework based on EJB technology simplifies network management applications by handling many details of application behavior automatically, without complex programming. By employing an EJB-based management framework, application developers on longer need to develop applications against specific operating system APIs or specific vendor’s middleware APIs. A management framework based on EJB technology has a number of benefits as follows: encapsulation of business logic, simplified application development, easy Enterprise Application Integration (EAI), extensibility, scalability, consistency, transaction management, database and directory access, container-managed persistence, and distributed object access, and so on [6]. Thus, we adopt the EJB framework to build a distributed COSS platform.

3. The Architecture of COSS Platform

3.1 Multi-tier MVC Architecture

The application architecture of the COSS based on EJB framework adopt the multi-tier Model-View-Controller (MVC) architecture which is highly manageable and scalable, and provides the overall strategy for the clear distribution of objects involved in managing service. A Model represents the structure of the data in the COSS, as well as COSS-specific operations on those data. A View presents data in some form to a user, in the context of some business function. A Controller translates user actions and user input into business method calls on the Model, and selects the appropriate View based on user preferences and Model state [9].

Figure 3 shows the multi-tier MVC architecture. The multi-tier architecture provides adaptable solutions for integration with existing and future systems and for deployment of new systems and interfaces as needs arise. The Client-tier is a Web browser interacting with the Presentation-tier. The Presentation-tier runs a Web server to handle client requests, and invokes Servlets and JSPs. The Business-tier runs an EJB server to handle EJB components. EJB components constitute the core business logic for the COSS. EJBs receive requests from the Presentation-tier. EJBs then satisfy the request usually by accessing some data sources or network elements, and the results to the Presentation-tier. Finally, the integration-tier typically consists of one or more databases and network management agents. Java Database Connectivity (JDBC) drivers are typically used for accessing databases and Simple Network Management Protocol (SNMP) is used for accessing network elements.
3.2 Distributed Platform

The distributed platform architecture for the COSS consists of Presentation-tier and Business-tier [see Figure 4]. Servlets and JSP pages can encapsulate state in the Presentation-tier with JavaBeans components inside the Servlet container. The COSS uses JavaBeans to encapsulate application state such as the contents of the ordering and the problem handling. COSS’s management components are implemented as entity or session beans depending on their functional requirements in the Business-tier. COSS performs management actions through a service management agent (SMA) and real management beans such as Inventory bean, Customer bean, Product bean, and Resource bean. A SMA is implemented as a session bean, and provides access to other beans and guides the service management on behalf of the client system. It creates customer accounts, provides service information, and guides the provisioning state model by communicating to internal COSS management beans.

Because Inventory bean, Customer bean, and Product bean generally require persistence, it is desirable that they are implemented as entity beans. However, Product Type bean and Resource bean are implemented as session beans, because they don’t require persistence.

The Inventory bean encapsulates the Network Resource Inventory (NRI) database and supports three functions: availability check, configuration attributes and access to inventories. The COSS uses the NRI database to query for availability for a specific product type. Among other information, NRI maintains management data concerning service availability and resources required by services. The Resource bean manages a set of inventory items whose function is to query for available inventories and to reserve inventory.

The Customer bean represents the COSS view of a customer. The individual Customer bean of COSS needs to access data across the tables of the Customer Service database to query for individual customers of a business unit.

The Product bean represents the class of product types that is associated with a service. An instance of a Product bean represents a service, which is provisioning for a customer. A Product Type bean encapsulates service-describing data and provides information such as service availability check requirements for the Product. Any instance of the Product Type bean can be used for guiding the installation process of the Product bean to which it is related. Thus, once the Product Type bean completes its task as requested by the SMA, it is returned to the pool of available Product Type beans until the SMA issues another request for the same type of service. However, once a Product bean is created, it will exist until the customer cancels the service. This coupling provides a more flexible way of adding new services [10].

The QoS policies are stored in a QoS policy directory. They are represented as entity beans in an EJB container. The JNDI API and the Lightweight Directory Access Protocol (LDAP) provide lookup services for this directory.
When the COSS performs management actions through accessing the network elements, the distributed platform also allows the client system or the internal manager beans to use the software component plugging in the platform.

4. Policy-based DiffServ QoS Management System Based on COSS Platform

In this section, we present a simple example. That is a policy-based QoS management system applying the COSS platform for DiffServ networks. The outline of our example model for policy-based DiffServ QoS management system is shown in Figure 5.

The policy-based DiffServ QoS management system based on the COSS platform conforms to the MVC architecture and Session Façade pattern. The policy-based DiffServ QoS management system roughly consists of a Web server and an EJ B-based QoS manager. We use a Web browser-based interface as Graphical User Interface (GUI). To create and administer high-level QoS policies to be enforced on the DiffServ network, an administrator uses the Web browser-based interface.

The topology information retrieved from the network through SNMP MIB-II tables (an ipAddrTable and an ipRouteTable) is stored in a Topology Database and is represented as Topology Node (TN) entity beans in the EJB container of the EJB-based QoS manager. A Topology Management Agent (TMA) session bean guides this procedure. A JDBC driver connects the EJB container to the Topology Database. High-level DiffServ QoS policies are represented as valid XML documents through a Java Servlet and low-level ones are represented as policy entity beans in the EJB container of the EJB-based QoS manager through a Policy Management Agent (PMA) session bean. In the EJB-based QoS manager, there are three low-level policy entity beans for managing QoS of the DiffServ network: Classifier (CL) policy, Meter & Action (MA) policy, and Queuing & Scheduling (QS) policy. The CL policy and MA policy are usually used in edge routers, while QS policy is usually used in core routers. They are stored in a QoS policy directory. A QoS Management Agent (QMA) session bean guides the QoS management operations processing those policy beans. The JNDI API and the LDAP provide lookup services for the QoS policy directory.
The EJB-based QoS manager uses SNMP to distribute those low-level policy beans, namely, the EJB-based QoS manager converts the low-level policies into the appropriate DiffServ MIB [11] values that will conform to device appropriately. The QoS monitoring information of the DiffServ network is also obtained from the DiffServ MIB. We have established a set of DiffServ routers in the Linux systems and have added an SNMP agent for the DiffServ MIB and MIB-II in each router.

PMA session beans and QMA session beans accomplish the QoS policy management by using three QoS policy entity beans above for managing QoS of the DiffServ network.

Figure 6 shows the sequence diagram for a set of QoS policy management procedures. As demonstrated in Figure 6, the QoS policy management procedures are composed of policy creation, policy validation, policy translation, and policy distribution. The CL policy bean classifies packet flows and assigns Classifier Identities (CIDs) on them. A MA policy bean meters packets and performs actions on them: marker, counter, absolute dropper, and so forth. QS policy beans queue and schedule, or drop packets. By using three QoS policy entity beans above, the PMA session bean provides the policy translation and the QMA session bean provides the policy distribution.
The most basic check of policy validation is that of validating the syntax of the policy specification. In our model we defined a XML Schema file to validate high-level QoS policies described as valid XML documents with an XML Schema. A Java Servlet receives a policy information from an administrator and creates an XML policy document and then validate them through an XML Schema. After a set of these actions, the Java Servlet requests a PMA session bean instance that it creates entity bean instances of low-level QoS policies. The PMA session bean provides the policy translation by properly setting attributes of the CL policy bean, the MA policy bean, and the QS policy bean. These beans represent the behavior and data of a business object. The PMA bean uses a CL policy bean, a MA policy bean, a QS policy bean to translate high-level QoS policies to low-level QoS policies. High-level policies related to edge routers may be applied to CL policy beans and MA policy beans. While high-level policies related to core routers may be applied to QS policy bean.

In the policy-based DiffServ QoS management system, the DiffServ QoS policies mostly conform to the SNMP DiffServ MIB. In other words, policy beans perform SNMP operations for DiffServ MIB. A QMA session bean provides the policy distribution through a CL policy bean, a MA policy bean, and a QS policy bean. A CL policy bean is usually deployed to the ingress interfaces of edge routers and applies to inbound traffic. MA policies are usually deployed to the egress interfaces of edge routers and apply to outbound traffic. A QS policy is usually deployed to the egress interfaces of core routers and applies to outbound traffic. A set of those actions is accomplished by using the DiffServ MIB. The DiffServ MIB describes the configuration and management aspects of DiffServ nodes. The MIB contains the functional elements of the datapath, using various tables. The idea is that RowPointers are used to combine the various functional elements into one datapath. The DiffServ MIB tables are categorized in four architectural DiffServ elements, which are classifier, meter, action, and queue. The main advantage of using SNMP for policy distribution is that it is likely to work across all the routers in a standard manner.

Distributed policies may not behave as stated in the policy definition. The QoS resulted from a policy distribution can be different from its specification. Therefore, an administrator must monitor the QoS resulted from a policy distribution. QoS monitoring is also accomplished by using the DiffServ MIB. The QMA session bean access policy definition in the CL policy bean, MA policy bean, and QS policy bean and compare the observed behavior of the network with the one defined in the policy. If degradation is verified, The QMA session bean notifies the administrator by alerting messages and updates the performance database.
5. The Implementation of the Policy-based DiffServ QoS Management System

The policy-based DiffServ QoS management system is implemented in the Windows 2000 Server platform with Java. The policy-based DiffServ QoS management system based on the multi-tier MVC architecture is highly manageable and scalable, and provides the overall strategy for the clear distribution of objects involved in the QoS management system application. The presentation-tier runs a Web server to handle administrator requests, and invokes Servlets and JSPs. We use Apache Tomcat 4.0.1 [12] for a Servlet and JSP container in the presentation-tier. Tomcat 4.0.1 is a free, open-source implementation developed under the Jakarta project at the Apache Software Foundation. And it implements the Servlet 2.3 and JSP 1.2 specifications from Java Software, and includes many additional features that make it a useful platform for developing and deploying web applications and web services. An EJB-based QoS manager within the business-tier runs an EJB server to handle low-level QoS policy beans. We use JBoss 3.0.3 [13] for an EJB server and use EJB 1.1 to implement EJB beans. JBoss is an open-source, standards-compliant, application server implemented in Java and distributed for free. AdventNet SNMP v2c APIs [14] are used for handling EJB-based QoS manager’s SNMP operations. AdventNet provides SNMP v2c API written in Java. It allows the developers of network management applications to develop management applications by simplifying SNMP interfaces. The Oracle 8i Enterprise Edition 8.1.6 for Windows NT [15] is used for storing performance and topology information derived from MIB tables.

6. Conclusion and Future Work

In this paper, we have introduced Component-based Operations Support System (COSS) Platform with QoS policy support. The COSS conforms to Model-View-Controller (MVC) model, and is based on IETF Policy-Based Management (PBM) and Enterprise JavaBeans (EJB) technologies. And we have showed the policy-based QoS management system applying the COSS platform for DiffServ networks. Because of the COSS employs QoS policies and component-based development model for building high-quality OSS, the COSS will have the potential to reduce significantly development cost and time-to-market for service management applications, and improve the maintainability, reliability and overall quality of applications.

We are currently at the stage of implementation of the business logic on the EJB-based QoS manager. We plan to experiment with and demonstrate the system on our testbeds using Linux-based DiffServ router.

[References]

Si-Ho Cha  
1995 received B.S. in Computer Science from Sunchon Univ.  
1997 received M.S. in Computer Science from Kwangwoon Univ.  
2000- under Ph.D. course in Computer Science of Kwangwoon Univ.  
1997-2000 worked as a Senior Researcher of Daewoo Telecom  
Research Area> Network Management, Service Management, Policy-based Networking

Jae-Oh Lee  
1987 received B.S. in Computer Science from Kwangwoon Univ.  
1989 received M.S. in Computer Science from Kwangwoon Univ.  
1993 received Ph.D. in Computer Science from Kwangwoon Univ.  
1995 Kolon Information Communications  
2000 Korea Telecom Research Center  
2002 WarePlus Inc.  
2002- Professor in the Dept. of Computer Engineering, Korea Univ. of Technology and Education  
Research Area> Distributed System and Network Management Platform, Software Component Technology, Object-oriented Distributed Processing Technology

Kuk-Hyun Cho  
1977 received B.S. in Electronic Eng. from Hanyang Univ.  
1981 received M.S. from Tohoku Univ., Japan  
1984 received Ph.D. from Tohoku Univ., Japan  
1984- Professor in the Dept. of Computer Science & Engineering, Kwangwoon Univ.  
1998-1999 the President of Open System Interconnection Association (OSIA)  
Research Area> Network Management, Object-oriented Networking, Distributed System