ATM Management Using HP DM CORBA and Java

JEREMY RIXON

Abstract—**This paper is a description of a prototype ATM management system built to demonstrate HP OpenView DM CORBA at the 1996 Network Management Forum in Barcelona. The emphasis of the demonstration was on the features and use of the services provided by DM CORBA. The GUI for the demonstration was implemented using Java. The paper first introduces DM CORBA, then describes the implementation of the prototype management system and a simple video-on-demand service built on top of it, and finally, we offer some comments about what we learned.**

Keywords—**ATM, network management, OpenView, CORBA, Java**

Source of Publication—**Proceedings of the International OpenView Forum Conference, Anaheim, USA, June 1997.**

1 INTRODUCTION

The purpose of the demonstration given at the 1996 Network Management Forum in Barcelona was to show some of the features of OpenView DM CORBA. It is an example of the type of application which can be produced using DM CORBA's distributed, object oriented framework.

The demonstration allows users to view, navigate and query the current topology of a simulated ATM network as well as provision of Permanent Virtual Circuits (PVCs) between hosts in the network.

It does not provide functionality to query the status of network elements or manage individual elements of the ATM network in any way. The demonstration is concerned with network topology management rather than network element management.

A simple video-on-demand service built on top of the management system demonstrates the ability to integrate DM CORBA applications into a wider network and service management model.

2 OVERVIEW

Figure 1 depicts the overall architecture of the system:

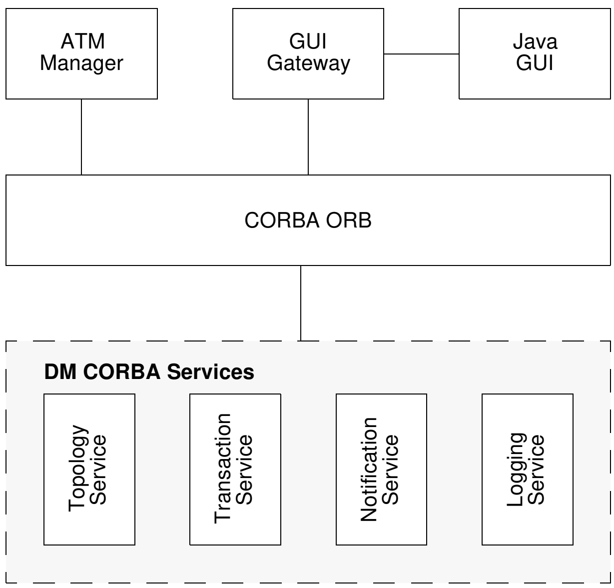

Figure 1: Architectural Components

The principle feature of CORBA DM which is demonstrated is the topology service. The topology service stores relationship information about CORBA objects and provides an extremely flexible query engine for retrieving that information.

Network management is an obvious candidate for demonstrating a topology service; a lot of network management is just remembering what is connected to what in the network and being able to display that information graphically. We chose to manage a simplified model of an ATM network for the demonstration.

Java was chosen for the GUI because of two strong features:

- GUI construction in Java is faster, easier and more robust.
- Java GUIs are inherently very distributable.

To show the distributed nature of DM CORBA, an installation of two hosts was used in the demonstration. Each host simulating and managing an ATM subnetwork.

The demonstration consists of two HPUX hosts, each running: (see figure 2)

- DM CORBA
- An ATM manager
- A GUI Gateway
- Multiple Java GUIs

The following sections describe each of these in turn. After those, a description of a simple video-on-demand service built on top of the management system is given.

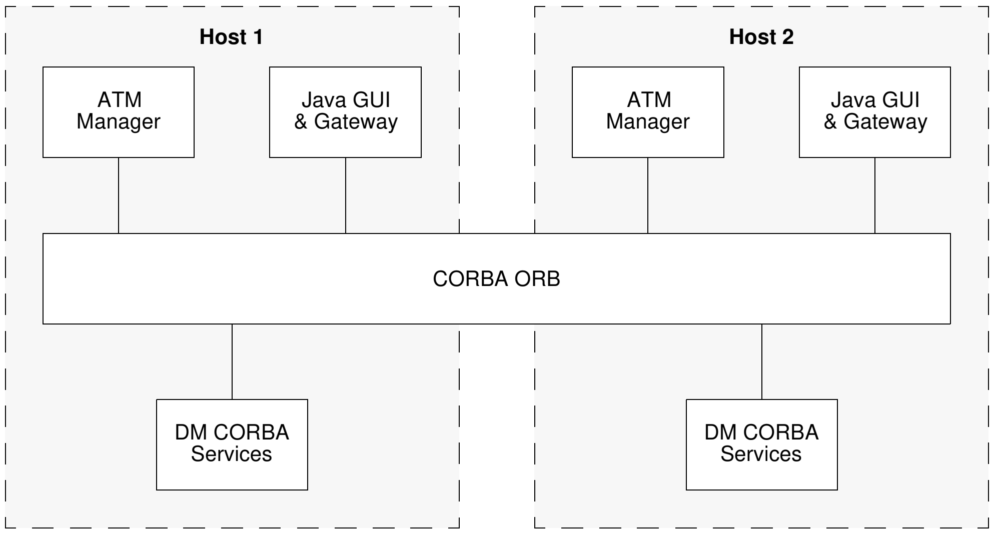

Figure 2: Distribution of services

3 DM CORBA

Since DM CORBA is very new, here is a brief description of the services which are used in the demonstration.

3.1 TRANSACTION SERVICE

The transaction service provides a method safely performing a series operations on objects which may be distributed across many hosts. A transaction has the following (ACID) properties[1]:

- It is **atomic**: it either completes or is entirely undone.
- It is **consistent**: invariant properties of the participants are preserved.
- It is **isolated**: intermediate state during the transaction is not visible.
- It is **durable**: if a transaction completes then the results are permanent.

3.2 TOPOLOGY SERVICE

The topology service stores information about the relationships between CORBA objects. “Associations” are formed between objects. Each object in an association has a “role” in that association. For example, an

1. Refer to the OMG document “CORBAservices: Common Object Services Specification” March 31, 1995 for more information about transactions.

ATM node object might have a “contained” role in a “containment” association with an ATM location (which would have a “contains” role in the association).

The functionality offered by the topology service can be split into three areas: metadata, data and queries.

3.2.1 METADATA

Metadata is the “type” information of the object relationships stored in the topology service.

Every object “managed” in topology has a type. Types exist in a type hierarchy (in most cases, the topological type of an object will simply be its IDL interface name).

Topology metadata restricts how objects may be related to each other by specifying the minimum and maximum (which can be infinity) cardinality of roles in associations. For example, an ATM physical connection object must be associated with exactly two ATM node objects.

Metadata is typically created during the installation of an application.

3.2.2 DATA

Once the metadata is established, relationship information between object instances can be stored.

All operations on the topology service are performed within a transaction. If no transaction is active when the operation is called, one will be started by the topology service. This ensures that relationship information stored in topology remains consistent.

Any restrictions specified on how objects may be related are enforced by the topology service. This ensures that the data stored is always consistent with the object model. The restrictions are checked at the end of a transaction, and the transaction is rolled-back if the state which would be committed is not consistent with the metadata restrictions.

3.2.3 QUERIES

While the relationship information can be retrieved using typical “get me the things related to this” type calls, the topology service offers a much more powerful and flexible interface for accessing this information: the topology query language (TQL).

TQL allows topology information to be retrieved using a text string as the query. For example, the following TQL finds all of the ATMNode objects which are associated with ATMLocation objects which are associated with the specific object named “ATMWorld” (in other words, it finds all the ATMNodes in the world):

```
AENAME "ATMWorld" - ATMLocation - ATMNode
```

The results are returned as a list of subgraphs satisfying the query (i.e., for the above query, the results would be a list of ATMLocation - ATMNode pairs). Alternatively, the results can be returned simply as the union of all the objects in the list of subgraphs - this is very often all that is needed as is this case in the above query (in fact, there is only one case in the demonstration where the results are needed as subgraphs).

Much more complex queries are possible which make use of association types/roles, logical operators or repeating units. A full description of the language is beyond the scope of this paper. Refer to the DM CORBA developer documentation for more information.

A restriction of the language is that all queries must have a starting point, i.e. there must be a reference to a specific object somewhere in the query. This means that you cannot ask for all objects of a particular type, for example. This is why the demonstration has an object “ATMWorld” which is the global starting point for queries.

The reason for this restriction is that global searches with no starting point are very inefficient in a distributed system such as DM CORBA.

3.3 NOTIFICATION SERVICE

The notification service allows objects (notification producers) to emit notifications when an event occurs. Clients can receive these notifications if they have previously registered to receive notifications from that notification producer. Arbitrary data can be passed along with the notification.

3.4 LOGGING SERVICE

The logging service provides a facility for distributed clients to log messages to a unified log. The log presents the illusion of being a single log but is actually stored in a distributed manner across all hosts of the installation. The messages can be retrieved later, with ordering maintained correctly.

4 ATM MANAGER

The ATM manager is a collection of CORBA objects which represent and manage an ATM network.

There are three basic types of objects in the ATM Manager:

- Objects which represent physical elements of the network such as switches, hosts and physical connections.
- Objects which represent logical elements of the network such as PVCs and locations.
- Objects which offer management functions for the ATM network such as PVC construction and deletion.

The topology of the network is stored in the topology service.

Each object has a unique name. A flat namespace is used for simplicity - in a real implementation, structured, hierarchical names would be used to avoid collisions.

4.1 ATM OBJECTS

This section describes all of the objects which exist in the ATM manager.

Many of these objects (ATMNode, ATMPhysConn, ATMPVC, ATMLocation, ATMWorld) have no methods or attributes - they exist solely so that they may have topological information about them stored and manipulated in the topology service. This is due to the bias of the demonstration: we are demonstrating topological management functions and aren't interested in simulating status information etc. of individual elements of the network.

4.1.1 ATMFACTORY

An ATMFactory object exists in all ATM Managers. It creates and deletes all of the other objects in the manager. It also adds the objects to topology on creation and removes them on deletion. Note that by removing the object from the topology service all associations which it was involved in are automatically deleted.

4.1.2 ATMNODE

An ATMNode object represents a node in the managed ATM network. It is an abstract base type with two concrete subtypes: ATMHost and ATMSwitch which are used for hosts and switches respectively.

4.1.3 ATMPHYSCONN

An ATMPhysConn object represents a physical connection between two nodes in the ATM network. It is an abstract base type with two concrete subtypes: ATMSSConn and ATMHSConn for switch to switch connections and host to switch connections respectively (host to host connections are not possible in an ATM network).

4.1.4 ATMPVC

An ATMPVC object represents a Permanent Virtual Circuit (PVC) in the ATM network. A PVC is a logical route for data between two hosts. A PVC must have two hosts as endpoints and traverse at least one switch.

4.1.5 ATMLocation

An ATMLocation object represents a logical location in the ATM network. An ATM network would be divided into locations based on geographic distribution. All ATMNode objects are associated with one ATMLocation. This provides a starting point for operations specific to a location.

4.1.6 ATMConnMgmt

An ATMConnMgmt object creates and deletes PVCs.

PVCs are created by specifying the two host endpoints, the name of the new PVC and, optionally, the path for the PVC. The path for the PVC is specified as a list of ATMSwitch objects. If a path is not specified, the ATMConnMgmt will execute a query in the Topology service to find the shortest path between the endpoints. (This is the one place where the structure of the returned subgraph is retained)

Once the PVC object is created by the ATMFactory, associations are created in topology between it and all of the nodes and physical connections along its path.

PVC Deletion is handled by calling ATMFactory to delete the object (this takes care of removing all associations to the PVC automatically).

ATMConnMgmt objects are “notification producers”. When a PVC is created or deleted, a notification is emitted. The data sent with the notification is a single text string which is suitable for forwarding without any processing to the Java GUIs.

4.1.7 ATMWorld

There is only one ATMWorld object (named “ATMWorld”). All ATMLocation objects are associated with it, this provides a starting point for global operations.

4.2 Object Model

The ATM object model shown in figure 1 maps very easily into topological metadata.

The only difference between the object model and the topological type hierarchy is that a common parent type ("ATM") was used for the topological type hierarchy. This was done for mostly housekeeping purposes: it makes it easy to find all of the ATM objects.

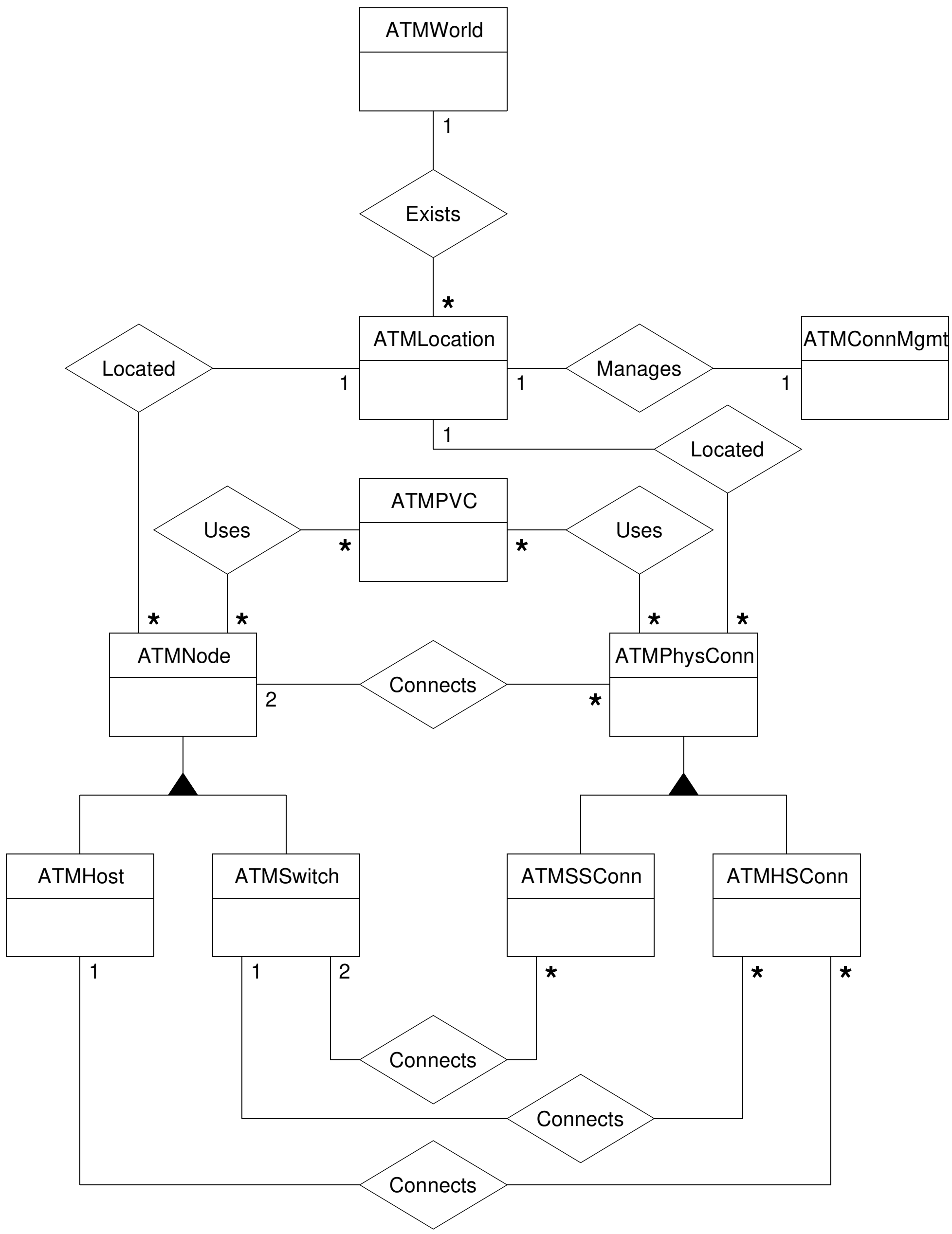

Figure 3: Fusion Model of Association Rules

5 GUI GATEWAY

The GUI Gateway is the glue which allows a Java GUI to communicate with CORBA objects and services. It accepts socket connections from multiple GUIs and has three basic functions: responding to commands from the GUI, passing on notifications from the ATM managers and logging interesting events.

ATM Objects are always referred to by their name when communicating with the GUI.

5.1 GUI COMMANDS

The command protocol between the GUI and the gateway is high-level and ASCII based. As an example, the GUI can retrieve a list of all of the nodes at a location with the command "get_nodes <location>" where

<location> is the name of the ATM location. The response is either a list of nodes and the relationships between them or an error message.

5.2 NOTIFICATIONS

The gateway finds all of the ATMConnMgmt objects (using a topology query) and then registers for notifications from all of them. When it receives a notification, it passes it on to all of the currently connected GUIs (the data is a text string suitable to send directly to a GUI).

5.3 LOGGING OF EVENTS

The following events are logged by the GUI gateway (using the logging service):

- Connection or disconnection of a GUI.
- Receipt of a command from a GUI.
- Receipt of a notification.

6 JAVA GUI

When the Java GUI starts up, it makes a socket connection to a GUI Gateway (the hostname and port number of the gateway are passed to the GUI on invocation). It then requests a list of all ATMLocation objects and then a list of all ATM objects in those locations. In a real management system, only objects in the local location would be retrieved initially, since immediately retrieving all of the information about the entire world is inherently non-scalable.

The GUI then displays a unified view of the whole ATM network and waits for a user to do something. The Java GUI can perform the following actions:

6.1 NETWORK TOPOLOGY DISPLAY

The GUI's basic functionality is displaying the ATM network. Either an ATM subnetwork at a specified location or a unified view of the entire ATM network can be displayed. Switches, hosts, physical connections and PVCs can be shown or hidden as desired. The switches and hosts are shown in two concentric circles with the switches in the inner ring. Lines between hosts are PVCs and other lines are physical connections. Figure 4 shows a unified view of a small ATM network.

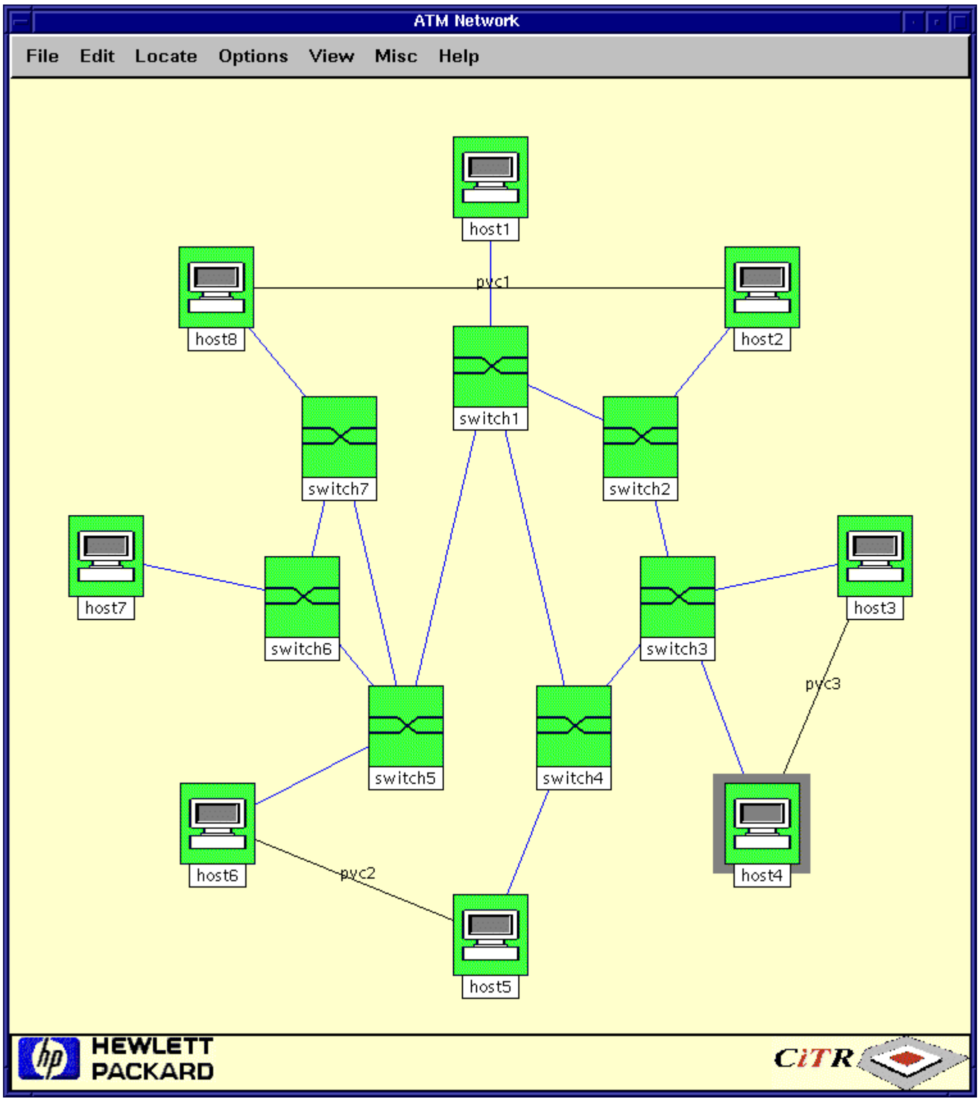

Figure 4: Unified View of ATM Network

6.2 PVC Creation and Deletion

PVC creation and deletion using the GUI is straight-forward. To create a PVC, select the "Add PVC" menu option, type in the name of the PVC and click on the source and destination hosts. To delete a PVC select the "Delete PVC" menu option and click on the PVC.

6.3 Query PVC Path Information

Information about the paths which PVCs follow through the network can be displayed in three ways:

6.3.1 PVC Path

A window showing the hosts, switches and physical connections which a PVC traverses can be displayed by selecting "Show PVC Path" from the menu and clicking on a PVC.

6.3.2 PVCs through switch

A window showing all the PVCs which traverse a switch and their host endpoints can be displayed by selecting "Show PVCs through switch" from the menu and clicking on a switch.

6.3.3 PVCs through physical connection

Similarly, a window showing all the PVCs which pass through a physical connection and their host endpoints can be displayed by selecting "Show PVCs through physconn" from the menu and clicking on a physical connection.

6.4 Asynchronous updating

When a PVC is created or deleted by another operator, a notification is produced by the ATMConnMgmt object which performed the operation. When the GUI gateway receives this notification, it sends a message to all of the connected GUIs so they can update the current display to reflect the changes.

7 Video on Demand

A primary advantage of developing applications using DM CORBA is that integration between them is facilitated by the platform. As a demonstration of this, a simple video-on-demand (VOD) system was developed which made use of the existing ATM functionality.

The following new components were added for the VOD service:

7.1 A new ATM object: ATMVideo

An ATMVideo CORBA object represents a video which is available to be sent over the ATM network to a viewer. When a request is received to show the video, this object does the following:

- Determines the ATM host where the video is to be sent. This is given to the ATMVideo object in the request.
- Queries the topology service to find the host where the video currently resides.
- Calls the ATMConnMgmt object to create a PVC between the two hosts.
- Waits for a notification from the ATMConnMgmt object that the PVC has been created. This is especially important in a demonstration system to ensure that the video doesn't start playing before that PVC appears in the GUI.
- Plays the video. In the demonstration system, this consists of executing an mpeg player and telling it to display on an X terminal specified in the original request for the video.
- Waits for the video to stop playing. In the demonstration, this means waiting for the mpeg player to exit.
- Calls the ATMConnMgmt object to remove the PVC

In our demonstration, we added this object to the ATM Manager process. This was for convenience sake, it could equally run in another CORBA server process.

7.2 A new topological type: ATMVideo

A new topological type "ATMVideo" was created which allowed the associations between ATMVideo objects and the ATMHosts where they reside to be stored in the topology service.

7.3 New commands for the GUI gateway

Two new commands were added to the GUI gateway which allow a GUI to:

- List all of the videos available. The gateway queries topology for this information.
- Show a video. The gateway calls the specified ATMVideo object to show itself on the specified host/xterm.

This, too, was for convenience sake - there are many ways of adding this functionality without modifying the original GUI gateway.

7.4 A new Java GUI: the VODGUI

The VODGUI is a very simple Java GUI which connects to a GUI gateway on start-up, retrieves a list of available videos and presents the titles in a list. When the user selects a video to play, the appropriate command is sent to the GUI gateway.

Two variables need to be specified when the VODGUI is run:

- The ATM host which the user is pretending to be using. This is the destination host for the PVC which will be created before the video starts.
- The name of the X terminal display to show the video on. Since the ATM hosts are only simulated, an actual display name is needed so the video can be shown on a screen.

8 EXPERIENCES

8.1 DM CORBA

8.1.1 BENEFITS

One measure of the usefulness of a platform is how much it reduces development time. It is almost tautological to state that the services provided by DM CORBA dramatically reduced the development time for this demonstration since the functionality of the demonstration was targeted squarely at those very services. It does bear saying, however, that the services provided by DM CORBA are necessary in any object-oriented distributed application and would have to be re-invented in some form or another.

The video on demand service showed that applications built using the DM CORBA platform are easy to extend and to integrate with other applications.

8.1.2 PROBLEMS

While we did find our share of bugs in DM CORBA during development, it was expected that we would since it was alpha code at the time. All the bugs we found have since been fixed.

CORBA clients and servers tend to produce large executable (tens of megabytes) which can cause performance problems on machines with limited RAM.

8.1.3 THINGS WE LEARNED

Use C++ wrapper classes for complex IDL generated code. The IDL to C++ language mapping is very low-level, and not at all pleasant to use directly. A C++ wrapper class around a complex object or structure will make it easier to use and understand. In particular, we wrote a C++ wrapper around the results structure returned by the topology query service which proved its usefulness again and again during development.

8.2 JAVA

8.2.1 BENEFITS

Java is an ideal environment for producing GUIs for distributed systems. A Java GUI can be run on almost any platform. Since our GUI makes a standard socket connection to a gateway process, the GUI can be run in a web browser anywhere on an intranet or even the internet (which raises obvious security issues which won't be dealt with here)

Using Java dramatically reduced the development time for the GUI; A similar GUI developed using Motif or Microsoft Windows would have required much more effort.

8.2.2 PROBLEMS

Java's AWT library is not as feature-rich as we would have liked. For example, we wanted some of the lines in the GUI to be dashed and found that it wasn't supported.

The expectation with Java GUIs is that they will look and act exactly the same on all platforms. This turns out not to be the case; We developed our GUI using the JDK appletviewer but when we displayed it on different platforms using web browsers, the differences ranged from cosmetic to unusable.

ACKNOWLEDGEMENTS

The author would like to thank the many people who have contributed to the demonstration and this paper. In particular: Greg Hall, Peta McRae, Robert Coote, Qinzheng Kong, Graham Chen, Norm Lawler and Jason Etheridge.