Journal of Network and Systems Management, Vol. 8, No. 2, 2000

Configuration Maintenance for Distributed Applications Management

Hanan L. Lutfiyya,[1] Andrew D. Marshall,[1] Michael A. Bauer,[1] Patrick Martin, and Wendy Powley

The MANDAS project has defined a layered architecture for the management of distributed applications. In this paper we examine a vertical slice of this architecture, namely the management applications and services related to configuration management. We introduce an information model which captures the configuration information for distributed applications and discuss a repository service based on the model. We define a set of services and management applications to support maintenance of configuration information, and describe how the different types of configuration information are collected. Finally, we present two management applications that use configuration information.

KEY WORDS: Distributed application management; configuration management; management services.

1. INTRODUCTION

Distributed computing systems typically consist of large numbers of heterogeneous computing devices connected by comunication networks, various operating system resources and services, and user applications running on them. The 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. 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 dis-

[1] Department of Computer Science, The University of Western Ontario, London, Ontario, Canada. E-mail: *hanan{flash,bauer}@csd.uwo.ca*
[2] Department of Computing and Information Science, Queen's University, Kingston, Ontario, Canada. E-mail: *martin{wendy}@qucis.queensu.ca*

1064-7570/00/0600-0219$18.00/0

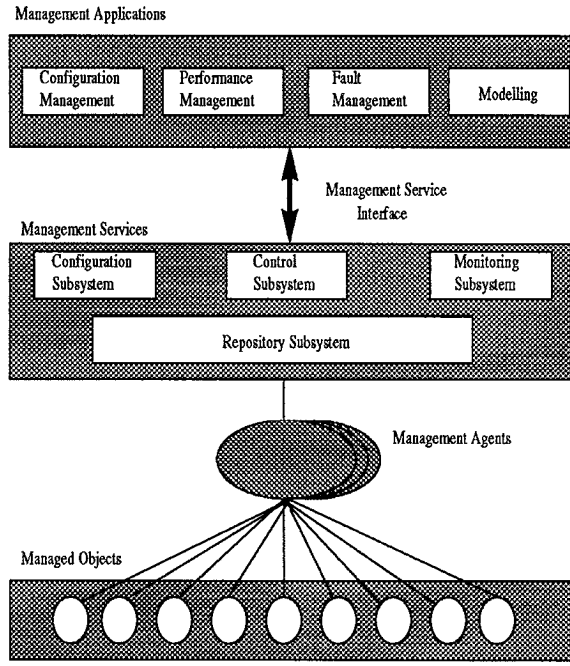

Fig. 1. MANDAS management architecture (adapted from Bauer *et al.* [1]).

tributed applications. We take management to include configuration management, fault management, performance management, and application metrics and modeling.

Figure 1 shows the architecture we have adopted to support applications management [1]. *Management applications* are used to perform management tasks, such as system configuration, analysis of performance bottlenecks, report generation, visualization of network or system activity, simulation, and modeling. The management applications make us of a set of *management services* which we organized as four subsystems: the *repository services subsystem*, the *configuration services subsystem*, the *monitoring services subsystem*, and the *control services subsystem.* The management services in turn interact with *management agents* through various protocols, for example SNMP or CMIP. Each management agent monitors and collects data about a set of *managed objects.*

The development of the management applications, the repository services, configuration services and monitoring services needed for configuration management is the subject of the paper. The remainder of the paper is structured as follows. Section 2 describes the information model used to represent configuration information. Section 3 describes the repository services. Sections 4 and

5 outline the services to maintain configuration information and to collect it, respectively. Section 6 presents two management applications which supply and use configuration information. Section 7 discusses related work, and Section 9 summarizes the paper.

2. INFORMATION MODEL

Based on a study of management systems, we have identified three different types of data in distributed applications management [2]:

- *Structural data*: Descriptions of the objects composing an application, the objects composing the environment in which the application runs, and the relationships among these objects.
- *Measurement data*: Data describing the run-time performance of the application objects which may be collected by monitoring an object (e.g., process CPU or disk use) or may be derived from collected data.
- *Control data*: Data describing the execution state of the application objects, such as the current application and system parameter settings, and the event notifications generated by the objects.

Structural data, which is the type of data discussed in this paper, is the primary input for configuration management and is used by other management applications in locating, requesting, and interpreting measurement, and control data. The structural data for a managed application presents two views of an application: the *runtime view* and the *code view*. The runtime view of an application consists of descriptions of objects related to the execution of the application, that is application instances and processes. The code view of an application consists of objects related to the construction of an executing application, that is source files, data files, interface files, object files, executable files, makefiles, and script files. The structural data for a managed application also contains descriptions of the environment in which an application executes. Environment objects include objects related to management, such as management agents and sensors, and objects related to the runtime environment such as hosts and network connections.

Structural data typically consists of many object instances of relatively few types, for example there may be many process instances active at one time and all can be described in the same way. These object descriptions can be complex and will typically outline the relationships the object has to other objects. For instance, a process description contains properties like 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 will primarily be via queries using object identifiers or attribute values, or following relationships to other objects. Structural data is also relatively static since it is only updated for "significant events" such as process creation and pro-

cess termination. Finally, the development of an information model is an iterative process, that is, as we gain more experience and understanding of distributed application management, the model will change and evolve. These properties suggest an *object-oriented* information model that provides mechanisms to support classification, generalization and aggregation is the most appropriate representation.

2.1. Modeling Constructs

The role of the information model is to represent the structural data for the managed applications. Our information model is a structurally object-oriented model based on the Telos language [3]. It uses the following constructs:

- *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.
- *Object*: An object is an identifiable collection of attribute values which describe an application entity. Every object has a unique object identifier or name.
- *Class*: A class is a collection of objects that share common properties. An object is related to a particular class via the *instance-of* relationship. Classes are related via the *generalization* or *is-a* relationship.
- *Metaclass*: A metaclass is a collection of classes that share common categories of attributes where a category groups attributes related to a particular aspect of an object. A class is related to a metaclass via the *instance-of* relationship. Metaclasses are related via the *is-a* relationship.

2.2. Modeling Concepts

The information model's class hierarchy is shown in Fig. 2. The model is defined at two levels of abstraction. The nodes above the horizontal line are metaclasses and the nodes below it are classes. Nodes at the same level of abstraction are related by the *is-a* relationship, which is represented by solid lines. The class nodes are *instances-of* metaclass nodes, which is represented by the dashed lines.

Metaclasses, as mentioned earlier, are used to group together classes whose attributes come from the same categories. The root metaclass `RepositoryObjectClass` defines the categories of attributes found in all classes. The metaclasses under `RepositoryObjectClass` introduce attribute categories appropriate for the different collections of objects represented by the classes, that is, hardware objects (`HardwareObjectClass`) such as nodes and networks, runtime objects (`RuntimeObjectClass`) such as processes and application

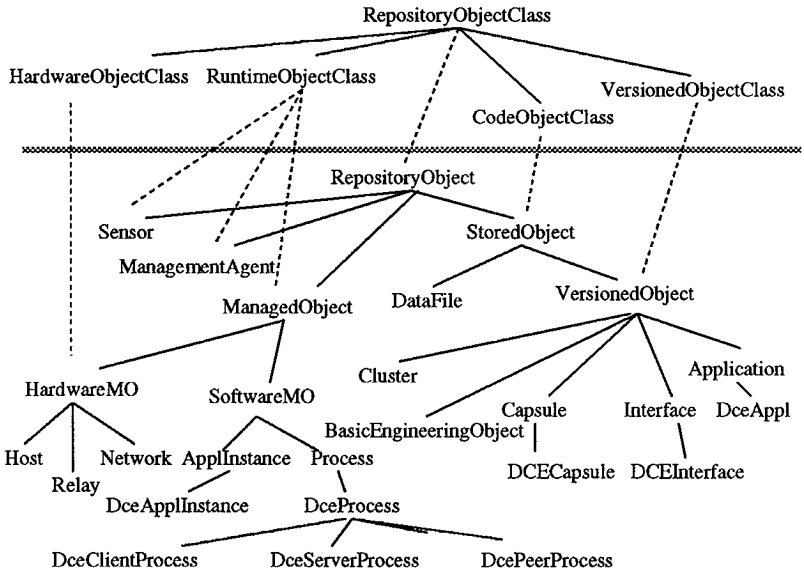

Fig. 2. Information model class hierarchy.

instances, and code view objects (`CodeObjectClass`) such as source files and executables. The `VersionedObjectClass` metaclass defines the categories of attributes related to the version information that is required for some of the code view objects. For example, the `RuntimeObjectClass` metaclass is defined as follows.

```
MetaClass RuntimeObjectClass
isa RepositoryObjectClass
with
  properties
    actions: Proposition;
    notifications: Proposition;
    monitorAttributes: Proposition;
    stateAttributes: Proposition
  relations
    mgmtRelationships: Proposition
end
```

where **actions** include properties related to managing an object, *notifications* are properties which describe the management notifications that can be generated by an object, `monitorAttributes` include the properties collected about an object, `stateAttributes` are the properties describing the current state of an object, and `mgmtRelationships` are links related to managing an object.

The root of the class hierarchy is `RepositoryObject` which defines properties common to all objects in the repository. The two main subtrees in the hierarchy, under `StoredObject` and `ManagedObject`, contain the application objects related to the code view and runtime view of an application, respectively. The other two subclasses of `RepositoryObject`, namely `ManagementAgent` and `Sensor` describe objects in the management system itself that are related to the managed applications.

The code components of a distributed application are modeled as instances of the classes in the subtree under the class `StoredObject` which defines the file-related properties of the objects. The class `VersionedObject` defines the properties related to the maintenance of versions for some components. The classes used to represent the components include the engineering structures of the Reference Model for Open Distributed Processing (RM-ODP) [4]. These structures provide a generic terminology which allows us to represent applications built using different middleware such as the Open Software Foundation's Distributed Computing Environment (DCE) or OMG's CORBA [5]. The different types of code components include the following:

`BasicEngineeringObject` defines the building blocks of the application, namely source files. They contain the procedures or *methods* that perform the processing of the system and a specification of the other software components on which this component depends in order to operate correctly.

Cluster defines groupings of related objects, namely object files. They are an abstraction used in reconfiguring an application.

Capsule defines the executables that make up the application. An executable is instantiated on a host as a process. It also specifies other executables and datafiles that this executable depends in order to operate properly.

`DataFile` defines the data files used by the managed applications.

Interface defines the components that specify the interfaces between executables in the application, for example, the IDL definitions of a client-server interface in a DCE application.

Application defines the managed applications. Each object of the class points to the necessary makefiles and scripts to create and deploy the particular application.

Several of the code view classes (**Capsule**, **Interface**, and **Application**) are fur-

ther specialized to capture particular properties of components of a managed DCE application. A similar kind of specialization can be performed for applications built with other middleware.

The runtime components of a distributed application are defined in the subtree under the class `ManagedObject` which lists properties common to all managed objects and is defined as follows.

```
class ManagedObject
in RuntimeObjectClass
isa RepositoryObject
with
   descriptions
      protocol: String
   stateAttributes
      operationStatus: String
   actions
      startManage: Action;
      stopManage: Action;
      changePriority: Action;
      settingControl: Action
   notifications
      creation: Notification;
      deletion: Notification;
      attributeValueChange: Notification;
      statusChange: Notification
   monitorAttributes
      measurementDataSet: {DataSource}
   mgmtRelationships
      mgmtAgent: ManagementAgent
end
```

`ManagedObject` is the ancestor class of all managed objects (hardware or software) and defines their common properties. In particular, it introduces attributes which describe basic management properties such as the actions to which a managed object must respond, the notifications which a managed object must gen-

erate, the data sources which may be used to store measurement data for the managed object, and the management agent responsible for the managed object. [Note: The notation "{DataSource}" indicates a multivalued attribute.]

The two main types of managed objects of interest are hardware objects (`HardwareMO`) and software objects (`SoftwareMO`). The two types of software objects we manage are application instances (`ApplInstance`) and processes (**Process**). Application instances and processes are further specialized to capture information particular to DCE applications. The relationships between the code and runtime objects of an application are represented as link, or object reference, attributes in the class definitions. For example, the definition of the **Process** class is as follows.

```
class Process
isa SoftwareMO
with
  stateInfo
    PID: Integer;
    PPID: Integer;
    UID: Integer;
    GID: Integer;
    processParameters: {String};
    relocatable: Boolean;
    processPriority: Integer
  dependencies
    componentOfAppInstance: AppInstance;
    instantiatedFrom: Capsule;
    dependsOn: {Process};
    dependedOnBy: {Process};
    usesDataFile: {DataFile};
    residesOn: Host
end
```

The **Process** class can, in turn, be refined to capture properties of processes in particular operating system or middleware environments. For example, the class hierarchy contains several subclasses of **Process** which refine the notion of process for the DCE middleware.

3. REPOSITORY SERVICES

The repository services provide the data management support required by configuration management and the other management applications. We discuss two aspects of the repository service, namely the Management Information Repository (MIR), which implements the information model and stores the structural data, and the MIR Browser.

3.1. Management Information Repository

The MIR, which is shown in Fig. 3, uses a basic client-server architecture. The *MIR Server* has two components: the Telos Repository, which provides the back-end database for the MIR, and the MIR Server Interface, which takes requests from MIR Clients and translates them into requests to the Telos Repository. A *MIR Client* also has two components: an application that requires access to the MIR and the MIR Client Library, which is an Application Programming Interface (API) to the MIR.

The Telos Repository used for the MIR prototype is the University of Toronto implementation of the Telos language [3]. This implementation uses ObjectStore as the underlying storage mechanism. [ObjectStore is a trademark of Object Design, Inc.] The MIR Server Interface, which provides functions for connecting to the repository, submitting and retrieving objects, and querying the MIR, is implemented in C++ runs on an IBM RS/6000 computer on top of OSF DCE [6]. [RS/6000 is a registered trademark of International Business Machines

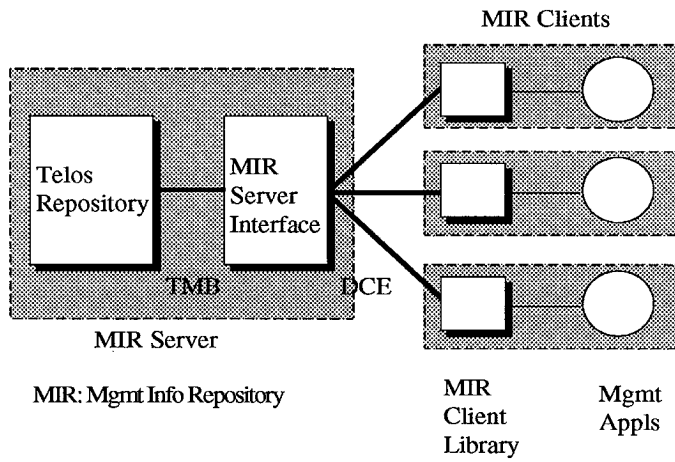

Fig. 3. MIR protoype.

Corporation.] The back-end of the interface communicates with the Telos Repository via the Telos Message Bus (TMB) API. Requests and results are passed along the TMB in the form of *s-expressions* which are strings that are parsed and understood by the Telos Repository. Although theoretically MIR Clients could use the TMB API directly to access the Telos Repository, we decided to build an intermediate layer (the MIR Server Interface) between the Telos Repository and the MIR Clients for the following reasons:

- *Database Independence*: The MIR Server Interface buffers the MIR clients from the specifics of Telos. This will allow us to change the underlying storage mechanism with minimal modifications to the overall system. Only the back-end 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 TMB allows only a single client to access the Telos Repository at a 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 which 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.
- *MANDAS Project Requirement*: One of the original design decisions of the MANDAS project was to interface the various components using DCE.

Clients, which consist of the Client Library and a MIR application, communicate with the MIR Server via DCE Remote Procedure Calls (RPCs). The Client Library is implemented in C++ and uses the C++ API provided by the MIR Server Interface. It presents applications with a view of the MIR which is consistent with the MANDAS Information Model. There are three basic object types used in the MIR: Management Meta Objects (MMOs), Management Schema Objects (MSOs), and Management Data Objects (MDOs). MSOs are similar to classes in an object-oriented programming language. They define the attributes that an MDO may contain. The attributes are further grouped into categories that are defined by an MMO. Each MSO is an instance of one or more MMOs (thus specifying the categories of the attributes), and each MDO is an instance of an MSO.

The MIR Client library provides functions for the following tasks:

- connect to, and disconnect from the MIR
- create, modify, and retrieve MMOs
- create, modify, and retrieve MSOs
- create, modify, and retrieve MDOs
- define query conditions and issue MDO queries

3.2. MIR Browser

The MIR Browser presents a graphical view of the contents of the repository, allowing users to examine the structure of the information model and to see the relationships between the objects. They may also browse the data stored in the MIR.

Figure 4 shows the MIR browser interface. The user connects to the MIR and selects a class or metaclass from which to start browsing the class hierarchy. To see the subclasses associated with a particular metaclass or class, the user clicks once on the object. Double clicking on an object displays the class description for the selected class in the text window to the right.

In Fig. 4, the `VersionedObject` class has been selected as the starting point for browsing the classes. The user has expanded the subtree to show the subclasses of `VersionedObject`. The class description for **Application** is shown in the text window to the right. The description shows the attributes and their types as well as the *instance-of* and *is-a* relationships of the class.

Classes which have associated data objects (i.e., instances of the class) are indicated by the data objects icon (as seen to the right of the **Class** objects in Fig. 4). Selecting the icon produces a text box containing the information for all the data objects associated with the class. In Fig. 4, the **Capsule** data objects icon has been selected and one of the data objects is shown. The user may cycle through the list of data objects, one at a time, by using the "Next" and "Previous" buttons.

4. CONFIGURATION INFO MAINTENANCE

Configuration information, which makes up the majority of the structural data, is stored in the MIR. The Configuration Maintenance Service enables management applications and other management services to retrieve and update the configuration information as a result of changes. This service, which we have implemented as a DCE server, is a client of the MIR and uses the MIR client library to provide registration, query and update services to other management entities.

We can categorize the service interfaces of the Configuration Maintenance Service into four groups: registration, deregistration, query, and modification services. To date, only the first three have been defined.

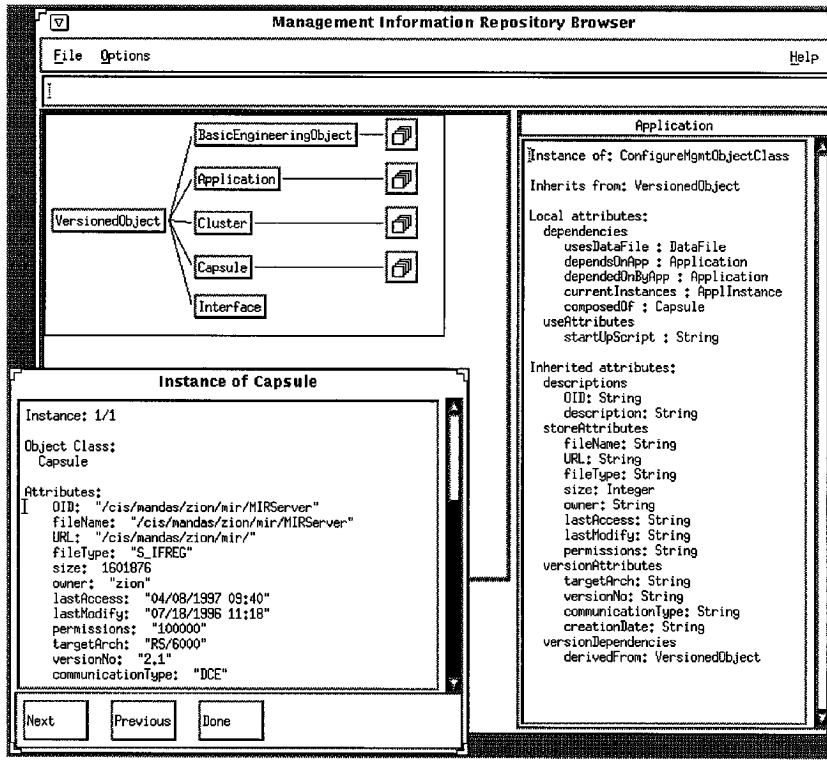

Fig. 4. MIR browser.

Registration Services allow for the registration of various system and application components. The entities that can be registered, and the services responsible for doing so are shown in Table I.

Deregistration Services allow for the deregistration of the distributed system components.

Table I. Registration Services

Registration service	Entity
`RegisterHost`	host
`RegisterExecutable`	executable file
`RegisterApplication`	application
`RegisterApplicationInstance`	application instance
`RegisterProcess`	process

Table II. Query Services

Query routine	Returns
`RetrieveAllApplications`	Information about registered applications in the system
`RetrieveAllCapsulesInApplication`	Information about all executable files pertaining to a specified registered application
`RetrieveAllHosts`	Information about hosts in the system
`RetrieveApplInstInApplication`	Application instances for a specified registered application
`RetrieveProcessesInAppInstance`	Information about a process of a specific application instance
`RetrieveProcessInfo`	Information about a specified registered process
`RetrieveCapsuleInfo`	Information about a specified capsule

Query Services allow other management services and management applications within our management framework to retrieve configuration information from the repository. The services available for retrieving information about system and application components Table II.

Modification Services allow other management services and management applications within our management framework to modify configuration information from the repository. These are similar to the registration service in the number of interfaces and the type of information that can be entered in (a table is not presented in order to save space).

5. COLLECTING CONFIGURATION INFO

This section describes the management applications and management services used to collect configuration information.

5.1. Management Applications

We developed two management applications that collect code-view and runtime information about distributed applications.

5.1.1. Code View Data Extractor

The code view of a distributed application includes configuration information about the source files making up the application, the executables, the interface descriptions, etc. The code view is represented in the information model mainly by the classes in the `StoredObject` subtree including `Application`, `Interface`, `Cluster`, `Capsule` and `BasicEngineeringObject`.

A distributed application may include many source files, many executables and many interfaces. Therefore the information required to describe the code view of a distributed application is likely to be extensive. To populate the MIR with the configuration information manually would be tedious and virtually impossible. The Code View Data Extractor (CVDE) automates much of this process. The CVDE combines input from the user as well as information from the application makefile(s) and from the Interface Definition Language file(s) to automatically extract the configuration information. The tool also performs the numerous MIR updates.

The CVDE prompts the user for general information abou the distributed application such as the application name, the top-level directory of the application code files, the architecture for which the application is intended, and the type of middleware used, for example DCE. It also prompts the user for information on each executable associated with the application such as the executable's name and the makefile and Interface Definition Language files associated with the executable. The CVDE uses the information from the user to guide it as it examines the code files associated with the application, extracts the configuration information from those files, and generates the appropriate objects of the **Application, Cluster, Capsule,** `BasicEngineeringObject` and **Interface** classes.

5.1.2. Configuration Management Application

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

Upon selecting the **Start** option the user is presented with 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 may choose. This list of applications is retrieved using `RetrieveAllApplications`. When the user selects an application, the CMA does the following:

1. Generates a unique application instance identifier for the chosen application and registers the application instance using the `RegisterApplicationInstance` registration service.
2. Retrieves information about the executables needed to start the application using the `RetrieveAllCapsulesInApplication` service.
3. Starts an agent for the application instance and then sends control commands to the agent to cause it to run the executables making up the application on the appropriate hosts.

The user can now retrieve information about the new application instance

by using the `RetrieveApplInstInApplication` and `RetrieveProcessesInAppInstance` service operations.

When the user selects the **Stop Application** option, a list of application instances is displayed. Once the user selects an application instance, the CMA identifies the component processes, using the `RetrieveProcessesInAppInstance` and sends control commands to the agent to cause it to terminate the processes. It also causes the application instance information to be removed from the MIR.

5.2. Monitoring Services

The monitoring services subsystem is responsible for monitoring the behavior of managed objects in a distributed system and determining changes in the configuration. This is done using management agents and instrumentation. An *instrumented manageable process* is an application process with embedded instrumentation. Instrumentation is provided to application developers through a C++ class library. More details on the design and implementation of the instrumentation may be found in Katchabaw *et al.* [7, 8]. The instrumentation components are described later and are depicted graphically in Fig. 5.

- 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.
- *Instrumentation code* provides an internal view of the managed process. There are two types of instrumentation code: *sensors* and *actuators*. Sensors encapsulate measurement data. They collect, maintain, and (perhaps) process this information within the managed process. Sensors exhibit monitor-like qualities in that they encapsulate measurement data and provide an interface through which probes (see later), 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.

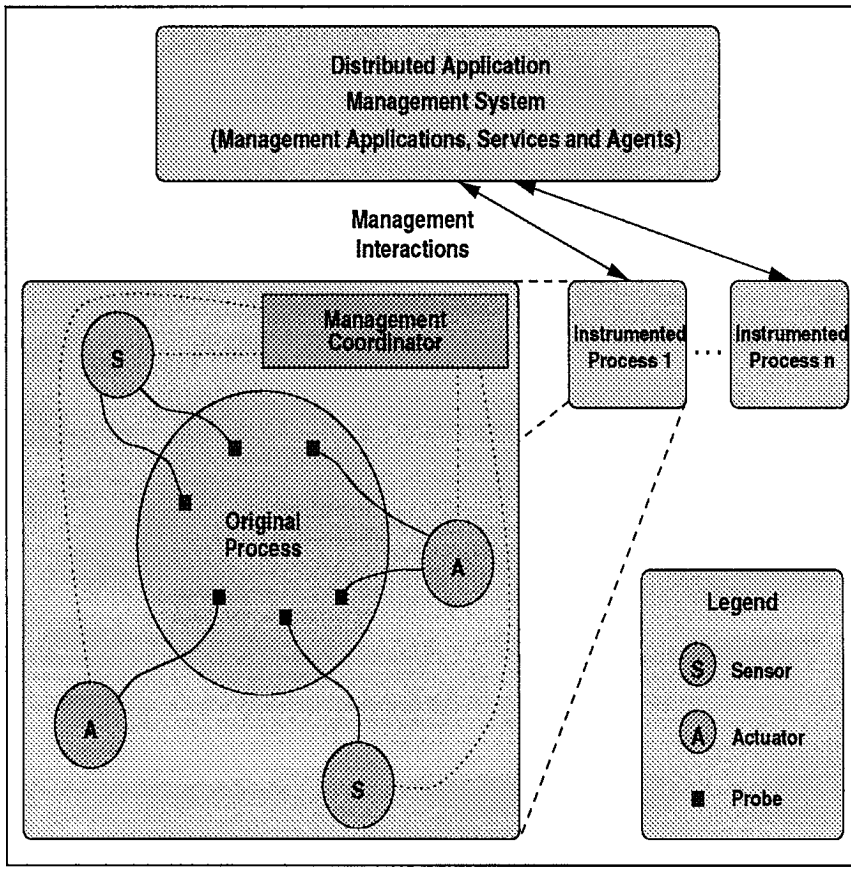

Fig. 5. Instrumentation architecture and environment.

The second type of instrumentation code is an actuator. The actuator encapsulates management functions which 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 injected, during development, into the instruction stream of the application at locations called *probe points.*

For the purpose of monitoring process start-up, we have a probe point called `Process_instrumentionInit` which is activated at process registration. This probe does the following:

1. *Retrieves the binding handle of a management agent.* This *binding handle* contains the information needed by a process to establish communication with the appropraite management agent.

2. *Registers the process with the agent.* The management agent 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, start time, executable that the process is instantiated from and the application instance. The management agent can then update the MIR using the `RegisterProcess` service.

Another probe called `Process_Terminate` is used to notify an agent of the termination of a process.

5.3. Operation

We now describe the operation of these services and management applications, highlighting the interactions of the management components that are in place. We do this by describing the sequence of operations that takes place when an example application is started.

To illustrate the operation, we will use a simple scheduling application that has the following characteristics:

- A server maintains a database of stored information. Users run a client program to schedule events involving themselves and other users, view their calendar, and respond to events being scheduled for them to attend.
- The client executable is compiled from the following source code files: `bind.c`, `calfuncs.c`, `delete_user.c`, `display_event.c`, `error.c`, `misc.c`, `new_event.c`, `new_user.c`, `show_users.c`, `term_server.c`, and `xscheduler.c`.
- The server executable is compiled from the following source code files: `client.c`, `queues.c`, `scheduler.c`, `scheduler_init.c`, `scheduler_mgmt_auth.c`, `scheduler_util.c`, and `server.c`.
- The interface is in `scheduler.idl`.

Each of the `.c` and `.idl` files are represented by an instance of the `BasicEngineeringObject` class. `scheduler.idl` is represented by an instance of the **Interface** class. The client and server executables are represented by an instance of the **Capsule** class. An instance of the **Application** class points to the necessary makefiles to create the `client` and `server` executables.

We now describe the sequence of operations that take place when an application instance is started.

1. Information about the application, which includes the list of executables

needed, are stored in the MIR. This information is gathered and stored using the CVDE prior to starting an application for the first time.

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. After the user selects an application, the following takes place:
   (a) A unique application instance identifier is generated for the chosen application and the application instance is registered using the `RegisterApplicationInstance` registration service.
   (b) Information about the executables is retrieved using the `RetrieveAllCapsulesInApplication` configuration service which returns information that includes the names of the executable files and the host on which each executable file should run. A management agent is started and control commands are sent to it to cause it to run the executables making up the application on the appropriate hosts.

   In our example, the **Start Application** uses the **Application** class to determine the scripts needed for startup. In this particular case, only the server is started. It is assumed that clients are started when needed.
3. When a process starts, it executes `Process_instrumentionInit` which does the following:
   (a) *Retrieves the binding handle of the agent.* The *binding handle* is a component-dependent structure that consists of information necessary to establish communication between processes. The information that the binding handle has includes host machine address, port number, etc. This is needed so that a process knows how to contact the agent responsible for managing it.
   (b) *Determines the executable.* Currently, this is implemented by having the executable name passes as a command line argument.
   (c) *Determines the application instance.* Currently, this is implemented by having the application instance identifier passes as a command line argument.
   (d) *Registers with the agent.* This notifies the agent of the existence of a new process, and provides basic information about the process that includes its name, process identifier, host, port number, start time, executable from which it was instantiated, and the application instance identifier.

At this point, information about an application instance and the processes that it consists of may be retrieved from the MIR using `RetrieveProcessesInAppInstance` and `RetrieveProcessInfo` services. In our example,

assume that a process was started from the server executable and that three processes were started from the client executable. If any management application or other management entity knows the application instance identifier of the scheduler application then it can retrieve all the processes of the application instance using `RetrieveProcessesInAppInstance`. If more information about processes is required it can be retrieve using `RetrieveProcessInfo`.

6. EXPERIENCE

A key aspect of our work has been to evaluate the proposed management services via prototypes and experimentation. This was done by examining management applications and developing a prototype of the management applications that made use of our configuration services and management applications. A prototype implementation demonstrating some of our ideas and concepts was shown at CASCON'96. The prototype platform was OSF/DCE [6] which was chosen because of its availability and our past experiences with it. We will now describe the prototype management applications that make use of the configuration services.

6.1. 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.

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 [9–11] 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.

The models require information about remote procedure call interactions between application objects that includes type of process (i.e., client, server or database). The `RetProcessInApplInstance` service is used to retrieve the processes in the application instance that is being examined. The `RetrieveProcessInfo` is used to retrieve information about a process. This information includes the executable that the process was started from. The `RetrieveCapsuleInfo` allows for the retrieval of link attributes that represent relationships between **Capsules.** These three services are used to deduce the application process dependencies.

6.2. Automating Fault Location

One aspect of distributed application management is fault management—detecting that the behavior of an application has deviated from the specification of its desired behavior. 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 [12], that automates the process of fault location. The steps in the fault location 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 services to determine the configuration of applications and hardware (for example, knowing the communication path between two processes determines 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).

7. RELATED WORK

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

7.1. Information Models

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

- ISO Management [13–15]
- Internet SNMP styled management [16–19]
- A CORBA based approach [20]
- An OSF/DCE based approach [6]
- DMTF/DMI [21]

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 which cover some of application management as defined in this paper. Other has work has recently appeared, but the focus is limited [22].

The IETF has proposed two extensions to its Management Information Base

(MIB) structure called the *Systems Application MIB* [23] and the *Application MIB* [24]. The System Application MIB maintains information about applications that have been installed and run 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 applications. In particular, it adds information on open files, open connnections 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 which are not used in the MIBs. Our model provides more detailed descriptions of the code and runtime views of a distributed application than can be formulated with the MIBs. Our model also describes the distributed runtime environment of an application while the MIBs can only describe each local node individually.

The DMTF has proposed the *Management Information Format (MIF)* [25] which is similar to MIBs with respect to its modeling capabilities and which has the same shortcomings as MIBs when compared with our model. We note, however, that Tivoli Systems has extended the MIF specification to cover distributed applications in their *Application Management Specification (AMS)* [26].

In another effort, the DMTF has proposed the *Common Information Model (CIM)* [27] which, in the same way as our model, applies object-oriented modeling techniques to network and systems management. A management schema in the CIM consists of a set of classes which defines a common framework for a description of the managed environment. The management schema is divided into three conceptual layers:

- *Core schema* captures notions applicable to all areas of management.
- *Common schema* captures notions that are common to particular management areas but independent of a particular implementation. The common areas are systems, applications, databases, networks and devices.
- *Exension schema* captures implementation-specific extensions of the common schema.

The CIM Application Schema Definition, developed by the DMTF Application Management Working Committee [28], 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 runtime environment. Our model, on the other hand, provides a more complete description of a distributed

application which can be used for configuration management, fault management, and performance management.

The only other work we have found that does model the distributed runtime environment is by Weinstock and Tewari [29]. They use an object model-to-model their environment. This work focuses on the modeling requirements and little has been done in determining how to collect and store the data defined in the model.

7.2. Configuration Services

A review of the academic literature and current research [30–34] on configuration management for distributed applications has been 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.

A review of the commercial side shows that Tivoli Systems [35] and Computer Associates [36] have products (based on one of the standards) that provide services for installation and version tracking.

8. DISCUSSION

Our work has taught us a number of lessons about the operation and performance of configuration management services and applications that we are using to extend our work.

- 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. We are working on tools that facilitate the instrumentation processes. We are currently developing a tool that allows a developer from a graphical user interface (GUI) to pick probes and points in a program that those probes are to be embedded into. The tool automatically does the insertion and generates a makefile. An extension of this will use the IDL to determine where the probe points are. By providing these types of tools we allow for the creation of manageable applications without putting an undue broken on the developer.
- The instrumented application processes should not be a burden on scarce

system resources due to excessive management overhead. We made use of a Distributed Applications Management Testbed [37] to carry out some initial performance measurements and analysis. In these experiments, it was found that using instrumentation to manage an application accounted for an average increase of only 3.4% (0.51 seconds) per application remote procedure call. This is quite small when compared to the effects of changing the application workload, a difference of 31.9% (4.2 seconds) per call. While the impact on responsiveness was relatively small, there was a more significant impact on resource utilization, with increases ranging from 7.4–42.4%, depending on the resource considered (memory and CPU utilization formed the two extremes here).

In other words, we could conclude that, given sufficient system resources, our instrumentation prototype is practical in light of its low impact on application responsiveness. However, if resources are scarce, or a heavy system load is causing resource contention, management may significantly impact performance. Further performance experimentation is still needed.

We should note that no instrumentation makes the application more difficult to manage.

- We continue to evaluate and refine our information model and configuration services by developing addition management applications. Currently, we are developing a management application that supports process resiliency. This is done by having a primary server process and backup server processes. The primary and the backup execute on separate nodes. The goal of the backup server process is to take over the role of the primary process, in case the primary process fails. This means that client processes need to be notified of this as well as the management information repository. The management information repository is examined to determine those processes that depend on the primary server process. These processes are then notified of the change the management information repository is updated.

9. SUMMARY

In this paper we have examined a vertical slice of the layered management architecture, namely the management applications and services related to configuration management. We introduced an information model which captures the configuration information for distributed applications and discussed a repository service based on the model. The information model is a structurally object-oriented model which captures both code and runtime information about a distributed application. We defined a set of services that should be available in the configuration services subsystem to support maintenance of configuration infor-

mation and how the configuration services, management applications and the repository service interact together using a sample distributed application. We then described how the different types of configuration information is collected. We presented two management applications that have been built which collect configuration information and we discussed how the monitoring services subsystem is used to determine changes to the configuration. Finally, we presented our current work based on our experience.

ACKNOWLEDGMENTS

The authors wish to express their gratitude to the anonymous reviewers whose feedback helped to improve the quality of the paper. The authors would also like to thank IBM Canada Ltd and the National Science and Engineering Research Council of Canada (NSERC) for their financial support.

REFERENCES

1. M. Bauer, P. Finnigan, J. Hong, J. Rolia, T. Teorey, and G. Winters, Reference architecture for integrated distributed systems management. *IBM Systems Journal*, Vol. 33, No. 3, pp. 426–444, 1994.
2. P. Martin and W. Powley, A management information model for distributed applications management. *Proc. CASCON'96*, Toronto, Canada, pp. 54–63, November 1996.
3. 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.
4. K. Raymond, Reference model of open distributed processing: A tutorial. In J. De Meer, B. Mahr, and S. Storp, (eds.), *Open Distributed Processing II* Elsevier Science B.V., North-Holland, pp. 3–14, 1994.
5. Randy Otte, Paul Patrick, and Mark Roy, *Understanding CORBA*, Prentice-Hall Inc., 1996.
6. Open Software Foundation, *DCE User's Guide and Reference*, 1992.
7. Michael J. Katchabaw, Stephen L. Howard, Andrew D. Marshall, and Michael A. Bauer, Evaluating the costs of management: A distributed applications management testbed, *Proc. CASCON '96*, Toronto, Canada, November 1996.
8. M. J. Katchabaw, Stephen 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, *Second International IEEE Workshop on Systems Management*, Toronto, Canada, pp. 104–111, June 1996.
9. J. Rolia, V. Vetland, and G. Hills, Ensuring responsiveness and scalability for distributed applications, *Proc. CASCON '95*, Toronto, Canada, pp. 28–41, November 1995.
10. J. A. Rolia and K. C. Sevcik, The method of layers. *IEEE Transactions on Software Engineering*, Vol. 21, No. 8, pp. 689–700, August 1995.
11. C. M. Woodside, J. E. Neilson, D. C. Petriu, and S. Majumdar, The stochastic rendezvous network model for performance of synchronous client-server-like distributed software. *IEEE Transactions on Computers*, Vol. 44, No. 1, pp. 20–34, January 1995.
12. C. Turner, Fault location in distributed systems, Master's thesis, The University of Western Ontario, September 1995.

13. ISO, Information Processing Systems, Open Systems Interconnection, Basic Reference Model, Part 4: Management Framework International Organization for Standardization, International Standard 7498-4, 1991.
14. ISO. Information Processing Systems, Open Systems Interconnection, Common Management Information Protocol Specification. International Organization for Standardization, International Standard 9596-1, 1991.
15. ISO. Information Processing Systems, Open Systems Interconnection, Systems Management Overview. International Organization for Standardization, International Standard 10040, 1991.
16. J. D. Case, M. Fedor, M. L. Schoffstall, and C. Devin, Simple Network Management Protocol (SNMP). The Internet Engineering Task Force, May 1990. Request for Comments 1157.
17. 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 Inernet Engineering Task Force, April 1993. Request for Comments 1441.
18. 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, April 1993. Request for Comments 1442.
19. M. T. Rose and K. McCloghrie, Structure and identification of management information for TCP/IP based Internets. The Internet Engineering Task Force, May 1990. Request for Comments 1155.
20. Object Management Group, The common object request broker architecture: Architecture and specification, 1991. OMG Document No. 91.12.1.
21. The Desktop Management Task Force, Desktop management interface specification, The Desktop Management Task Force, 1994.
22. C. Gbaguidi, S. Znaty, and J-P. Hubaux, Multimedia resource: An information model and its application to an MPEG2 video codec, *Journal of Network and Systems Management*, Vol. 6, No. 3, pp. 313–331, September 1998.
23. Internet Engineering Task Force, Definitions of system-level managed objects for applications, Internet Draft, April 1997.
24. Internet Engineering Task Force, Application management MIB, Internet Draft, March 1997.
25. Desktop Management Task Force, Desktop management interface specification Version 2.0, March 1996.
26. Tivoli Systems, Applications management specification 1.1. Retrieved from *http://www.tivoli.com/amsreg/AMS+Spec+Registration.html*, July 1997.
27. Desktop Management Task Force, Common Information Model (CIM) Version 1, April 1997.
28. Desktop Management Task Force, The common information model application schema definition, February 1997. Retrieved from *http://hplbwww.hpl.hp.com/people/arp/dmtf/papers/applicationschema/appman5.htm*, July 1997.
29. J. A. Weinstock and R. Tewari, A general object model for the management of distributed systems, *Second International IEEE Workshop on Systems Management*, Toronto, Canada, pp. 104–111, June 1996.
30. F. Cristian, Automatic reconfiguration in the presence of failures, *IEEE Software Engineering Journal*, Vol. 8, No. 2, pp. 53–60, March 1993.
31. M. Endler, A model for distributed management of dynamic changes, *Fourth IFIP/IEEE International Workshop on Distributed Systems: Operations and Management DSOM'93*, Long Branch, New Jersey, October 1993.
32. F. Faure and D. Marquie, Service dynamic management: A configuration micromanager, *Fourth IFIP/IEEE International Workshop on Distributed Systems: Operations and Management DSOM'93*, Long Branch, New Jersey, October 1993.
33. Christine Hofmeister, Elizabeth White, and James Purtilo, Surgeon: A packager for dynamically

reconfigurable distributed applications. *IEEE Software Engineering Journal*, Vol. 8, No. 2, pp. 95–101, March 1993.
34. Jeff Kramer, Editorial: Configurable distributed systems, *IEEE Software Engineering Journal*, Vol. 8, No. 2, pp. 51–52, March 1993.
35. Tivoli Systems, Tivoli applications management specification, 1995.
36. Computer Associates, Change and application configuration management product direction. *http://www.cai.com/products/addbm/ccm/ccm.htm.*
37. M. J. Katchabaw, S. L. Howard, A. D. Marshall, and M. A. Bauer, Evaluating the costs of management: A distributed applications management testbed, *Proc. CAS Conference*, Toronto, Canada, pp. 29–41, November 1996.

Hanan L. Lutfiyya is an associate professor of Computer Science at the University of Western Ontario. Dr. Lutfiyya graduated from the University of Missouri-Rolla with a degree in Computer Science. Her research interests are in distributed applications and systems management, distributed systems theory and software engineering. She has convened and chaired a workshop on applications management at CASCON97 and is an active organizer of the Systems Management Workshop.

Andrew D. Marshall is a Research Associate with the MANDAS Project at the University of Western Ontario. He is also a doctoral candidate in Computer Science at UWO. His research interests include management of distributed applications and systems, software engineering for distributed systems and software reengineering.

Michael A. 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.

Patrick Martin is an Associate Professor at Queen's University of Kingston. His research interests include data warehousing and resource management in database management systems.

Wendy Powley is a Research Associate with the MANDAS Project at Queen's University at Kingston. Her research interests include information repositories and data warehouses.