36

Incorporating Manageability into Distributed Software

R. Chadha and S. Wuu
Bellcore
445 South Street
Morristown, NJ 07960, USA
+1(201) 829-4869 (tel), +1(201) 829-5889 (fax)
chadha@bellcore.com

Abstract

In today's world, where software applications have evolved from mainframe-based applications to client/server-based distributed systems, the need for effective, easy to develop, and open management systems is becoming painfully evident. The goal of this work is to develop cost-effective methods and tools for managing software in a manner analogous to the way in which network elements are managed in today's networks. In order to provide a flexible, interoperable, standards-compliant solution to this problem, we have developed an infrastructure that conforms to the Telecommunications Management Network (TMN) suite of standards. The TMN is based on the OSI/CMISE systems management standards. Although telecommunications providers, both in the domestic and international markets, have been pressing equipment and operations systems suppliers to start developing TMN-compliant products, the supplier community has been reluctant to move towards OSI/CMISE management systems, largely due to the difficulty and expense of implementing OSI/CMISE interfaces to management applications.

This document gives a description of the components of an infrastructure that provides a middleware layer which shields software developers from much of the complexity of OSI/CMISE management implementation. It presents techniques that can be used to simplify the process of implementing TMN-compliant interfaces. This layer will considerably ease the process of incorporating standards-based management interfaces into software components for the purpose of management.

Keywords

Managing distributed software, CMISE, CORBA

1 INTRODUCTION

In today's world, where software applications have evolved from mainframe-based applications to client/server-based distributed systems, the need for effective, easy to develop, and open management systems is becoming painfully evident. The quality of large, distributed software applications will ultimately depend upon the effectiveness of their management systems. Management systems are required to perform fault, configuration, accounting, performance, and security management. As stated in the abstract, our work focuses on the problem of managing distributed software applications. This entails a study of the whole lifecycle of an end-to-end management application, starting with the definition of an information model, or management information base (MIB); designing the interface between the management application and the managed resources (which in this case are distributed software applications); defining the functionality of the management application; and so on.

This paper describes an architecture for distributed systems management based on OSI systems management. The use of OSI management provides the inherent advantage of making available standard, open interfaces for management applications. OSI management systems have not been widely implemented and deployed, in spite of support from international standardization bodies and government organizations. Even though OSI management offers a powerful, object-oriented management model, proprietary protocols and SNMP (Case et al., 1990) are still the preferred solutions for a large section of the market. This has been attributed mainly to the difficulty of OSI management implementation. In this paper, we show how we have hidden much of the complexity of OSI implementation from developers who need to make their applications manageable. We have analyzed, from a developer's viewpoint, the steps required to make an application manageable. In order to make a software application manageable, it will be necessary for developers to instrument their applications so that an agent will be able to retrieve management information from these applications; also, these software applications must be designed to emit the appropriate event reports, as defined by the information model. As mentioned earlier, it is naturally desirable that this be accomplished with a minimum of effort, so as not to add a disproportionate burden to the developer's task. With this in mind, we examine the steps required to make an application manageable, and suggest ways to partially automate this task.

A preliminary description of this work was given in (Chadha and Wuu, 1996a), where we described an end-to-end management system that managed a set of simple CORBA software components. In this paper, we describe in greater detail the steps required to build an agent for the managed resources, and to make applications manageable. This paper is organized as follows. Section 2 outlines the contributions of this work. Section 3 describes the system architecture and information model used for our prototype development, and the various components that make up the system. In Section 4, we describe the steps required to build an agent for our system. The steps for incorporating manageability into software components appear in Section 5. Some related work is discussed in Section 6, with conclusions in Section 7.

2 CONTRIBUTIONS OF THIS WORK

The main contribution of this work is that it demonstrates the feasibility of applying OSI management to distributed systems management. In this paper, we show how OSI management and the OMG CORBA (OMG, 1994) framework can be easily integrated, and why this approach is a good one. We demonstrate that the task of making a distributed CORBA application manageable can be reduced to the task of implementing some automatically generated CORBA interfaces.

The most important feature of our software management infrastructure is that the developers of the distributed software application being managed do not have to have any expertise in OSI systems management. Their task is to write server code to implement management operations (such as a function to "get" an attribute value), and to make calls to remote objects in order to send them information about events that occur in the application that the management application needs to know about. Thus they only deal with CORBA interfaces and are completely shielded from details about the CMIP protocol (CCITT, 1991) and about populating ASN.1 data structures (CCITT, 1988), which can be a formidable task, and which is an integral part of OSI management.

The second advantage of our approach is that it provides a precisely defined method for going from a GDMO (CCITT, 1992b) information model (which is the starting point of any OSI management application) to an implementation. The use of CORBA interfaces which are automatically generated from a GDMO information model provides a uniform starting point for all developers who need to make their applications manageable.

Finally, due to its implementation of OSI management standards, the management infrastructure described in this paper provides an open interface for management applications. Implementing such an open interface will make the distributed software application attractive to customers who wish to implement their own management applications in order to customize the latter to fit their needs.

3 AN END-TO-END MANAGEMENT SYSTEM

In this section, we describe a prototype implementation of a management infrastructure for managing distributed software applications. The following sections describe the architecture of this system, the implementation platforms and the information model used.

3.1 The information model

We have developed an information model (using GDMO) to represent managed software applications. This model defines a new managed object class called `softwareProcess`. This managed object class inherits all the properties of the managed object class `software`, which is defined in (CCITT, 1992d). The purpose of this managed object class is to encapsulate information about network and software configuration (including distributed software dependencies), performance (including process health and status), and faults. In addition to the attributes inherited from the `software` managed object class, the `softwareProcess` managed object class contains the following attributes: `serverList, peerList, clientList, processId, processingEquipmentName, processName, processCreationTime, processUpTime, processUserId, processGroupId, processEfUserId, processEfGroupId, processArguments, threadsUsed, cpuTime, filesOpened`. The first three attributes listed here are pertinent for monitoring distributed processes, while the rest provide information about the process and its health. These attributes are more fully described in (Chadha and Wuu, 1996a). The `softwareProcess` managed object class can also emit a number of notifications inherited from the `software` managed object class. The complete GDMO description of this information model can be found in (Chadha and Wuu, 1996b).

3.2 System Architecture

Figure 1 shows the architecture of the system and the platforms chosen to implement this prototype. All applications and platforms in this prototype run on Sun SPARCstations™ running Solaris™ 2.3. There are essentially three distinct components in this architecture: the manager, the agent, and the managed application (or managed resources). The interface between the manager and the agent is a standard CMIP interface. The manager and agent communicate using SunLink™[1] OSI 8.0, a full 7-layer OSI stack. The managed software application is implemented using the Common Object Request Broker Architecture (CORBA™[2]) (OMG, 1994). The interface between the agent and the managed application is CORBA. CORBA was developed by the Object Management Group (OMG), and defines mechanisms by which distributed objects transparently make requests and receive responses. It provides interoperability between applications on different machines in heterogeneous distributed environments. CORBA provides an Interface Definition Language (IDL) for defining the interface to an object. Section 3.3 discusses the translation of a GDMO description of an object into a CORBA IDL description. Our choice of CORBA as the distributed application platform was motivated by the emergence of CORBA as one of the prime candidates for the next generation of distributed computing platforms. The following subsections describe the manager, agent, and managed resources platforms in more detail.

1. Sun SPARCstation, Solaris, and SunLink are registered trademarks of Sun Microsystems, Inc.
2. CORBA is a registered trademark of Object Management Group, Inc.

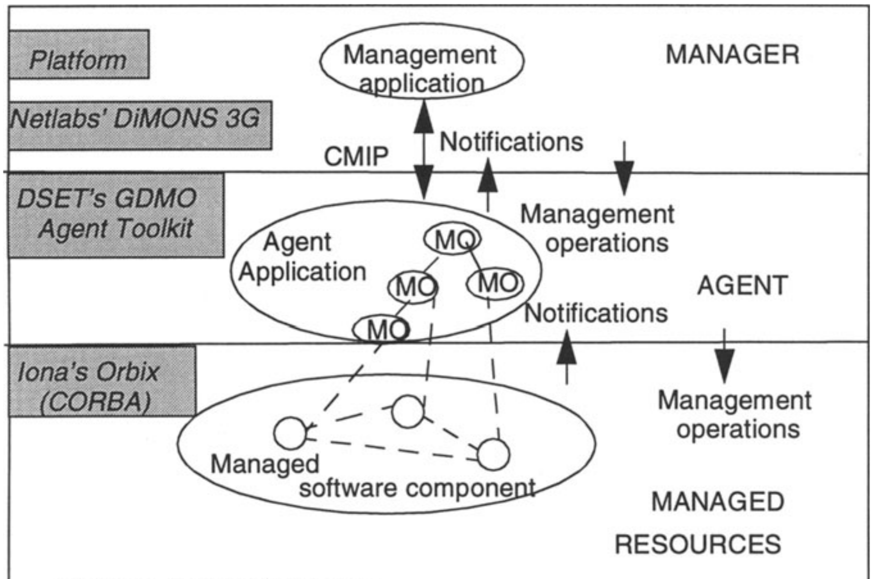

Figure 1 System Architecture.

3.2.1 The manager platform: Netlabs™[3]' DiMONS 3G

In order to reduce the complexity of building an OSI manager, we used Netlabs' DiMONS 3G management platform. The DiMONS 3G platform includes GDMO and ASN.1 compilers, which are used to incorporate new GDMO object definitions into the platform. An API called the Portable Management Interface (PMI) is provided for manipulating managed object information and building management applications. Access control lists are used to govern access to management information. It should be noted that Netlabs no longer supports the DiMONS 3G product, and much of their technology has been licensed by Sun Microsystems, Inc. and is incorporated in SunSoft's Solstice™ Enterprise Manager™[4] product. We plan to migrate to Sun's platform in the near future.

3.2.2 The agent platform: DSET™[5]'s GDMO Agent Toolkit

The agent was built using DSET's GDMO Agent Toolkit. This toolkit provides a suite of tools for building lightweight agent applications with low memory requirements. It provides GDMO and ASN.1 compilers that generate C data structures for

3. Netlabs is a registered trademark of Netlabs, Inc.

4. Solstice and Enterprise Manager are registered trademarks of Sun Microsystems, Inc.

5. DSET is a trademark of DSET Corp.

access by a program. It also provides a C/C++ API called ASN.C/ASN.C++ to enable programmers to manipulate ASN.1 data structures with ease. The platform is built on top of the Distributed Systems Generator, a DSET proprietary product.

3.2.3 The managed resources platform: IONA's Orbix™[6]

IONA's Orbix (Iona, 1996) is a commercially available implementation of CORBA. This product provides a CORBA IDL compiler, which generates a C++ class for each IDL interface. Each operation in an IDL interface is mapped into a C++ member function. An IDL interface consists of *operation* and *attribute* specifications. Each attribute is mapped into a pair of C++ member functions: one to read (get) the value, and the other to write (set) the value. IDL attributes can also be "readonly"; these attributes map to a single function, which returns the attribute value. IDL allows one interface to inherit from another, thereby creating an inheritance hierarchy. The object-oriented nature of IDL makes it relatively easy to define a mapping between the object models of CORBA IDL and GDMO; such a mapping will be discussed in the next section.

3.3 GDMO to CORBA IDL translation

The OSI Systems Management suite of standards has defined a language to be used for defining managed objects, namely GDMO. However, in order to implement OSI management, it is necessary to translate this GDMO specification into a language closer to the implementation platform. Since we have chosen the CORBA platform for implementing managed applications, we need a way to translate GDMO to CORBA IDL. Fortunately, the X/Open Joint Inter-Domain Management Taskforce (XoJIDM) has been working on this problem for the past several years, and has developed a GDMO to CORBA IDL specification translation algorithm. Commercial compilers have recently become available for performing this translation. For a complete description of this translation algorithm, see (X/Open, 1994).

Figure 2 shows a fragment of an object description in GDMO, and its translation into CORBA IDL. The GDMO fragment defines an attribute (`administrativeState`) and the management operations that can be performed on it (`GET` and `REPLACE`). The corresponding CORBA IDL contains two methods, one to perform the management operation `GET` (`administrativeStateGet()`) and one to perform the management operation `REPLACE` (`administrativeStateSet()`).

The XoJIDM taskforce has also been working on a Dynamic Interaction Translation document, which provides a mapping between CMISE services and OMG services. The GDMO to CORBA IDL specification translation algorithm mentioned

6. Orbix is a registered trademark of IONA Technologies Ltd.

```
ATTRIBUTES administrativeState
GET-REPLACE;

administrativeState ATTRIBUTE WITH ATTRIBUTE SYNTAX
 Attribute-ASN1Module.AdministrativeState;
MATCHES FOR EQUALITY;
```

Figure 2 (a) Fragment of GDMO definition of an attribute.

```
Attribute-ASN1Module::AdministrativeStateType
administrativeStateGet() raises (CMIP_ATTRIBUTE_ERRORS);

void administrativeStateSet(in
Attribute_ASN1Module::AdministrativeStateType Value)
raises (CMIP_ATTRIBUTE_ERRORS);
```

Figure 2 (b) Translation of GDMO into CORBA IDL.

above only addresses the syntactic translation of GDMO to CORBA IDL, and ignores the dynamic policies of object behavior defined by other GDMO templates. For example, NameBinding templates define the policies governing the lifecycle of a managed object (i.e. rules governing creation, deletion, copying, and naming of managed objects). The specification translation also does not make use of any of the OMG Common Object Services (OMG, 1995). The Dynamic Interaction Translation document is intended to deal with issues such as support for conditional packages, distribution of event reports using the OMG Event Service, and support for a Management Information Repository.

The XoJIDM Dynamic Interaction Translation provides several scenarios for CMIP-CORBA interworking. In one scenario, the managing system is CORBA-based, and the agent is in an OSI managed system, which communicates with the manager using CMIP. In such a case, a gateway function must be part of the managing system in order to translate the manager's CORBA requests into CMIP requests, to translate the agent's responses from CMIP to CORBA, and to translate CMIP event reports received from the agent into CORBA (see Figure 3). In another scenario, the situation is reversed: the managed system is CORBA-based, whereas the managing system is OSI-based. A gateway function is now required in the managed system, to translate between CORBA and CMIP (see Figure 4).

Finally, it is possible to have a scenario where both the managing and managed systems are CORBA-based, and they communicate using either CORBA, or two CMIP-CORBA gateways (one in the agent and one in the manager).

4 BUILDING THE AGENT: CMIP-CORBA INTERWORKING

In our architecture, the managing system is OSI-based, and the managed system belongs to the CORBA world, a situation similar to that depicted in Figure 4. This

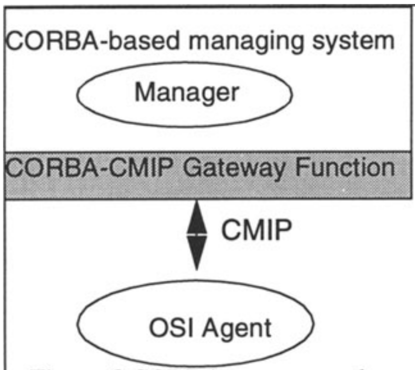

Figure 3 CORBA manager and OSI agent.

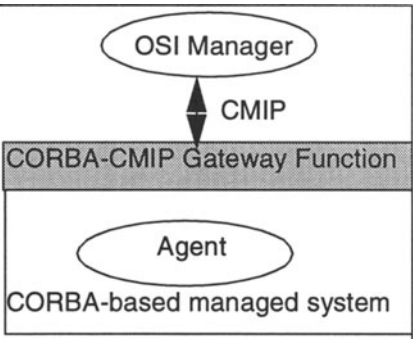

Figure 4 OSI manager and CORBA agent.

situation requires us to build a gateway function in the managed system. Rather than build such a system from scratch, we chose to use the infrastructure provided by the DSET GDMO Agent Toolkit, which handles much of the CMIP communications tasks, and leaves only a small portion of the gateway to be implemented by us. Thus in our system, the gateway function will be incorporated inside the agent application in the managed system.

Our agent application has to perform a number of tasks. It has to maintain a Management Information Tree (MIT) for the managed object instances; it has to translate management operations into CORBA operations to be performed on the managed software components; and it has to set up a CORBA server to receive notifications from the managed software components, which it must then translate into CMISE event reports and send to the manager. In order to be able to receive notifications sent from CORBA applications (which are the managed resources), the agent starts up a thread that implements a CORBA server. This thread waits for notifications to arrive from the managed software components, and translates them into notifications that are sent to the manager. The tasks performed by the agent are described in detail in the following subsections.

4.1 The agent's Management Information Tree

When an agent starts up, it can create instances of managed objects in its MIT. For our purposes, let us assume that it creates two managed object instances. The first instance created is an instance of the `system` managed object class. This instance is used as the root of the MIT. The second instance created is an instance of the `eventForwardingDiscriminator` managed object class, defined in (CCITT, 1992a). This is a special-purpose managed object class, whose function is defined in (CCITT, 1992c). This managed object class contains attributes which determine how incoming event reports from managed resources will be disposed of.

Two of the more important attributes of this managed object class are the destination attribute, which specifies a list of managers to whom event reports should be forwarded, and the discriminatorConstruct attribute, which allows the specification of a filter that ensures that only event reports satisfying this filter are forwarded to managers whose address is specified by the destination attribute.

Whenever a managed software component comes up, it makes a call to the agent CORBA server (implemented as a thread within the agent application) and sends it an objectCreation (CCITT, 1992a) notification. This notification contains all the information needed by the agent to create an instance of the appropriate managed object class. This managed object class will typically be softwareProcess, or a more application-specific managed object class that inherits all properties from the softwareProcess managed object class. This instance is created by the agent, and an objectCreation notification is sent to the manager. If this software component goes down or terminates, an objectDeletion notification (CCITT, 1992a) is emitted.

4.2 Mapping CMIP requests to function calls in the agent

One of the tasks of the gateway function in Figure 4 is to take a CMIP request (received from the manager), translate it into an appropriate method call on a CORBA interface, and invoke that method. Using the DSET Toolkit, CMIP requests received from the manager are automatically mapped to function calls; however, it is the user's responsibility to specify *what* functions will be called. As an example, suppose that the manager issues an M-GET operation on a managed object with distinguished name X. This M-GET request is received by the DSET agent, and transmitted to the managed object with distinguished name X. Now, for every attribute of this managed object, the user is expected to have specified a function to be called whenever an M-GET is issued on this attribute. Thus, for every attribute of the managed object X, the corresponding user-specified function is automatically called. Figure 6 below illustrates the operation of this strategy. Note that the gateway function is actually implemented inside the agent here.

Since the user is required to specify what function is called when a CMIP request is received, let us examine how this relates to the XoJIDM translation specification. Using the example in the previous section, suppose the manager issues an M-GET request on managed object X. Suppose X has two attributes, a1 and a2. According to the XoJIDM translation specification, M-GET requests for these two attributes map to the two methods

a1Get() and a2Get().

Therefore, we wrote a function that is called whenever a GET has to be performed on the attribute a1 (and similarly for attribute a2); this function does the following:

1. Makes a call to a method called a1Get() implemented in the CORBA object which is represented by the managed object to which a1 belongs

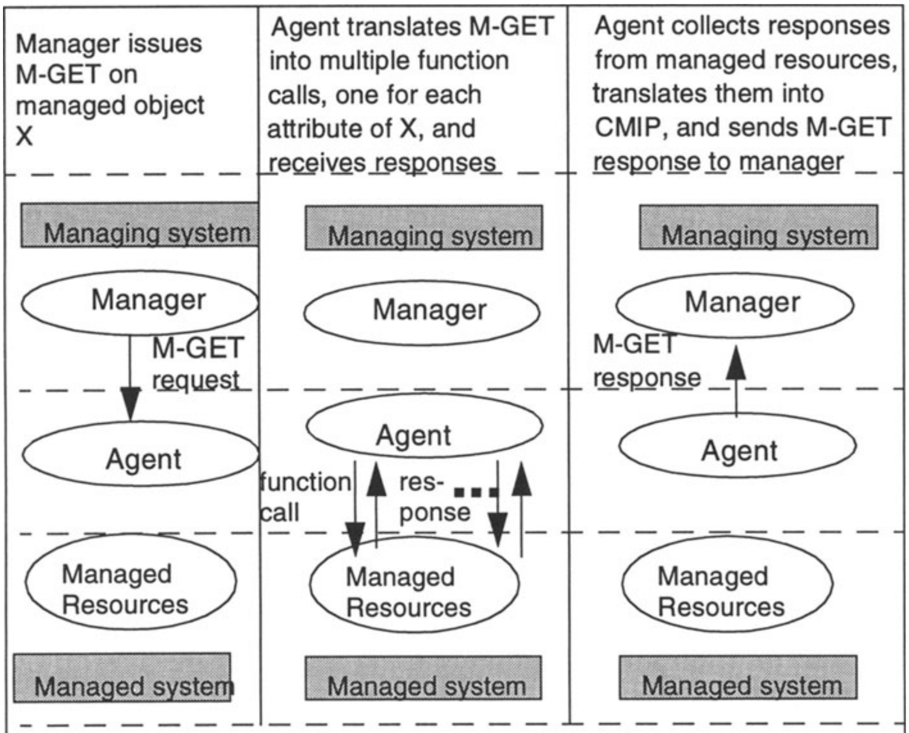
Manager issues M-GET on managed object X
Agent translates M-GET into multiple function calls, one for each attribute of X, and receives responses
Agent collects responses from managed resources, translates them into CMIP, and sends M-GET response to manager
Managing system
Managing system
Managing system
Manager
M-GET request
Agent
Managed Resources
Managed system
Manager
Agent
function call
res-ponse
Managed Resources
Managed system
Manager
M-GET response
Agent
Managed Resources
Managed system

Figure 6 Manager performs M-GET.

2. Stores the return value in the data structure allocated for that managed object in the agent.

The DSET Toolkit handles the formatting and dispatching of the M-GET response.

4.3 Mapping CORBA event reports to CMIP event reports

The XoJIDM translation specification maps event reports to methods. These methods can therefore be invoked by the CORBA managed resources whenever an event report is to be forwarded to the agent. Once such a method invocation is received by the agent, the following must be done:

1. The agent forwards this event report to all Event Forwarding Discriminator objects and Log objects in its MIT.
2. The Event Forwarding Discriminator objects check their discriminator construct and scheduling attributes to determine whether this event report needs

to be forwarded to a manager; if so, the event report is forwarded to all managers listed in the Event Forwarding Discriminator's `destination` attribute.

3. The Log objects check their discriminator construct and scheduling attributes to determine whether this event report needs to be logged; if so, appropriate log records are created in the agent's MIT.

The DSET Toolkit simplifies this entire process by providing an API for notifications. Thus, instead of performing steps 1 through 3 listed above, a call to an appropriate function in this API conveys all the notification information to the Toolkit engine, which takes care of steps 1 through 3.

We implemented this process in our agent by setting up a thread which is a CORBA server that waits for notifications from managed objects. Whenever a notification is received, it populates the necessary data structures and makes a call to the appropriate function in the DSET notification API.

4.4 How dependent is this architecture on the DSET Toolkit?

In designing this architecture, one of our primary objectives was to ensure that our design would survive platform and tool changes. The preceding sections described much of the work that we did in order to integrate the management of CORBA software components using an agent built using the DSET GDMO Agent Toolkit. However, the fact that we are using the XoJIDM specification translation makes the architecture flexible. Once implementations of gateway functions (as depicted in Figures 3 and 4) based on the XoJIDM work become available on commercial platforms, it should be a simple task to migrate our approach to any commercial platform which provides such a gateway implementation. At this stage, since no commercial gateway implementations were available, we were forced to implement our own gateway function, and were able to successfully leverage DSET's GDMO Agent Toolkit for this purpose.

5 MAKING SOFTWARE MANAGEABLE

The first step to making a CORBA application manageable is to select the managed object classes that are going to be used to represent the managed information. This can be done by developing new GDMO descriptions of managed object classes, or re-using existing ones. This GDMO document can then be translated automatically into CORBA IDL. The IDL thus generated consists of two parts: one contains interfaces for which the managed application is a client, and the other contains interfaces for which the managed application is a server. The managed application will act as a server when management operations are performed (see Section 5.1), and will act as a client when it needs to send notifications to the agent (see Section 5.2).

5.1 Instrumenting managed software components to respond to management operations

In order to implement the server part of the IDL, the developer must write code to implement the functions specified therein. An example of such a function is a function to "get" the value of an attribute. For example, using our example from Section 4.3, a function to retrieve the value of an attribute `a1` would be called `a1Get()`, and would return the current value of this attribute. Much of the task of implementing these functions can be simplified too. First, consider attributes that can be obtained from the operating system (such as, for example, the `processId`, `processUpTime`, and `cpuTime` attributes of the `softwareProcess` managed object class). Clearly, the code to obtain the values of such attributes is independent of the managed software component. Thus, these functions can be implemented once, and this code can be linked in with every managed software component. This makes the instrumentation for these attributes almost trivial. For other attributes that require explicit population by each developer (e.g. the `affectedObjectList` attribute, which specifies the object instances which can be directly affected by a change in state or deletion of a given managed object), much of the code can be written once, leaving only a few constants and parameters to be filled in by the developer. Thus the burden of instrumenting a managed software component is reduced to a minimum.

5.2 Emitting notifications from managed software components

The client side of the IDL is used for sending notifications; thus the semantics of the notifications (i.e. *when* does a notification need to be sent?) need to be analyzed and then implemented by populating the notification parameters and calling the notification functions wherever and whenever required. For example, in some cases, whenever the value of some attribute of interest changes, an `attributeValueChange` notification is sent to the agent. This notification contains information such as the name of the object sending the notification, the type of event, and the attribute that has changed along with its new value. Another example is that an `objectCreation` notification must be sent when the application starts up. The developer must incorporate client calls in order to send these notifications to the agent.

6 RELATED WORK

Although OSI management standards have been in existence for a number of years, OSI management systems have not been deployed on a large scale. This is partly due to the difficulty of implementing OSI management systems. A large number of researchers have been looking at the problem of simplifying the implementation of management systems. Some have opted for non-standards-based approaches, by defining their own tools and protocols for management. The obvious disadvantage of such approaches is the lack of standardization.

A lot of work has been done in the area of simplifying the task of implementing OSI management systems. The OSIMIS (OSI Management Information Service) platform (Pavlou et al., 1993) is an object-oriented development environment in C++ based on the OSI management model. It provides APIs for hiding the details of the underlying management services. An OSI management library is described in (Deri and Mattei, 1995). This paper also describes tools for facilitating the development of OSI management applications. Their approach for representing ASN.1 syntax using strings resembles that used by the IBM cmipWorks platform (Geiger et al., 1994). In addition to these platforms, there are a number of commercially available development tools and platforms (some of which were used in our work and mentioned in Section 3.2) that speed the process of OSI management implementation. However, none of these platforms provide any method for integrating OSI management and the OMG CORBA framework, which is one of the main features of our work.

7 CONCLUSION

In this paper, we described the internals of the service management infrastructure outlined in (Chadha and Wuu, 1996a). Special attention was paid to the details of building an agent, and methods for simplifying the process of making software components manageable. The construction of a CORBA-CMIP gateway allows transparent management of CORBA software components using OSI management, and hides the complexity of OSI management from the software developers.

8 REFERENCES

Case, G., Fedor, M., Schoffstall, M., Davin, J. (1990) "*A Simple Network Management Protocol (SNMP)*", Request for Comments 1157.

CCITT Recommendation X.208 (1988), ISO/IEC 8824, Spec. of Abstract Syntax Notation One (ASN.1).

CCITT Recommendation X.711 (1991), ISO/IEC 9596-1, Information Technology - OSI, Common Management Information Protocol (CMIP) - Part 1: Specification.

CCITT Recommendation X.721 (1992a), ISO/IEC 10165-2, Information Technology - OSI - Management Information Services - Structure of Management Information - Part 2: Definition of Management Information.

CCITT Recommendation X.722 (1992b), ISO/IEC 10165-4, Information Technology - OSI - Management Information Services - Structure of Management Information - Part 4: Guidelines for the Definition of Managed Objects.

CCITT Recommendation X.734 (1992c), ISO/IEC 10164-4, Information Technology - OSI - Systems Management: Event Report Management Function.

CCITT Recommendation M.3100 (1992d), Generic Network Information Model.

Chadha, R., Wuu, S. (1996a), "*Managing Distributed Systems using OSI Management*", Proceedings of the Second Intl. IEEE Workshop on Systems Management, Toronto Ontario, Canada, pp. 117-126.

Chadha, R., Wuu, S. (1996b), "*Service Management Infrastructure*", Bellcore TM-25364.

Deri, L., Mattei, E. (1995), "*An object-oriented approach to the implementation of OSI management*", Computer Networks and ISDN Systems 27, pp. 1367-1385.

Geiger, G., Allen, W., Majtenyi, A., Reder, P. (1994), "*IBM cmipWorks: Technical paper*", IBM.

Iona (1996), "*Welcome to Iona*", IONA Technologies Ltd., http://www.iona.ie.

OMG (1994), "*Common Object Services Specification*", Volume I, OMG Document Number 94-1-1.

OMG (1995), "*Common Object Request Broker Architecture 2.0 Specification*".

Pavlou, G. Bhatti, S. N., Knight, G. (1993), "*The OSI Management Information Service: User's Manual, Version 1.0*".

X/Open (1994), "*GDMO to OMG IDL Specification Translation Algorithm*", X/Open Company Ltd.

BIOGRAPHY

Ritu Chadha obtained her Ph. D. in Computer Science from the University of North Carolina at Chapel Hill in 1991. The subject of her dissertation was the mechanical generation of loop invariants for the purpose of program verification. She joined IBM in 1991, where she worked on software testing tools. In 1992, she joined Bellcore as a Research Scientist. She has worked on areas in software testing, modeling and simulation, distributed systems, and network and service management. She is currently working on issues related to Web site management. Dr. Chadha is also a part-time faculty member at Rutgers University.

Sze-Ying Wuu joined Bellcore in 1990 as a Research Scientist. She has worked on areas in distributed systems, distributed directories, multi-media conferencing systems, Internet and Intranet, network and service management, and web management. She was granted a patent in 1986 for her research work in distributed directory technology. She previously was the Engineering Manager at JvNCnet, an Internet Service Provider, where she was responsible for network management, operations, and engineering. She also worked with Bellcore as a consultant on telecommunication signaling and call processing systems from 1986 to 1989. Sze-Ying received an MS in Computer Science from the University of Wisconsin-Madison in 1985.