40

The Abstraction and Modelling of Management Agents *

Graeme S. Perrow, James W. Hong, Hanan L. Lutfiyya,
and Michael A. Bauer
Department of Computer Science
University of Western Ontario
{graeme,jwkhong,hanan,bauer}@csd.uwo.ca

Abstract

Management agents play an important role in distributed systems and network management. Agents are used to gather information, create, delete, and change the state of managed objects, and forward notifications of events from managed objects to managers. All management agents perform the same basic operations, yet there is no precise specification of the capabilities and architecture of *generic* management agents. As a result, developing management agents at present is difficult and time-consuming. This paper presents the design of a generic management agent and describes the architecture and service interface of such an agent. We also present an implementation of a management agent creation tool for automating the creation of management agents (CMIP, SNMP and other) which all bear the generic agent architecture. The use of this tool greatly reduces the time needed, and therefore the cost of developing management agents.

[**Keywords:** management agents, general agent architecture, CMIP agent, SNMP agent, extensible agent, automated agent development tool]

1 Introduction

Management systems contain three main types of components that work together: *managers*, which make decisions based on collected management information, *management agents*, which collect management information, and *managed objects*, which represent actual system or network resources being managed. Management agents perform operations requested by managers and notify managers of pre-determined events of interest to the manager. Agents are said to operate "on behalf of" managers, so that the manager's workload is greatly reduced, the load is distributed around the system or network, and efficiency is increased.

Agents play an important role in any management system. Agents are used to gather information, create, delete, and change the state of managed objects, and forward notifications of events

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

A. S. Sethi et al. (eds.), *Integrated Network Management IV*

from managed objects to managers (see Figure 1). A management agent is defined as an entity that provides a mechanism that performs management operations on managed objects and emits notifications on behalf of managed objects.

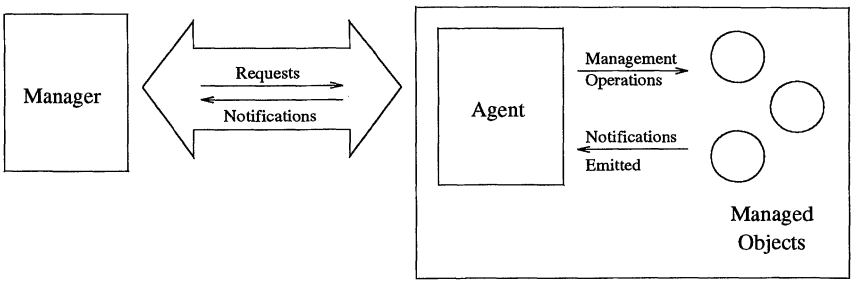

Figure 1: Manager-Agent-Managed Object interaction

To date, most of the research on systems and network management has concentrated on management protocols such as CMIP [6, 14] and SNMP [1, 2, 15]. There has also been a lot of work done on managed object definition; both the OSI [7] and the Internet [13] have created a managed object specification language for their respective management frameworks. These languages can be used to describe and define managed objects. There are even compilers available to parse these definitions and generate code that implements the managed object [10]. These compilers greatly facilitate the development of managed objects. However, relatively little work has focussed on facilitating the development of management agents.

Presently, the development of management agents is difficult, time-consuming and *ad hoc*. There are many decisions that must be made in the development of agents, such as, what services to offer, what relationships the agent should have with the environment (*i.e.*, hardware or software resources, user interface, *etc.*). Part of the reason for the difficulty in developing management agents is that these design issues have not been separated out from implementation details. For example, there is a set of services that is required from all or most agents; these include accepting monitoring and control requests from managers, executing these requests, returning results, notifying the manager of pre-determined events of interest and communicating with other entities. These services are independent of the underlying management protocols provided by the environment.

Some work has been done in the area of developing management agents. Both Bull [8] and DEC [9, 16] provide frameworks to create basic agents that handle management requests built into the standard management protocols (SNMP and CMIP). However, some of the operations (such as self-description, logging, user-defined) that we believe are key requirements for management agents are missing, and adding these operations to the agents would be a non-trivial task. Management by Delegation (MbD) [3, 4] has the opposite problem: creating large, powerful agents is easy, but if a small, simple agent is required, it would be far too large and intrusive to be useful. What is needed is a way to create agents so that the creation of a basic, simple agent is just as easy as creating a larger, more powerful agent with dynamically extensible functionality.

To facilitate the development of agents we have identified a generic architecture for agents. This architecture describes the services that the agents should or could provide, the components com-

prising an agent and how the components satisfy the services. The result is a specification of the capabilities of a *generic* management agent. This specification aids in the development of new management agents, since it saves the developer from designing the components, services, and interface of the agent to be created, and allows the developer to concentrate on customization of the agent. We then show that based on this architecture that a good deal of the creation of an agent can be automated based on a few pieces of user-supplied information, such as the management protocol, the basic management operations, and the resources to be managed.

The rest of this paper is organized as follows. Section 2 discusses the functional requirements of management agents. Section 3 presents our model of the architecture and service interfaces of a generic management agent. Section 4 describes a prototype management agent creation tool that can generate CMIP and SNMP agents automatically with the user's inputs from its graphical user interface. Section 5 concludes the paper with a summary and some future work.

2 Functional Requirements of Management Agents

In this section, we identify the key functional requirements for a management agent. An agent must be able to respond to requests from managers, other agents, and managed objects. The following classes of operations must be supported:

1. **Self-Description Operations:** An agent should be able to describe itself to other entities (*e.g.*, managers or other agents), so that they can discover what operations a particular agent is capable of, and what resources it can manage.

2. **Common Management Operations:** These represent operations common to the standard management protocols. They include operations to get information from managed resources, set information within managed resources, send commands to managed resources, and create and delete managed objects. Note that creating and deleting managed objects refers to creating and deleting abstract entities representing the managed resources, not the real resources themselves.

3. **User-Defined Operations:** Agents will require specific user-defined operations, such as performing different types of analysis on collected information. Hence, there must be some way to incorporate user-defined operations into an agent. Agents should also be able to execute services periodically, *i.e.*, a manager could request that this agent collect a certain piece of information every five minutes, without the need for the manager to send a request every five minutes.

4. **Logging Operations:** Agents should be able to log requests, notifications, and other management information for analysis, security, or statistical purposes.

3 The Generic Agent

This section presents an architecture for a generic management agent. It describes the components of the agents and the services that the agent provides. Using this architecture, the need for the developer to spend weeks or months designing the agent is greatly reduced. It allows agent developers

to create any type of management agent quickly and easily. The user specifies the type of agent desired and its capabilities, and the code for the agent is generated.

3.1 Architectural Components

In this section, we describe each of the components of the architecture, as well as the information necessary to automate creation of that component. The components of a generic management agent are shown in Figure 2. When a request arrives, it is first given to the Coordinator, where it is parsed. It is then passed onto the Request Verification component, where it is verified. The request is then sent onto the appropriate component where it is executed, and any results are returned to the requester.

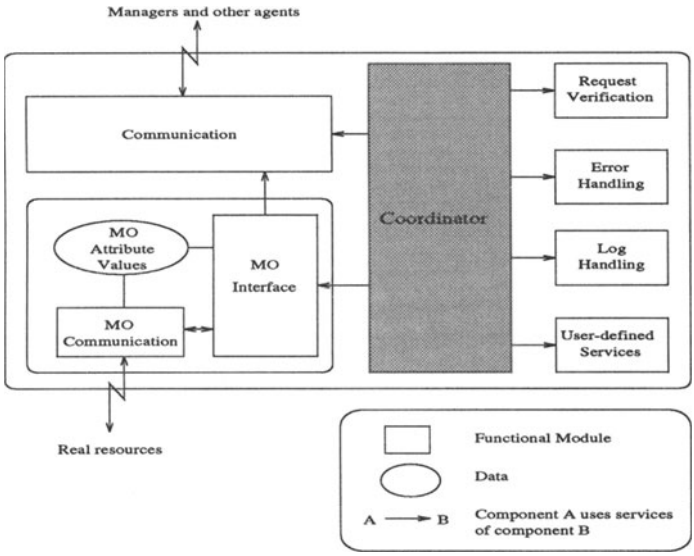

Figure 2: The Generic Agent Architecture

- **Coordinator:** The Coordinator is the central component of the agent. It parses requests and passes required information to the appropriate component. To be able to parse the requests, knowledge of the management protocol that is being used is required. The Coordinator must also know which of the services listed in Section 3.2 are provided by this agent, in order to be able to describe the agent to other management entities.
- **Communication:** The communication component provides communication services to other components, such as services to allow the agent to receive requests, return information to the requester, and forward notifications from managed objects to other entities. The management protocol to be used is all that this component needs to know.

- **Request Verification:** This component verifies each incoming request to make sure that the following conditions hold:
  - this agent supports the request,
  - any managed objects that are referenced are valid,
  - the requester is authorized to make a request, and
  - the requester has permission to perform the given request with the given managed objects.

  To check these conditions, this component must have knowledge of which agent services are supported, as well as which managed object classes can be managed. It does not, however, need to know which management protocol is being used.

- **Managed Object Interface:** This component contains the managed objects, which are abstractions of the managed resources. Each managed object "represents" a single resource for the purposes of managing it. Note that a managed resource may or may not be a physical resource. For example, a management system could have a managed object representing an application, which itself contains several managed objects each representing a different process which is part of that application. This component provides the interface for the agent to interact with the managed objects which, in turn, interact with the resources they represent.

  This component does not need to know the management protocol that is being used, since it does not have to "communicate" per se with managed objects. Managed objects may need to communicate with external resources, but the method or protocol that they use to do this communication is a decision made by the managed object developer, and is not relevant to the design of the agent. The only information that the agent developer must supply in order to create this component automatically is the list of management operations which should be supported.

- **Log Handling:** This component contains services that allow the agent to log information, such as management requests (get information, set information, creation of a managed object, *etc.*), the execution of dynamic services (see Section 3.2.3), or notifications received from managed objects. The Log Handling component can be entirely automated with no information from the user.

- **User-Defined Services:** This component stores the services that have been added to the agent, and provides a way for the agent to execute these requests. The execution could involve simply running the service and returning the result, starting the service as a background process, or even providing a multi-tasking environment in which the services run [4].

- **Error Handling:** This component hides some of the details about the management protocol from the rest of the agent. It translates internal error codes into a format recognizable by a manager, which can then diagnose and possibly correct the problem. Knowledge of the management protocol to be used by this agent is necessary for this component's creation.

An obvious advantage to having modular components is that each component is not dependent on the implementation details of the other components. For example, to automate the Error Handling

component, we only need knowledge of the management protocol. Whether or not the agent supports, for example, user-defined services or logging has no effect on the code for this component. This code reuse implies another advantage: increased reliability. The same Error Handling code is used in *all* agents with the same protocol regardless of other services offered and managed object classes supported. If this code is tested and found to be stable in a particular implementation, it can be assumed that the code will work in other implementations, since the implementation details of the other components do not affect this one. Once the code is stable, it can also be optimized, resulting in faster and more efficient agents.

3.2 Agent Service Interface

This section describes the operations that comprise the service interface. These operations represent the vast majority of the operations that management agents provide to other management entities, such as managers. There are, of course, exceptions: agents which provide specialized operations that are not listed here. However, using the User-defined operations described below, even these agents can fit into this architecture, with the specialized operations added as user-defined services.

There are four classes of operations offered: *self-description* operations that allow the agent to describe itself and its capabilities to other management entities; *common management* operations that are used to manipulate managed objects; *user-defined* operations that allow an agent's capabilities to be extended; and *logging* operations that allow the agent to log requests, notifications, and other management information. The detailed specification of the interfaces described below can be found in [11].

3.2.1 Self-Description Operations

Self-description operations can be used by managers, managed objects, or other agents to ask an agent for information about itself.

- **DescribeMyself:** Allows the agent to describe itself and its (built-in) capabilities.
- **GetMOList:** Lists the managed objects that are being managed. This operation basically traverses the tree of managed objects, and returns the list of objects. A parameter can be given that allows the requester to limit or control the traversal of the managed object tree.
- **ListPeriodicServices:** Allows a management entity to query an agent to find out which services have been scheduled to run as periodic services. A periodic service is a service which has been dynamically added and has been scheduled to be executed periodically (*e.g.*, every five minutes).
- **ListServices:** Allows a manager to query an agent to find out what services have been added.

3.2.2 Common Management Operations

Management operations can be used by managers and other agents to create and delete managed objects, perform actions on managed objects, and get and set attribute values within managed objects, and can also be used by managed objects to send notifications of events to managers.

- **Action:** Send an action command to one or more managed objects. Actions are defined within managed objects, and are operations that managed objects perform on themselves. For example, a file managed object may have actions called `rename`, `touch`, `remove`, or `copy`.
- **Create:** Creates a new managed object of the specified class, with the specified name, and with the specified attribute values (if present).
- **Delete:** Deletes the specified managed object(s).
- **ForwardNotification:** Allows a managed object to send a notification to a manager.
- **Get:** Returns the value of the specified attribute(s) from the specified managed object(s).
- **Set:** Sets the value of each of the specified attribute(s) in each of the specified managed object(s) to the specified value.

3.2.3 User-Defined Operations

User-defined operations allow the agent's functionality to be extended. "Services" can be statically or dynamically added to the agent, and can be executed at any time thereafter. These services can also be scheduled to be periodically executed.

- **AddNewService:** Dynamically adds new functionality to the agent. Each agent has three types of operations that it can perform: static operations (this list), static user-defined operations (code which is written by the user and is linked in with the agent at build-time), and dynamic user-defined operations (called "services" – user-written code that is added to the agent dynamically, at run-time). This operation allows a management entity to send a program to an agent, telling the agent to add the program to its list of services. That service is then accessible by any management entity, by using the ServiceHandle assigned to it by this operation.
- **ExecuteService:** Executes the specified (previously added) service once and returns the results.
- **StartPeriodicService:** Schedules the specified service to be executed at the beginning of each successive period of specified length. Any service that has been added using AddNewService can be used in a periodic service. If the service specified returns a value, the return value after each execution will be ignored. Note that the service is started and runs to completion each time it is executed; is it not "woken up" periodically.
- **StopPeriodicService:** De-schedules the specified periodic service.

3.2.4 Log Operations

Logging operations allow the agent to log information about events that have occurred or about operations that have been performed on managed objects [5].

- **DescribeLog:** This operations allows a management entity to query a log that has been started to find out its current state, size, etc.
- **LockLog:** Locks or unlocks the specified log. Locking a log allows records currently in the log to be read, but no new records can be added.
- **StartLog:** Create a new log and begin logging. When started, the log is both enabled and unlocked.
- **StopLog:** Stops the specified log. Once a particular log has been stopped, it cannot be restarted.

4 Prototype Implementation

As a proof of concept, we have developed a prototype tool that can automatically generate management agents (CMIP and SNMP) which possess the general agent architecture introduced in Section 3. In this section, we describe this prototype called the Management Agent Creation Tool (MACT) and how it is used to create management agents.

4.1 The Management Agent Creation Tool

MACT is a Motif-based tool that allows the user to specify the type of agent desired using a simple graphical user interface [11, 12]. The tool allows the user to enter the required information, ensures that the information entered is valid, and then generates the code for the specified agent.

As described in Section 3.1, the following information must be supplied by the user (an agent developer) in order to be able to generate the agent code:

- which managed object classes should be supported,
- which agent operations should be supported, and
- which management protocol will be used.

MACT makes it easy for the user to specify this information. When a user starts MACT, a main window requesting the user to select the management protocol (either CMIP or SNMP) is displayed. When the user selects CMIP, the user interface for generating CMIP agents, as shown in Figure 3, is displayed. The MACT user interface for generating SNMP agents is shown in Figure 4. When the user completes the input of the desired agent including the target platform (*e.g.*, Sun4 or AIX) and the create operation is requested, MACT generates all the source code including a Makefile into a directory. All the user has to do is to exit the tool and type `make` to generate an executable of the desired agent.

Currently, the OSIMIS implementation of the CMIP protocol and the ISODE implementation of the SNMP protocol are supported. We describe these in more detail below.

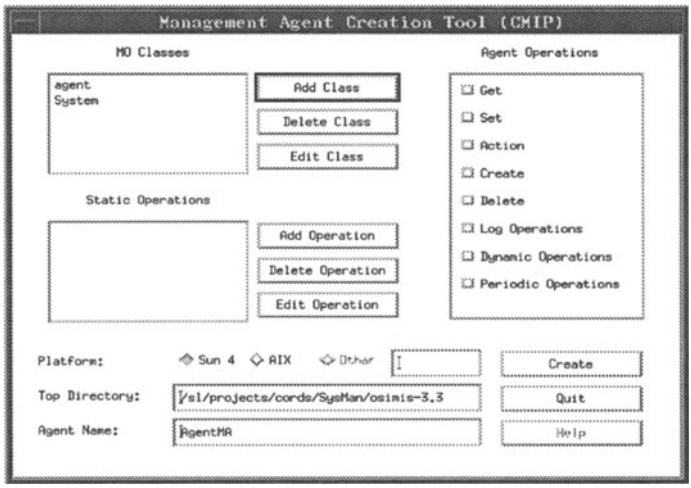

Figure 3: MACT User Interface for Creating CMIP Agents

4.2 OSI Management Agents

The OSI Management agent that MACT generates is based on OSIMIS (OSI Management Information Service) Version 3.3, which is an implementation of the OSI management framework, developed at University College London [10]. It was developed using the ISO Development Environment (ISODE), and provides a GDMO compiler, which generates C++ code for managed objects from specifications made in the GDMO language.

The architecture of the OSIMIS agent is somewhat less structured than the architecture we presented in Section 3.1. There is a class called Coordinator, but it provides the functionality of our Communication component. There is also a class called CMISAgent, which provides the functionality of our Coordinator. One method within the CMISAgent class acts as a (limited) request validation component, while another method is the error handling component. There are no user-defined service nor log handling components. The GDMO compiler provided with OSIMIS ensures that all managed objects have an identical interface, which provides the functionality of the Managed Object Interface component. Unfortunately, each component in the OSIMIS agent is highly dependent on the implementation details of the other components.

OSIMIS agents provide the same management operations as listed in Section 3.2.2. However, none of the other services listed in Section 3.2 are offered by OSIMIS agents. Logs are supported, but only for notifications, and they are driven by the managed objects, not the agent.

To validate our architecture, the OSIMIS agent was enhanced to include the other agent operations outlined in Section 3.2 that were not already present. A Log Handling component and operations were added, making the agent capable of logging requests without generating a notification. The log information is now generated by the agent, not by the managed object. User-defined service and self-description operations have also been added to the OSIMIS agent. Note that, as stated

in Section 3.1, the code to provide the log, user-defined service, and periodic service operations is protocol-independent; in fact, the same code is used for these operations in both the CMIP and SNMP agents.

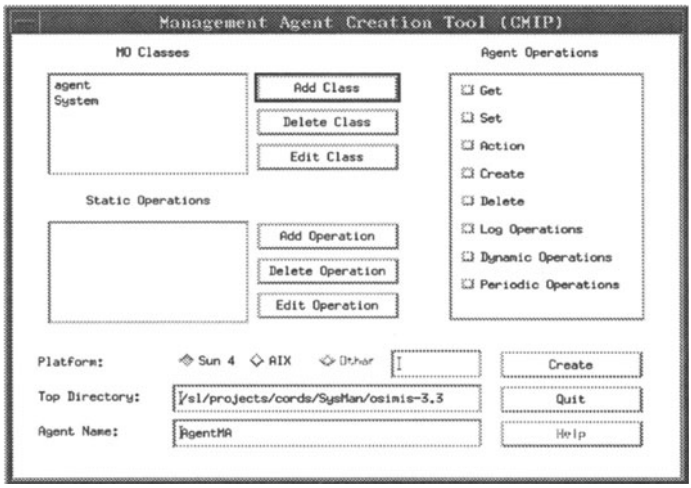

Figure 4: MACT User Interface for Creating SNMP Agents

4.3 SNMP Management Agents

The SNMP agent that MACT generates is based on the ISODE (ISO Development Environment) version 8.0. The SNMP agent was not designed or developed using an object-oriented methodology. As such, there is no clear distinction between components, and both error handling and request validation are done *ad hoc*. As in the OSIMIS agent, there are no user-defined service or log handling components. The Managed Object Interface consists of a list of C functions, each of which can get or set the values of specific groups of variables. It seems that every part of this agent is highly dependent on the implementation details of many other parts.

Nevertheless, we can identify functions and sections of code that act as the Coordinator and the communication component. It would not be difficult to write a single request validation routine and a single error handling routine, which would reduce the large amount of code repetition present within the agent.

The only operations that SNMP agents provide are Get, Set, Trap (for sending notifications), and Get-Next, which is used in the GetMOList operation. None of the other services listed in Section 3.2 are offered by SNMP agents. The SNMP agent has been enhanced to include other agent operations outlined in this paper that were not already present. Log Handling, User-Defined Service, and Self-Description operations have all been added to the SNMP agent. The Action, Create, and Delete operations have not been implemented since they are not supported in SNMP agents. The SNMP

standard specifies that variables are created only on agent initialization, and cannot be created afterward, nor can they be destroyed. SNMP also does not support actions on variables, and so the Action operation is also unnecessary.

5 Concluding Remarks

We have described the role and importance of management agents within management systems. We then outlined the requirements for generic management agents, and presented a general architecture for these agents. We have identified four kinds of basic services which are common to all agents, and the interfaces to each of those services. We have also outlined the information that is required from the agent developer in order to be able to create the desired agent.

We have developed a prototype tool called MACT that automates much of the development process of management agents. Using MACT will greatly reduce the time needed, and therefore the cost of creating management agents, and will eliminate the need for the agent developer to "reinvent the wheel". Because the code is reused in different agents, it is more robust than an *ad hoc* solution. One of the most important benefits of using MACT is that most agents will not require much (if any) code written by the agent developer. The only code that needs to be written is the code for the managed objects to access real resources being managed and the code for any user-defined routines, which can easily be added to the agent.

MACT has been sparingly used by our group members for developing various CMIP and SNMP management agents. We have used MACT to generate a number of management agents including UNIX system management agent, a generic distributed application management agent as well as a number of specific application management agents. Our generic management agent combined with MACT, in our opinion, provides an excellent framework for providing "extensible" agents.

We are in the process of enhancing the functionality of MACT. We hope to develop and add to it a managed object class library browsal and definition tool, which would allow the user to browse through the existing managed object classes, modify existing or define new managed object classes on the fly. We plan to develop and experiment with both the static and dynamic operations to extend the capabilities of agents for various purposes. We also plan to develop more management agents using MACT for network, system and application management.

References

[1] J. Case, M. Fedor, M. Schoffstall, and C. Davin. A Simple Network Management Protocol (SNMP). *Internet Request for Comments 1157*, May 1990.

[2] 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.

[3] Germán Goldszmidt. On Distributed System Management. In *Proceedings of the 1993 CAS Conference*, pages 637–647, Toronto, Canada, October 1993.

[4] Germán Goldszmidt, Shaula Yemini, and Yechiam Yemini. Network Management by Delegation - the MAD Approach. In *Proceedings of the 1991 CAS Conference*, pages 347–359, Toronto, Canada, October 1991.

[5] ISO. Information Technology – Open Systems Interconnection – System Management – Part 5: Event Report Management Function. *International Organization for Standardization, International Standard X.736*, November 1990.

[6] ISO. Information Technology – Open Systems Interconnection – Systems Management Overview. *International Organization for Standardization, International Standard X.701*, June 1991.

[7] ISO. Information Technology – Structure of Management Information – Part 4: Guidelines for the definition of managed objects. *International Organization for Standardization, International Standard X.722*, July 1991.

[8] Paul Miller. Bull's CMIP Agent Development Kit – A Platform for the Rapid Development of CMIP Agents & Objects. In *Proceedings of NOMS94*, Orlando, FL, February 1994.

[9] Oscar Newkerk, Miriam Amos Nihart, and Steven K. Wong. The Common Agent – A Multiprotocol Management Agent. *IEEE Journal on Selected Areass in Communications*, 11(9):1346–1352, December 1993.

[10] G. Pavlou, S.N. Bhatti, and G. Knight. The OSI Management Information Service User's Manual. Version 1.0, February 1993.

[11] G. S. Perrow. The Abstraction and Modelling of Management Agents. *MSc. Thesis*, Dept. of Computer Science, University of Western Ontario, London, Ontario, Canada, September 1994.

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

[13] M. Rose and K. McCloghrie. Structure and Identification of Management Information for TCP/IP-based Internets. *Internet Request for Comments 1155*, May 1990.

[14] Marshall T. Rose. *The Open Book: A Practical Perspective on OSI*. Prentice Hall, Englewood Cliffs, NJ, 1990.

[15] Marshall T. Rose. *The Simple Book: An Introduction to Internet Management, Second Edition*. Prentice Hall, Englewood Cliffs, NJ, 1994.

[16] M. Sylor and O. Tallman. Applying Network Management Standards to System Management; the Case for the Common Agent. In *Proceedings of the IEEE First International Workshop on Systems Management*, Los Angeles, CA, April 1993.

About the Authors

Graeme S. Perrow is currently working as a software engineer at Comnetix Computer Systems, Mississauga, Ontario. He received his BMath in Computer Science from the University of Waterloo in 1992 and his MSc in Computer Science from the University of Western Ontario in 1994. His research interests include network management, software engineering and information systems. He can be reached via electronic mail at `gperrow@comnetix.com`.

James W. Hong is a research associate and adjunct professor in the Department of Computer Science at the University of Western Ontario. He received his BSc and MSc from the University of Western Ontario in 1983 and 1985 respectively and his doctorate from the University of Waterloo in 1991. He is a member of the ACM and IEEE. His research interests include distributed computing, software engineering, systems and network management. He can be reached via electronic mail at `jwkhong@csd.uwo.ca`.

Hanan L. Lutfiyya is an assistant professor of Computer Science at the University of Western Ontario. She received her B.S. in computer science from Yarmouk University, Irbid, Jordan in 1985, her M.S. from the University of Iowa in 1987, and her doctorate from the University of Missouri-Rolla in 1992. She is a member of the ACM and IEEE. Her research interests include distributed computing, formal methods in software engineering and fault tolerance. She can be reached via electronic mail at `hanan@csd.uwo.ca`.

Michael A. Bauer is Chairman of the Department of Computer Science at the University of Western Ontario. He received his doctorate from the University of Toronto in 1978. He has been active in the Canadian and International groups working on the X.500 Standard. He is a member of the ACM and IEEE and is a member of the ACM Special Interest Group Board. His research interests include distributed computing, software engineering and computer system performance. He can be reached via electronic mail at `bauer@csd.uwo.ca`.