Services supporting management of distributed applications and systems

by M. A. Bauer
R. B. Bunt
A. El Rayess
P. J. Finnigan
T. Kunz
H. L. Lutfiyya
A. D. Marshall
P. Martin
G. M. Oster
W. Powley
J. Rolia
D. Taylor
M. Woodside

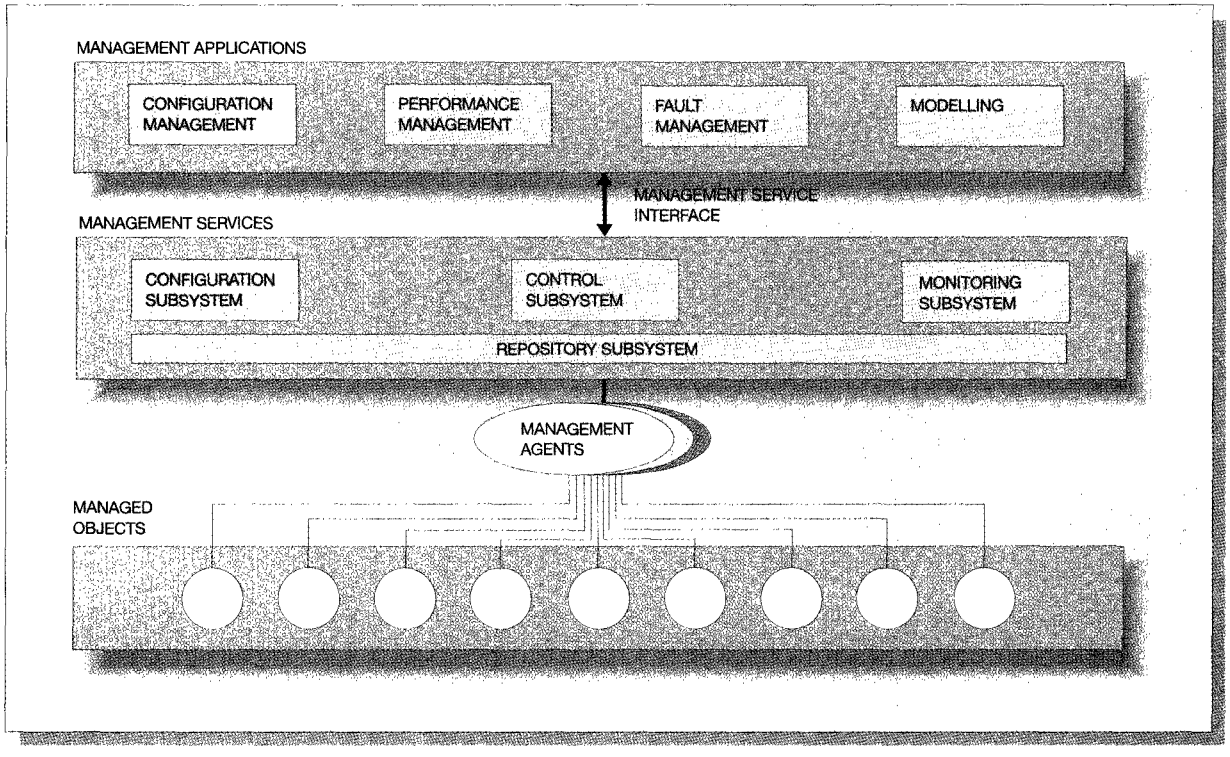
A distributed computing system consists of heterogeneous computing devices, communication networks, operating system services, and applications. As organisations move toward distributed computing environments, there will be a corresponding growth in distributed applications central to the enterprise. The design, development, and management of distributed applications presents many difficult challenges. As these systems grow to hundreds or even thousands of devices and similar or greater magnitude of software components, it will become increasingly difficult to manage them without appropriate support tools and frameworks. Further, the design and deployment of additional applications and services will be, at best, ad hoc without modelling tools and timely data on which to base design and configuration decisions. This paper presents a framework for management of distributed applications and systems. The framework is based on a set of common management services that support management activities. The services include monitoring, control, configuration, and data repository services. A prototype system built on the framework is described that implements and integrates management applications providing visualisation, fault location, performance monitoring and modelling, and configuration management. The prototype also demonstrates how various management services can be implemented.

Distributed systems must be managed to ensure that they operate as intended. Distributed application management focusses specifically on tools and services for managing the processes and files of applications running within a distributed system. It may be said to sit on top of network management (for the communications infrastructure) and system management (for the system services and devices such as file systems, operating systems, disk drives, and printers). The management environment has to collect data about the existence and behaviour of services and devices of many kinds. The data have to be kept for analysis based on different viewpoints, such as the performance of a particular service, or the availability of a set of services, and for analysis over different time scales. The management system also has to provide automatic reactions of various kinds to maintain services and service quality.

©**Copyright** 1997 by International Business Machines Corporation. Copying in printed form for private use is permitted without payment of royalty provided that (1) each reproduction is done without alteration and (2) the *Journal* reference and IBM copyright notice are included on the first page. The title and abstract, but no other portions, of this paper may be copied or distributed royalty free without further permission by computer-based and other information-service systems. Permission to *republish* any other portion of this paper must be obtained from the Editor.

0018-8670/97/$5.00 © 1997 IBM

Figure 1 MANDAS management framework



Compared to network and system management, there has been little research on distributed application management.[1-4] The distinguishing difficulty of this level of management is that applications are, by their nature, more specialised, more distinctive, and more heterogeneous than network devices or file services. Management tends to be developed specifically for one application or one domain. Domain independence, environment independence, and scalability are necessary characteristics of an effective solution for application management.

In our previous work[2] we proposed a conceptual framework[5] with a central role for several powerful midlevel management services. These services interacted with managed applications and management agents, which were integrated by a common data repository. The management services were in turn exploited by management applications. We have continued to work on our management framework, in particular by concentrating on the design, implementation, and use of management services. This paper presents our results to date on management services.

First we review our management framework and explain each of the management service subsystems that we have been developing as part of the project. Next we describe a prototype system developed to evaluate these services and experimental management applications. We then position our research with respect to related work. Finally we present the lessons we have learned and conclude with directions for future research.

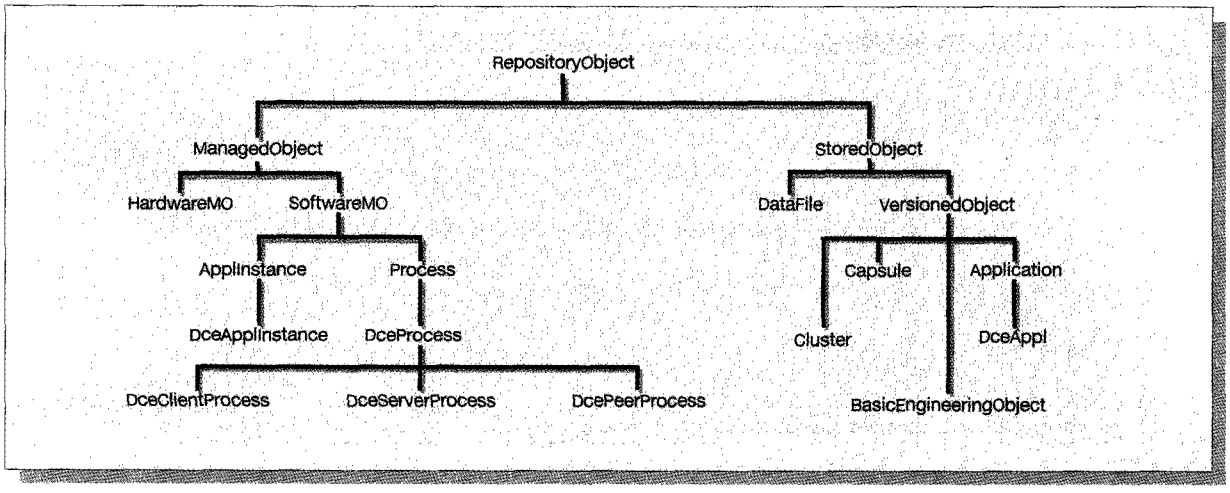
Management framework and services

MANDAS, the management framework developed in our previous work, is depicted in Figure 1.

Management applications are used to perform management tasks, such as system configuration, analysis of performance bottlenecks, fault detection and location, report generation, visualisation of network or system activity, simulation, and modelling.

Playing a central role in our framework is a set of *management services*, which in our previous work we

Figure 2 Information model class hierarchy



organised as four subsystems: the *repository services subsystem*, the *configuration services subsystem*, the *monitoring services subsystem*, and the *control services subsystem*.

Managed objects are abstractions of real managed resources such as processes or devices. *Management agents* carry out management activities on behalf of management services and applications. For example, they may configure the managed resources, monitor their behaviour, and perform control actions on them. Some agents may be capable of both monitoring and controlling functions; others may even possess some analysis capabilities. Management services may communicate with these agents using SNMP,[6] CMIP,[7] or proprietary protocols.

Each of the management services subsystems is described in detail.

Repository services subsystem. The repository subsystem provides the database management services needed by the management applications and the other subsystems. Three types of data must be handled by the repository subsystem: structural data, control data, and measurement data.

Structural data consist of descriptions of managed objects, their relationships, and their environments. These include, for example, application configurations, workload characteristics, and network topology. Structural data typically consist of many instances of relatively few types of objects. These descriptions can be complex and will typically outline the relationships an object has to other objects. For example, there may be many process instances active at one time, all of which can be described in the same way. The description of each process instance contains properties such as process identifier, parent process identifier, start time and end time, and relationships to its host, its application, its executable file, its data files, and other processes. Retrievals from the structural data are primarily queries using object identifiers or attribute values, or following relationships from one object to other objects. Finally, structural data are relatively static since they are only updated for "significant events," such as process creation and process termination. These properties suggest that an object-oriented information model is the most appropriate representation.

We have defined an object-oriented information model to describe the structural data. A portion of its class hierarchy is shown in Figure 2; a detailed description of the model is available.[8] The classes in the model describe a distributed application in terms of its code and run-time components. The code components of a distributed application include the files that are used to build an application, such as source code files (basic engineering objects), object code files (clusters), executable files (capsules), and data files. We have chosen to use generic terminology from the engineering structures of the Reference Model for Open Distributed Processing

(RM-ODP)[9] in order to accommodate different styles of applications. The run-time components of a distributed application include the run-time managed objects such as application instances and processes. The relationships between the code and run-time objects of an application are represented as link or object reference attributes in the class definitions.

Control data capture information related to the operation of an application and are of two types. The first type is the set of environment and initialisation values for an application instance. The second type is the set of *event notifications* generated by a managed object. For example, a process would send a notification to its management agent for events such as process creation, process termination, or remote procedure call (RPC) timeout. A history of the event notifications may be maintained for use in management applications, such as fault management.

Measurement data describe the run-time operation of an object. They may be data collected by monitoring the object, such as process CPU use or process disk I/O, or they may be derived from collected data, such as "resource use = CPU use + disk I/O." Measurement data provide the primary input for the performance management, performance modelling, and fault management applications. The data provide both the current state and the execution history of an application in terms of resource use and quality of service provided to users.

Accesses to the measurement data consist mainly of updates by the management agents collecting the data and retrievals by the management applications analysing the data. Updates are typical database updates in that the new value is independent of the current value, but atypical in that they happen in real time. The real-time characteristic of updates means that the overhead incurred by an update must be minimised and so favours a design in which updates can be performed on databases local to the management agents. Retrievals, on the other hand, do not have real-time requirements like updates, but they may involve data about more than one managed object or summaries of the data across one or more dimensions. For example, an analysis of resource usage may require examining usage totals for an entire system, an application, or an individual process, on an hourly, daily, or weekly basis. It will also be necessary to maintain historical data for certain types of analyses. Thus, the service to provide storage and management of measurement data must support both complex retrievals and real-time updates. One approach to providing the service is with a data warehouse[10] that will allow collection of the measurement data at distributed, independent sites and will also integrate copies of the data in a common database for querying and analysis.

The different characteristics of the three types of data meant that no single type of database system could efficiently support all the data. This fact led to a repository subsystem composed of two different repositories: the Management Information Repository to store structural and control data, and the Measurement Data Warehouse to store measurement data.

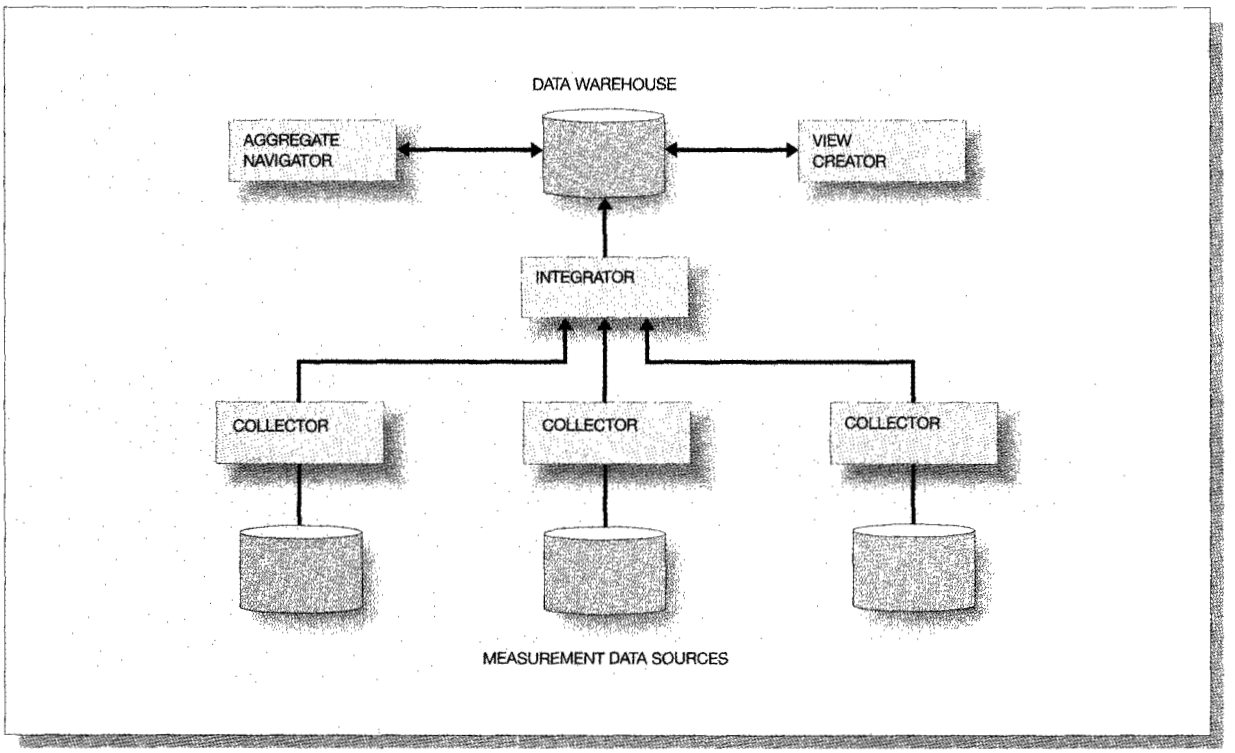
Management Information Repository. The Management Information Repository (MIR) implements the information model and is used to store structural data and control data. It can be implemented with a client/server architecture in which the MIR server stores the data and provides retrieval and update functions to clients. Clients (users or management applications) retrieve data from the server in two ways, by issuing queries to the server searching for objects matching certain criteria, or by using a browsing paradigm to locate objects by exploiting the structure of the information model.

Clients interact with the MIR server through a collection of functions called the MIR client library. The MIR client library presents clients with a view of the MIR that is consistent with the MANDAS information model. It also encapsulates communication so that interaction with the MIR server can be independent of other management services. There are three basic object types used in the MIR: management metaobjects, management schema objects, and management data objects. Schema objects are similar to classes in an object-oriented programming language. They define the attributes that a data object may contain. The attributes are further grouped into categories that are defined by a metaobject. Each schema object is an instance of one or more metaobjects (thus specifying the categories of the attributes), and each data object is an instance of a schema object.

The MIR client library provides functions to connect to and disconnect from the MIR, to create, modify, and retrieve metaobjects, schema objects, and data objects, and to define query conditions and issue data object queries.

Measurement Data Warehouse. The Measurement Data Warehouse, shown in Figure 3, stores the measurement data and the event histories. The data

Figure 3 The Measurement Data Warehouse



warehouse approach allows us to separate the real-time updates of the measurement data from the complex data analysis performed by management applications. The Measurement Data Warehouse is made up of the following components:

- A *collector* is responsible for propagating data from a measurement data source into the warehouse. The measurement data source can be of various types, for example, a file, or an agent that emits a data stream using a standard or proprietary protocol. The collector takes the data from the source, converts it into repository format, then passes the data to the integrator.
- The *integrator* combines data from the various data sources, translates the data into changes to be applied to the views in the warehouse, then applies the changes to the warehouse data.
- The *data warehouse* is typically a relational database management system that maintains the measurement data in a form suited to analytical processing. Aggregates along various dimensions may be maintained as materialised views of the base data to support faster querying of the measurement data.
- The *view creator* allows the system manager to define the views of the data that are to be kept in the warehouse.
- The *aggregate navigator* translates high-level data requests from the analytical processing into queries on the views in the data warehouse.

Configuration services subsystem. Structural data that capture the structure and relationships of a system and application are referred to as *configuration data*. Configuration data have both static and dynamic aspects. Static configuration data describe the organisation of the system and applications at start-up. Changes made to the system or application while the system is running create dynamic configuration data. Both a user request to start an application and a process spawning a subprocess represent the types of events that change a system's configuration. The configuration services subsystem is responsible for detecting, collecting, and maintaining descriptive and

location information about the entities of the distributed system.

Configuration information is stored in the MIR. A service[11] is needed to enable management applications and other management service subsystems to update the configuration information as a result of changes. This service uses the MIR client library to provide a registration service and a query service to other management entities. The registration services permit registration of various system and application components in the MIR. The entities that can be registered, and the services responsible for doing so, are shown in Table 1. The services available for retrieving information about system and application components from the repository are shown in Table 2.

Monitoring and control services subsystem. In our original management framework,[2] we considered monitoring and control services as two separate subsystems. Our experiences to date suggest that they should be combined into one subsystem, supported by process instrumentation.

The monitoring subsystem is responsible for monitoring the behaviour of managed objects in the distributed system. Measurement data are collected by, or from, management agents and stored via the repository subsystem. Subsequently, data may be retrieved from the repository for analysis. Results may be returned to the repository for other uses, such as for later display or for further analysis.

The monitoring subsystem must be able to determine appropriate agents and information in response to requests for past, current, or future information. Such requests could originate from administrators via management applications or come from other management service subsystems. Accordingly, the monitoring subsystem is responsible for initiating the collection of information at appropriate times, delegating monitoring requests to remote monitoring components or to subordinate systems and devices, coordinating the collection of data from multiple agents, and ensuring that ongoing monitoring activities continue even under failures.

The control subsystem encompasses the set of components responsible for controlling the behaviour of managed objects. Control activities may also be carried out by interacting with management agents. The control subsystem requests may come from various management applications, from the monitoring subsystem, or from the configuration subsystem. For example, the monitoring subsystem or management tools may trigger appropriate control actions to be taken when exceptions on managed objects arise.

Table 1 Registration services

Registration Service	Entity
RegisterHost	Host
RegisterExecutable	Executable file
RegisterApplication	Application
RegisterApplicationInstance	Application instance
RegisterProcess	Process

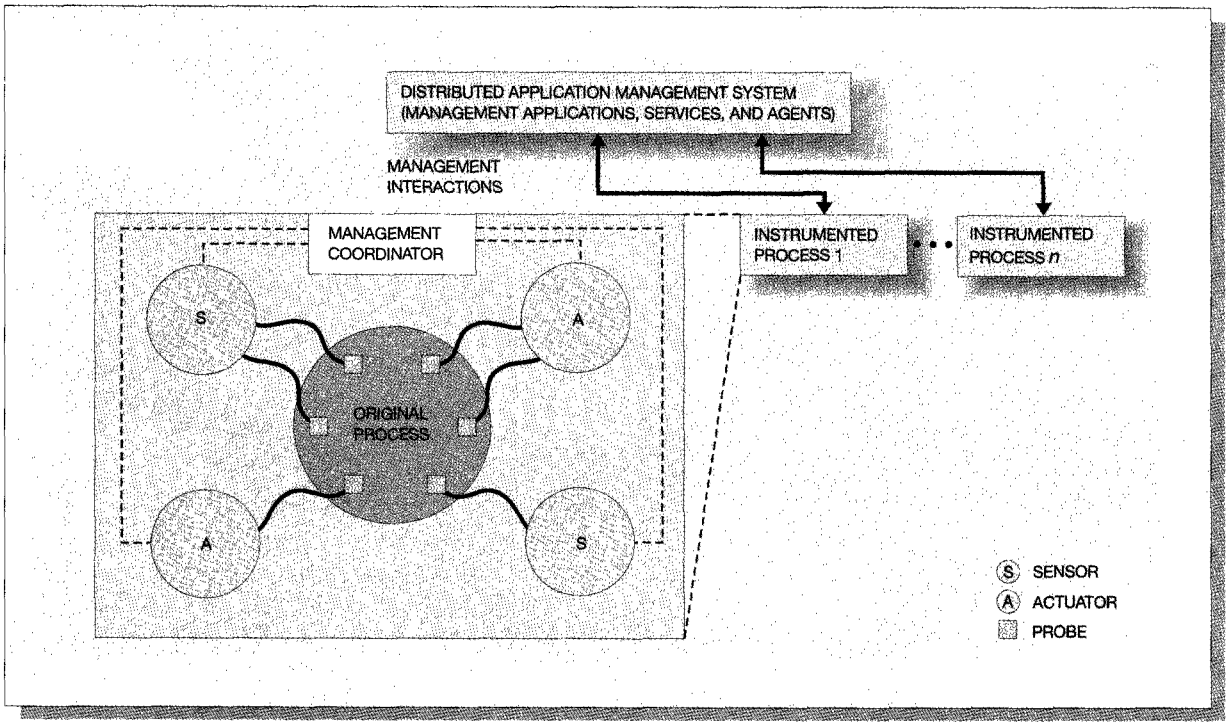
Table 2 Query services

Query Routine	Returns
RetrieveAllHost	Information about hosts in the system
RetrieveExeInApp	Information about executables in a specific application
RetrieveAppInstance	Processes in a specific application instance
RetrieveProcInAppInstance	Information about a process in a specific application instance

Both monitoring and control aspects of distributed application management make use of management agents and require processes to be instrumented. An *instrumented manageable process* is an application process with embedded instrumentation. Such a process can maintain management information, respond to management requests, and generate event reports. Thus, instrumentation provides aspects of both monitoring and control services. More details on the design and implementation of the instrumentation are available.[12,13] The instrumentation components, depicted graphically in Figure 4, are described below:

- The *management coordinator* facilitates communication between management agents and an instrumented process. Its role includes message routing for requests, replies, and reports flowing between the management system and the instrumentation code. The management coordinator, upon receiving incoming requests from management agents, invokes the appropriate functionality in the instrumentation code. Similarly, the instrumentation code sends requests or reports to the appropriate management agents through the management coordinator.

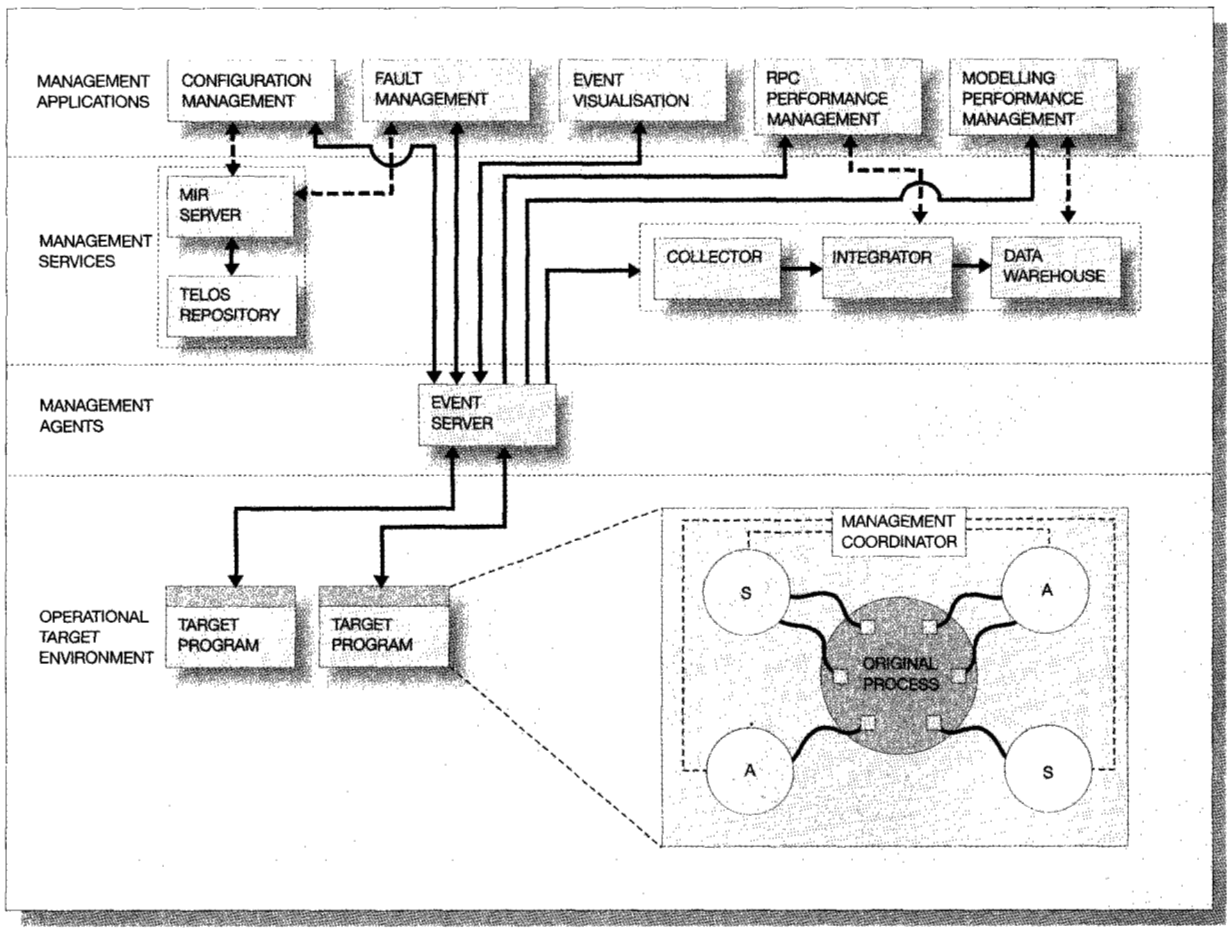
Figure 4 Instrumentation architecture and environment



- *Instrumentation code* provides an internal view of the managed process. There are two types of instrumentation code: *sensors* and *actuators*. Sensors encapsulate management information. They collect, maintain, and (perhaps) process this information within the managed process. Sensors exhibit monitor-like qualities in that they encapsulate management information and provide an interface through which probes, other sensors, and the management coordinator can access their state in a controlled manner. Sensors can be provided for a variety of performance metrics, to measure resource usage, to collect accounting statistics, and to detect faults. Sensors get their input data from probes inserted at strategic points in the process code or by reading other sensors. Sensors provide their information to the management coordinator in the form of periodic reports, alarm reports (when exceptional or critical circumstances arise), or in response to explicit requests. The second type of instrumentation code is an actuator. The actuator encapsulates management functions that exert control over the managed process to change its operation.
- *Instrumentation probes* are embedded in the process to facilitate interactions with the instrumentation code (the sensors and actuators). Probes may be implemented as macros, function calls, or method invocations that are injected, during development, into the instruction stream of the application at locations called *probe points*.

We now illustrate these concepts with an example. We may specify that a client process should receive a response to a remote procedure call within x seconds. A probe is injected before and after each remote procedure call. The probe before the remote procedure call includes code to initialise a variable with the current time. The probe after the remote procedure call includes code to determine the elapsed time. This elapsed time is passed to a sensor. The sensor determines if a threshold has been exceeded. If it has, an event report is sent through the management coordinator to the management agent. If the management system decides to change the threshold from x seconds to x' seconds, a message is sent through the management coordinator, which then informs the sensor.

Figure 5 Management system prototype implementation



A prototype management system

A key aspect of our work has been to evaluate and refine the proposed framework and services via prototypes and experimentation. A prototype implementation (Figure 5) demonstrating some of our ideas and concepts was shown at the Centre for Advanced Studies conference (CASCON) in 1996. The prototype platform was the Open Software Foundation's Distributed Computing Environment** (OSF DCE**),[14] which was chosen because of its availability and our past experience with it. In this section we describe the management applications that made use of the management services, our management agent, our use of instrumented processes to implement aspects of the monitoring and control services, the implementation details of the MIR and Measurement Data Warehouse, how the configuration services subsystem was implemented, how the prototype operates, and the interactions among its components.

Management applications. To validate the management services as well as to investigate aspects of management applications, it was important to have, as part of the prototype system, several management applications.

Event visualisation. POET, Partial Order Event Tracer,[15] is a tool for collecting and visualising event traces from the execution of distributed applications. Although POET was originally intended for use as a debugging tool,[16] its event displays are also useful for visualising the operation of an application in production use.

POET's notion of an event is fairly general, which allows it to remain independent of the target system generating the events. In the context of DCE, for example, events represent the creation and deletion of processes, threads and mutexes,[17] RPC communication, and thread-synchronisation events such as mutex lock and unlock. POET displays show different event types with different combinations of open and filled circles and squares. Each entity (such as a process or a mutex) is visualised as one "trace line"; events occurring within this entity are placed along the trace line in such a way that the underlying partial order on events is preserved. Events representing an interaction between entities are connected by an arrow showing the direction of information flow. In the case of an RPC, for example, the *call* event and the *receive* event are paired, with the call event occurring in a client process thread and the receive event occurring in a server process thread. In this context, POET can be used to view an application's execution on an ongoing basis, scrolling the display automatically when the current display fills up. This allows a user to focus on the current activity of the application or system being monitored.

POET provides abstraction operations to group events into abstract events and traces into clusters. These abstractions can, in turn, be grouped again into higher-level abstractions, leading to tree-structured abstraction hierarchies. The user can navigate the abstraction hierarchies to visualise the execution at an appropriate abstraction level.[15,18] The abstraction hierarchies can be built manually or by using automatic tools.[19,20] Various forms of pattern specification and matching to automatically scan the incoming event trace are also supported. Some examples of these facilities are the flagging of potential race situations, the detection of user-specified global predicates, or the automatic abstraction of certain user-specified communication patterns.

Performance management. Delays caused by poor performance at the application level or network level can seriously affect the usability and effectiveness of a distributed application, or an entire distributed environment. Both application developers and managers of a distributed system must therefore take steps to ensure that their systems are performing well.

To that end, we have developed two performance-related management applications, both of which share a common subset of performance data stored in the Measurement Data Warehouse.

Measurement and modelling of RPC traffic. RPC performance is critically important to the distributed applications we are interested in. A real-time performance monitoring tool was used to visualise DCE RPC performance data obtained from the Measurement Data Warehouse. RPCs are decomposed by DCE runtime software into a series of protocol data units (PDUs), the size of which is implementation-dependent. These PDUs are further decomposed by network software into a series of IP (Internet Protocol) packets on the network. The superposition of these two processes determines the arrival process. The tool provides a visual picture of some of these characteristics in "real time," and can be used to observe the effects of application, system, and network factors on traffic patterns. Network-level performance is presented as a set of times: the interarrival times for all IP packets, the interarrival times for PDUs, and the interarrival times for IP packets in PDUs. The use of this information in performance visualisation is illustrated in Sun et al.[21]; its use in performance prediction and workload modelling is discussed in detail in Sun.[22]

Automatic construction of predictive models. Predictive performance models for distributed application systems can be used by distributed application developers and performance management staff to make quantitative comparisons between software design and system configuration alternatives. Models and their performance evaluation techniques[23–25] can also be used in capacity planning, to determine whether specific application objects should be placed in the same server, or how server processes should be allocated to nodes for a given workload mix of application requests.

We build these models based on data, collected about applications as they run, that are stored in the Measurement Data Warehouse. The models require information about hardware resource demands and about remote procedure call interactions between application objects. Some information is collected from the operating system using management agents, some from probes in the "midware," and some from probes inserted in the application.

Automating fault location. One aspect of distributed application management is fault management—detecting that the behaviour of an application has deviated from the specification of its desired behaviour. This deviation is referred to as a *failure*, and is manifested through observed *symptoms*. Symptoms are detected and reported by instrumentation and

management agents. At the source of a failure is a *fault* (or possibly several faults). Symptoms alone do not provide enough information to allow the fault to be corrected: many faults may give rise to the same symptom. We have developed a management application, the fault management tool,[26] that automates the process of fault location. The steps in this process are: reducing the number of symptoms for further examination, determining a set of system objects that could be the source of the fault, examining the failure history of these objects to determine an order in which to test them, then, finally, testing each object in turn. The tool makes use of configuration and repository services to determine the configuration of applications and hardware (to find the set of objects that could be the source of the fault), and of management agents and instrumentation for determining the possible causes of a fault (through status checking and testing).

Configuration management application. The configuration management application can retrieve configuration information about distributed applications from the MIR, including the structure of executing distributed applications, the source code, and the network topology. The user interacts with the configuration management application to start, stop, view, and manage applications.

The POET event server as management agent. The POET-based event visualisation management application described earlier has its data supplied by a separate process called the POET event server. In previous work we used management agents that conformed to CMIP, with all the communication layers that entails. Since the POET event server is able to accept messages from applications and forward these messages to the visualiser, we chose to experiment with it to see if it can be used with other management applications as a "lighter-weight" management agent.

The existing POET infrastructure was enhanced to support a wider variety of management applications as follows. Each management application interested in event information defines its own target-system instrumentation, using the underlying POET event-collection functions. Events are typed, and event types are grouped into classes, one for each management application. As part of the registration of each event class, the event server is provided with the name of a class-specific client. When events of a class other than POET appear in the event stream, the event server will buffer them in a bounded buffer and, if necessary, start up the corresponding client. It is up to the class-specific clients to query the event server, using the standard protocol, for their event data. Also, once the connection is established, such clients can return control information via the event server to the embedded instrumentation. The base instrumentation package provides for the registration of "callback" functions by the class-specific instrumentation packages. These callback functions will be invoked when control information reaches the instrumentation stubs.

This design has allowed the integration of event data collection for performance analysis, fault detection, and visualisation, all using the same basic infrastructure.

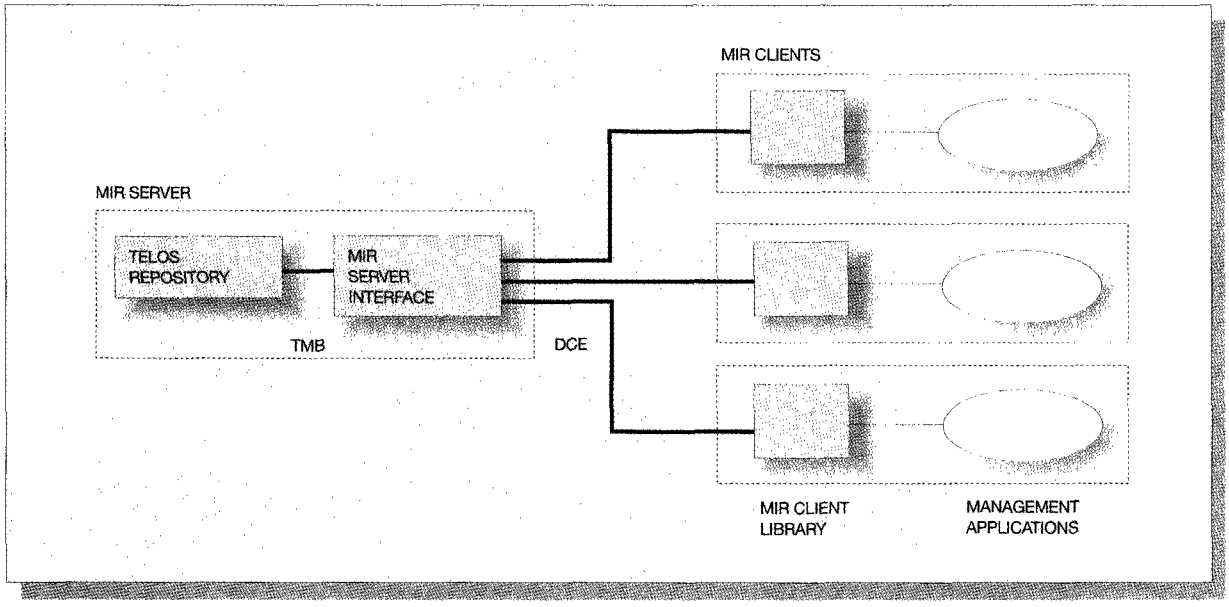
Instrumentation. We developed an instrumentation library for the prototype that includes the following sensors and actuators:

- Fault detection sensors. These sensors collect information about remote procedure call timeouts and response times.
- RPC statistics sensors. These sensors compute RPC statistics (e.g., average service time) at the end of each reporting interval, then report them to the management coordinator.
- Process control actuators. These actuators control process termination, process suspension, and process priority modification and change the length of the interval between RPC statistics event reports.

The following probes were inserted into source code by hand:

1. Process_instrumentationInit( ). For a process to be manageable, it is necessary that the process entry point have this instrumentation probe. It:
   - Retrieves the *binding handle* of the POET event server. The binding handle contains the information needed by a process to establish communication with the appropriate POET event server.
   - Performs registration. The POET event server is notified of the existence of a new process and is provided with information about the process, such as its process identifier, parent process identifier, host, and start time.
   - Creates the management coordinator and sensors. A management thread is created that becomes the management coordinator, the sensors, and the actuators.

Figure 6 MIR prototype



2. Process_rpcRequestBegin( ) and Process_rpcRequestEnd( ). These are inserted before and after each remote procedure call. They update the RPC statistics and fault detection sensors.

3. Process_instrumentationShutdown( ). This probe notifies the POET event server of the termination of a process.

Management Information Repository. The MIR prototype is depicted in Figure 6. The MIR server has two components: the Telos Repository, which provides the backend object-oriented database for the MIR, and the MIR server interface, which takes requests from an MIR client and translates them into requests to the Telos Repository, returning query results when necessary. Telos[27] is a conceptual modelling language that provides the concepts and facilities necessary to represent the different types of data objects and their relations relevant to distributed application management.

The Telos Repository used for the MIR prototype is the University of Toronto implementation of the Telos language. This implementation uses ObjectStore** as the underlying storage mechanism.

Although MIR clients could theoretically use the interface to the Telos Repository directly, an intermediate layer (the server interface) was built between the repository and the clients for the following reasons:

- *Database independence*. The MIR server interface buffers the MIR clients from the specifics of the Telos Repository. This will allow changes to the underlying storage mechanism with minimal modifications to the overall system. Only the backend of the MIR server interface would need to be modified in order to accommodate such a change. The MIR client interface would remain stable, thus requiring no change to the possibly numerous MIR clients.
- *Multiplexing*. The communication mechanism used by the Telos Repository allows only a single client to access the repository at a given time. The MIR server uses threads to service and coordinate multiple clients, sending one request at a time to the Telos Repository.
- *Extended query capabilities*. The Telos Repository provides very limited query capabilities. The MIR server interface extends these capabilities to include conjunctive queries based on *instance-of* conditions, *is-a* relations, and queries by attribute value. The MIR server interface generates a query that can be processed by the Telos Repository, sends the request to the repository, then filters the

result to return only the objects requested by the MIR client.

The MIR server interface is implemented in C++ and runs on an IBM RISC/6000* (RS/6000*) processor on top of OSF DCE. The MIR server interface communicates with the Telos Repository via the Telos message bus (TMB) API (application programming interface). Strings that are parsed and understood by the Telos Repository, called *s-expressions*, are passed from the MIR server interface to the Telos Repository. Requests are satisfied by the repository by returning s-expressions to the MIR server interface.

Clients communicate with the MIR via DCE RPCs using the functions provided by the MIR API (the client library). The client library provides users with both a C and a C++ interface to the MIR.

In the CASCON '96 demo, the MIR was used to store the configuration information for distributed applications. The configuration management service used the MIR to register applications and their associated processes. Information such as process identifier, parent process identifier, host, start time, and many other parameters were stored about each process. Static information about an application, such as references to the executable code, pointers to the source code, and other information, was also stored in the MIR. When an application was started, an application instance was registered with the MIR for use by other management services and applications.

Measurement Data Warehouse. At the time of CASCON '96, the Measurement Data Warehouse was in the initial design stages. For this reason, we used a simplified version of the warehouse to demonstrate the concept. This warehouse comprised three main components, the collector, the integrator, and the underlying database, DATABASE 2*/6000 (DB2*/6000).

For the CASCON '96 demo, processes associated with the target application were instrumented to collect information about each RPC, including the RPC identifier, the hosts of the processes involved, and the start and completion times for each RPC. A table was set up in DB2 to store these data.

The RPC information was passed from the instrumented processes to the collector via the POET event server. In a fully implemented Measurement Data Warehouse, the data might come from an event stream, a flat file, or possibly another database. The collector was implemented as a POET event-server client and was responsible for gathering the RPC information and passing it to the integrator. The interprocess communication between the collector and the integrator was done using IBM's CORBA** implementation, DSOM.[28]

The sole responsibility of the integrator in the CASCON '96 demo was to submit the data to DB2. This was done using embedded SQL (structured query language). The integrator will serve a much larger role in a full-scale data warehouse. The data stored in the data warehouse were retrieved and used by the performance monitoring tool, where they were used to provide an application-level view of system performance.

Implementation of configuration services. The configuration services subsystem described earlier was to be provided by a DCE server with the interface shown in Tables 1 and 2. This server process is an MIR client, hence it uses the MIR client library to store the information it collects in the MIR. Because of time constraints, this server was not ready in time for use in the demonstration (it has since been completed), so dynamic configuration information was maintained by the configuration management application.

The prototype in operation. We now describe the operation of the prototype, highlighting the interactions of the management components that are in place. We begin by describing the sequence of operations that take place when an application instance is started:

1. Information about the application, which includes the list of executable components needed, is stored in the MIR. Currently, this information is entered by hand by the developer or installer of the application.

2. The configuration management application is started. The user selects the Start option. This presents another menu with two options, one being Start Application. Selecting this option results in a window appearing with a list of applications, from which the user selects an application to be started.

3. After an application is selected, the configuration management application:

   - Generates a unique application instance identifier for the chosen application and registers

the application instance using the RegisterApplicationInstance registration service
- Retrieves information about the executable components using the RetrieveExeInApp configuration service. This returns information that includes the names of the executable files and the host on which each executable file should run.
- Starts an instance of the POET event server to act as management agent for the application instance
- Sends control commands to the event server to cause it to run the executable components making up the application on the appropriate hosts

4. When a process starts, it executes Process_instrumentationInit( ), which registers it.

5. The POET event server forwards the registration information to the MIR, where it is stored. It uses the RegisterProcess service to do this.

The user can now retrieve information about the new application instance through the configuration management application. The information is retrieved using the RetrieveAppInstance and RetrieveProcInAppInstance query service operations.

Once the processes in an application instance are operating, the probes Process_rpcRequestBegin( ) and Process_rpcRequestEnd( ) will update the information in the RPC statistics sensor. Periodically, the RPC statistics sensor computes statistics from the raw data collected by the probes and sends them to the POET event server. The POET event server forwards them to the Measurement Data Warehouse, where the performance-related management applications can retrieve them.

Suppose that a remote procedure call from one process to another fails. The fault detection sensor will generate a symptom, which will be passed to the POET event server, then to an instance of the fault management tool. As previously described, the fault management tool will attempt to determine the source of the fault.

When the user of the configuration management application selects the Stop Application option, a list of application instances is displayed. Once the user selects an application instance, the configuration management application identifies its component processes using the MIR and, using the control services, instructs each of the processes to stop. It also causes the application instance information to be removed from the MIR.

Related work

In this section we review related work in the areas of management frameworks, information models, and management services.

Management frameworks. Two management frameworks related to our work are Tivoli** and the Java** management application programming interface. Tivoli provides a framework for management applications that helps to tie together management information from many heterogeneous sources. The sources can include SNMP agents, other monitors, and even the MANDAS management services infrastructure. Attributes for each source are defined using application description files.[29] Special-purpose agents called *sentries* acquire management data from managed components. The application description files are used to interpret the data and make them available to Tivoli-based management applications. The layer of abstraction introduced between management applications and monitors helps to conceal the specifics of each monitor to provide for more easily developed management applications.

Our framework is similar to the Tivoli framework in that we also provide a common model for describing managed objects and the data collected about them. Both have developed a set of services that maintain the information within a common repository and allow it to be queried.

The Java management application programming interface (JMAPI)[30] provides a framework for management applications that uses Java and internet technologies to interact with sources of management information. The high level of connectivity offered by Java helps to simplify the problem of connecting with a heterogeneous collection of monitors. In this way the JMAPI competes with, yet complements, Tivoli. Furthermore, JMAPI provides a framework for the development of management application user interfaces in Java; however, it does not provide any specific management applications.

Other management frameworks, which have less in common with our work, include:

- OSF DME.[31] This provides a single methodology (independent of the underlying operating system) for deploying, updating, and controlling software in

a heterogeneous environment, thus simplifying these operations.

- NMF (Network Management Forum) OMNIPoint.[32] The focus is on network management and services management.
- IBM SystemView*.[33] This product focusses on the management of system services.
- IBM TMN (Telecommunications Management Network) WorkBench.[34] This focusses on the management of switching and transmission components. The TMN WorkBench eliminates much of the routine work involved in generating code for data structures from a GDMO (Guidelines for Definition of Managed Objects) specification. The TMN Workbench allows the use of GDMO ASN.1 (Abstract Syntax Notation 1), and SMI (Structure of Management Information) notations for the specification of managed objects, which can then be used by CMIP agents.

These four frameworks are intended for the management of entities at the network or systems levels, in contrast to our work, which is at the application level.

Information models. A choice of standards for configuration information models does exist. While the following standards are not explicitly associated with management, they can be adapted easily for the management domain:

- ISO management[7,35,36]
- Internet SNMP-styled management[6,37–39]
- A CORBA-based approach[40]
- An OSF DCE-based approach[14]
- DMTF/DMI[41]

Choices of standard management models relevant to distributed applications did not exist when we began our work. Recently, the Internet Engineering Task Force (IETF) and the Desktop Management Task Force (DMTF) have proposed models that cover some of application management as defined in this paper.

The IETF has proposed two extensions to its Management Information Base (MIB) structure: the *system application* MIB[42] and the *application* MIB.[43] The system application MIB maintains information about applications already installed and running on a node. This information is limited to what can be obtained without instrumenting the application code, such as the application packages installed and their component files and executables, the application instances started and the processes making up an instance, and system-level measurements such as CPU usage and memory usage of a process. The application MIB extends the system application MIB structure to include attributes that require instrumentation of the application. In particular, it adds information on open files, open connections to other processes, and transaction statistics. The MIB descriptions are low level compared to our information model. Our model provides powerful object-oriented abstraction mechanisms, like inheritance, and more detailed descriptions of the code and run-time views of a distributed application than can be formulated with the MIB structures. Our model also describes the distributed run-time environment of an application, while the MIB structures can only describe local nodes individually.

The DMTF has proposed the *Management Information Format* (MIF),[41] which is similar to MIB structures with respect to its modelling capabilities and has the same shortcomings as MIB structures when compared with our model.[29] In another effort, the DMTF has proposed the *Common Information Model* (CIM),[44] which in the same way as our model applies object-oriented modelling techniques to network and systems management. The CIM application schema definition, developed by the DMTF Application Management Working Committee,[45] is the component of the CIM closest to our work. The application schema deals with the installation and deployment of an application over its lifetime but does not include information relevant to other aspects of management, such as descriptions of the run-time environment. Our model, on the other hand, provides a more complete description of a distributed application that can be used for configuration management, fault management, and performance management.

Monitoring and control of distributed applications. Some work has been done in implementing specific tools for monitoring and controlling distributed applications. Some of these tools are:

- Meta toolkit.[4] This toolkit is a system for managing distributed applications developed using the ISIS distributed programming toolkit.[46]
- Huang and Kintala tools.[47] This set of tools provides services for detecting whether a process is alive or dead, specifying and checkpointing critical data, recovering checkpointed data, logging events, locating and reconnecting to a server, and replicating user-specified files on a backup host.

- Megascope tool.[48] This is a utility for monitoring, reporting, managing, and presenting status information about the resources of computer systems and environment services in large, heterogeneous distributed environments. Megascope is an autonomous distributed application, built on top of OSF DCE. It extends the basic functionality of DCE by adding to it a service that provides the cell resource information.
- MAL (Management Adaptation Library).[49] This work focusses on instrumentation. A separate thread is used to support management.

The Meta toolkit is meant to be within a specialised environment that is provided by ISIS. Our work is more general in that we do not make any particular assumptions about the underlying environment. The Megascope tool is specific to DCE services while our work also encompasses distributed applications that use DCE services (or any other midware environment). The services provided by the Huang and Kintala tools can be implemented by an agent, or by sensors and actuators. MAL is similar to our work in instrumentation except that we further developed an architecture for instrumentation that includes both sensors and actuators.

Configuration services. A review of the academic literature and current research[50–54] on configuration management finds it to be concerned primarily with the development of languages and environments for the implementation of reconfigurable systems. Most of these languages and systems adopt the principle of the strict separation between a *module configuration language*, which describes the overall static and dynamic structure of the program, and a *module programming language*, which is used to implement the algorithms within the application program. Reconfiguration facilities are usually restricted to a class of changes and are embedded into the module configuration language.

Summary. As can be seen, much of the work just described focusses on a single aspect of management. For example, much of the work in monitoring tools does not explicitly consider integration of services for the collection, storage, and analysis of data about the internal behaviour of processes.

In contrast to the bottom-up approach seen in much of the related work, we have chosen to consider the requirements of management applications. This has allowed us to extract a set of common requirements with respect to an information model, configuration services, and repository services.

Lessons learned

Our work on the MANDAS framework has taught us a number of lessons concerning the management of distributed applications.

The framework developed in our initial work[2] treated monitoring and control services as two separate subsystems. We now see them as one subsystem implemented through a combination of instrumentation and management agents, with the only difference between control and monitoring being the direction of information flow: out of a managed process for monitoring and into it for control.

The management agents used in managing distributed applications may, and likely will, have a range of capabilities and rely on a variety of communication mechanisms. We have developed agents that use standard protocols, such as SNMP or CMIP, to communicate with management applications and service subsystems. We have also developed alternative agents, such as those described in this paper, using alternative protocols. We need to balance the ease of using agents with standard services, for example, a CMIP communication interface, where there is no need to translate management requests into lower level primitives, against the "weight" of the implementation of such an agent. To facilitate development of management applications it is most important to isolate the management agents from the management applications, so that developers can concentrate on functionality rather than protocol details.

Our development of the prototype emphasised the importance of the information model in the framework. The information model provides a common description of all aspects of an application that can be used by all the management applications. The model and its associated MIR support integration of the set of management applications into the management framework.

Distributed application management requires the collection and storage of different types of data with very different characteristics and access requirements. No one type of database system can adequately support all the requirements, so the repository service must consist of multiple database systems. We found that a reasonable approach was

to use two database systems: one system with a powerful object-oriented model to store the structural data and a second system to handle the measurement data with their frequent updates and relational database-like querying requirements.

Acquiring the data needed to perform management is a complex and tedious task, so a management framework must include tools to support data acquisition. This includes tools to automatically extract configuration information from application files and tools to support the automatic generation of instrumentation code.

A distributed system running a number of distributed applications has the potential to generate vast quantities of management data. To be scalable, efficient means of handling large amounts of data and approaches to filtering or aggregating data will be needed.

We underestimated the importance of the human/computer interface in managing distributed applications. More research is needed to develop effective ways of screening and presenting relevant data to the user.

Future directions

One of the motivations for developing a management framework is to enable the development of tools for automating some of the steps involved in generating and running manageable applications. We have developed, and are continuing to develop, such tools. Two examples of these tools, one aimed at creating applications, the other at creating management agents, are described here.

An automated model builder for distributed applications has been integrated into a prototype of IBM's Distributed Application Development Toolkit (DADT). With DADT, an application designer can specify object interfaces and choices for midwares. DADT then generates "wrapper" code to cause the appropriate object interactions with the specified midwares. The model-building system[55] has three major components:

- An instrumentation package and tool for capturing information about application level objects
- A statistical analyser that deduces some model parameters from measurement data that cannot be measured directly[56]
- A model generator that gathers information from the management information repository and creates models for performance evaluation

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, and what relationships the agent should have with the environment (i.e., hardware or software resources, user interface, etc.). We have identified a set of services, independent of the underlying management protocols, 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 predetermined events of interest, and communicating with other entities. We have also identified a generic architecture for agents that describes the services that the agents should or could provide, the components of an agent, and how the components satisfy the services. Based on this infrastructure, we have developed a tool that allows the user to specify the type of agent using a simple graphical user interface.[57,58] The tool allows the user to enter the required information, ensures that the information entered is valid, then generates the code for the specified agent. We are currently examining how to extend this tool to a Java-based network environment.

We have developed a number of prototype management applications, but determining the kinds of management applications that will be needed in the field remains an open problem. We believe that the use of *policies* may help drive this search. Policies originate with the business needs and objectives of the enterprise and represent the operational requirements of the systems, services, and applications. We believe that the specification and enforcement of policies will give insight into the management applications needed and the services to support these applications.

Our current management applications have to know about the types and locations of management agents. Management applications could be more easily developed if these details were hidden by a service that takes requests for data and determines the best way of getting those data. We have begun work on such a service.

Finally, we have had some success in integrating different management applications into a prototype. What is still needed is a consistent user interface used

by all the management applications—a "management station" with a consistent user interface, rather than a collection of disparate tools.

Acknowledgments

James P. Black and Vidar Vetland contributed to the work described in this paper. This research was supported by the IBM Centre for Advanced Studies and the Natural Sciences and Engineering Research Council of Canada. We would also like to thank the referees of this paper for their excellent comments and constructive suggestions for improving the paper.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of the Open Software Foundation, Inc., Object Design, Inc., Object Management Group, Tivoli Systems Inc., or Sun Microsystems, Inc.

Cited references and notes

1. M. A. Bauer, N. Coburn, D. L. Erickson, P. J. Finnigan, J. W. Hong, P.-Å. Larson, J. Pachl, J. Slonim, D. J. Taylor, and T. J. Teorey, "A Distributed System Architecture for a Distributed Application Environment," *IBM Systems Journal* **33**, No. 3, 399–425 (1994).
2. M. A. Bauer, P. J. Finnigan, J. W. Hong, J. A. Rolia, T. J. Teorey, and G. A. Winters, "Reference Architecture for Distributed Systems Management," *IBM Systems Journal* **33**, No. 3, 426–444 (1994).
3. M. A. Bauer, H. L. Lutfiyya, J. W. Hong, J. P. Black, T. Kunz, D. J. Taylor, T. P. Martin, R. B. Bunt, D. L. Eager, P. J. Finnigan, J. A. Rolia, C. M. Woodside, and T. J. Teorey, "MANDAS: Management of Distributed Applications and Systems," *Proceedings, International Workshop on Future Trends in Distributed Computing Systems (FTDCS95)*, Cheju, Korea (August 1995), pp. 200–206.
4. K. Marzullo, R. Cooper, M. D. Wood, and K. P. Birman, "Tools for Distributed Application Management," *IEEE Computer* **25**, No. 8, 42–51 (August 1991).
5. This framework (the *MANDAS Management Framework*), its subsequent refinements, and the results reported in this paper were developed as part of the MANDAS (MANagement of Distributed Applications and Systems) project.
6. J. D. Case, M. Fedor, M. L. Schoffstall, and C. Devin, *Simple Network Management Protocol (SNMP)*, The Internet Engineering Task Force, Request for Comments 1157 (May 1990).
7. Information Processing Systems—Open Systems Interconnection, *Common Management Information Protocol Specification*, International Organization for Standardization, International Standard 9596-1 (1991).
8. P. Martin and W. Powley, "A Management Information Model for Distributed Applications Management," *Proceedings of CASCON '96*, Toronto, Canada (November 1996), pp. 54–63.
9. K. Raymond, "Reference Model of Open Distributed Processing: A Tutorial," J. De Meer, B. Mahr, and S. Storp, Editors, *Open Distributed Processing II (C-20)*, Elsevier Science B. V., Amsterdam (1994).
10. J. L. Wiener, H. Gupta, W. J. Labio, Y. Zhuge, H. Garcia-Molina, and J. Widom, "A System Prototype for Warehouse View Maintenance," *Proceedings of the Workshop on Materialized Views: Techniques and Applications*, Montreal, Canada (June 1996), pp. 26–33.
11. Z. I. Najib, *Maintaining Configuration Information in Distributed Systems*, master's degree thesis, The University of Western Ontario, London, Canada. In preparation.
12. M. J. Katchabaw, S. L. Howard, H. L. Lutfiyya, and M. A. Bauer, "Efficient Management Data Acquisition and Run-Time Control of DCE Applications Using the OSI Management Framework," *Proceedings of the Second International IEEE Workshop on Systems Management*, Toronto, Canada (June 1996), pp. 104–111.
13. M. J. Katchabaw, H. L. Lutfiyya, A. D. Marshall, and M. A. Bauer, "Policy-Driven Fault Management in Distributed Systems," *Proceedings of the Seventh International Symposium on Software Reliability Engineering (ISSRE '96)*, White Plains, NY (October 31–November 2, 1996).
14. *Introduction to OSF DCE*, Open Software Foundation, Cambridge, MA (1992).
15. D. J. Taylor, "The Use of Process Clustering in Distributed-System Event Displays," *Proceedings of CASCON '93, Vol. 1, Software Engineering*, Toronto, Canada (October 1993), pp. 505–512.
16. D. J. Taylor, T. Kunz, and J. P. Black, "A Tool for Debugging OSF DCE Applications," *Proceedings of the 12th Annual International Computer Software and Applications Conference*, Seoul, Korea (August 1996), pp. 440–446.
17. The term "mutex" (an abbreviation of "mutual exclusion") refers to a mechanism to ensure that only one thread at a time can perform a particular action. Most often, the action in question is accessing a shared variable. Typically, a thread acquires a mutex associated with a shared variable (the mutex mechanism guarantees that it can only do so if no other thread currently holds the mutex), updates the shared variable, then releases the mutex.
18. T. Kunz, "Visualizing Abstract Events," *Proceedings of CASCON '94*, Toronto, Canada (November 1994), pp. 334–343.
19. T. Kunz, "Reverse Engineering Distributed Applications: An Event Abstraction Tool," *International Journal of Software Engineering and Knowledge Engineering* **4**, No. 3, 303–323 (September 1994).
20. T. Kunz and J. P. Black, "Using Automatic Process Clustering for Design Recovery and Distributed Debugging," *IEEE Transactions on Software Engineering* **21**, No. 6, 515–527 (June 1995).
21. Y. Sun, R. Bunt, and G. Oster, "Measuring RPC Traffic in an OS/2 DCE Environment," *Proceedings of the Fifth International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '97)*, Haifa, Israel (January 1997), pp. 51–54.
22. Y. Sun, *Measuring and Modelling RPC Performance in OSF DCE*, master's degree thesis, Department of Computer Science, University of Saskatchewan, Saskatoon, Canada (August 1997).
23. J. Rolia, V. Vetland, and G. Hills, "Ensuring Responsiveness and Scalability for Distributed Applications," *Proceedings of CASCON '95*, Toronto, Canada (November 1995), pp. 28–41.
24. J. A. Rolia and K. C. Sevcik, "The Method of Layers," *IEEE Transactions on Software Engineering* **21**, No. 8, 689–700 (August 1995).
25. C. M. Woodside, J. E. Neilson, D. C. Petriu, and S. Majum-

dar, "The Stochastic Rendez-vous Network Model for Performance of Synchronous Client-Server-like Distributed Software," *IEEE Transactions on Computers* **44**, No. 1, 20–34 (January 1995).
26. C. Turner, *Fault Location in Distributed Systems*, master's degree thesis, The University of Western Ontario, London, Ontario (September 1995).
27. 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, Toronto, Canada (August 1990).
28. CORBA (Common Object Request Broker Architecture), defined by the Object Management Group (OMG), is a step toward standardization for interoperability among different hardware and software products. For more information, see http://www.omg.org/. IBM's Distributed System Object Model (DSOM) is consistent with the CORBA specification.
29. *Tivoli Applications Management Specification 1.1*; available from http://www.tivoli.com/amsreg/AMS+Spec+Registration.html/.
30. *Java Management API (JMAPI)*; available from http://www.javasoft.com/products/JavaManagement/.
31. *The OSF Distributed Management Environment (DME) Architecture*, Open Software Foundation, Cambridge, MA (May 1992).
32. Network Management Forum, *Discovering OMNIPoint*, PTR Prentice Hall, Englewood Cliffs, NJ (1993).
33. *SystemView Structure*, SC31-7038, IBM Corporation (March 1993); available through IBM branch offices.
34. *TMN Workbench for AIX: Power Tools for Application Development*, available from http://www.training.ibm.com/telmedia/tmnworkb.htm.
35. Information Processing Systems—Open Systems Interconnection, *Basic Reference Model—Part 4: Management Framework*, International Organization for Standardization, International Standard 7498-4 (1991).
36. Information Processing Systems—Open Systems Interconnection, *Systems Management Overview*, International Organization for Standardization, International Standard 10040 (1991).
37. J. D. Case, K. McCloghrie, M. T. Rose, and S. Waldbusser, *Structure of Management Information for Version 2 of the Internet-Standard Network Management Framework*, The Internet Engineering Task Force, Request for Comments 1441 (April 1993).
38. J. D. Case, K. McCloghrie, M. T. Rose, and S. Waldbusser, *Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2)*, The Internet Engineering Task Force, Request for Comments 1442 (April 1993).
39. M. T. Rose and K. McCloghrie, *Structure and Identification of Management Information for TCP/IP Based Internets*, The Internet Engineering Task Force, Request for Comments 1155 (May 1990).
40. *The Common Object Request Broker Architecture: Architecture and Specification*, Object Management Group, OMG Document No. 91.12.1 (1991).
41. *Desktop Management Interface Specification Version 2.00*, The Desktop Management Task Force, Hillsboro, OR (1996).
42. C. Kalbfleisch, C. Krupczak, R. Presuhn, and J. Saperia, *Application Management MIB*, Internet Engineering Task Force, Internet Draft (July 1997; available at ftp://ftp.ietf.org/internet-drafts/draft-ietf-applmib-mib-04.txt).
43. C. Krupczak and J. Saperia, *Definitions of System-Level Managed Objects for Applications*, Internet Engineering Task Force, Internet Draft (April 1997; available at ftp://ftp.ietf.org/internet-drafts/draft-ietf-applmib-sysapplmib-08.txt).
44. *Common Information Model (CIM) Version 1*, The Desktop Management Task Force, Hillsboro, OR (April 1997).
45. *The Common Information Model Application Schema Definition*, The Desktop Management Task Force, Hillsboro, OR (February 1997).
46. K. Birman and R. Cooper, *The ISIS Project: Real Experience with a Fault Tolerant Programming System*, Technical Report TR90-1183, Department of Computer Science, Cornell University, Ithaca, NY (1990).
47. Y. Huang and C. Kintala, "Software Implemented Fault Tolerance: Technologies and Experience," *Proceedings of the 23rd International Symposium on Fault Tolerant Computing*, Toulouse, France (June 1993), pp. 2–9.
48. B. Obrenić, K. S. DiBella, and A. S. Gaylord, "DCE Cells under Megascope: Pilgrim Insight into the Resource Status," *DCE—The OSF Distributed Computing Environment: Client/Server Model and Beyond, International DCE Workshop, Karlsruhe, Germany*, Number 731 in Lecture Notes in Computer Science, Springer-Verlag (October 1993).
49. P. Trommler, A. Schade, and M. Kaiserswerth, *Object Instrumentation for Distributed Applications Management*, Technical Report RZ 2730 (88162), IBM Research Division (July 1995).
50. F. Cristian, "Automatic Reconfiguration in the Presence of Failures," *IEE Software Engineering Journal* **8**, No. 2, 53–60 (March 1993).
51. M. Endler, "A Model for Distributed Management of Dynamic Changes," *Proceedings, Fourth IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM'93)*, Long Branch, NJ (October 1993).
52. F. Faure and D. Marquie, "Service Dynamic Management: A Configuration Micro-Manager," *Proceedings, Fourth IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM'93)*, Long Branch, NJ (October 1993).
53. C. Hofmeister, E. White, and J. Purtilo, "Surgeon: A Packager for Dynamically Reconfigurable Distributed Applications," *IEE Software Engineering Journal* **8**, No. 2, 95–101 (March 1993).
54. J. Kramer, "Configurable Distributed Systems," *IEE Software Engineering Journal* **8**, No. 2, 51–52 (March 1993).
55. M. Qin, R. Lee, V. Vetland, and J. Rolia, "Automatic Generation of Layered Queueing Models of Distributed Applications," *Proceedings of CASCON '96 (CD-ROM version)*, Toronto, Canada (November 1996).
56. J. Rolia and V. Vetland, "Parameter Estimation for Performance Models of Distributed Application Systems," *Proceedings of CASCON '95*, Toronto, Canada (November 1995), pp. 42–51.
57. G. S. Perrow, *The Abstraction and Modelling of Management Agents*, master's degree thesis, University of Western Ontario, London, Canada (September 1994).
58. G. S. Perrow, J. W. Hong, M. A. Bauer, and H. Lutfiyya, *MACT User's Guide Version 1.0*, Technical Report 434, Department of Computer Science, University of Western Ontario, London, Canada (September 1994).

Accepted for publication June 2, 1997.

Michael A. Bauer *Department of Computer Science, Middlesex College, University of Western Ontario, London, Ontario, Canada N6A 5B7 (electronic mail: bauer@csd.uwo.ca).* Dr. Bauer is Senior Director, Information Technology Services, at the University

of Western Ontario. He is also a professor in, and former chair of, the Department of Computer Science. His research interests include distributed computing, applications of high-speed networks, and software engineering.

Richard B. Bunt *Department of Computer Science, University of Saskatchewan, Saskatoon, Saskatchewan, Canada S7N 5A9 (electronic mail: bunt@cs.usask.ca).* Dr. Bunt is a professor in, and former chair of, the Computer Science Department at the University of Saskatchewan. His research interests include performance evaluation and distributed systems.

Asham El Rayess *Department of Systems and Computer Engineering, Carleton University, Ottawa, Ontario, Canada K1S 5B6 (electronic mail: asham@sce.carleton.ca).* Mr. El Rayess received the B.Eng. degree in electrical engineering from the University of Damascus, Syria in 1992. He is currently pursuing a Ph.D. degree in the Department of Systems and Computer Engineering at Carleton University. His research interests include distributed applications management systems, midware environments, and software performance engineering. He currently holds a research fellowship from IBM Canada's Centre for Advanced Studies.

Patrick J. Finnigan *IBM Software Solutions Division, Toronto Laboratory, 1150 Eglinton Avenue East, North York, Ontario, Canada M3C 1H7 (electronic mail: finnigan@vnet.ibm.com).* Mr. Finnigan is a staff member at the IBM Toronto Software Solutions Laboratory, which he joined in 1978. He received the M.Math. degree in computer science from the University of Waterloo in 1994, and is a member of the Professional Engineers of Ontario. He was principal investigator, at the IBM Centre for Advanced Studies of the Consortium for Software Engineering Research (CSER) project, migrating legacy systems to modern architectures, and is also executive director of the Consortium for Software Engineering Research, a business/university/government collaboration to advance software engineering practices and training, sponsored by Industry Canada.

Thomas Kunz *Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1 (electronic mail: tkunz@uwaterloo.ca).* Dr. Kunz received the Dr. Ing. degree from the Technical University of Darmstadt, Federal Republic of Germany, in May 1994. He is currently an assistant professor at the University of Waterloo. His research interests include load balancing in distributed systems, distributed debugging, management of distributed applications and systems, mobile computing, and reverse engineering and program understanding.

Hanan L. Lutfiyya *Department of Computer Science, Middlesex College, University of Western Ontario, London, Ontario, Canada N6A 5B7 (electronic mail: hanan@csd.uwo.ca).* Dr. Lutfiyya received the Ph.D. degree from the University of Missouri at Rolla in 1992. She is currently an assistant professor at the University of Western Ontario. Her research interests include management of distributed systems, fault tolerance, and software architecture.

Andrew D. Marshall *Department of Computer Science, Middlesex College, University of Western Ontario, London, Ontario, Canada N6A 5B7 (electronic mail: flash@csd.uwo.ca).* Mr. Marshall is a research associate with the MANDAS project at the University of Western Ontario, where he is a doctoral candidate in computer science. His research interests include management of distributed applications and systems, software engineering for distributed systems, and software reengineering.

Patrick Martin *Department of Computing and Information Science, Queen's University, Kingston, Ontario, Canada K7L 3N6 (electronic mail: martin@qucis.queensu.ca).* Dr. Martin is an associate professor at Queen's University. His research interests include data warehousing and resource management in database management systems.

Gregory M. Oster *Department of Computer Science, University of Saskatchewan, Saskatoon, Saskatchewan, Canada S7N 5A9 (electronic mail: oster@cs.usask.ca).* Mr. Oster is a research assistant with the MANDAS project at the University of Saskatchewan. His current work includes building tools for monitoring the performance of distributed systems.

Wendy Powley *Department of Computing and Information Science, Queen's University, Kingston, Ontario, Canada K7L 3N6 (electronic mail: wendy@qucis.queensu.ca).* Ms. Powley is a research associate with the MANDAS project at Queen's University. Her research interests include information repositories and data warehouses.

Jerome Rolia *Department of Systems and Computer Engineering, Carleton University, Ottawa, Ontario, Canada K1S 5B6 (electronic mail: jar@sce.carleton.ca).* Dr. Rolia is an assistant professor in the Department of Systems and Computer Engineering at Carleton University. His research interests include the performance engineering, analytic modelling, and management of distributed application systems.

David Taylor *Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1 (electronic mail: dtaylor@uwaterloo.ca).* Dr. Taylor is a professor of computer science at the University of Waterloo, where he has been a faculty member since 1977. His research interests include distributed-systems software and software fault tolerance. During the 1991–1992 academic year he spent a sabbatical at the Centre for Advanced Studies, IBM Canada Ltd. Laboratory.

Murray Woodside *Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada K1S 5B6 (electronic mail: cmw@sce.carleton.ca).* Dr. Woodside is professor of computer systems engineering at Carleton University, and holder of the OCRI/NSERC Industrial Research Chair in Performance Engineering of Real-Time Software. His research interests include software engineering methods that address performance, and modelling of distributed systems.

Reprint Order No. G321-5656.