A Management Information Repository for Distributed Applications Management[1]

November 1995

External Technical Report

ISSN 0836-227-95-393

Patrick Martin
Dept. of Computing and Information Science
Queen’s University at Kingston
Kingston, Ontario
Canada K7L 3N6
office: (613) 545-6063
fax: (613) 545-6513
(*martin@qucis.queensu.ca*)

Abstract

The Management of Distributed Applications and Systems (MANDAS) project addresses problems arising in the management of distributed applications. Specifically, we are studying the areas of configuration management, fault management, performance management, and application metrics and modeling. We are also investigating the tools, techniques and services needed to support the above management applications. The MANDAS *Management Information Repository* (MIR) provides database support for the management applications and supports their integration into a single management environment.

In this paper we examine the problem of distributed applications management to extract the requirements for a MIR. Based on the requirements, we present an information model for distributed applications management and outline a prototype MIR developed for the MANDAS project.

Keywords: distributed applications management, information modeling, repositories.

[1] The author thanks IBM Canada Ltd. and NSERC for their support of this research.

1. Introduction

Distributed computing systems typically consist of large numbers of heterogeneous computing devices connected by communication networks, various operating system resources and services, and user applications running on them. These resources and applications are becoming indispensable to many enterprises, but as distributed systems become larger and more complex, more things can go wrong, potentially interrupting or crippling critical operations. Thus management support is often cited by end users as the single most important aspect of a distributed system. Management issues at the network and systems levels have received a great deal of attention but the same thing cannot be said for the application level.

The objective of the Management of Distributed Applications and Systems (MANDAS) project is to address problems arising in the management of distributed applications. We take management to include configuration management, fault management, performance management and application metrics and modeling. More specifically, we include within applications management the following [12]:

1. The software tools and techniques that designers of distributed applications and system administrators use, called *management applications*, to ensure the ongoing and effective design and operation of the systems and applications.
2. The data and information about the applications and systems required by the management applications, and the means of collecting, storing and maintaining them.
3. The tools, techniques and services needed to support the above.

In order to integrate the management applications and manage the data they require, we see a logically centralized database, which we call the *Management Information Repository* (MIR), at the heart of the system. The development of a MIR is the focus of our part of the MANDAS project and the subject of the paper.

The remainder of the paper is structured as follows. Section 2 discusses the MANDAS project and its view of distributed applications management. Section 3 outlines data management requirements for distributed applications management. Section 4 describes the information model used by the MIR and Section 5 describes a prototype MIR developed for the MANDAS project. Section 6 discusses related work and Section 7 summarizes the paper.

2. Distributed Applications Management

MANDAS considers distributed applications that rely on the support of middleware, such as DCE [8] and CORBA [7], for communication between software components. Midware environments make it easier to build distributed applications by providing a bridge between heterogeneous systems in the form of common interface description techniques and communication paradigms. The midware can hide many of the complexities of distributed systems through features such as location transparency and data format translation. Application software that relies on transparencies is more robust with respect to evolving environments. Transparencies, however, can make it more difficult to understand why distributed applications and their systems behave the way they do. Midware also poses performance costs that need to be understood.

Our approach towards characterizing application behaviour starts at the midware and continues upwards into the application domain. When it is appropriate, application behaviour is then correlated with other information such as operating system and communication network resource consumption. For example, Rolia [11] describes the information that needs to be collected or deduced to support the performance characterization and modeling of distributed applications. The services provided by each application process must be characterized separately according to properties such as the average CPU time needed by a process to complete a specific operation and the operation’s average number of requests for service from other servers’ operations. Monitoring systems for midware environments should collect this information and make it available to more general system monitors. In our architecture [1] the system level monitor interacts directly with management applications and with the MIR. SNMP and CMIP [13] have been chosen to support our system level monitoring infrastructure.

Figure 1 shows the management architecture we have adopted to support distributed applications management [1]. The *management applications* access the available *management services* through a common *management service interface*. The management services in turn interact with the *management agents* through various protocols, for example SNMP or CMIP. Each management agent is responsible for some collection of *managed objects*. The Management Information Repository provides services to the three component subsystems as well as providing a common interface to the set of management agents.

3. Data Requirements

To provide database support for distributed applications management it is necessary to understand the requirements of the management applications and the management services that will use the database. We derive these requirements from the properties of the types of data used in distributed applications management, including its physical properties and its semantic properties, and the characteristics of the types of accesses to the data. We distinguish three types of data: observation data, management data and control data. Haritsa et. al. [3] provide a similar

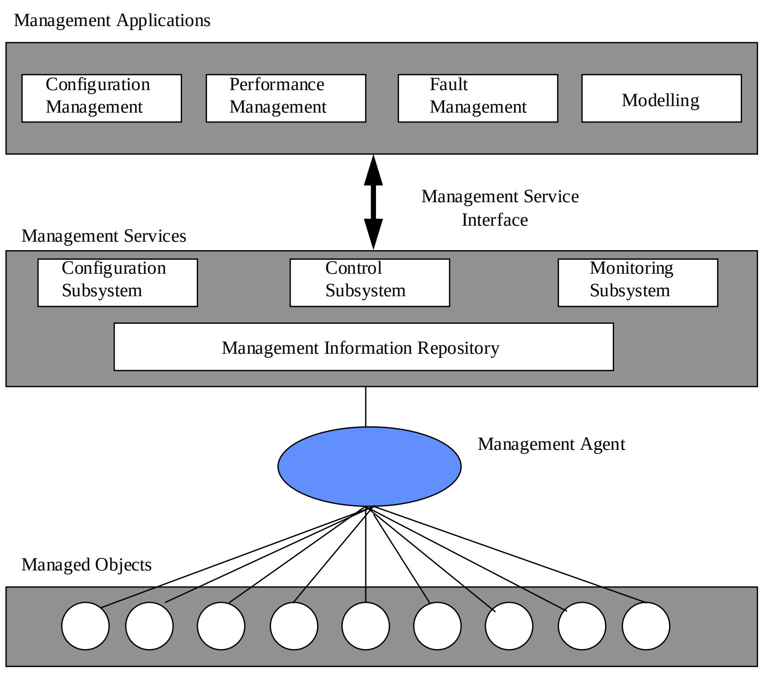

Figure 1: MANDAS Management Architecture

classification for network management data.

3.1 Observation Data

The observation data is the raw information that is received from the monitoring processes and includes a variety of variables which depend upon the managed object. For example, application level variables include process up time and the number and type of each request made from one process to another; midware variables include server queue lengths, RPC invocations and RPC response times, and resource-level variables include memory and CPU usage.

Observation data provides the primary input for the performance management, modeling and fault management tasks. It represents the current state of an application in terms of resource usage and quality of service provided to users. During normal execution, observation data arrives from a monitoring process at a regular frequency but under fault conditions data may be generated at a higher rate than normal. Another possible situation is where observation data only arrives when an extraordinary event occurs or when it is requested by a management task.

Two distinct groups of transactions, “updates” and “reads”, access the observation data. The updates are performed by the monitoring processes. They are typical database updates in that the new value is independent of the current value, but atypical in that they will happen in real-time. The real-time characteristic of updates means that the overhead incurred by an update must be minimized.

Communication overhead and contention for MIR resources can be minimized by physically distributing the observation data so that monitoring processes write to local, possibly heterogeneous, databases which are under the umbrella of the MIR. This distributed database approach also enhances the scalability of the MIR.

Transaction management overhead can also be minimized. Since an update typically accesses only one portion of the database and would not interfere with any other update explicit concurrency control among updates is not required . Similarly, if observation data is stored in versions then read transactions can access the data they want without conflicting with any update transactions.

3.2 Management Data

Management data is composed of “static” information such as an application’s configuration and the network topology. This data provides the primary input for the configuration management and is used by other management applications in locating, requesting and interpreting observation data. Management data can be both read and written by the operator(s) or by MIR control processes. For example, changing the configuration of an application is usually initiated by an operator but system migration of a process would trigger an update by control processes. Most of the data is stored at system initialization time and changed at a moderate rate consistent with classical DBMS applications. Since it is possible that multiple transactions may access the same data at the same time, standard database transaction management would be used.

Management data must represent the structures and relationships of a variety of management entities and concepts. Descriptions of resource level entities, such as those provided by the SNMP and the CMIP SMIs (Structure of Management Information) [13], must be maintained as well as descriptions of midware services and application level entities like programs and processes.

3.3 Control Data

Control data captures the current setting of application and system tuning parameters such as the maximum number of requests to a server. The process for changing an existing set of control settings is either initiated by the operator or triggered automatically as a function of the information contained in the sensor data. In either case, no more than one process should be able to update a set of control variables at one time otherwise control setting would be at the mercy of the transaction processing order. Therefore, concurrency control is not explicitly required for control data. It is required implicitly, however, as part of ensuring the atomicity of updates to set of related control variables. As with observation data, it may be necessary to maintain a history of the control data however these changes will occur at a moderate rate relative to the observation data.

4. Information Modeling

4.1 Modeling Concepts

An important aspect of an information model for the MIR is the ability to represent, and to relate, the different types of data objects relevant to distributed applications management. For example, some relationships exist because collecting observation data about objects at the application level (e.g. processes) involves collecting data at the resource level (e.g. CPU) of the system. That is observations about higher level objects are *derived from* observations of lower layer objects. Distributed applications management must therefore, to some degree, encompass systems management and network management concepts.

The information model must also capture the structure of the management model, that is, it should describe managers, management agents and managed objects. The managed objects consist of the application components and resources used by those components. Management agents provide views of managed objects to the managers. Managers provide the interface to the management applications.

Objects may undergo changes during operation, for example an application process may be moved from one site to another. Performance management requires that a history of an object be maintained so that the effectiveness of the changes can be evaluated. This requirement implies that versions must be part of the model.

We describe a distributed application in terms of a set of concepts which include the engineering structures of the Reference Model for Open Distributed Processing (RM-ODP) [10]. This allows us to model the application in terms of its individual components such as objects, operating system processes and nodes. The structure provides enough abstraction to represent applications built using midware such as DCE or OMG's CORBA.

The engineering structures of the RM-ODP are defined as follows:

- **Basic Engineering Objects** (BEOs) are the building blocks of the application. They contain the procedures or *methods* that perform the processing of the system.

- **Clusters** are groupings of related objects. They are an abstraction used in reconfiguring an application.
- **Capsules** are the operating system processes required by the application.
- **Nodes** are the stations in a distributed environment.

The relationships among these structures are shown in Figure 2.

In addition, we also define the following concepts:

- **Applications** are the entities to be managed. They are described by the set of *requests*, or transactions, processed by the application and by the set of component clusters.
- **Application Instances** are run-time instances of applications.
- **Configurations** define an assignment of capsules to nodes for an application instance.

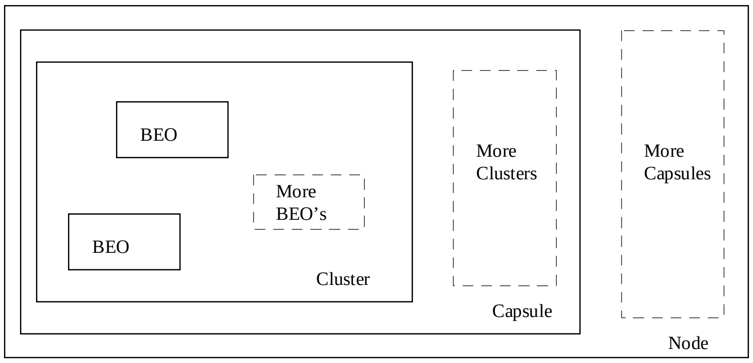

Figure 2: Engineering Structures of RM-ODP

4.2 Modeling Constructs

We use a structurally object-oriented model, based on the Telos language[6], to define the structures of our information model. We chose an object-oriented model for two main reasons:

1. the entities to be modeled may have complex structures and relationships;
2. there will be a large number of similar entities, for example processes, which may be grouped into classes.

The model uses the following constructs:

- *object*: An object is an identifiable collection of attribute values which represents an entity or component such as a process, management agent or resource. Every object has a unique object identifier (OID) or name.
- *class*: A class is a collection of objects that share common properties. An object is created as an *instance of* a particular class. Classes are related via the *is-a* relationship.

- *attribute*: An attribute is a particular property of an object. Each attribute has a *type* which may be *primitive type*, such as string, integer or real, or a user-defined class. In the latter case, the value for the attribute is a *reference* to an instance of that class. Attributes may be *single-valued* or *multi-valued*. The attributes defined for a class are grouped into *categories* where a category is related to a particular aspect of an object. For example, management objects have one which describes the management interface of the object and one category which describes the state of the object. In our Telos implementation, attribute categories are implemented through metaclasses.

The current class hierarchy for the information model is shown in Figure 3. The root of the hierarchy is the class *MIObject*. The main subtrees correspond to *ManagedObjects* with subtrees for the two types of objects - *Resources* and *ApplicationComponents*, *ManagementAgents* and application source code objects (*ASObject*). The class *VersionedMIObject* implements versions in its subclasses.

We present several example class definitions to illustrate the features of the model. The class *MIObject* is defined as

```
class MIObject
with
    State
        name: String;
        description: String
 end
```

It contains the single attribute category *State* and defines the two properties *name* and *description* which are inherited by all objects. The class *ManagedObject*, defined as

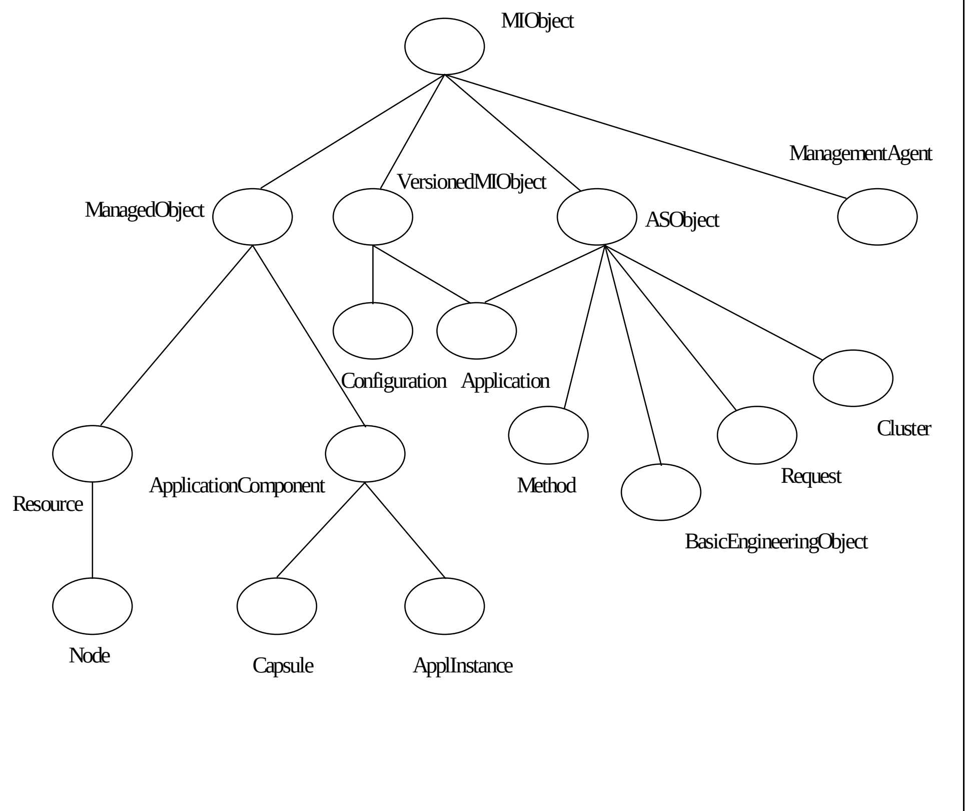

Figure 3: Information Model Class Hierarchy

```
class ManagedObject
isa MIObject
with
    ObservationData
end
```

is a specialization of *MIObject* and the ancestor class for all instances of managed objects. It introduces the attribute category *ObservationData* which is used by all subclasses to define the observation data related to a managed object. The class *ApplicationComponent*, defined as

```
class ApplicationComponent
isa ManagedObject
end
```

groups the collection of managed objects that are components of a distributed application.

There are subclasses of *ApplicationComponent*, namely *ApplInstance* and *Capsule*. We view an application as composed of a set of interacting processes or capsules and possibly instances of another application. For example, a financial application may use an independent database application. We assume for now that applications do not wish to manage objects beyond the level of detail of capsules. The definition of the *Capsule* class is as follows:

```
class Capsule                           (* process *)
isa ApplicationComponent
 with
    State
        capID: Integer;                 (* process id *)
        portID: Integer;                (* port number for appl communication *)
        upTime: Integer;                (* time process was started *)
        operStatus: String;             (* operational status of process *)
        residentHost: Node;             (* current process location  *)
        components: {Cluster};
        exposedMethods: {Method};
    ObservationData
        cpuUsage: DataFile;             (* data for CPU use (system and user) *)
        memoryUsage: DataFile;          (* data for memory use *)
        diskIO: DataFile                (* data for disk use *)
end
```

Properties in the *State* of a capsule define relationships with several other types of objects: the *Node* on which the capsule is running, the set of *Clusters* that make up the capsule and the set of *Methods* exposed at the interface to the capsule (i.e. available to other capsules). Relationships are represented as links or references in an object-oriented model like the one used here. Capsules also contain links to the observation data that is collected about them by the management agents. The three pieces of information, *cpuUsage*, *memoryUsage* and *diskIO*, are collected and maintained in data files outside of the MIR. Each of the data files is described in the MIR and represented as an object of class *DataFile*.

5. MIR Prototype Design

The structure of the MIR prototype is shown in Figure 4. It is a basic client-server system. The *MIR Server* has two components: the *Telos Repository* and the *MIR Server Interface*. The Telos Repository [2] provides the back-end database for the MIR. The MIR Server Interface accepts requests from MIR Clients and translates them into requests to the Telos Repository. Communication between the Telos Repository and the MIR Server Interface is via the Telos

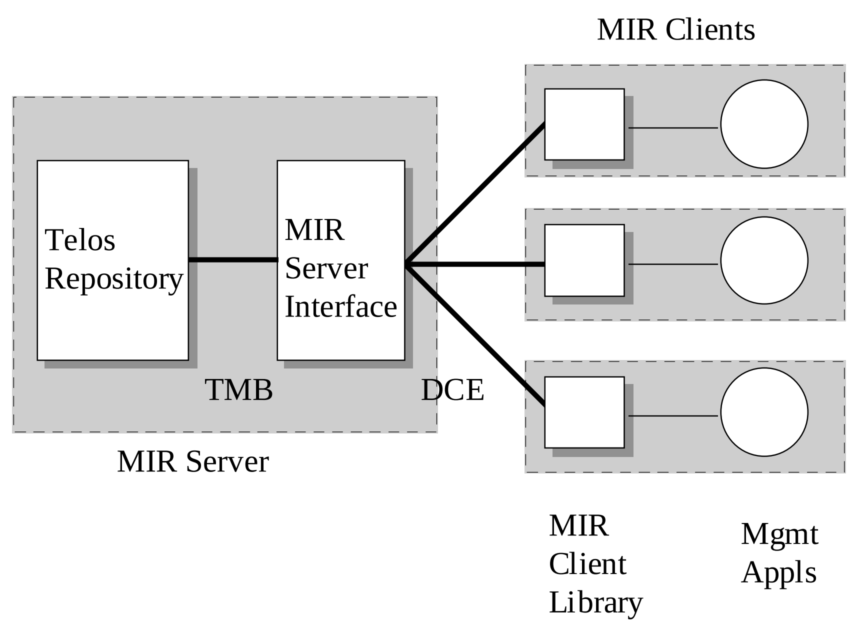

Figure 4: MIR Prototype

Message Bus (TMB). Communication between the MIR Clients and the MIR Server Interface is via DCE RPCs.

The Telos Repository has limited query capabilities. Since management applications are likely to need to search for objects based on attribute values as well as by browsing links, we provide a sophisticated filtering mechanism in the MIR Server Interface. Data may be retrieved from Telos and then filtered according to client-specified conditions before it is returned to the client. The Telos Repository also handles requests in a serial manner. The MIR Server Interface is implemented with DCE threads and interacts with multiple clients at one time. Concurrent client requests are buffered in the MIR Server Interface and passed serially to the Telos Repository.

A *MIR Client* has two components: the *MIR Client Library* and a *Management Application.* The MIR Client Library is a collection of functions to allow interaction with the MIR Server. It presents management applications with a view of the MIR which is consistent with the MANDAS Information Model. It provides functions to perform the following:

- connect to, and disconnect from, the MIR;
- create, modify and retrieve MIR schema class definitions;
- create, modify and retrieve MIR objects;
- define query filters and issue queries.

Observation data is not stored in the MIR for performance reasons. The MIR is used to store the management data, the control data and descriptions of the observation data. A management application which needs to access observation data first accesses the MIR to obtain information on the observation data such as its location, how it is stored (e.g. file, relational database) and how it is structured. The client then interacts directly with the observation data source.

6. Related Work

While there has not been previous work on distributed applications management, the role of databases in network and systems management has been studied. Haritsa et. al. describe MANDATE (Managing Networks using Database Technology), a proposed MIB (Management

Information Base) to support network management [3]. Their approach is to have operators interact solely with the MIB - the database *embodies* the network for the operator. Similar to our work, the MIB, like the MIR, is the focal point for integration of the management applications. MANDATE also proposes an object-oriented model to support management. Implementation in MANDATE is client-server with sophisticated client caching while our implementation is based on distributed databases.

The work of Wolfson et. al. [14] has a different focus than either our work or MANDATE. They propose a model of network management actions as data manipulation operations and provide an SQL-like language for specifying network management functions. They also discuss the use of database triggers as a mechanism for automatically initiating management events.

Finally, Hong et. al. [5] investigate the use of an X.500 directory system as a repository for systems management information. The X.500 system has the advantages that it is a distributed database and it is a widely available standard. However, X.500 is intended as a white pages system and not as a general DBMS. There are questions about its general modeling capabilities and its performance.

7. Summary

Distributed systems management is crucial to the success of future distributed systems and is an interesting application area for database systems research. We have introduced the MANDAS project in distributed applications management and explained the role played by the Management Information Repository in the integration and support of a collection of management applications. We outlined the requirements for the MIR, based on our view of distributed applications management, and discussed an information model and prototype implementation of the MIR.

We are currently integrating the MIR with components provided by other MANDAS groups including a system for generating and assigning management agents [9] and a performance modeling management application [4]. We also plan to explore a number of extensions to the MIR. We will examine how to incorporate versions and triggers into the model and the implementation. We will also study the relationships among the MANDAS information model and management models such as SNMP and CMIP [13] from network and system management.

Acknowledgements

The author would like to thank the members of the Database Systems Group at Queen’s University who contributed to the development of the MIR: Wendy Powley, Dave Barrowman, Peter Zion, Yichun Ding, Daniel Boudreault and J.P. LeBlanc.

References

[1] M. Bauer, P. Finnigan, J. Hong, J. Rolia, T. Teory and G. Winters. Reference Architecture for Integrated Distributed Sysyems Management. *IBM Systems Journal 33(3*), pp. 426-444, 1994.

[2] E. Buss, R. De Mori, M. Gentleman, J. Henshaw, H. Johnson, K. Kontogiannis, E. Merlo, H. Muller, J. Mylopoulos, S. Paul, A. Prakash, M. Stanley, S. Tilley, J. Troster and K. Wong. Investigating Reverse Engineering Technolgies for the CAS Program Understanding Project. *IBM Systems Journal 33(3)* , pp. 477-500, 1994.

[3] J. Haritsa, M. Ball, N. Roussopoulos, A. Datta and J. Baras. MANDATE: Managing Networks Using DAtabase TEchnology. *IEEE Journal on Selected Areas of Communication 11(9)* , pp. 1360-1372, Dec. 1993.

[4] G. Hills, J. Rolia and G. Serazzi. Performance Engineering of Distributed Software Process Architectures. To appear in *Proc of International Conference on Performance Tools’95*, Heidelburg, Sept. 1995.

[5] J. Hong, M. Bauer and J. Benett, Integration of the Directory Service in Distributed Systems Management. *Proc. of 1992 International Conference on Parallel and Distributed Systems*, Hsin Chu, Taiwan, pp. 142 - 149, Dec. 1992.

[6] J. Mylopoulos, A. Borgida, M. Jarke and K. Koubarakis. *Telos: A Language for Representing Knowledge about Information Systems (revised).* Technical Report KRR-TR-89-1, Department of Computer Science, University of Toronto, August 1990.

[7] OMG. *The Common Object Request Broker: Architecture and Specification.* Object Management Group, Framingham MA, 1993.

[8] OSF. *The OSF Distributed Computing Environment Rationale*. Open Software Foundation, Cambridge MA, 1991.

[9] G. Perrow, J. Hong, H. Lutfiyya and M. Bauer. The Abstraction and Modelling of Management Agents. To appear in *Proc. of the 4th IFIP/IEEE International Symposium on Integrated Network Management*, 1995.

[10] K. Raymond. Reference Model of Open Distributed Processing: a Tutorial. *Open Distribted Processing II (C-20)*, J. De Meer, B. Mahr and S. Storp (Editors), Elsevier Science B.V. (North-Holland) , pp. 3 - 14, 1994.

[11] J. Rolia. Distributed Application Performance, Metrics and Management. *Open Distribted Processing II (C-20)*, J. De Meer, B. Mahr and S. Storp (Editors), Elsevier Science B.V. (North-Holland) , pp. 235-246, 1994.

[12] J. Rolia, C. Woodside, V. Vetland, R. Bunt, D. Eager, M. Bauer, J. Hong, H. Lutfiyya, J. Black, T. Kunz, D. Taylor, P. Martin, T. Teory and P. Finnigan. Distributed Application Management, The MANDAS Project. To appear in *Proc. of the 6th IFIP/IEEE International Workshop on Distributed Systems: Operation and Management*, October 1995.

[13] M. Sloman. *Network and Distributed Systems Management*, Addison-Wesley Publishing Co., Wokingham England, 1994.

[14] O. Wolfson, S. Sengupta and Y. Yemini. Managing Communication Networks by Monitoring Databases. *IEEE Trans. on Software Engineering 17(9)*, Sept. 1991.