Modeling and Management of Distributed Applications and Services Using the OSI Management Framework *

James W. Hong, Michael J. Katchabaw, Michael A. Bauer and Hanan Lutfiyya
Department of Computer Science
University of Western Ontario
London, Ontario, Canada
{jwkhong,katchab,bauer,hanan@csd.uwo.ca}

Abstract

As more and more mission-critical applications and services get deployed in distributed computing environments, managing these applications and services for increased reliability and performance is becoming an increasingly important and necessary task. Management of these resources software involves monitoring, analyzing, and controlling their behavior. The OSI Management Framework has been proposed for managing networks and their devices. This paper describes our work investigating the use of the OSI Management Framework for managing distributed applications and services. We focus on abstracting distributed applications and services as managed objects using its object-oriented modeleling methodology. We then show how the captured applications and services can be managed using its manager-agent paradigm in a prototype distributed management system. Our experience thus far indicates that the OSI Management Framework provides rich and powerful mechanisms and facilities for managing distributed applications and services.

[**Keywords:** distributed applications management, object-oriented modeling, OSI management framework, management interface instrumentation, management applications]

*This research work is supported in part by the Natural Sciences and Engineering Research Council of Canada.

Appeared in the Proceedings of the International Conference on Computer Communication, Seoul, Korea, August 1995

1 Introduction

Today's computing environment consists of interconnected networks of heterogeneous computing hardware and software platforms. Users' demands for more powerful and sophisticated applications and services are also increasing. At the same time, users demand that these applications and services be more reliable and efficient. Consequently, the underlying operating systems and other subsystems supporting these applications and services are becoming larger and more complex.

Systems administrators face difficult and challenging tasks in managing such a computing environment. They must be able to plan, model, configure, deploy, monitor, control the resources as well as account for their usage. They must be concerned with the well being of not only the hardware resources, but also of the multitude of software resources running on them. Unfortunately, they are seriously lacking tools and facilities for carrying out these management tasks. Worse yet, most distributed applications and services that are being developed and deployed are not "equipped" to be managed.

Our recent work has focused on the management of distributed applications and the services required to support the management of applications. In this work, we focused on instrumenting distributed applications so that they could be managed [6]. The approach to the instrumentation entailed the development and insertion of a *management interface* into the application components (*e.g.*, processes) that were to be managed. We designed and implemented a generic management interface so that this interface could be used by most (if not all) types of applications. This work also involved the development of a management system for managing the instrumented distributed applications [5]. As part of this, we developed our own simple protocol for interactions between the manager application and managed applications. Other research addressing similar issues arising in the management of distributed applications also made use of proprietary protocols and software systems [19, 20].

Recently, we have become interested in the use of standard management frameworks for the purpose of managing distributed applications and services. This is beneficial for the following reasons. First, the managers and agents we develop using the management standards will be interoperable with other standard compliant management applications. Second, there is often existing tools which can greatly reduce the amount of work we have to do.

The OSI Management Framework [9], originally proposed as a set of management standards for network management, provides mechanisms for the modeling, monitoring, control and coordination of resources. One of the key components of the OSI management framework is their object-oriented modeling methodology which provides a powerful mechanism for modeling various resources using classes and inheritance. The framework also provides a powerful mechanism called a containment hierarchy which allows the resources to be grouped into logical groups at run time.

Using these two mechanisms of the OSI management framework, we show how the distributed applications and services can be easily represented as *managed objects* and then managed using the standard tools and mechanisms provided in the framework. Our hypothesis is that if we can model and manage distributed applications and services using the OSI management framework, it will be a giant step towards managing distributed systems (*i.e.*, managing networks, systems and applications) using a single, uniform method.

For the proof of concept, we developed distributed application management agents which contain the managed object classes and thus capable of creating them at run-time. These agents handle the managed objects on behalf of managers for the purpose of managing real applications and ser-

Appeared in the Proceedings of the International Conference on Computer Communication,
Seoul, Korea, August 1995

vices they represent. We demonstrate this by using the prototype applications management tools we developed in the OSI Management Information Service (OSIMIS) management environment [21, 22].

The rest of this paper is organized as follows. Section 2 presents an overview of the OSI Management Framework. Section 3 presents modeling of distributed applications and services using the concepts of class hierarchy and containment hierarchy provided in the OSI framework. Section 4 describes our experience with the prototype implementations using OSIMIS. We conclude the paper in Section 5 with a summary and some possible future work.

2 Overview of the OSI Management Framework

Open Systems Interconnection (OSI) management entails those activities needed to control, coordinate and monitor the resources which allow communications to take place in the OSI environment [9, 14]. The OSI Management Framework defines the services and protocols used to transfer management information between open systems. It also defines the abstract syntax and semantics of the information transferred in management protocols.

The OSI Management Framework uses an object-oriented methodology for modeling open system resources to be managed. These resources may be physical or logical and are managed through an abstraction called a Managed Object (MO). A *managed object* is a management view of a resource. That is, it logically characterizes the actual resource for the purpose of managing it. A managed object is defined in terms of attributes it possesses, operations that may be performed upon it, notifications it may issue and its relationships with other managed objects. The set of managed objects within a system, together with their attributes, constitutes that system's Management Information Base (MIB).

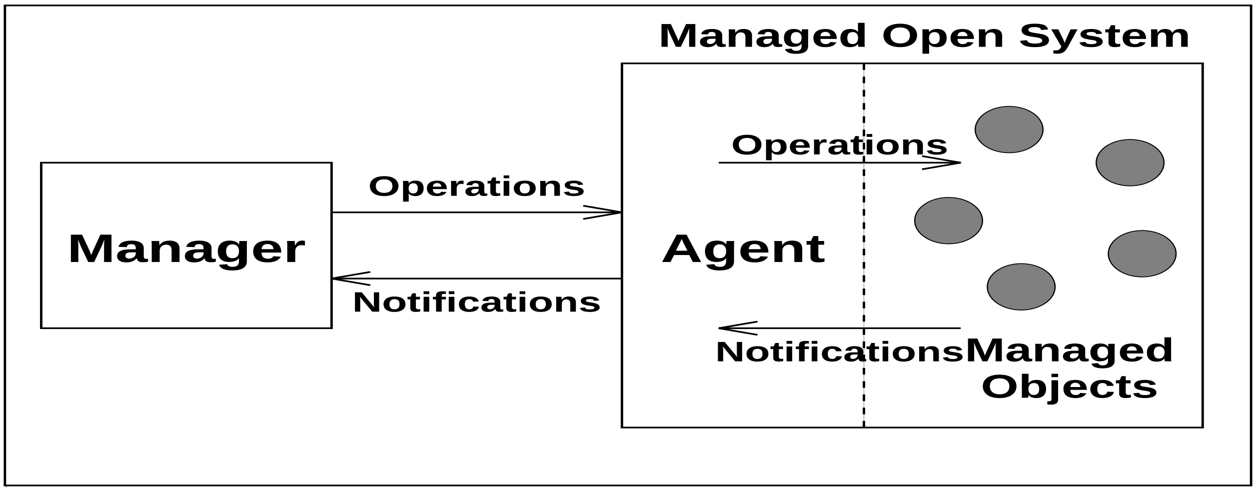

Figure 1: OSI Manager-Agent Interaction Model

MOs are handled by *agents* (management applications in agent roles) and are accessed by *managers* (management applications in manager roles) in order to implement management policies. Figure 1 illustrates the interactions between a manager and an agent in the OSI Management Framework. A manager performs management activities by issuing management operations to and receiv-

Appeared in the Proceedings of the International Conference on Computer Communication,
Seoul, Korea, August 1995

ing notifications (or event reports) from agents. An agent, upon receiving a request from a manager, carries out the operations on the managed objects. An agent may also forward notifications emitted by managed objects to a manager.

The OSI management standards that present guidelines on how to define managed object classes include the Management Information Model (MIM) [12] which defines the model for managed objects, Definition of Management Information (DMI) [13] which defines system managed objects, and templates that can be imported into a variety of managed object class definitions, and Guidelines for the Definition of Managed Objects (GDMO) [15] which provides guidance, methods and notational techniques for specifying managed object classes.

OSI management provides the Common Management Information Service (CMIS) [10] which is a set of primitives that can be used by managers and agents for exchanging management information. OSI management also provides the Common Management Information Protocol (CMIP) [11] which is used to implement CMIS. The following generic management operations are available to the users of CMIS:

Get – read management information from a managed object.
Set – write management information to a managed object.
Action – perform an action on a managed object.
Create – create a managed object.
Delete – delete a managed object.
Cancel-Get – cancel a previously-issued get request.
Event-Report – send an event report to manager.

Managers use the first six operations for performing operations on managed objects through agents and agents use the last operation (**Event-Report**) for notifying predefined events to managers.

3 Modeling of Distributed Applications and Services

We will first discuss an abstraction of distributed applications and services and then show how this abstraction can be modeled using the OSI management framework.

We define a distributed application to be an application composed of one or more components (*e.g.*, processes) running on one or more processing nodes throughout the distributed computing environment, interacting with each other to carry out the tasks of the intended application. This definition allows us to abstract most (if not all) distributed computing services and applications as distributed applications[1].

There are largely three models of applications in distributed computing systems. As shown in Figure 2, they are the 1) *centralized* model, 2) *client-server* model, and 3) *peer-to-peer* model. A centralized application typically consists of a single process and an example is a legacy system running on a mainframe. A client-server application typically consists of one or more clients interacting with one or more servers for utilizing various services offered by the servers. Examples of client-server applications are transaction services (such as banking systems), airline reservation systems and telecommunication customer service systems. A peer-to-peer application consists of multiple processes which do not have clear distinctions of their roles as client-server applications. Some

[1] Hereafter, when we refer to distributed applications we are referring to both distributed applications and services.

Appeared in the Proceedings of the International Conference on Computer Communication,
Seoul, Korea, August 1995

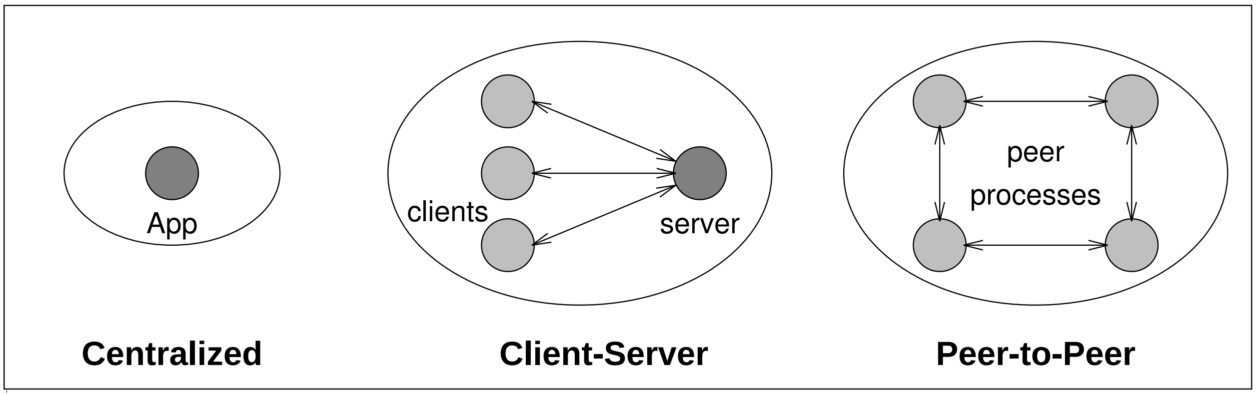

Figure 2: Application Models in Distributed Computing Environments

processes act as servers, some act as clients while others may act as both clients and servers. Examples of peer-to-peer applications are electronic message systems and distributed directories (such as Directory System Agents in the X.500 Directory Service [3]).

These abstractions of distributed applications can be modeled using the OSI management framework [9, 12]. As described in Section 2, OSI management model represents the resources to be managed as *managed objects* (or MOs). A named set of managed objects sharing the same set of attributes, actions and notifications is a *managed object class*. Attributes represent characteristics of the resources to be managed. Actions are used to perform various specific operations on the managed objects in addition to the common management operations defined as part of the standard protocol (namely the CMIP). Notifications are used to capture and relay a predefined set of events occuring in the MOs to the manager from an agent that handles the MOs.

Since the resources we are interested in managing are distributed applications, they need to be represented by managed objects as well. This is achieved by defining a managed object class or classes representing distributed applications and instantiating them as needed. We have analyzed and defined a managed object class called appObj which consists of a set of attributes, actions and notifications. The name of application, version number, number of component processes, its operating status are some examples of the attributes. We believe these are essential or common to most (if not all) distributed applications and services. A GDMO definition of the appObj managed object class is given in Appendix A. If specific application managed object classes requiring other attributes, actions and notifications are needed, they can be defined as subclasses of the appObj MO class. That is, a new application managed object class can inherit those attributes, actions and notifications of the appObj MO class and add its own characteristics. Thus, a hierarchy of application MO classes can be formed. Instances of the appObj MO class are referred to as application managed objects (or App-MOs).

We have analyzed and defined another MO class called procObj representing a process (*i.e.*, a thread of execution such as a UNIX process). We have also defined it such a way that the attributes, actions and notifications are generic to most (if not all) processes that may be running on various distributed computing environments. A process identifier, resident host name, resource usage are examples of attributes included in procObj. A GDMO definition of the appObj managed object class is given in Appendix B. Like the appObj MO class, the procObj can be specialized into a number of MO classes by inheriting the class and adding more characteristics. Thus, a hierarchy of process MO classes can be formed. Instances of the procObj MO class are referred to as process managed objects (or Proc-MOs). Full GDMO definitions for the appObj and procObj MO classes can be found in [8].

In order to monitor the performance and other application-specific aspects, MO classes representing applications should have application-specific attributes, actions and notifications defined. This can be achieved by inheriting the procObj MO class and adding the application-specific characteristics. To demonstrate this, we have defined a MO class that represents a trading service in a distributed computing environment. The trading service has been implemented as one or more co-operating servers [25]. We wanted to be able to obtain all the information and perform the actions defined in the procObj MO class and obtain the application-specific information for the purpose of monitoring and control. The new MO class, traderObj, is given in Appendix C.

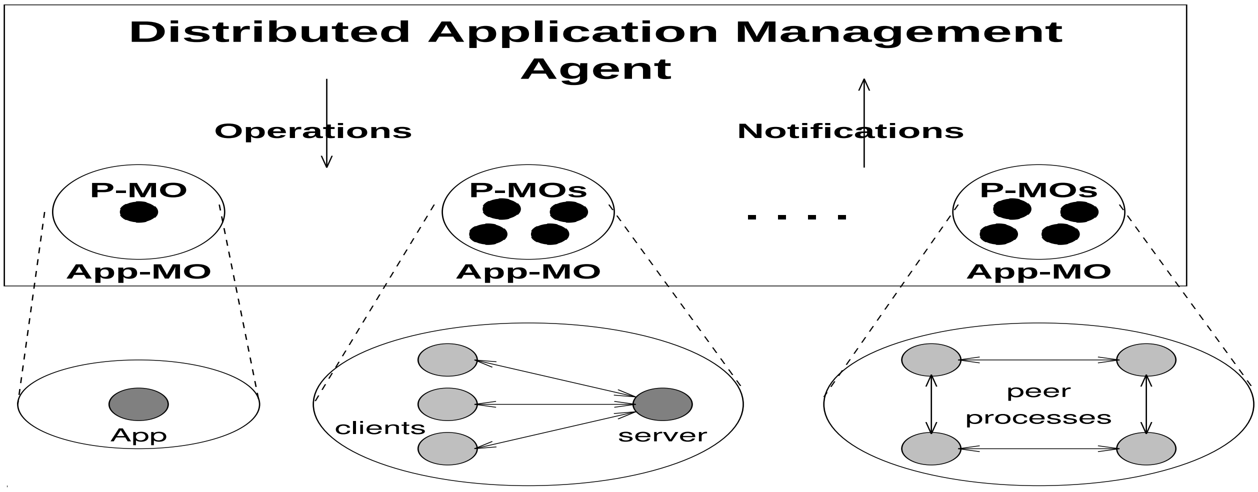

Figure 3: Modeling of Distributed Applications Using the OSI Management Model

In the operation and management environment, these MO classes are instantiated to create managed objects. In many cases, creating managed objects triggers creation of real resources. For example, instantiating the procObj MO class creates a Proc-MO which in turn creates a corresponding UNIX process. Note that the appObj MO class is truly an abstract class since it does not have any real resource associated with it when the appObj MO class is instantiated (*i.e.*, when an App-MO is created). Instead, it creates one or more component Proc-MOs since a typical application is composed of one or more processes. This can be easily represented by the MO *containment* relationship provided in the OSI management framework. Thus, a distributed application can be represented by an App-MO containing one or more process managed object Proc-MO as shown in Figure 3.

As described in Section 2, the managed objects are handled by management agents and provide access to real managed resources for monitoring and controlling their behavior. Access to real resources may take various forms depending on the location of real resources (*i.e.*, local or remote) as well as the type of communication paradigm supported by the real resources. Figure 3 illustrates a distributed application management (DAM) agent overseeing a number of App-MOs, each of which is composed of one or more Proc-MOs. Note that a DAM agent can be used to manage a single application or multiple applications. In a large distributed computing environment, being able to manage multiple applications by a single agent would be desirable.

Appeared in the Proceedings of the International Conference on Computer Communication,
Seoul, Korea, August 1995

A DAM agent is like any other OSI management agent in that it basically receives management requests from a DAM manager and performs those requests on managed objects (App-MOs and Proc-MOs in this case) on behalf of the manager. It also receives notifications from managed objects and relays them to the manager. In this way, all the powerful features provided by the OSI management operations and services (which were originally developed for managing networks and their devices) can be utilized for managing distributed applications.

4 Prototype Implementation

For the proof of concept, we developed a distributed applications management system consisting of the DAM Manager, the DAM agents and a management information repository (MIR). The DAM Manager manages distributed applications through the DAM agents by requesting appropriate management operations. The DAM agents contain the managed object classes modeled and defined in Section 3 and thus capable of instantiating them at run-time. The DAM agents interact with the real resources (*i.e.*, application component processes) for carrying out the manager's requests and relaying notifications to the manager if any. The management information repository (MIR) is used by both the managers and agents for storing various kinds of management information. Figure 4 is an architecture of our prototype implementation of a distributed applications management system.

Our prototype implementation used the OSI Management Information Service (OSIMIS) [22] as a development platform. We first give a brief overview of OSIMIS and then present the prototype implementation.

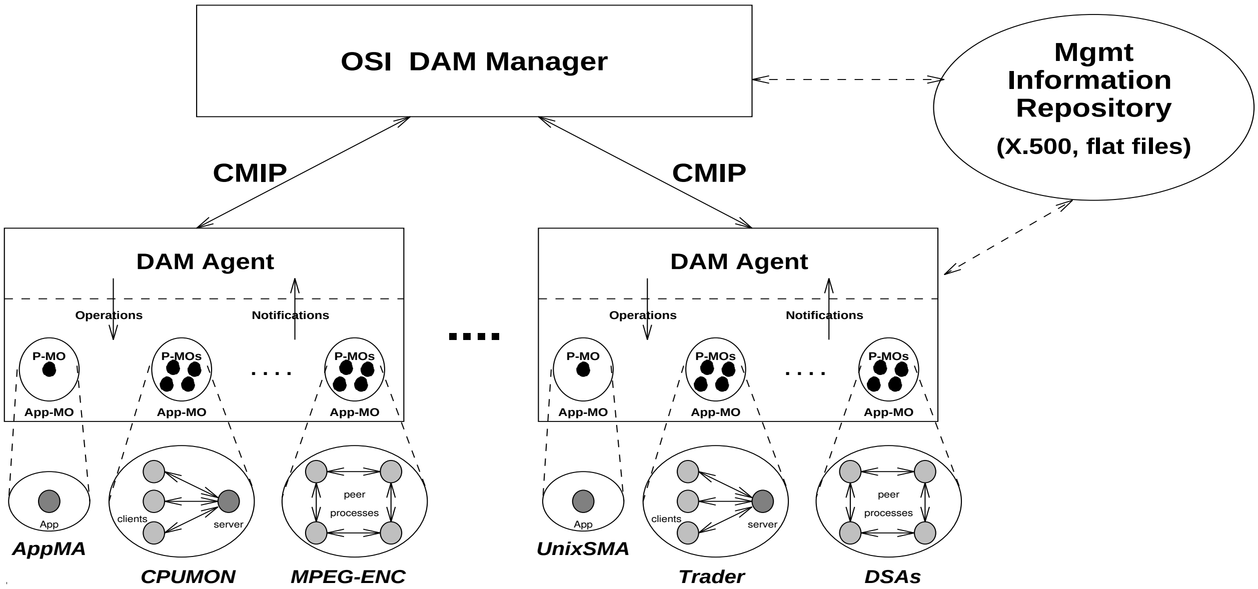

Figure 4: Implementation Architecture of the Distributed Application Management System

4.1 OSIMIS

OSIMIS is an OSI management system developed at UCL. It provides a set of facilities for developing manager applications, agent applications, and managed objects. It has been developed using

Appeared in the Proceedings of the International Conference on Computer Communication,
Seoul, Korea, August 1995

the ISO Development Environment (ISODE) [26] and has generic and specific parts. The generic parts include an object-oriented infrastructure for the development of OSI agents and managers and a set of generic manager applications. The specific parts are agents supporting specific MOs and associated managed resources. OSIMIS provides an object-oriented API (in C++), which hides the details of management information access through the OSI CMIS/P.

There is a considerable amount of code an OSI management application developer must provide for management application entities (agents and managers) communicating over the OSI protocol stack as well as for encoding and decoding of data being exchanged. OSIMIS combined with ISODE are providing much of this code and thus the developer can save considerable amount of time and effort.

OSIMIS also provides a GDMO compiler [4] which is used to parse the managed object class definitions (specified in GDMO [15]) and to generate source code. This eliminates the burden of writing much of the code for managed objects by hand. With this compiler, developing an agent requires the developer to define the classes of objects the agent is supposed to be interacting with and provide the "hand-written" code for accessing the real resources. The GDMO compiler generates the necessary code for accessing those managed objects by the agent.

4.2 OSI DAM Manager

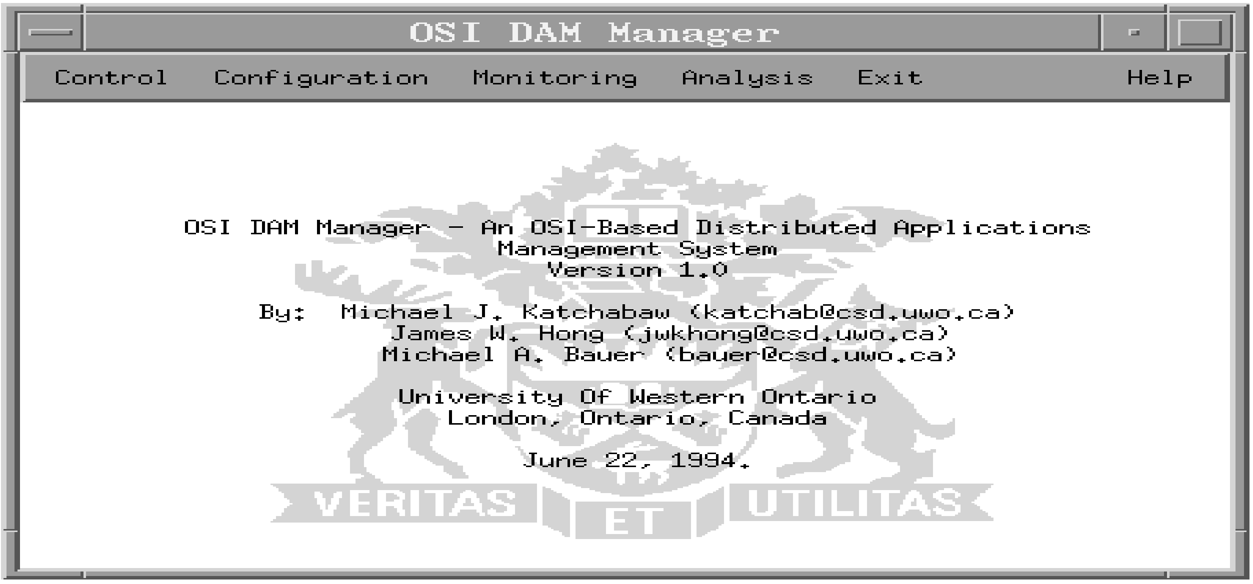

Figure 5: Motif-Based OSI DAM Manager

OSI DAM Manager is an OSF/Motif-based manager application designed to manage distributed applications [17]. The DAM Manager is composed of four toolsets: *configuration*, *control*, *monitoring*, and *analysis*. The configuration toolset allows the user to configure the distributed application to be managed. For example, the user can assign the components of an application to run on specific hosts. The configuration toolset also allows the user to start up and shut down the application. The monitoring toolset allows the user to select one or more distributed applications that are running and to monitor their activities. The analysis toolset allows the user to analyze the monitored

Appeared in the Proceedings of the International Conference on Computer Communication,
Seoul, Korea, August 1995

data. Finally, the control toolset allows the user to dynamically create, delete, modify and perform actions on managed objects.

The DAM Manager interacts with DAM agents to manage distributed applications. The interactions between the DAM Manager and DAM Agents are through CMIS/P (see Figure 4). Now we elaborate these DAM agents below.

4.3 OSI DAM Agent and Managed Objects

Since we are interested in managing distributed applications, we have developed an OSI distributed applications management agent (DAM agent) by defining managed object classes corresponding to managed applications and their components. We have used OSIMIS facilities to achieve this. First, we defined a GDMO definition of distributed application MO class (appObj). As mentioned in Section 3, one of the most important components of any distributed application is its component processes. Thus, we defined a GDMO definition of a process MO class (procObj) and defined them as subordinate objects of application MOs in the MO containment hierarchy. Object instances of the managed application object class and process object class are App-MOs and Proc-MOs respectively (see Figure 3).

We then used the OSIMIS GDMO compiler to generate the source code (in C++) for accessing these managed objects by a DAM agent. The real resources these Proc-MOs are interacting with are UNIX run-time processes that are part of the distributed application being managed. Interaction methods between the MOs and the real managed resources (applications) depends on the location of real resources as well as the type of communication paradigm the instrumented real resources provide. If the real resources are located on a local machine then the interaction method may take the form of local communication (*e.g.*, system calls or procedure calls). If the real resources are located on a remote machine then the interaction method may take the form of Internet socket communication, a remote procedure call (RPC) or some others.

We then used an automatic management agent creation tool called MACT [24, 23] to create various DAM agents by specifying the MO classes to be included and the types of management operations to be performed. MACT also includes all the necessary run-time libraries and generates the necessary source code needed to create a DAM agent.

Deploying a distributed application in this management environment is equivalent to instantiating an appObj class with the configuration information, which may include the name of an application and where the component processes are to be run. As part of the App-MO creation, one or more component Proc-MOs would be automatically created. Each Proc-MO, in turn, creates an application component process as part of its creation procedure. These processes may be distributed throughout the computing environment as is the case in our prototype implementation.

Shutting down an application is simply equivalent to deleting an existing App-MO instance, which would also delete all of its subordinate Proc-MOs. Each Proc-MO then is responsible for shutting down its corresponding real process. The manager also has the flexibility of creating and deleting Proc-MOs individually as well. This would allow the manager to perform control actions depending on the status of the application and the user's needs. For example, the user may wish to create more processes to provide more processing capability to the application or delete some unnecessary or inefficient processes. Both starting up and shutting down applications or individual component processes can be initiated by the DAM Manager.

Appeared in the Proceedings of the International Conference on Computer Communication,
Seoul, Korea, August 1995

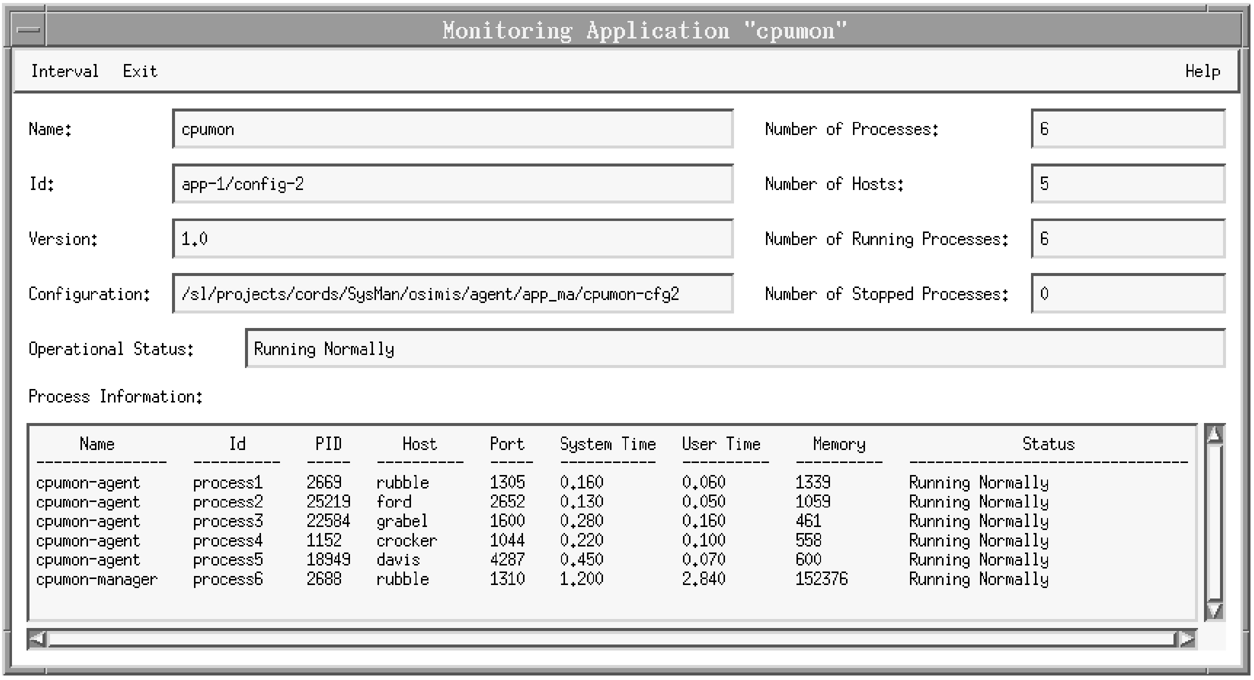

Figure 6: A Display of Monitored Application and Component Process Information

Figure 6 displays the information on a managed application being monitored by the monitoring toolset of the DAM Manager. The figure displays the management on a simple distributed application called cpumon which monitors CPU load of processing nodes in a distributed computing environment. The application consists of a load monitor process which periodically interacts with the load monitoring agent processes running on the individual monitored hosts for retrieving their CPU loads. The display shows that the cpumon application consists of six processes. It also displays the monitored information of component processes being managed as well. Such status and resource usage information from individual component processes could be monitored through the management interface we have instrumented into the processes being monitored [6].

4.4 Management Information Repository

We have integrated a management information repository (MIR) into our distributed applications management system for storing various useful management information. Such information can be added or updated by the DAM agents on their status information (such as up or down), their presentation address, and what applications they are managing. Information such as this can be then utilized by the OSI DAM Manager for locating appropriate DAM agents for sending management requests.

For example, when a DAM agent is started, it registers (or updates) its status and address information to the MIR. When an application is started (after receiving a create App-MO request), it updates information on the applications it currently manages by adding to the list of managed applications in the MIR. When an application is shutdown (after receiving a delete App-MO request), it also updates information by removing the application from the managed application list.

When the OSI DAM Manager wishes to perform various management operations (in response to

Appeared in the Proceedings of the International Conference on Computer Communication,
Seoul, Korea, August 1995

the user's actions), it looks up the required information from the MIR. For example, the Manager needs to locate the DAM agent which manages the application the user wishes to monitor or control a running application. This can be achieved by querying the MIR for the DAM agent which currently manages the searched application. We have implemented the MIR as a heterogeneous set of data sources including the X.500 Directory Service. This is an extension of our earlier work which used the X.500 Directory Service for storing various network resource information for network management [7].

5 Concluding Remarks

Management of distributed applications and services are needed to ensure the continued operations and acceptable performance in distributed computing environments. In this paper, we have reduced popular models (namely centralized, client-server and peer-to-peer) of distributed applications into abstract application managed objects containing one or more process managed objects using the OSI management model. Using this abstraction, most (if not all) distributed applications and services can be captured and managed. Using OSIMIS, we then developed a prototype distributed applications management system consisting of the OSI DAM Manager, the DAM Agents and the management information repository.

Given our previous work on the development of our own management framework and protocols for distributed applications [5], we found the use of a standard protocol and framework to be a great advantage. First, whether it be the OSI Management Framework (CMIP) or the Internet Management Framework (SNMP) [1, 2], there is a well-defined set of services and framework. This simplifies to conceptual modeling of the management problem. Second, there is often existing tools, like OSIMIS in our work, which can greatly simplify the design and implementation of managers and agents. Third, our prototype suggests that there are many reusable components which can be used to implement other managers and agents, thus facilitating the development of manageable distributed applications. We have developed a management agent creation tool called MACT to partially automate the development of management agents [23].

Although we have been successful in using the OSI management framework as a basis for managing distributed applications, our experiences are still limited. There is clearly a need to investigate the role of the OSI management framework, and that of SNMP, in more complex distributed application environments. We are currently working on the management of more complex distributed applications, such as X.500 and ODP Traders [16, 25], and within a computing environment having several such applications. It is only by examining the management of these more complex applications that we will be able to assess the utility of the existing frameworks in this context.

As indicated above, an alternative to the OSI Management Framework is the Internet Management Framework. Some work has been done in investigating the use of the Internet Management Framework for other than network management (such as UNIX systems management [18]). Would the Internet Management Framework provide sufficiently powerful mechanisms for managing distributed applications? We are also currently investigating this question.

Acknowledgments

We would like to thank George Pavlou and his research team at UCL for providing us with OSIMIS, especially the pre-release version of the GDMO compiler which saved us a great amount of

development time.

References

[1] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Introduction to version 2 of the Internet-standard Network Management Framework. *Internet Request for Comments 1441*, April 1993.

[2] J. D. Case, M. S. Fedor, M. L. Schoffstall, and J. R. Davin. A Simple Network Management Protocol. *Internet Request for Comments 1157*, 1990.

[3] CCITT. *The Directory - Overview of Concepts, Models and Services, CCITT X.500 Series Recommendations*. CCITT, December 1988.

[4] J. Cowan. *OSIMIS GDMO compiler user manual*. Department of Computer Science, University College London, UK, August 1993.

[5] J. W. Hong and M. A. Bauer. Design and Implementation of a Generic Distributed Applications Management System. *Proc. of the GLOBECOM '93*, pages 207–211, Houston, TX, November 1993.

[6] J. W. Hong, M. A. Bauer, and H. M. Abdu. A Framework for Developing Manageable Distributed Applications. *Proc. of the 1993 Korea Telecom International Symposium*, pages 148–157, Seoul, Korea, November 1993.

[7] J. W. Hong, M. A. Bauer, and J. M. Bennett. Integration of the Directory Service in the Network Management Framework. *Proc. of the Third International Symposium on Integrated Network Management*, pages 149–160, San Francisco CA, April 1993.

[8] J. W. Hong, M. J. Katchabaw, M. A. Bauer, and H. Lutfiyya. *Distributed Applications Management Using the OSI Management Framework*. Technical Report #448, Dept. of Computer Science, University of Western Ontario, London, Canada, January 1995.

[9] ISO. *Information Processing Systems - Open Systems Interconnection - Basic Reference Model - Part 4: Management Framework*. International Organization for Standardization, International Standard 7498-4, 1991.

[10] ISO. *Information Processing Systems - Open Systems Interconnection - Common Management Information Service Definition*. International Organization for Standardization, International Standard 9595, 1991.

[11] ISO. *Information Processing Systems - Open Systems Interconnection - Common Management Information Protocol Specification*. International Organization for Standardization, International Standard 9596-1, 1991.

[12] ISO. *Information Processing Systems - Open Systems Interconnection - Structure of Management Information Part 1: Management Information Model*. International Organization for Standardization, International Standard 10165-1, 1991.

Appeared in the Proceedings of the International Conference on Computer Communication, Seoul, Korea, August 1995

[13] ISO. *Information Processing Systems - Open Systems Interconnection - Structure of Management Information Part 2: Definition of Management Information*. International Organization for Standardization, International Standard 10165-2, 1991.

[14] ISO. *Information Processing Systems - Open Systems Interconnection - Systems Management Overview*. International Organization for Standardization, International Standard 10040, 1991.

[15] ISO. *Information Technology - Open Systems Interconnection - Management Information Services - Structure of Management Information Part 4: Guidelines for the Definition of Managed Objects*. International Organization for Standardization, International Standard 10165-4, 1991.

[16] ITU-TS. *Draft ODP Trading Function*. ITU-TS SG7.Q16 Draft Recommendation, July 1994.

[17] M. J. Katchabaw, J. W. Hong, and M. A. Bauer. *OSI DAM Manager User's Guide Version 1.0*. Technical Report, *in preparation* Dept. of Computer Science, University of Western Ontario, London, Canada, October 1994.

[18] B. Krupczak. UNIX Systems Management via SNMP. *Proc. of the Third International Symposium on Integrated Network Management*, pages 289–299, San Francisco CA, April 1993.

[19] K. Marzullo, R. Cooper, M. D. Wood, and K. P. Birman. Tools for Distributed Application Management. *IEEE Computer*, 25(8):42–51, August 1991.

[20] D. M. Ogle, K. Schwan, and R. Snodgrass. Application-Dependent Dynamic Monitoring of Distributed and Parallel Systems. *IEEE Transactions on Parallel and Distributed Systems*, 4(7):762–778, July 1993.

[21] G. Pavlou, S. N. Bhatti, and G. Knight. *The OSI Management Information Service User's Manual*. Version 1.0, University College London, UK, February 1993.

[22] G. Pavlou, G. Knight, and S. Walton. Experience of Implementing OSI Management Facilities. *Proc. of the IFIP Second International Symposium on Intergrated Network Management*, Washington DC, April 1991.

[23] G. S. Perrow, J. W. Hong, M. A. Bauer, and H. Lutfiyya. The Abstraction and Modelling of Management Agents. *Proc. of the Fourth International Symposium on Integrated Network Management*, Santa Barbara CA, May 1995.

[24] G. S. Perrow, J. W. Hong, M. A. Bauer, and H. Lutfyya. *MACT User's Guide Version 1.0*. Technical Report #434, Dept. of Computer Science, University of Western Ontario, London, Canada, September 1994.

[25] A W. Pratten, J. W. Hong, M. A. Bauer, H. Lutfiyya, and J. M. Bennett. A Resource Management System Based on the ODP Trader Concepts and X.500. *Proc. of the Fourth International Symposium on Integrated Network Management*, Santa Barbara CA, May 1995.

[26] C. J. Robbins and J. P. Onions. *The ISO Development Environment: User's Manual, Version 8.0*. X-Tel Services Ltd., June 1992.

Appeared in the Proceedings of the International Conference on Computer Communication,
Seoul, Korea, August 1995

Appendix A – GDMO Defintion for Application MO Class

```
-- ##############  Class Templates  ##############

appObj MANAGED OBJECT CLASS
    DERIVED FROM        top;
    CHARACTERIZED BY    appObjPackage;
    REGISTERED AS       { uclManagedObjectClass 70 };


-- ##############  Name Binding Templates  ##############

 appObj-system                   NAME BINDING
     SUBORDINATE OBJECT CLASS    appObj AND SUBCLASSES;
     NAMED BY
     SUPERIOR OBJECT CLASS       system AND SUBCLASSES;
     WITH ATTRIBUTE              appObjId;
     CREATE                      WITH-AUTOMATIC-INSTANCE-NAMING;
     DELETE                      DELETES-CONTAINED-OBJECTS;
     REGISTERED AS               { uclNameBinding 70 };


-- ##############  Package Templates  ##############

appObjPackage PACKAGE
    ATTRIBUTES
        appObjId               GET,
        appName                GET-REPLACE,
        appVersion             GET-REPLACE,
        appConfig              GET-REPLACE,
        appNumProcs            GET,
        appNumHosts            GET,
        appRunningProcs        GET-REPLACE,
        appStoppedProcs        GET-REPLACE,
        appOperStatus          GET-REPLACE;

      ACTIONS
        setrunproc,
        delProcess,
        setDownProcess,
        setUpProcess;

    NOTIFICATIONS
        objectCreation,
        objectDeletion,
        attributeValueChange;        -- for attributes

    REGISTERED AS       { uclPackage 70 };

-- ## Attribute, Action, Notification Templates omitted for simplicity
```


Appeared in the Proceedings of the International Conference on Computer Communication, Seoul, Korea, August 1995

Appendix B – GDMO Definition for Process MO Class

```
-- ##############  Class Templates  ##############

procObj MANAGED OBJECT CLASS
    DERIVED FROM        top;
    CHARACTERIZED BY    procObjPackage;
    REGISTERED AS       { uclManagedObjectClass 80 };

-- ##############  Name Binding Templates  ##############

procObj-appObj                  NAME BINDING
    SUBORDINATE OBJECT CLASS    procObj AND SUBCLASSES;
    NAMED BY
    SUPERIOR OBJECT CLASS       appObj AND SUBCLASSES;
    WITH ATTRIBUTE              procObjId;
    CREATE                      WITH-AUTOMATIC-INSTANCE-NAMING;
    DELETE                      ONLY-IF-NO-CONTAINED-OBJECTS;
    REGISTERED AS               { uclNameBinding 80 };

-- ##############  Package Templates  ##############

procObjPackage PACKAGE
    ATTRIBUTES
        procObjId               GET,
        procName                GET-REPLACE,
        procParms               GET-REPLACE,
        procPID                 GET,
        procHost                GET-REPLACE,
        procMPort               GET,
        procSysTime             GET,
        procUsrTime             GET,
        procMemory              GET,
        procOperStatus          GET;

    ACTIONS
        getRUsage,
        checkPID;

    NOTIFICATIONS
        objectCreation,
        objectDeletion,
        attributeValueChange,   -- for attributes
        objectRegistration,
        processTermination,
        processDown,
        processUp;

    REGISTERED AS       { uclPackage 80 };

-- ## Attribute, Action, Notification Templates omitted for simplicity
```


Appeared in the Proceedings of the International Conference on Computer Communication,
Seoul, Korea, August 1995

Appendix C – GDMO Definition for Trader MO Class

```
-- ##############  Class Templates  ##############

traderObj MANAGED OBJECT CLASS
    DERIVED FROM        procObj;
    CHARACTERIZED BY    traderObjPackage;
    REGISTERED AS       { uclManagedObjectClass 110 };


-- ##############  Name Binding Templates  ##############

traderObj-appObj                NAME BINDING
    SUBORDINATE OBJECT CLASS    traderObj AND SUBCLASSES;
    NAMED BY
    SUPERIOR OBJECT CLASS       appObj AND SUBCLASSES;
    WITH ATTRIBUTE              procObjId;
    CREATE                      WITH-AUTOMATIC-INSTANCE-NAMING;
    DELETE                      ONLY-IF-NO-CONTAINED-OBJECTS;
    REGISTERED AS               { uclNameBinding 110 };


-- ##############  Package Templates  ##############

traderObjPackage PACKAGE
    ATTRIBUTES
        traderNumClients        GET,
        traderNumResources      GET,
        traderRegistrationRate  GET,
        traderExportRate        GET,
        traderFindRate          GET,
        traderListRate          GET,
        traderAveLookupTime     GET;

    ACTIONS
        getTraderSpecific;

    REGISTERED AS       { uclPackage 110 };


-- ## Attribute, Action Templates omitted for simplicity
```


Appeared in the Proceedings of the International Conference on Computer Communication,
Seoul, Korea, August 1995