Supporting Dynamic Policy Change Using CORBA System Management Facilities

Stephen Howard, Hanan Lutfiyya, Michael Katchabaw and Michael Bauer
Department of Computer Science, The University of Western Ontario
London, Ontario, Canada N6A 5B7
email: {showard,hanan,katchab,bauer}@csd.uwo.ca

Abstract

Automation of management tasks is an effective counter-measure to the growing complexity of distributed systems. An increasingly popular view redefines the role of the management system to include automated validation and enforcement of policy. This research proposes an architecture for a policy-driven management system which can adapt dynamically to policy change. We show how this architecture can be implemented in a CORBA distributed object computing environment on top of the recently adopted System Management Common Management Facilities.

Keywords

Policy-driven management, CORBA management, distributed system management, distributed object computing

1 INTRODUCTION

New levels of automation are needed to address the growing complexity of distributed systems. We share the view that *policy-driven* management is an appropriate means to automate more of the management function. As an ideal, we believe that human involvement can be reduced to defining policy and roles and that intervention should be necessary only for critical failures. We are realistic in the sense that we recognise the infancy of the current state of research relative to that vision.

This paper proposes a basic architecture for a policy-driven management system which can support application, system and network management at a level of automation beyond that of today's management systems. We pursue the problem in a CORBA (OMG, 1995) environment, believing that CORBA is reflective of the direction in which distributed computing is being steered and thus, new challenges for management (particularly application-level management) will be found there. With its focus on object orientation, and a rich set of standard object services, CORBA presents the opportunity to explore new approaches to building management systems. A particular focus of our work is to investigate the suitability of the new OMG System

Integrated Network Management V (IM '97), San Diego, CA, May 12-16, 1997, chapter 39, pages 527-538

Management Common Management Facilities (X/Open, 1995) to support our architecture.

The remainder of the paper is organized as follows: In Section 2, we describe the general notion of policies and policy-driven management. Section 3 presents our architecture. In Section 4, we introduce the recently adopted OMG management services and summarize a few of the design issues we face in attempting to use these to support our architecture. Section 5 highlights some related work and Section 6 draws some conclusions and outlines our next steps.

2 POLICY-DRIVEN MANAGEMENT

A basic premise of this research is that the management system exists to uphold management policy. In practice, policy exists at many levels. Corporate policy makers often begin by defining high-level strategic policy, which is then refined into tactical or goal-oriented policy and then further refined into low-level activity-based policy. The policy refinement process will be difficult to automate without constraining the expression of higher-level policy in unnatural ways. Other research in this area which strives to encompass the policy refinement hierarchy within the management architecture generally allows for the storage of policies at various levels and then facilitates relationship tracking among these levels of refinement. The refinement process itself is left to human policy administrators.

Our research currently addresses only low-level policy, making the assumption that policies have been refined to the point where they can be expressed in a formal syntax. While our work requires basic policy definition services, the design of these is not our focus and therefore we have borrowed concepts from other excellent work in this area (Becker, Raabe and Twidle, 1993, Marriott, Mansouri-Samani and Sloman, 1994, Koch, Krell and Krämer, 1996). The remainder of this section outlines our view of policies; how they are specified and how they are organized.

2.1 Policies

In our architecture, low-level policies are treated as objects. A simple example is presented to illustrate some of the attributes included in a policy specification. The example policy defines a requirement to monitor CPU load in a distributed environment and react when an unusually high load is encountered.

```
POLICY reportHighCpuLoad {
  MODE obligation
  SUBJECT (...)/systemManager
  EVENT highCPU@(...)/*/systemMO
  ACTION $S->displayWarning
  EVENT-RECIPIENT (...)/eventLogger
  DESCRIPTION Report CPUs with extraordinarily high load
}
```

Integrated Network Management V (IM '97), San Diego, CA, May 12-16, 1997, chapter 39, pages 527-538

Each policy carries a name and a mode (either *obligation* or *authorization*). In the example, a management application called `systemManager` is the policy *subject* and is ultimately responsible for enforcing this policy. The subject is specified using a *domain expression* as described in Section 2.2.

The circumstance in which enforcement is required is defined by the *event* clause. The event expression in the example contains a single term identifying the *primitive event* `highCpuLoad` (discussed in greater detail in Section 2.3) and the set of managed objects from which that event may emanate. These managed objects are the *targets* of the policy and are specified using domain expressions. It is assumed that a managed object called `systemMO` exists on each host to collect CPU utilization (and other) information through operating system instrumentation. Note that "(...)" is not valid syntax but is used here in place of a full domain path. The "*" in the target expression is a wildard causing inclusion of any `systemMO` managed object below the current level in the domain hierarchy.

The enforcement actions to be taken by the subject are indicated in the *action* clause. In this example, when the `systemManager` receives a `highCpuLoad` event notification from any `systemMO` object, it invokes its own `displayWarning` method (the $S syntax indicates a method on the subject) perhaps to inform a human manager of the condition.

Event-recipient objects receive the same event notifications as the subject but do not carry out enforcing actions. In this example, an `eventLogger` object records the events in its log.

A description may be included with the policy to provide information for policy browsers and other policy administration applications.

2.2 Policy Domains

Policy domains (Becker, Raabe and Twidle, 1993, ISO/IEC JTC1/SC21/WG4, 1992, Sloman and Twidle, 1994) are used to group objects which share common management policy. The objects in a domain can be managed objects or other domains, allowing for the construction of domain graphs. An object can belong to multiple domains, in which case the domains are said to overlap.

Domains are used as the basis for object naming; that is, domains correspond to naming contexts and thus impose a structure and allocation scheme for the local naming of managed objects. *Domain expressions* are used in policy specifications to identify the policy's subject(s), target(s) and alternate event recipients. Domain expression resolution yields a set of object references.

2.3 Primitive Events

Primitive event specifications define significant circumstances and are bound to policies through the policy specification. In our research, we deal with three primitive event types. For *scheduled* events, the circumstance is time-based, thereby providing a mechanism for scheduling management tasks. *Initialization* events, which are associated with the instantiation of managed objects, provide a means to specify managed object initialization tasks. A great variety of other circumstances can be realized using *alarm* events which are triggered by changes in managed object state.

Like policies, primitive events are treated as objects. To illustrate some of the attributes of a primitive event, the following example defines the `highCpuLoad` event referred to in Section 2.1.

```
EVENT highCpuLoad {
  TRIGGER ALARM cpuLoad > 95
  ATTRIBUTES cpuLoad
  DESCRIPTION CPU load too high
}
```

Each primitive event has a unique name. The *trigger* identifies the event's type. For an alarm-type event, a predicate defines a condition based on managed object attributes. For a scheduled event, a time expression is given to define when the fixed, relative or repeating time when the event occurs. Initialization events require no extra information as they are always tied to object instantiation. In this example, the event is triggered if the managed object's `cpuLoad` attribute exceeds a threshold value of 95 percent. If so, the management system must generate an event report carrying the `cpuLoad` attribute value (as well as other standard information such as an identifier, event source and time stamp).

2.4 Compound Events

Within the `EVENT` attribute of the policy specification, it is possible to define *compound events* using pre-defined composition operators. We are currently investigating a small set of operators but have defined a mechanism which allows new operators to be added easily. Event composition semantics are complex and impose event synchronization and buffering requirements. A detailed discussion of this topic falls outside the scope of the paper. The following simple example demonstrates our intent.

```
(e1@!d1 and e2@d2) then e3@d3
```

It is here that assumed that e1, e2 and e3 are primitive events and that d1, d2 and d3 are domain expressions specifying the managed object(s) at which the events originate. The example requires that an e1 event from *all* managed objects within the d1 domain and an e2 event from *any* managed object in the d2 domain must precede an e3 event from *any* managed object in the d3 domain.

Integrated Network Management V (IM '97), San Diego, CA, May 12-16, 1997, chapter 39, pages 527-538

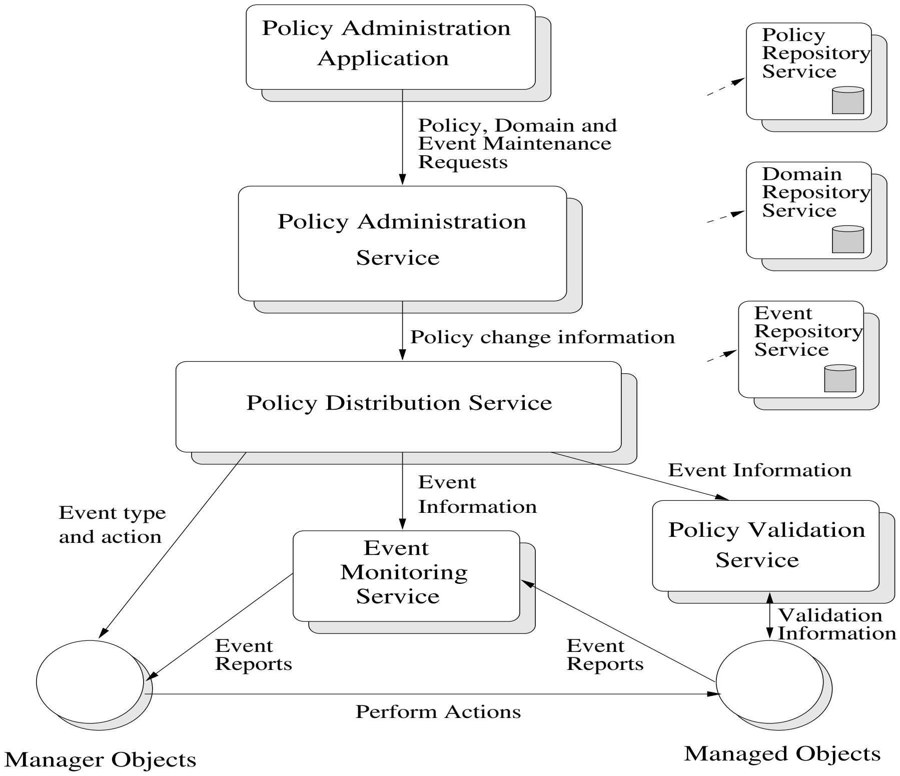

Figure 1 Management System Architecture

3 MANAGEMENT SYSTEM ARCHITECTURE

The management architecture we propose is shown in Figure 1. This architecture supports policy administration, policy violation detection, management task scheduling, managed object initialization, event filtering/forwarding and reactive control. A key aspect of the architecture is that it is designed to facilitate changes in policy at run-time. The management system is able to adapt dynamically to accommodate these changes. The architectural components are described below.

3.1 Managed Objects

Managed objects are used to model system resources. One managed object may encapsulate a single resource or many resources to provide an appropriate management view. For application-level management, the managed object may be an instrumented process. For device-level management, the managed object is probably a “stand-in” object which interacts in a private protocol with one or more devices to represent them to management. For system level management, the managed object interacts with operating system instrumentation. The management system is unaware of any interaction between managed objects and real resources.

In the example of Section 2.1, it is intended that the `systemMO` managed objects (of which there is one per host) are simple application level objects that collect CPU load data (and probably other system-level information) from their respective operating systems.

From a developer's perspective, managed objects inherit heavily from predefined managed object classes. This inheritance will provide basic manageability as well as resource class-specific characteristics and functionality. Common to all managed objects are attributes and methods which support basic object lifecycle operations, naming, domain association, collection, and so on.

3.2 Manager Objects

A policy's *subject* resolves to one or more manager objects to which responsibility for the policy is assigned. Event and action information from the policy must be distributed to these objects so the manager object knows when and how to enforce the policy (that is, what methods to invoke in what circumstances). Manager objects are also managed objects, allowing them to be managed by higher-level management applications.

3.3 Policy Administration Applications

Authorized administrators must have a way to add and remove policies, define events, build domain hierarchies, browse policy information, and so on. A policy administration application provides a user interface to facilitate this type of activity. It is conceivable that more than one type of policy administration application may exist. Policy administration applications use the operations of the Policy Administration Service (PAS) to interact with the management system.

3.4 Policy Administration Service (PAS)

The Policy Administration Service controls external access to policy, event and domain information. A client views the PAS as an aggregate of the three repositories (see Section 3.5). Internally, however, there is a requirement to control the use of these "raw" services through an addional layer of functionality.

At a given instant, the management system is configured for a particular set of policies, domains and events. Because such configurations are generated and adjusted automatically, information integrity is vital. As a result, the PAS must ensure that: domain expressions are resolvable, referenced primitive events really do exist, enforcement actions specify valid objects and method signatures, new policies do not conflict with existing ones, expired policies do not linger, and so on.

3.5 Repository Services

Three repositories provide persistent storage for policies and associated information. Each repository is treated as a separate service, each with a well-defined service interface. Although the PAS imposes consistency requirements across the repositories at a higher level, at the repository level each individual service operates independently of the others.

The **Policy Repository Service (PRS)** allows for the storage and retrieval of policies. A policy may exist in an inactive or active state (initially, a policy is inactive). A policy can only be removed after it has been deactivated. The PRS allows clients to add or remove policies, activate or deactivate existing policies, retrieve and/or change the attributes of policies and retrieve policies matching certain criteria.

The **Domain Repository Service (DRS)** facilitates the organization of managed objects into policy domains. New domains can be defined and positioned in the domain hierarchy and existing (empty) domains can be removed. Managed object *types* can be added to, or removed from existing domains. Adding and removing managed objects themselves are operations on the managed objects, not on the DRS. Policies are not added to domains; the association between a policy and a domain is determined by the policy's target (which is a domain expression). The DRS allows clients to query a domain for the managed objects it contains, its subdomains and its parent domain. The DRS is capable of resolving a domain expression to yield a set of managed object references.

The **Event Repository Service (ERS)** stores primitive events independently of the policies which use them. This provides some opportunity for reuse of an event across multiple policies. The ERS provides a single interface which allows clients to add, remove or alter event specifications, to inquire about the attributes of a particular event, or to retrieve events matching certain criteria.

3.6 Policy Distribution Service (PDS)

The Policy Distribution Service analyzes policy changes and then initiates and coordinates adaptive restructuring. Manager objects must be informed of policy responsibility assignments, and about the events and enforcing actions associated with new policies. The Policy Validation Service requires event triggering conditions to update the appropriate initializer, validator, and scheduler objects. Event forwarding and filtering information must be passed to the Event Monitoring Service so it can maintain its event channel graphs.

Integrated Network Management V (IM '97), San Diego, CA, May 12-16, 1997, chapter 39, pages 527-538

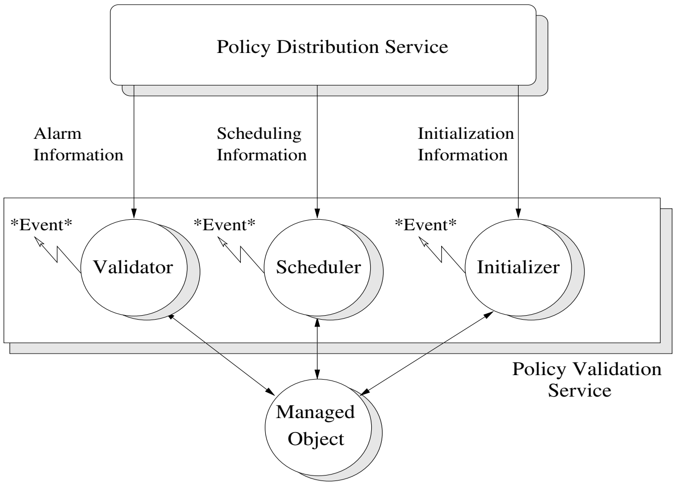

Figure 2 Policy Validation Architecture

3.7 Policy Validation Service (PVS)

Each policy includes an event expression describing the situation for which enforcement is necessary. Primitive events in that expression contain triggering conditions which become the basis for validation. The three types of triggers (see Section 2.3) call for three sub-components of the PVS as shown in Figure 2. Each of these is a generic validator which accepts rules from the PDS and then validates managed object behaviour against these rules, generating event reports upon validation failure. The roles of these components are described briefly below.

1. An *initializer* generates an event report when a managed object is created. This provides an opportunity for policy subjects to carry out initialization tasks on the object.
2. A *validator* holds validation rules from alarm-type event specifications. For example, consider the `systemMO` object with attribute `cpuLoad`. When this (or any) management attribute changes, the managed object invokes a generic `validate` method on its validation object(s) (there could be more than one if the managed object belongs to multiple domains). This method validates the change against currently held rules involving this attribute (in this case `cpuLoad` $>$ `95`).
3. A *scheduler* handles schedule-type events and maintains a queue of scheduled times when event reports are to be generated. The content of these reports is dictated by the event specification.

3.8 Event Monitoring Service (EMS)

Conceptually, events originate at the managed object level (actually with initializer, validator and scheduler objects) and are reported to management applications. In our work, policy and event specifications must contain sufficient information to allow the system to establish event routing paths and apply necessary filters automatically. The Event Monitoring Service (EMS) accomplishes this using graphs of CORBA event channels. With information received from the PDS about new, deleted or changed event specifications, the EMS can restructure its event channel graphs accordingly. Event communication is based strictly on asynchronous, push-style event flow.

Any object in the system which can generate or recieve events must contain methods through which the EMS can control the object's registration with event channels. In the example of Section 2.1, the EMS would ask the `systemManager` object to register as a consumer to a particular event channel which has been set up to buffer and propagate `highCpuLoad` events. A similar request would be made on the managed object end (the mechanics of this are oversimplified here for the sake of brevity).

4 DESIGN ISSUES

Included in the OMG *Common Facilities Architecture* (OMG, 1994) are basic requirements for *System Management Common Facilities*. OMG has recently adopted an initial set of services based on a proposal from X/Open (X/Open, 1995). The major ones are outlined here (detailed coverage is beyond the scope of this paper):

- The *Managed Sets Service* is a simple collection-like service which allows managed objects to be grouped.
- The *Instance Management Service* is a specialization of the CORBA Lifecycle Service. An *Instance Manager* acts both as a factory for a single type of managed objects and as a managed set of these objects. It can report the interface definition of its supported type and can have associated with it *Initialization Policy Objects* and *Validation Policy Objects*. A *Library* object is used to collect, create and locate instance managers.
- The *Policy Driven Base Service* provides the interface for managed objects to be managed by instance managers and to be associated with *policy regions* (domains).
- The *Policy Management Service* is essentially a domain service through which policies and managed objects are bound.

The standards assume that policies are hard-coded methods in the validation policy object which are invoked from the managed object at points where state changes must be verified. In our architecture, policies are objects. Although we maintain the concept and referential relationships of the validation and initialization objects, we redefine their roles to that of generic processors that can accept rules at runtime from the Policy Distribution Service. This way, all validator objects can be instantiated from a single class (and all initializer objects from another). A generic *validate* method is

Integrated Network Management V (IM '97), San Diego, CA, May 12-16, 1997,
chapter 39, pages 527-538

invoked by the managed object (an attribute-value pair is passed to define the state change). We add equally generic scheduler objects but at the managed object level (rather than being assocatiated with the instance manager).

Managed objects in our implementation will inherit our own *managedObject* interface which will, in turn, inherit the CORBA *PolicyDrivenBase* interface. This will provide managed objects with all the management capability defined in Section 3.1.

Our Event Monitoring Service builds its event channel graphs (described in Section 3.8) using standard CORBA event channels (the implementation must support filtering rules).

Much of the functionality of our Domain Repository Service can be realized by "wrapping" the *PolicyRegionsInstanceManager* defined within the CORBA Policy Management Service.

Beyond the standard services and the "retrofits" described, we add policy administration and distribution services, repository services for policies and events, and applications for policy administration.

5 RELATED WORK

There is a good base of research dealing with categorization, refinement, specification and organization of management policy (Becker, Raabe and Twidle, 1993, Marriott, Mansouri-Samani and Sloman, 1994, Moffett, 1994, Wies, 1995, Sloman and Twidle, 1994).

We have found two other groups who share our pursuit of a policy-driven management system which adapts dynamically to changes in policy. Researchers at Imperial College continue to play a lead role in policy-driven management in general, and their recent work (Marriott and Sloman, 1996) extends into dynamimic policy distribution and into the CORBA environment. While we share goals, our approach differs in how policy distribution, validation and event monitoring are realized. Koch *et al.* (Koch, Krell and Krämer, 1996) also propose an architecture similar to ours. They are beginning to explore policies as CORBA objects and are investigating approaches to policy distribution. They place more emphasis on the policy hierarchy issue than we do and, again, our approach to such things as event monitoring and policy distribution are quite different.

X/Open, the consortium responsible for the CORBA System Management Common Management Facilities (X/Open, 1995), will continue to pursue a richer set of management services for CORBA. While the available services represent an important base for management application developers, the level of automation we pursue exceeds what these services deliver. Our attempt to employ these services as a foundation seems to be novel.

The Joint X/Open/NM Forum Inter-Domain Management (JIDM) Taskforce (Soukouti, 1995) is addressing the inevitible coexistence of distributed computing environments and the desire for management systems to extend beyond their native domain. Their work includes algorithms for translating specifications among CORBA IDL, SNMP MIB and OSI GDMO. They have also proposed gateway solutions to

Integrated Network Management V (IM '97), San Diego, CA, May 12-16, 1997, chapter 39, pages 527-538

inter-domain scenarios such as using OSI management for CORBA objects. While the value of this work is readily apparent, our approach is based on a "purist" view in which the management model is more closely coupled with the OMG architecture.

6 CONCLUSIONS AND FUTURE WORK

In this paper we have proposed a general architecture for a policy-driven management system in a distributed object computing environment. A key requirement for the management system is that it be able to adapt dynamically to changing policy. The CORBA environment has been chosen as the target for an implemetation of our architecture. Of particular interest in the implementation is whether the recently adopted system management services can be used. In this paper we have mapped out a strategy for doing this.

The implementation is ongoing. Because the CORBA management facilities are new, and no vendor products currently implement them, we must build prototype versions of these in addition to our own management services. We are currently constructing our prototype management system on OS/2 using IBM's implementation of CORBA (SOMobjects 3.0). We anticipate several rounds of architecture and prototype refinement before we can begin to address more complex (and higher-level) management issues.

REFERENCES

[1] Becker, K., Raabe, U., Sloman, M. and Twidle, K. (editors) (1993) *Domain and Policy Service Specification: IDSM Deliverable D6, SysMan Deliverable MA2V2*.

[2] ISO/IEC JTC1/SC21/WG4 (1992) *Management Domains Architecture* (Working Draft).

[3] Koch, T., and Krell, C., and Krämer, B. (1996) Policy Definition Language for Automated Management of Distributed Systems. *Proceedings of the IEEE Second International Workshop on Systems Management*, Toronto, Canada.

[4] Marriott, D., Mansouri-Samani, M. and Sloman, M. (1995) Specification of Management Policies. *Proceedings of the Fifth IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM '94)*, Toulouse, France.

[5] Marriott, D. and Sloman, M. (1996) Implementation of a Management Agent for Interpreting Obligation Policy. *Proceedings of the Seventh IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM '96)*, L'Aquila, Italy.

[6] Moffett, J. (1994) Specification of Management Policies and Discretionary Access Control, in *Network and Distributed Systems Management* (ed. M.S. Sloman). Addison-Wesley.

Integrated Network Management V (IM '97), San Diego, CA, May 12-16, 1997, chapter 39, pages 527-538

[7] OMG (1994) *Common Facilities Architecture* (OMG Document Number 94.11.9).

[8] OMG (1995) *The Common Object Request Broker: Architecture and Specification (Revision 2.0)* (OMG Document Number PTC/96.03.04).

[9] Sloman, M. and Twidle, K. (1994) Domains: A Framework for Structuring Management Policy, in *Network and Distributed Systems Management* (ed. M.S. Sloman). Addison-Wesley.

[10] Soukouti, N. (1995) *Toward Managing CORBA Objects via OSI Network Management Mechanisms* (Draft Paper).

[11] Wies, R. (1995) Using a Classification of Management Policies for Policy Specification and Policy Transformation. *Proceedings of the Fourth International Symposium on Integrated Network Management (ISINM '95)*. Santa Barbara, California.

[12] X/Open (1995) *Systems Management: Common Management Facilities, Volume 1* (Preliminary Specification).

ACKNOWLEDGEMENTS

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

ABOUT THE AUTHORS

Stephen L. Howard is a Ph.D. student in the Department of Computer Science at the University of Western Ontario.
Hanan L. Lutfiyya is an Assistant Professor in the Department of Computer Science at the University of Western Ontario. Dr. Lutfiyya received a Ph.D. in computer science from the University of Missouri at Rolla.
Michael J. Katchabaw is a Ph.D. student in the Department of Computer Science at the University of Western Ontario.
Michael A. Bauer is a Professor in the Department of Computer Science and Senior Director of Information Technology Services at the University of Western Ontario. Dr. Bauer received a Ph.D. in computer science from the University of Toronto.

Integrated Network Management V (IM '97), San Diego, CA, May 12-16, 1997, chapter 39, pages 527-538