Efficient Management Data Acquisition and Run-time Control of DCE Applications Using the OSI Management Framework*

Michael J. Katchabaw Stephen L. Howard Hanan L. Lutfiyya Michael A. Bauer

Department of Computer Science
The University of Western Ontario
London, Ontario, Canada N6A 5B7
E-mail: {katchab,showard,hanan,bauer}@csd.uwo.ca

Abstract

The goal of a management system in a distributed computing environment is to provide a centralized and coordinated view of an otherwise distributed and heterogeneous collection of hardware and software resources. The management software required to achieve this goal will, within a policy framework, monitor, analyze and control network resources, system services and distributed application programs.

In our research, we investigate the use of the Open System Interconnection (OSI) network management standards as a basis for a generic architecture around which such management systems can be built. We are particularly interested in the suitability and flexibility of the OSI standards in the context of managing application programs. The work described in the current paper focusses on efficiency and dynamic control of management data acquisition and on the run-time control of application programs. We describe a prototype management system developed to explore these issues in the management of DCE applications.

1 Introduction

Managers of distributed computing systems require sophisticated management software to help them address the complexity of interconnected networks, diverse host architectures, multi-layered middle-ware, multi-process applications and distributed data. Provision of such distributed management systems is, in fact, critical to the acceptance of distributed computing as a main-stream computing technology. Unfortunately, the distributed systems management problem is a complex one, and most successes to date have been limited to system and network management.

Although the need for distributed application management is apparent, relevant research is just beginning. The Open Software Foundation (OSF) is currently attempting to standardize performance instrumentation and interface specifications for monitoring Distributed Computing Environment (DCE) [9] services and other DCE-based applications [10, 11]. The Object Management Group (OMG) is close to adopting X/Open's System Management Common Management Facility [14][1] for the CORBA distributed object computing environment which, together with other standard CORBA services [8], provides a foundation for building management applications. The Internet Engineering Task Force (IETF) has chartered the *Application MIB Working Group* to define a set of managed objects for the monitoring and control of distributed applications in an SNMP environment.

Our current research investigates the management of distributed application programs using the OSI Management Framework [4, 6]. Figure 1 illustrates the basic structure and interactions within this framework.

The OSI standards use object-oriented concepts for modeling open system resources. A *managed object* (MO) is an abstraction of one or more real resources and one which is deemed to be an appropriate management view. A managed object is defined in terms of the attributes it possesses, the operations that may be performed upon it, the notifications it may issue and its relationships with other managed objects. The

*This research work is supported by the IBM Centre for Advanced Studies and the Natural Sciences and Engineering Research Council of Canada.

[1] A revised version of the referenced document has been submitted to OMG and is being considered for adoption.

Appeared in the Proceedings of the Second International Workshop on Systems Management, Toronto, Canada, June 1996

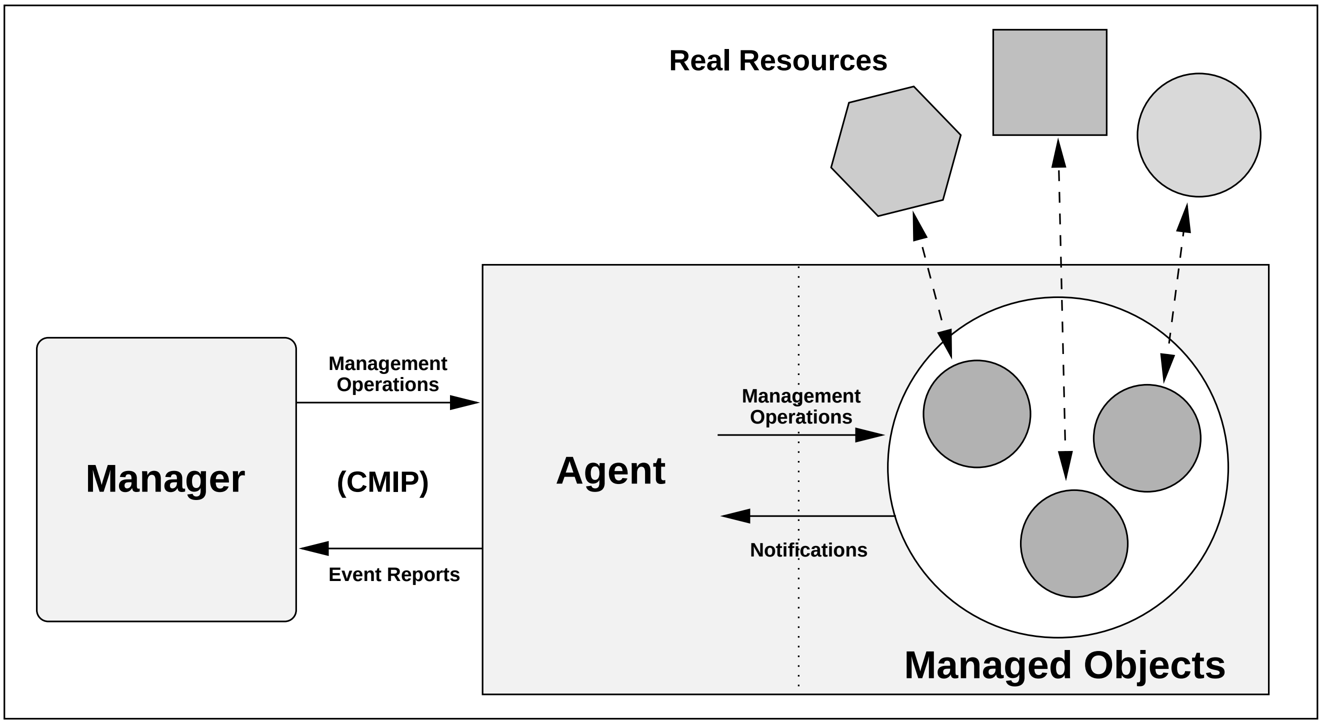

Figure 1. OSI Management Framework

set of managed objects within a system, together with their attributes, constitutes that system's *management information base* (MIB). The managed objects in the MIB are organized into a hierarchy called the *management information tree* (MIT).

A *management agent* is responsible for a particular collection of managed objects. The agent and its managed objects serve to decouple management applications from their managed resources. This approach has many advantages. It facilitates many-to-many relationships between management applications and managed resources. It provides a means to distribute the management function; localized agents can perform such tasks as data aggregation, filtering, analysis and logging, ultimately reducing the flow of data to the management application. An agent receives management requests from a management application(s) and carries out the operations on the appropriate managed objects. Conversely, notifications emitted by managed objects are routed to appropriate management applications.

Management applications serve to implement management policies on behalf of human managers. In the OSI Management Framework, the management application views the managed domain precisely as it is modeled by managed objects; it knows nothing of the real resources. All communication between the management application and the managed objects is through the agent and uses the the Common Management Information Protocol (CMIP) [5].

We promote this framework as a foundation upon which to build a single unified architecture suitable for all facets of the distributed application management problem. That is, we pursue the notion that performance management, configuration management, fault management, security management and accounting management can all be carried out within the context of a single management architecture. Furthermore, we believe that the architecture and mechanisms employed are based on requirements which are equally applicable to network and system-level management.

In our previous work [2, 3], we developed a basic agent architecture and focussed on defining suitable abstractions for distributed applications and services. Interaction between the agent and the real resources (which falls outside the OSI standards) was dealt with in a simplistic and ad-hoc manner. Because of the focus of that work, little attention was paid to efficiency and generality in this part of the architecture.

The work described in this paper concentrates on the management services necessary at the agent-application interface to exert run-time control over application programs and to achieve efficiency in the acquisition of management data. The latter is largely dependent both on techniques employed to carry data between application entities and management agents and on the ability of the management system to allow dynamic control over the acquisition mechanism itself. Generality is an important theme in this work. The architecture and techniques proposed are designed with flexibility in mind – flexibility to support different management functions (performance management, configuration management, fault management, accounting management, and security management).

An experimental management system is described in which DCE applications are managed using the OSI-based architecture. The object-oriented modeling of OSI is shown to be sufficiently powerful to describe an appropriate managed object hierarchy. Communication of management information between application entities and management agents uses standard DCE remote procedure calls (RPC).

The paper is organized into five sections. Section 2 outlines some of the major objectives guiding the current work. Section 3 describes how the OSI Management Framework is used as the basis for a generic management architecture to meet these objectives. Section 4 describes the prototype management system developed for managing the performance of DCE applications. Section 5 concludes with a summary of the project status and directions for future work.

2. Objectives

A number of general objectives were established to guide our study of the agent-application interaction and to influence the refinement of the management architecture. These include:

- **Support for Application Control**

  The management system must be able to exert control over its managed resources. For application management, this includes system-level process control (starting, terminating, suspending and resuming processes) as well as process/application registration and process/application run-time status inquiry.

- **Data Acquisition**

  A management system which imposes undue load on CPUs, networks and other resources is of little value. Efficiency is a complex issue but there are a number of architectural objectives which can be established in its support:

  - The most predominant distributed management system, SNMP-based network management, uses the *pull* or polling model to gather management information from network resources. While this approach may imply simplicity at the device level, it imposes serious limitations on efficiency.

    Alternatively, we feel that the management system should support both *pull* and *push* models. The former is required to satisfy management inquiries and to facilitate control imperatives issued to the managed resources. The *push* model should be used for the efficient transport of most management data. Using this model, unsolicited management data, including timed event reports and alarms can be sent from application processes to the management system without the overhead of polling.

  - Two basic acquisition mechanisms are required. Interval-based sampling should be used to report "normal" management information. Efficiency is enhanced by collecting this data at the resource level and sending it in bunches at prescribed intervals. Exceptional conditions (often controlled by one or two threshold levels associated with a particular attribute) require higher-priority notification channels. For these, alarm-style notifications should be sent immediately to the management system.

  - It must be possible to dynamically control various aspects of the management function. For example, a performance monitoring subsystem should include controls which can be used to establish what information is collected, to adjust the sampling interval (including disabling sampling altogether), to modify alarm threshold values and dictate the granularity of the data gathered. Such controls are important for efficiency, problem diagnosis, tuning and other factors.

  - The management system must be prepared to accept data and event notifications from multiple application processes concurrently. Similarly, the application process must support concurrency so it can deal with its normal application communication as well as requests from one or more management agents.

Ideally, an architecture of the agent-application interaction would be used to manage all applications with respect to configuration, performance, reliability, accounting and security. This architecture would adapt easily to the wide variety of interprocess communi-

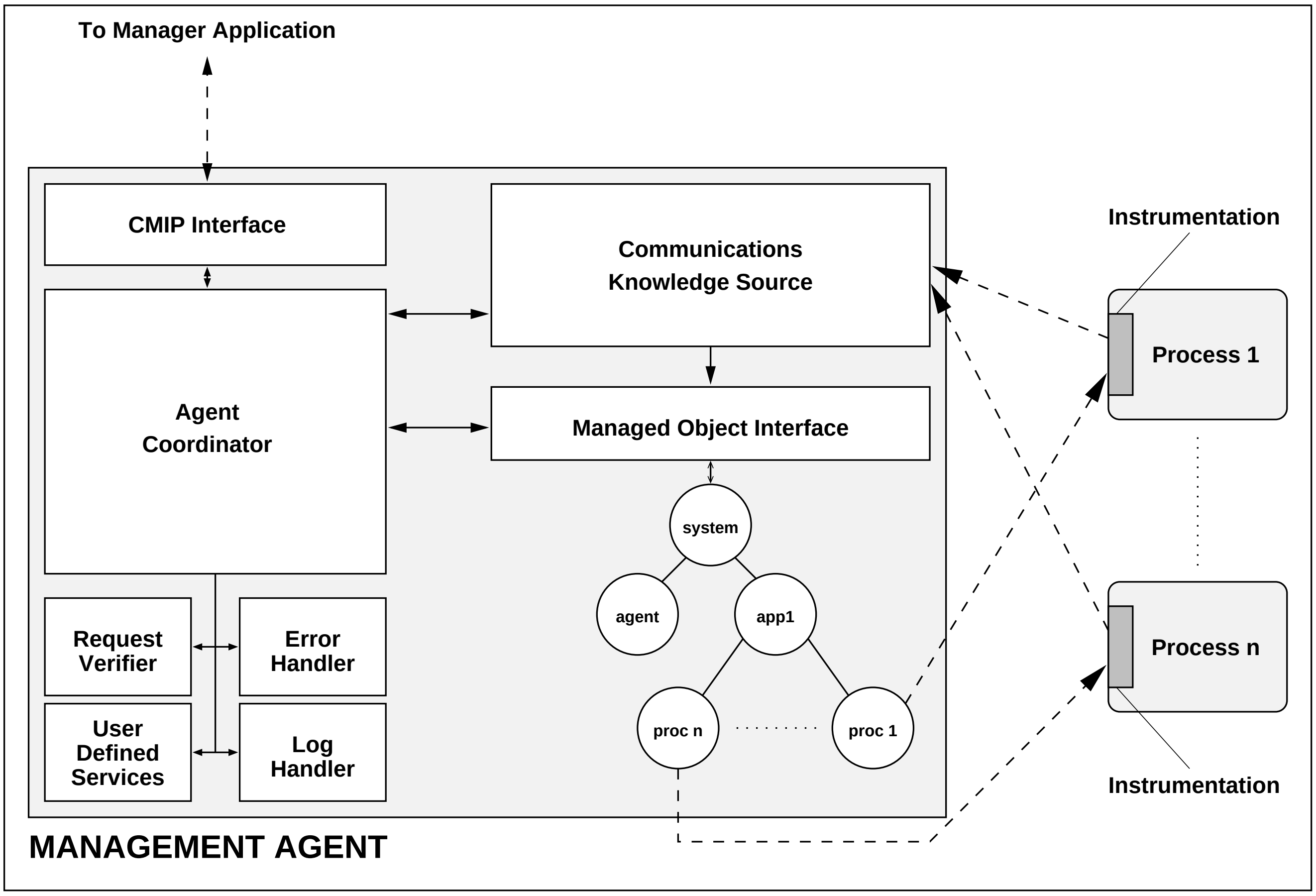

Figure 2. Agent-Application Interaction Architecture

cation schemes employed by distributed applications (sockets, Sun RPC, DCE, and so on).

3. A Generic Management Architecture

The generic management architecture of the agent-application interaction described in this section refines the architecture developed in our previous work [2, 3] to improve service at the agent-resource interface and to enhance its generality for a much wider class of applications based on several different communications paradigms.

3.1. Component Overview

The agent-application interaction architecture shown in Figure 2 is used as the basis for our research. The most relevant components of the architecture are described below.

Agent Coordinator: The role of the agent coordinator is to coordinate activities taking place within the agent. This entails the internal routing of requests and notifications to the appropriate components and the synchronization of access to the managed objects.

Communications Knowledge Source: The communications knowledge source provides an interface for management information and event notifications from the instrumented application. The knowledge source contacts the agent coordinator to synchronize exclusive access to managed objects and performs operations on managed objects through the managed object interface. Knowledge sources must be tailored to the communication mechanism used by the managed applications (for example, a DCE knowledge source is required to interact with DCE applications).

Managed Object Interface: The managed object interface is used to coordinate requests on managed objects and to handle emitted notifications.

Managed Objects: A distributed application is modeled by a single application managed object containing (in the object-oriented sense) one or more process managed objects. Process managed object classes are specialized through inheritance to reflect their unique characteristics including communication environment (sockets, DCE RPC, etc.), process role (client, server, or peer), and application type (database, etc.). Each process managed object has attributes for its name, host, process id, priority, status, resource utilization, communication statistics, and so on. These attributes depend on the specialization of the process managed object class (for example, DCE client processes have different performance metrics than DCE server processes). Process managed objects use the appropriate mechanism to communicate with instrumented application processes in the system (for example, DCE RPC would be used to communicate with DCE application processes).

Instrumented Processes: Code instrumentation is required in application processes to make them manageable. This includes both a mechanism which "listens" for incoming management requests from agents and another to send management information and event reports to agents.

The architecture described is flexible and can be used to construct various management system configurations to meet functional and efficiency requirements. For example, the architecture can support the use of multiple agents. Perhaps it is desirable to partition management functionality, using one agent for configuration management, another for fault location and yet another to handle the registration of application processes. There is also nothing in the architecture that precludes interaction between agents or between management applications. For example, a performance management agent might report "hot spots" to a fault locator agent.

3.2. Interactions

As stated above, our present focus is on the interactions between the management agent and the application processes. We are concerned with generality, efficiency of data acquisition and on run-time controls.

3.2.1. Management Information Acquisition. Acquiring management information is critical to the management of distributed applications, and as a result, it is important to do so well. In our architecture, this acquisition is done in one of two ways:

1. **On Demand:** The on demand approach to collecting management information is the typical approach employed by traditional management systems. Managers send management requests for information to agents, which either return the information immediately, or contact the appropriate managed processes for the information, depending on what is being requested. This is done using the operation:

   retrieveManagementInformation: By indicating the type of information desired (resource utilization, communication statistics, performance metrics, or all possible data), this operation will cause the process instrumentation to collect and return the information specified.

2. **Event Reports:** Event reports are another way of acquiring management information, in a somewhat different manner. In this case, the flow of information is from the processes to the agents and then on to the managers, if the managers are interested in the information. Event reports are typically simple periodic reports from the processes, made using the operation:

   sendManagementInformation: Sends unsolicited management information describing the process's state, resource consumption, performance, and so on.

   When significant events occur that can affect the normal operation of the application, alarm reports can be generated using the operations:

   notifyCommunicationsFailure: Notifies the agent that a request from one application process to another failed due to a communication error.

   notifyServiceFailure: Notifies the agent that the process was unable to successfully contact some system service (for example, the DCE cell directory service).

   notifyCongestion: Notifies the agent that the process is experiencing congestion due to over-utilization and that the expected quality of service has degraded.

   Finally, additional reports include registration reports, informing of the initialization of processes, and termination reports, informing of their termination. These reports are generated using the operations:

notifyProcessRegistration: Notifies the agent of the existence of the new process, and provides basic information for contacting the process (process name, process id, host, presentation address, and so on).

notifyTermination: Notifies the agent of the termination of the process in the context of the normal operation of the application.

3.2.2. Process and Management Control. While acquiring management information is critical for management, it is equally important to be able to use this information to control the operation of distributed applications to meet management policies. There are two kinds of control that are needed: control of the managed processes themselves, and control of the management of the processes. The architecture proposed here supports both types.

1. **Control of the Managed Processes:** The following operations are required to exert control over the distributed application processes:

   controlTerminateProcess: Instructs the process to terminate as quickly as possible.

   controlSuspendProcess: Temporarily suspends the functioning of a process.

   controlWakeUpProcess: Awakens a process that was previously suspended.

   controlChangeProcessPriority: Changes the priority of an executing process.

2. **Control of the Management of the Processes:** To manage a distributed application effectively, we must be able to change the way management is done at run-time in order to react to operational needs. Unfortunately, this is an area of control often overlooked. Operations required include:

   controlManageProcess: Activates management in a process, informing the process which agent will be managing it.

   controlUnmanageProcess: Deactivates management in a process when it is no longer needed.

   controlChangeManagementSettings: Used to tailor the event reports generated by the processes. For periodic reports, this operation can change the interval between reports (*i.e.* how often the **sendManagementInformation** operation is invoked). For alarm reports, it can change thresholds that generate alarms (*i.e.* what causes the **notifyCommunicationsFailure**, **notifyServiceFailure**, and **notifyCongestion** operations to be invoked). In both cases, this operation can change the information content of the reports.

The services and operations outlined here consititute a core set. It is probably that extensions will be made to the set as the management system is refined and applied to specific contexts.

4. Prototype Implementation

A prototype management system based on the generic architecture presented in the previous section was developed as a proof of concept. The management system was used in the context of DCE applications in various Unix environments.

The prototype builds upon University College London's OSI Management Information Service (OSIMIS) [12, 13] which provides an object-oriented infrastructure for developing management applications and OSI agents. OSIMIS includes a GDMO compiler [1] to parse managed object class definitions (specified in GDMO [7]) and generate the agent code required to access those managed objects. Most of our development effort was confined to the agent-application interaction; the definition of managed object classes, application process instrumentation and a custom DCE knowledge source within the agent.

In the paragraphs below, we discuss the prototype and how it satisfies the established objectives.

4.1. Managing DCE Applications

Using OSIMIS, we first developed generic management agents. Since we were interested in managing DCE applications, these generic agents then had to be specialized to communicate with application processes using DCE RPC. This involved deriving a DCE knowledge source from the generic communications knowledge source. In addition, we developed DCE process managed object classes, again derived from more generic objects. We further derived managed object classes specifically for DCE client, DCE server, and DCE peer processes. Each of these derived classes has attributes, event reports, and control operations specific to the class.

In addition to equipping the agents appropriately, the DCE applications themselves must prepared for management. This involves adding instrumentation to each managed process so it can both respond to management requests and send event reports to agents. In this regard, an instrumentation library based on the architecture of the previous section was created and "hooks" were injected at appropriate points throughout the application code. Instrumentation is typically placed at process entry and exit points, and surrounding interprocess communication (naturally, additional hooks may be added as necessary). At run-time, management threads are created within each application process to handle the interactions with agents and maintain the necessary management information for reporting resource utilization, various statistics, and process performance.

Management, in the case of DCE applications, involves a sequence of remote procedure calls between agents and managed processes. When a process is started, it first registers with an agent which, in turn, informs the process that it is now being managed. Periodically, the managed process sends information reports to the agent for analysis (or other use). The nature of the information and the reporting interval are dynamically adjustable. Under exceptional circumstances (such as communication failures, server failures, or server congestion), alarm reports are generated and sent immediately to the agent(s). Thresholds associated with these alarms are also dynamically configurable. Additional management information may be requested at any time by the management system (outside of the normal reporting interval). In addition, a variety of control requests (for termination, suspension, resumption, and so on) may be imposed on the process during operation. When the process is about to terminate under normal conditions, it notifies its agent so that management can be discontinued.

Through this set of interactions, DCE applications can be managed quite effectively.

4.2. Experience

Initial experiments using the prototype management system were conducted in our test laboratory to verify the operation of the system and assess its suitability for managing distributed applications. In these tests, we instrumented DCE applications for the purpose of managing their performance. As the applications executed, the instrumentation in the component processes generated periodic event reports describing process resource utilization, remote procedure call response times, service throughput, and so on. This collected data can be used as input to performance analysis (although analysis was outside the scope of these first experiments). Using the controls built into the management system, we were able to "tune" the acquisition mechanism so the type, quantity and relevance of the management information was appropriate and the collection seemingly efficient. Upon dramatically increasing application loads, alarms were raised to report congestion at the servers. Similarly, forcing abnormal process termination caused alarms due to the resulting communication failures. This initial experimentation with the prototype indicated that the system was quite suitable for this type of management.

After our initial experimentation, the prototype was used in a cooperative research project involving several Canadian universities. This project integrated various aspects of performance modeling and management with respect to DCE applications. Our prototype was used as the basis for interacting with the managed applications for control and data acquisition. The project gave us some feedback on the flexibility of the architecture as the prototype was used in a different environment and had to be integrated with various software components developed at other universities.

As a third application of the prototype, we explored its suitability to another management problem – fault management. In this case, new fault manager applications were designed to replace the performance manager applications. By interacting with the agents and instrumented processes, defined classes of application faults could be detected, located, and corrected quite effectively. Only minor extensions to the prototype code were needed to support the new alarm reports required by the fault manager applications.

Based on these experiments, we believe that the architecture is suitable for a broad class of distributed management problems.

5. Concluding Remarks

The work we describe here is part of an ongoing structured attack on the distributed systems management problem. There are many complex issues yet to be resolved. We believe that using a flexible and generic architecture makes it easier to address these issues and lends uniformity both to the research and to the end product.

The research detailed in this paper focusses on the interaction between management agents and application processes. We have presented a core set of generic management services and operations for the efficient acquisition of management information and for effective run-time control over distributed applications. Through a prototype management system, we have demonstrated that the OSI Management Framework is a suitable infrastructure for the provision of these services and that our generic agent-application architecture is sufficiently rich to support a wide range of management functions.

The work is ongoing. We next plan to carry out performance measurement and analysis on the management system itself. There are many questions outstanding regarding the overhead of the management system and how and when the control mechanisms can be used most effectively. We need to examine data reduction techniques; for example, how and where analysis and aggregation are carried out to minimize overhead and maximize function. We need to look at how management function should be partitioned across agents. Should separate agents be assigned to each function (fault location, configuration management, performance management, and so on)? Is co-operation among agents useful?

We have a number of ongoing projects in areas of fault location, configuration management and security management. All are based on the generic architecture described here.

References

[1] J. Cowan. *OSIMIS GDMO Compiler User Manual.* Department of Computer Science, University College London, UK, August 1993.

[2] 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.

[3] J. W. Hong, M. J. Katchabaw, M. A. Bauer, and H. Lutfiyya. Modeling and Management of Distributed Applications and Services Using the OSI Management Framework. *Proceedings of the International Conference on Computer Communication*, October Seoul, Korea, August 1995.

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

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

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

[7] 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.

[8] OMG. *Common Object Services Specification*, volume 1. Object Management Group, March 1994. OMG Document Number 94.1.1.

[9] OSF. *Introduction to OSF DCE.* Open Software Foundation, first edition, 1992.

[10] OSF. *OSF Request For Comments 32.0 - Requirements for Performance Instrumentation of DCE RPC and CDS Services.* Open Software Foundation, June 1993.

[11] OSF. *OSF Request For Comments 33.0 - Standardized Performance Instrumentation and Interface Specification for Monitoring DCE-Based Applications.* Open Software Foundation, July 1995.

[12] 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.

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

[14] X/Open. *Systems Management: Common Management Facilities*, volume 1. X/Open Company Limited, January 1995. Preliminary Specification.