Web–based Network Management System using the CORBA JIDM Facilities integrated with Java and XML Technologies

Amir Djalalian and Moshe Zukerman

Electrical and Electronic Engineering Department
The University of Melbourne
Parkville, Vic. 3052
Australia
Phone: +61-3-9344-9209
Fax: +61-3-9344-9188
email: {a.djalalian, m.zukerman}@ee.mu.oz.au

Running Head: Web-based Network Management
Web–based Network Management System using the CORBA JIDM Facilities integrated with Java and XML Technologies

Amir Djalalian and Moshe Zukerman

Electrical and Electronic Engineering Department
The University of Melbourne
Parkville, Vic. 3052
Australia
Phone: +61-3-9344-9209
Fax: +61-3-9344-9188
email: {a.djalalian, m.zukerman}@ee.mu.oz.au

Abstract

This paper proposes and justifies an architecture that supports a common web interface to heterogeneous network management systems such as those supporting the CORBA-based, OSI-based or Internet-based management reference models. To this aim, we integrate the CORBA JIDM facilities with Java and XML technologies. We propose the combination of HTTP and TCP/IIOP protocols for issuing management operations and receiving event notifications. The proposed architecture is compatible with both the OSI and CORBA management reference models. The steps required for designing the building blocks of such architecture are identified.

Keywords:
TMN, CMIS, CMIP, CORBA, JIDM, HTTP, XML, JAVA, JSP.

1.  INTRODUCTION

Given recent Internet technology advances, we have now the opportunity to use the web to integrate heterogeneous distributed systems such as network management applications. Global availability and inexpensiveness of the Internet has encouraged many network management architects to choose the Internet as the ultimate distributed platform that provides cost effective global accessibility to the management functions. In this paper, we address the process of integrating the Telecommunications Management Network (TMN) [1] based network management systems with Internet based management applications. As a result, the user (the network operator) will be able to use the web to manage any set of telecommunications equipments such as ATM, SDH, ADSL, Cable Modem and WDM.
Furthermore, recent advances in the Common Object Request Broker Architecture (CORBA) [2] have made it possible for Internet applications to communicate efficiently, taking advantage of a well-defined object oriented model. In particular, Internet Inter-ORB Protocol (IIOP) [2], which is the TCP/IP implementation of General Inter-ORB Protocol (GIOP) [2], is becoming the standard transport used by Internet-based applications to communicate to each other using an object-oriented notion. One of the common practices among the Internet application developers is to provide a CORBA view (i.e. CORBA Wrappers, Adaptors or Gateways) for their legacy application so their clients can use their services via the Internet.

For example, using IIOP, it is possible to provide a transformation to/from a legacy application using a CORBA object. These objects must have at least two interfaces: one to the Internet world via IIOP and the other to the legacy system via proprietary protocols. They provide a consistent view of the legacy system to the outside world. Therefore IIOP can provide an efficient mechanism for intercommunication and integration of network management applications over the Internet.

Furthermore, the Object Management Group (OMG) has already defined a standard CORBA view of the network management based on both the Internet-based and the OSI-based management reference models. The CORBA view of these models is defined in the form of the CORBA facilities in the Joint Inter-Domain Management (JIDM) Interaction Translation definition [3]. Also, the JIDM defines an interface for a gateway that allows for integration into legacy and proprietary systems. However, a mechanism for the presentation of the management information to the user over the web has not been specified. The majority of web-based applications communicate with browsers using Hypertext Transfer Protocol (HTTP) in the form of Hypertext Markup Language (HTML) documents. Very few applications use applets to present management information. Assuming that the presentation of the management information to the user is best achieved via an HTTP and Java enabled browser, rather than a custom built application, an end-to-end solution for a web-based distributed network management system can be achieved using IIOP between the application servers, and also using HTTP or TCP sockets between the application servers and the browsers.
A traditional mechanism for a browser to establish communication to a web-based application is the Common Gateway Interface (CGI). However, recent advances in the Java server side technology have lead to a more efficient mechanism, using the object oriented programming model. In this paper, we aim to propose a new architecture which will take advantage of these recent advances, in particular Java Servlets [4], Java Server Pages (JSP) [5] and Java Beans [6], to achieve global accessibility to network management functions and domains. We will address and resolve issues such as security, access control, event notification reception and generation of XML DTD, Servlet, and JSP from the management information.

In short, we propose the use of Java technologies to encapsulate the IIOP interaction with the JIDM facilities into Java beans. The results of the management operations performed on managed domains will readily be available to those JSPs that incorporate such beans. JSPs will use this result to generate HTML or eXtensible Markup Language (XML) contents dynamically. Such contents are presented to the user over an HTTP connection. We also address the problem of using HTTP to forward event notifications to the user; thus, proposing the use of applets in this particular aspect of management information exchange.

The remainder of this paper is organized as follows. In section 2 we provide background on available technologies and standards which the proposed architecture is based on, and in Section 3, we describe in detail the proposed architecture and its internal components.

2. RELATED TECHNOLOGIES AND STANDARDS

We now review the currently available technologies and standards that our proposed architecture is based on.

2.1 OSI Management Information Model and GDMO

Open Systems Interconnection (OSI) Management Information Model (MIM) [7] defines the concept of Managed Object (MO). The fundamental idea of MO is to expose the management view of a managed resource to the management applications. When the definition of how a particular resource is to be managed is expressed in terms of MOs, with the properties defined in [7], it is then guaranteed that Common
Management Information Services (CMIS) [8] and Common Management Information Protocol (CMIP) [9] can support the management communication between different management systems.

OSI management model defines the concept of *MO domain*. MO domain (or managed domain, or domain) represents a collection of MOs controlled by a single management agent or an agent. An agent is part of the management application that performs management operations on behalf of the managers. A manager is part of the management application that issues management operations and receives event notifications. In an MO domain, an MO is an instance of a particular MO class. An MO class is defined based on the following elements:

- **Properties**, or attributes visible at the MO boundary and their respective value range.
- **Management operations** on the attributes.
- **Behavior** it exhibits in response to management operation.
- **Notifications** it emits and the conditions in which it emits them.
- **Packages** it may include.
- Its position in the inheritance hierarchy of MO classes.

It is possible to collect a set of attributes, notifications, operations and behaviors into a *package*. By including a package in the MO class definition, an MO will include all the elements within that package. This simplifies the MO definition.

MIM provides the modeling concepts and terminology that are used for modeling MOs and management information. However, the model on its own is not sufficient to allow MO definitions. The Guidelines for the Definition of Managed Objects (GDMO) [10] provides link between the abstract model in the MIM and the concrete requirements for specifying particular MO classes that permit the management of particular resources in the OSI environment.

The process of defining a MO class involves the specification of the following elements:

- Attributes
• Operations on the attributes
• Operations on the MO as a whole
• Notifications
• Behavior definition
• Attribute groups
• Packages
• Naming

GDMO uses the template notion for defining the MO classes and its elements. A template identifies what items shall be specified in order to produce a valid definition for a particular element. GDMO has defined the following nine templates:

• MO class template
• Package template
• Parameter template
• Attribute template
• Attribute group template
• Behavior template
• Action template
• Notification template
• Name binding template

For clarity, we outline the makeup of the “MO Class” template as an example:

Example 1.
<class-label> MANAGED OBJECT CLASS
DERIVED FROM <super-classes>
CHARACTERIZED
Where the “DERIVED FROM” construct specifies the super class in which the MO class is derived from. The “CHARACTERIZED BY” construct lists any mandatory packages which are to be included in all instances of the class. “CONDITIONAL PACKAGES” construct lists any packages whose inclusion in an instance of the class is dependent upon a condition that will be evaluated at instantiation time. The “REGISTERED AS” construct provides a mechanism that allows the identification of this MO class globally.

Understanding the structure of these templates is a necessary factor in constructing Document Type Definition (DTD) that are used by XML documents that convey information about MOs.

2.2 JIDM Interaction Translation

JIDM Interaction Translation is the definition of a set of CORBA facilities that enable interworking between management systems who are based on different management models.

In order to be able to interwork between a particular pair of management reference models, JIDM defines:

- A translation scheme between the different object models of both management reference models, referred to as Specification Translation.
- A dynamic conversion mechanism between the protocols and behaviors used in both domains, referred to as Interaction Translation.

JIDM presents a set of CORBA-based facilities to provide interoperability between CORBA and alternative telecommunication management models, specifically OSI management and Internet management. The JIDM module comprises a collection of IDL interfaces that together define a basic set of services for
developing systems management applications based on CORBA. Following the JIDM reference model, these interfaces may be used between manager applications and JIDM Frameworks, or between JIDM frameworks and agent applications.

There are three levels of interfaces being defined:

- **Generic interfaces, management model independent** – a framework to access a managed domain, independently of the management reference model being used. These are called *JIDM facilities*.

- **Generic interfaces, management model specific** - two management reference models are considered, OSI management, i.e. CMIP-based, and Internet management, i.e. Simple Network Management Protocol (SNMP) -based. Here, we only concentrate on the OSI management facilities that provide a CORBA view of the OSI management reference model. This set of facilities extends the generic JIDM facilities to support all CMIS interactions in CORBA, and to support OSI specific concepts such as scoping, filtering and multiple replies both in pure CORBA environments and in interworking environments (gateways).

- **Specific interfaces, information model and management model dependent** - not covered in this paper.

JIDM defines facilities that allow CORBA manager objects to connect to MO domains given their titles and allows CORBA manager objects to:

- Create a new member of the managed domain (a new MO) and assign a name to it.
- Obtain a reference to a member of the managed domain (an existing MO) given its name. This allows the manager to perform management operation directly on the member.
- Operate on collections of those members of the managed domain, which meet some criteria. (i.e. scoped/filter operation)
- Receive event notifications from remote MO domains.

And most importantly JIDM gateway facilities allow:
• A CORBA management application to internetwork with a non-CORBA, (e.g. SNMP or CMIP), MO domain, this is called the manager side gateway.

• A CORBA agent application to offer a management interface based on some management-specific protocol such as CMIP or SNMP but not CORBA thus making the managed domain visible to a non-CORBA manager application. This is called the gent side gateway.

Note the significance of the manager side gateway in our architecture. We need to have a single entry point into the MO domains that may support either CORBA based or OSI based reference models. This entry point, being JIDM facilities, allows any manager application to exchange management information with either a CORBA or a non-CORBA based MO/domain via IIOP.

Note that each MO domain is associated with a title. This title uniquely and globally identifies that domain in a given management environment. For example, in the OSI management environment AE-title is being used whereas in the Internet management environment IP address or the hostname is being used.

The same concept applies to the managers. Managers must, uniquely, be identified for the purpose of reception of event notifications.

In order for a manager to perform management operation on MOs, it must first establish communication with a given MO domain. The manager must first call access_domain(in Key key, in Criteria criteria) method defined in the JIDM::ProxyAgentFinder interface. Where “key” specifies the system management environment in which the managed domain resides on, (i.e. OSI, Internet or CORBA), and “criteria” specifies, beside other things, the title associated to the MO domain. Thus, the combination of the key and criteria is sufficient to locate a domain. The returned value of access_domain(…) is an object of the type JIDM::ProxyAgent (or OSIMgmt::ProxyAgent if the MO domain exists within an OSI-based management environment). A proxy agent object presents a session established with an MO domain. Using this object, manager is able to obtain references to:
• CosNaming::NamingContext object within that domain by invoking get_domain_naming_context() method. Further more by invoking the resolve(…) operation on the naming context object, one can obtain further references to the MOs. Note that for an OSI management environment, resolve(…) will return a reference to a proxy managed object located within the CMIP/CORBA gateway. This object, as indicated from its name, acts as a proxy and provides a CORBA view of the actual MO located within the OSI management environment.

• CosLifeCycle::FactoryFinder object by invoking get_domain_factory_finder() method. By invoking find_factories(…) method on this object, one can obtain references to the specific object factory which provides a mechanism for the creation of new MOs. Note that if the MO domain is within an OSI management environment, the returned value of find_factories(…) method is a reference to the proxy managed object factory. This object handles the communication to/from the actual factory located within the OSI management environment.

A manager who wishes to receive event notifications from one or more MOs, it must first create an instance of JIDM::EventPort by invoking the create_event_port(in Key key, in Criteria criteria) method defined in JIDM::EventPortFactory. Values of the input parameters “key” and “criteria” are described above and are meant to identify the event port globally. After creation of the event port, the manager must register with it as a consumer. MOs who wish to report events to a manager application, must first obtain a reference to the manager’s JIDM::EventPort object by invoking the find_event_port(in Key key, in Criteria criteria) defined in JIDM::EventPortFinder and then register with it as a supplier. In the case of MOs within the OSI management environment, the event port resides within the JIDM’s OSI gateway. Such event ports are capable of receiving event notifications from the OSI-based MOs via CMIP and forwarding them to the manager applications via IIOP, thus, providing a CMIP to IIOP translation of the event.

Above introduction provides a sufficient background sought for our architecture.

2.3 Servlets, JSP and Java Beans
A servlet is a web component that generates dynamic content. Servlets are small, platform independent Java classes compiled to an architecture neutral bytecode that can be loaded dynamically into and run by a web server. Servlets interact with web clients via a request response paradigm implemented by the servlet container. This request-response model is based on the behavior of the HTTP. Browsers invoke servlets in the same manner they invoke CGI programs. Parameters passed via the GET/POST requests are visible within the servlet in the form of attributes of the request object. Servlets can redirect requests and controls to other servlets or JSPs.

The JavaServer Pages technology enables the authoring of Web pages that create dynamic content easily but with maximum power and flexibility. A JSP page is a text-based document that describes how to process a request to create a response. The description intermixes template data with some dynamic actions and leverages on the Java Platform allowing incorporation of Java components such as Java Beans into the documents.

In our architecture we propose the combination of servlets, JSPs and Java Beans to achieve IIOP-HTTP conversion that provides us with a suitable presentation mechanism.

Java Beans, like any other Java application, can be used for communicating with the JIDM facilities using IIOP. They also provide an efficient way of creating dynamic contents for JSPs, which can either be in HTML or XML format. We can design a set of Java beans that encapsulate the communication to the JIDM server. These Java beans communicate the management information to the JSP when they are invoked upon. JSPs/servlets can convert the results of management operations into a HTML/XML format and transmit them to the browsers using HTTP thus achieving IIOP-HTTP transformation.

2.4 XML

XML is a subset of the Standardised Generalised Markup Language (SGML). Markup languages (such as XML, or SGML) are means to add structure and convey information about documents and data that it contains. XML is a markup language used for representing structured data in textual form. A specific goal of XML was to eliminate the complexity that was present in SGML without removing its descriptive power.
XML is similar in concept to HTML. They are both text-based document that can be transferred over HTTP, however HTML only conveys information regarding the visual presentation of the document and its contents, whereas XML is used to convey information about the content itself. An XML document can optionally have a description of its grammar attached. The grammar for an XML document is described using a DTD that may or may not reside in the same document. The DTD describes the allowable elements in the XML document and describes the structure of those elements.

An XML document can be categorised as either:

- Well formed – (i.e. it is structured according to the rules defined in the XML specification).
- Valid – (i.e. it is "well formed" and conforms to the XML syntax rules and follows the guidelines of a DTD).

The power of XML, compared to that of HTML, lies in the descriptive capabilities of document contents. XML applications are able to act on the contents based on their type whereas HTML applications, e.g. web browsers, can only act on the visual presentation of the document contents.

In order, for an application, to be able to process an XML document, it must first be able to parse it. XML parsers are used for reading, creating and manipulating XML documents. Most XML parsers produce a Document Object Model (DOM) that is the tree view of the XML document in the memory. XML based applications can manipulate XML documents through the generated DOM interface.

In the proposed architecture, either XML or HTML can be used to convey management information that are exchanged between the servlets and the browsers, however, the use of XML is strongly recommended as it enables the browsers to perform further processing on the management information, thus, taking the burden of trivial and yet expensive data processing away from the application servers.

For example, when the operator first establishes communication to the web server via a browser, s/he will first perform a network discovery operation to find out about the managed domains, MOs and their attribute values, notifications they emit … etc. Obviously, the browser may retrieve massive amount of
information. Assuming that the operator wishes to view all MOs within their respective containment tree, if the browser has no mean of understanding the semantic of the data, then it cannot construct such view.

In order to increase the performance of the server side components in our architecture, it is recommended that the client be able to perform such trivial tasks, thus the use of XML is recommended.

3. PROPOSED ARCHITECTURE

In this section, we identify each component in our architecture and explain how they fit together. For each component, we will explain its purpose and operation in relation to the whole.

3.1 JSP/Servlets/Java Beans

JSPs provide an efficient mechanism for generating and presenting dynamic contents to the user. In our architecture, these dynamic contents are the management information obtained from managed domains as a result of some management operation. Dynamic contents of these JSPs are to be produced by Java beans, therefore, we need to design a set of Java beans that will use the client stubs, generated from the JIDM’s IDLs, to communicate with objects within JIDM environment. For example, beans who are responsible for obtaining the current status of an MO must call cmis_get(…) method defined in JIDM’s OSIMgmt::ProxyAgent interface. Beans that are responsible for altering the state of an MO must call cmis_create(…), cmis_create_sync(…), cmis_set(…) and cmis_action(…) methods defined in JIDM’s OSIMgmt::ProxyAgent interface. These beans are invoked upon by JSPs who report the result of such management operations to the user.

Although a user is able to open a JSP directly by pointing to its URL, such a practice is not recommended in our architecture. It is preferable that a user be authenticated and authorised before they are allowed to open a particular JSP. Therefore we recommend an entry point servlet that will perform the necessary authentication/authorisation checking before dispatching the request to a particular JSP. This servlet, based on the information obtained from the request object, must be able to determine which JSP to dispatch the request to. We will discuss security related issues in later sections.

3.2 GDMO to XML
In this section we examine the possibility of creating DTDs and XML Style Sheet (XSL) from GDMO templates. These DTDs will be used for generating valid XML contents, embodied by a JSP, that convey information about a particular MO. These documents are used for exchanging and presenting management information to the user.

Referring to the GDMO templates, we can define a set of XML DTDs that can be used for generating valid XML documents who are presenting the information in the same manner as the templates do.

For example, assuming that a DTD has been defined for the GDMO MO Class template, mentioned above, the definition of the class “1.2.3.4.5” can be represented using the following XML document.

```
<?xml version="1.0" standalone="yes"?>
<MANAGED_OBJECT_CLASS
    class-label="managedObjectClassB">
    <DERIVED_FROM>
        managedObjectClassA
    </DERIVED_FROM>
    <CHARACTERIZED_BY>
        mandatoryPackage
    </CHARACTERIZED_BY>
    <CONDITIONAL_PACKAGES>
        conditionalPackage
    </CONDITIONAL_PACKAGES>
    <REGISTERED_AS>
        “1.2.3.4.5”
    </REGISTERED_AS>
</MANAGED_OBJECT_CLASS>
```

Combining the descriptive power of XML and the dynamic capabilities of JSPs, management information processing tasks can be carried out by the client side. JIDM facilities provide single entry point to management functions of CORBA and non-CORBA based MOs. Java beans communicate the management information to and from JSPs and JIDM facilities. JSPs present the management information in
either XML or HTML formats and passes them to the browsers using HTTP. And finally, assuming that XML is being used, XML enabled browsers process the management information based on the user’s requirement, e.g. sorting, tabulating, etc.

### 3.3 Event notification

OSI management model divides the communication between the management application and the MO into two categories, *management operations* and *event notifications*. Management operations are initiated by the manager role, whereas, event notifications are initiated by MOs when a certain condition is met. In many cases it is desirable for the event notifications to reach the manager with minimum delay to ensure the integrity of network monitoring and operation. One of the drawbacks of the HTTP is that it does not provide a call back mechanism thus making it impossible for a MO to send its event notifications to browser on a real-time basis.

As we discussed above, programming model of the JIDM facilities specifies that the JIDM::EventPort to be used as the media for exchange of event notifications between MOs and manager objects. The MO must register as event supplier and the manager object must register as event consumer of this event channel.

Knowing that the Java applets can establish TCP connection to any application server residing on the same host, it is possible to implement a mechanism where event notifications are sent to the browser as soon as they are received by a JIDM::EventPort object. The advantages of applets are that they can be launched and run from within a web page, which allows the coexistence of both HTTP and TCP. Although an event reporting mechanism may be implemented using Java applets that can act as a consumer of the JIDM::EventPort objects, such approach would not be practical since untrusted applets can only open socket connections to the host where they are loaded from, i.e. only event ports residing on the same host are reachable by the applet, whereas JIDM facilities are distributed and may reside on any host in the Internet. Therefore, an intermediate component is needed to store and forward the events from JIDM::EventPort to the applets. These intermediators, which we have identified as *Event Forwarding Agent* (EFA) in Figure 1,
must reside on the same host as the applet’s source. EFA is the consumer of JIDM::EventPort and the supplier to the applets.

Due to the restrictions imposed on applets, they cannot act as servers, i.e. they cannot listen for network connections. This prevents them to act as a conventional CosEventComm::PushConsumer object which requires the applet to act as a server and the CosEventComm::PushSupplier as a client of an event channel. However, if we follow the pull model specified in the OMG CORBA Event services [11], where CosEventComm::PullConsumer object act as the client and CosEventComm::PullSuppliers object acts as the server to an event channel, we can successfully implement the proposed event notification reporting mechanism using applets. Most vendors implement the pull consumer as the server who accepts connection and the pull consumer the client who request a connection to an event channel. Therefore, in our architecture, we propose that the applet be implemented as a pull consumer who establishes network connection with the event channel within EFA. This means that the supplier object within EFA should register as a pull supplier to the EFA’s event channel. It is needless to say that the client stubs for the pull consumer interface must be included, together with the applet, in the jar file being downloaded by the browser.

There is no restriction on supplier component of the EFA when it comes to interaction with the JIDM::EventPort. It can register as a pull or push consumer with JIDM::EventPort. It is very important that the EFA’s event channel be capable of maintaining any event notification until all pull consumers (i.e. applets) have received the event. Usually, in a network management system there would be a large amount of events being generated, which most likely will overwhelm the internal cache on the EFA’s event channel. Therefore a local database is needed for maintaining all received events. This is well suited with EFA’s store and forward behaviour and is particularly useful for maintaining a history of all events.

Although the push or pull model of the CORBA event services would be sufficient to forward all events to the client’s browser, there are no built in mechanism for the CORBA event service to filter incoming events before forwarding them to its consumers. Ability to filter event prior to delivery, based on a given criteria, is an important requirement for the network management systems. Operators cannot afford to
view hundreds of irrelevant alarms being generated every minute. Therefore it is strongly desirable for them to be able to choose the type of events they wish to receive. For example, a user might only be interested in event notifications generated by a particular MO.

CORBA Notification Service [12] solves the problem. The notification service could be looked upon as a much more powerful version of the event service. Although it supports decoupled communication, *push/pull* and typed/untyped models defined in CORBA event service, it provides many additional features, which enable you to implement sophisticated, intelligent event-based communication. As well as typed and untyped data events, there is also the new native "structured" event. This allows the transmission of a well-defined data structure in addition to the untyped Any. Due to the well-defined message structure, extra information can be associated with the event such as filtering and quality of service (QoS) details. The notification service supports content based filtering on event data. If the EFA is implemented using a CORBA notification service, then consumers can indicate what types of events they want to receive by providing a filter constraint. This will reduce the network traffic between the client and the EFA and would lead to a more manageable view of the network.

### 3.4 Security and Access Control

Although security is one of the most important requirements of network management systems, in this paper we only focus on some of the relevant issues related to TMN and Internet security. This topic is vast and in many cases independent of the application. Internet security can be applied to any application being deployed on the Internet regardless of their domain (e.g., finance, network management, stock market).

The first and most important aspect of the architecture, which needs to be secure, is the communication between the browser and the web server. It is crucial to provide protection against eavesdropping between these two points.

Secure Socket Layer (SSL) is a network layer encryption scheme [13] that allows the client to establish a secure communication to a secure server using an encrypted port. The port is managed by a software called the SSL Record Layer, implemented on top of TCP. Higher-level software, i.e. the SSL
Handshake Protocol, uses the SSL Record Layer and its port to contact the client. The SSL-Handshake Protocol on the server arranges authentication and encryption details with the client using public-key encryption. After this negotiation, all messages are exchanged in an encrypted form and, therefore, are protected against unauthorised entities that may have the ability of capturing such messages online. What is described above is the minimum requirement for a secure communication and it certainly does not control the user access to a specific managed entity. There should be other measures to control the monitoring and operation capabilities of a user. Simply allowing any authenticated user to perform management operation on any domain that s/he wishes, would jeopardise the integrity of the network being managed. Each user, besides being authenticated, must be authorised prior to performing a specific management operation. For example, it may be desirable to limit ones activity to a specific managed domain, or to a limited number of MOs within that domain. The aim is to achieve multi-level authorisation. Not all authenticated users have the same level of authorization.

One way of achieving strong authentication and authorization on the Internet is server side authentication/authorisation mechanism. Such mechanism requires a digital certificate issued by the Certificate Authority (CA) to be installed on the browser. This may not be an X.509 certificate. In most cases these certificates are issued by the same organization that provides the services and may have a particular format. They contain encrypted information regarding the registered user. They also require a special plug-in to be installed on the web server. The plug-in, besides authenticating the user, allows the server to extract the user specific information from the certificate and pass them to the servlet as part of the request object.

Advantages of such mechanism are

- The servlet can use this information to authenticate the browser. (i.e. is s/he who s/he claims to be?). This may be redundant, since the web server has done so before invoking the servlet.

- The servlet can verify the authority of the user against an internal database. (i.e. is he allowed to perform the requested operation?). Imagine the access control granularity that can be achieved. This is vital in performing CMIS management operations.
• Cookie-less session/state management. Since every incoming request has unique information pertaining the user, there is no need for cookie-based session/state management mechanisms. Servlets can create and maintain session objects using the information extracted from the certificate as session id.

We shall now discuss access control to both OSI and CORBA based management reference models. Referring to the JIDM specification, we can see that CMIS operations defined in OSIMgmt::ProxyAgent and OSIMgmt::ManagedObject take an argument of the type X711CMI::AccessControlTypeOpt as their input parameter. This parameter is to be used as input to the access control functions in each management environment; however, since JIDM specification does not define any interface for the CORBA based MOs and their access control, it is inconclusive to rely on the JIDM facilities for this purpose. We propose the use of policy repository. This is a database where user profiles are kept. It can be configured to include information such as the

• MO domains accessible by the user.
• Operation that s/he is allowed to perform on manage objects, such as create, delete, get set …etc.

The policy repository is maintained and accessed via authorisation server. See Figure 1. Since all the management operations are performed via Java beans, it is the responsibility of the beans to verify authorisation of each user for a given request against the policy repository. It is needless to say that the user information that is obtained from the digital certificate must be passed from servlet to JSP and then to the beans. This information is vital for obtaining ones profile from the policy repository.
Figure 1: The web-based network management systems architecture
4. CONCLUSIONS AND FUTURE WORK

We have evaluated recent advances in Internet technology and have demonstrated their suitability for our proposed web-based network management architecture. Given that the interaction between various components, such as browser to servlet, servlet to JSP, JSP to Java beans, Java beans to a dummy implementation of subset of JIDM interfaces has been verified via a proof of concept applications in various industrial projects, the communication aspects of the proposed architecture in achievable. However, the reader is reminded that prior to full implementation of the proposed architecture, a full implementation of JIDM facilities is required.

Our future research shall concentrate on the auto-generation of various components of the proposed architecture from a given Management Information Base (MIB). We aim at auto-generation of Java beans, servlets and JSPs using agent technology.

Acknowledgements

The authors wish to acknowledge the support by Australian Research Council and Telstra Research Laboratories (TRL).
REFERENCES


6. Sun Microsystems, Graham Hamilton (Editor), *JavaBeans™*, v1.01, July 1997


Biographies

Amir Djalalian received a BSc (double major in Physics and Computing) from the University of Western Sydney, Australia and a BEng (Honours) from The University of Melbourne, Australia. He is currently working towards his Ph.D degree at The University of Melbourne. He has worked as an architect and designer of various network management systems and has extensive experience with CORBA and Internet based applications. His research interests include network management systems for the next generation networks, application of mobile/intelligent agents in network management systems and CORBA based TMN.

Moshe Zukerman received his B.Sc. in Industrial Engineering and Management and his M.Sc. in Operation Research from Technion - Israel Institute of Technology and a Ph.D degree in Electrical Engineering from The University of California at Los Angeles in 1985. Dr. Zukerman was an independent consultant with IRI Corporation and a post-doctoral fellow at UCLA during 1985-1986. During 1986-97 he served in Telstra Research Laboratories (TRL), first as a research engineer and between 1988-97 as a project leader. He is the recipient of the Telstra Research Laboratories Outstanding Achievement Award in 1990. In 1997 he joined Melbourne University where he is now an Associate Professor responsible for promoting and expanding telecommunications research and teaching in the Electrical and Electronic Engineering Department. He is an IEEE Senior Member and has served as a session chair and member of technical and organizing committees of numerous national and international conferences. He served on the editorial board of the Australian Telecommunications Research Journal during 1991-1996, as a Guest Editor of IEEE JSAC for an issue on "Future Voice Technologies", as the Lead Editor for a JSAC issue on "Analysis and Synthesis of MAC Protocols", and on the editorial boards of Computer Networks and the International Journal of Communication Systems. Dr. Zukerman has published over hundred papers in scientific journals and conference proceedings and has been awarded nine national and international patents.