Network Modeling For Management Applications Using Intelligent Mobile Agents

Tony White, Bernard Pagurek, Andrzej Bieszczad
(tony, bernie@sce.carleton.ca, bieszczad@lucent.com)

Systems and Computer Engineering, Carleton University
1125 Colonel By Drive, Ottawa, Ontario, Canada K1S 5B6

ABSTRACT

A network model is a fundamental part of a network management solution. Traditionally, network models have provided views with static behavior and limited state of the network elements that they represent. This paper discusses an alternative approach to the creation and maintenance of network models that relies on the use of mobile agents and the principle of delegation. In the intelligent network model proposed, behavior and state are part of the model and both may be dynamically updated. A mobile code environment being used to support the research is briefly described.

1. Introduction

The telecommunication networks that are in service today are usually conglomerates of heterogeneous, very often incompatible, multi-vendor environments. Management of such networks is a nightmare for a network operator who has to deal with the proliferation of human-machine interfaces and interoperability problems. Legacy network management systems are very strongly rooted in the client/server model of distributed systems. This model applies to both the Internet Engineering Task Force (IETF) and Open Systems Interconnection (OSI) standards. In the client/server model, there are many agents providing access to network elements and considerably fewer managers that communicate with the agents using specialized protocols such as the Simple Network Management Protocol (SNMP) [1] or the Common Management Information Protocol (CMIP), see [2], for example. The agents are providers of data to analyzing facilities centered on managers, with a network model used to store aspects of network element state locally; behavior is not dynamically updated. Very often a manager has to access several agents before any intelligent conclusions can be inferred and presented to human operators. The process often involves substantial data transmission between manager and agent that can add a considerable strain on the throughput of the network. The concept of *delegation of authority* [3] has been proposed to address this issue. Delegation techniques require an appropriate infrastructure that provides a homogeneous execution environment for delegated tasks. Although *delegation* is quite a general idea, the static nature of management agents still leaves considerable control responsibility in the domain of the

manager. Intelligent agents [4] and mobile agents seem a natural, and desirable, method for implementing delegation [5], [6], [7] in communication networks. Intelligent networks (IN), in particular, would seem to benefit from the deployment of mobile agents [8], [9].

In this paper we are concerned with the delegation of authority for network model creation and maintenance using mobile agents called *deglets* and *netlets* [10].

The network models that are used in network management systems are, most often, proprietary and provide only views of the components in the network or provide limited behavior for a small set of network elements. This limitation arises as a consequence of the inability of these models to extend their behavior dynamically; i.e. all component classes must be known at deployment time. This makes it very difficult to provide a network model that can deal with heterogeneous components; both cross vendor and cross technology. Figure 1 demonstrates this situation. In the top diagram, an application running on a network management compute platform (NMCP) uses a network model containing virtual representations of the components discovered in the network with only data derived from the management information base (MIB) as might be available through access an SNMP agent.

In the diagram below it, a more desirable situation is shown where the MIB contains behavior that can be exported and used by the network model being constructed.

An emerging technology that provides the basis for addressing problems with legacy management systems is network computing based on Java technology. Java [11] can be considered a technology rather than merely as another programming language as a result of its 'standard' implementation that includes a rich class hierarchy for communication in TCP/IP networks and an industrially supported network management infrastructure (JMAPI) [12], [13]. Jini [14], a recently announced framework for the support of plug-and-play networks, promises to ease the configuration of heterogeneous networks of the future. Java incorporates facilities to implement innovative management techniques based on mobile code. Using this technology and these techniques it is possible to address many interoperability issues and work towards plug-and-play networks by applying mobile agents that can take care of many aspects of configuring and maintaining networks

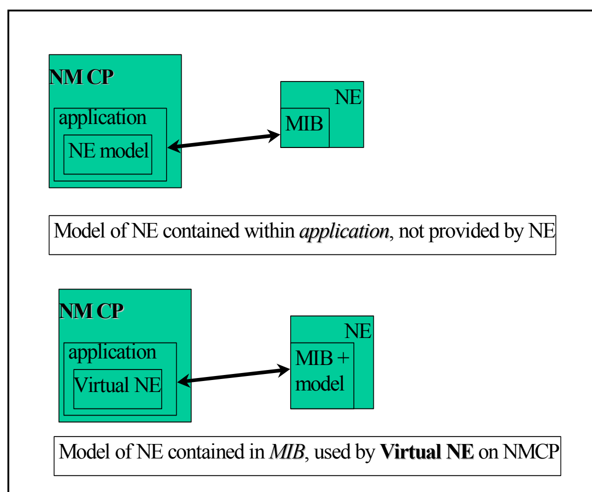

Figure 1: Intelligent and Traditional Network Models

in an autonomous fashion.

This paper continues with a brief description of a Java mobile code environment that has been used to support delegation of network management activity. A section then outlines the process of network model creation by mobile code delegation techniques. Two applications using the framework are then described. Finally, a summary of the key messages of the paper is provided along with the authors' views on the possibilities for mobile agents in Network Management.

2. Mobile Code Framework

A homogeneous execution environment for mobile code is considered extremely advantageous for the agent-based management of heterogeneous networks. Typically, a Mobile Code Framework (MCF) contains the following components [15]:

- mobile code daemon (MCD),
- migration facility (MF),
- interface to managed resources (VMC),
- communication facility (CF),
- security facility (SF).

Certain of these components are shown in Figure 2.

It is assumed that a mobile code daemon runs within a Java virtual machine on each network element. While not possible today, it is the authors' belief that Java-enabled network devices will be common in future products; several vendors are already developing such devices. The mobile code daemon receives digitally signed mobile agents and performs authentication checks on them using the security facility before allowing them to run on the network element. While resident on the network element, mobile agents interact with resources managed by the component through a virtual managed component (VMC). The VMC can be thought of as an enhanced MIB, containing both data and behavior that relate to the device. The VMC is the *only* interface to managed resources and ensures secure access to them. The VMC provides get, set, event and notification

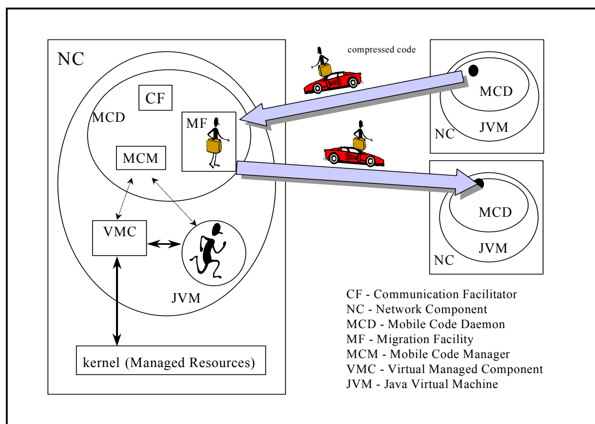

Figure 2: MCF Components

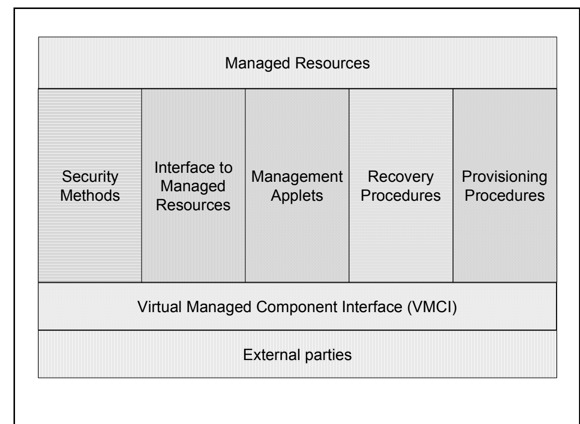

Figure 3: The Virtual Managed Component

facilities (similar to an SNMP agent) with an access control list mechanism being used to enforce security. VMCs are designed to contain information that encompasses repair procedures, provisioning procedures, a managed information base (MIB) and vendor-related information. This is shown graphically in Figure 3.

It is possible that several VMCs may be resident on a device. The advantage here is that separate security policies can be used for the distinct VMCs and sets of agents can be prevented from accessing each other's data.

The migration facility is used to move a mobile agent from one network element to another, either within the same region or between regions. Mobile code environments are connected with default migration patterns in order to form mobile code regions with gateways between them. Individual mobile agents may use the default migration destination or use other algorithms in their choice of migration destination. A communication facility (CF) is provided for inter-agent communication. This comprises two components. A communicator facilitator (CF) provides an inter-agent communication facility on each network element and a mediator (MED) that resides on the mobile code region gateway provides a regional communication facility.

The MCF provides support for different types of mobile code; however, deglets and extlets are relevant to the research reported here. A deglet is an agent that has an activity delegated to it; e.g. the discovery of components in a network. An extlet is an agent that represents an extension of the behavior of the MCF on a particular device; e.g. the installation of behavior associated with a newly discovered component in the network.

While these framework elements provide sufficient facilities for mobility, they apply to the Java language environment. In order for mobile agents to be universally accepted, a language neutral standard is now being developed. The Mobile Agent Services Interoperability Facilities (MASIF) [16] specification, and its first implementation in the GRASSHOPPER [17] framework, promises to provide mobile agent frameworks that inter-work in the future. The MASIF dependence upon the Common Object Request Broker Architecture (CORBA) and CORBA services seems an excellent approach to multi-language mobile agent systems. However, it is unclear, given the increasing number of implementations of mobile agent systems in Java whether the multi-language facilities and execution environments will ever be required.

The research reported here was conducted prior to the acceptance of the MASIF standard.

3. Network Model

The previous section introduced the VMC as being the interface used by mobile agents in accessing the resources of the network element. The network model requires a similar interface on the NMCP that we call a

virtual managed resource (VMR). Together, the VMC and VMR comprise the virtual network element (VNE). This is shown in Figure 4. It is the VMR that the mobile agent constructs when arriving at the NMCP for the purpose of network model construction. The VMR and VMC interact in order to ensure that the view maintained on the NMCP is up to date. As the device vendor potentially provides both the VMC and VMR, the protocol used for interaction may be proprietary. Again, it is intended that remote access to the network element be supported through the VMR in the same way that local access to network element resources is through the VMC. Stated another way, the network model consists of standard parts that are effectors of NM activity in the network.

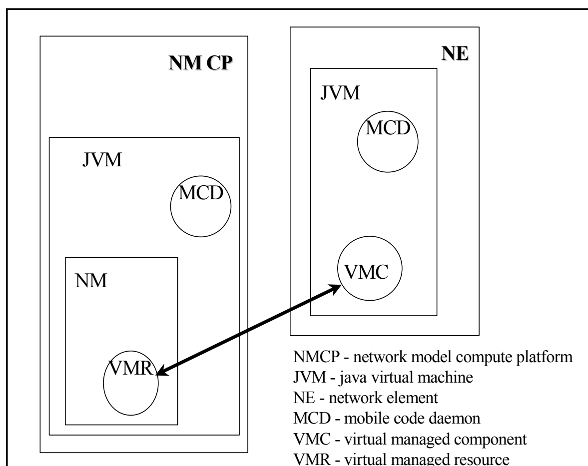

Figure 4: VNE Components

To date, three distinct classes of VMR have been identified. These are:

- Atomic: one-to-one correspondence with a device or service in the network, e.g. switch, router or printer.
- Table: one-to-many correspondence with a device or service in the network, e.g. virtual channel connection (virtual circuit). The table is *homogeneous*.
- Aggregate: one-to-many correspondence with many atomic, table or aggregate VMRs; e.g. a subnetwork.

4. Creation and Maintenance

While the plug-and-play network of tomorrow will demonstrate a high degree of self-configuration and repair, the need to provide a model of the network on a network management workstation will still remain. For example, the presentation of a graphical view of the network will still provide the principal interaction point for network operators managing the network and it is likely that WWW browsers will provide the environment in which such graphical views are displayed. Web based network management is already a fertile area of research and several products are currently available commercially. See, for example, [18]. The difference in future network management solutions is that it is highly likely that a wide range of computational platforms will be used for network management activity. Certain of these platforms will exploit relatively low bandwidth communication with the network such as provided by wireless access. Mobile agents can be used to achieve an 'intelligent' network model that is constructed in real time during the discovery of the properties of a network element.

First, a VMC is injected into the NMCP, this being the network model application. This is achieved using the injection client, shown below. The injection client creates a mobile code object of the requested class (Class name:) in a running mobile code daemon at the specified URL[1] (URL:) with a user-specified name (Mobile code:) and properties (Properties:). The Authority field determines the domain for code sharing that can occur; i.e. all mobile code classes with the PUBLIC authority are defined and resolved within the same network class loader. In a very restricted sense, this is a type of agency.

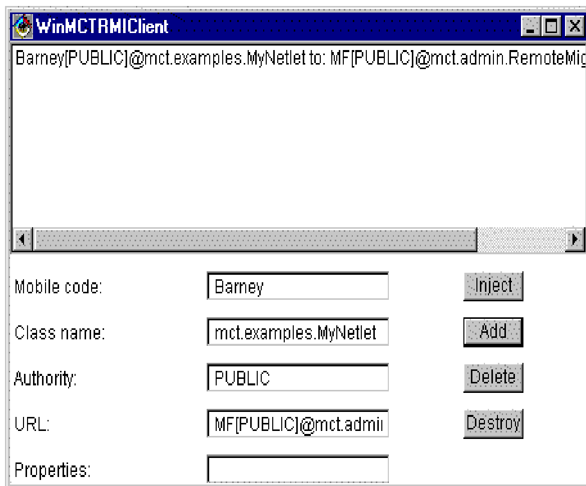

Figure 5: The Mobile Code Injection Client

Referring to Figure 6, a netlet is injected into the network from the network management workstation and, using either a pre-configured itinerary or using a default migration strategy, visits network elements (NEs). The 'discovery' netlet carries with it the IP address of the host that requires a full or partial network model and the name of the network model being constructed. The name of the network model is the name of the VMC injected initially; e.g. mct.vmc.nm. Two forms of 'discovery' have been implemented.

First, the agent is given an itinerary of devices to visit that has been generated by the action of a standard

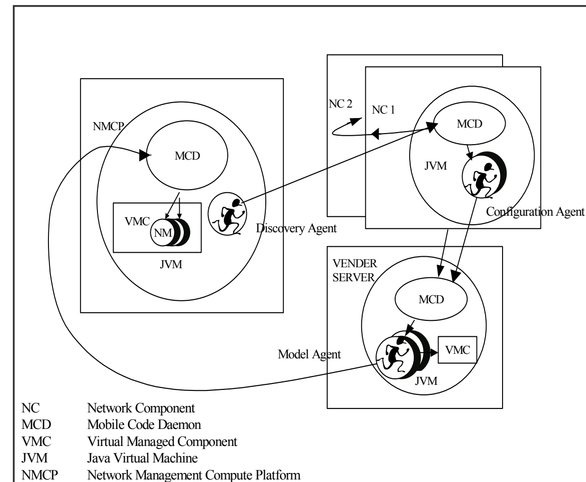

Figure 6: Network model creation with mobile agents

network discovery algorithm. Second, the agent is allowed to follow the default migration path connecting mobile code daemons in the network. In this latter case, it is assumed that the mobile code daemons form a logically connected graph.

The 'discovery' agent may be used to identify all components in the network or may have an associated discovery condition such as, "Include only those components with a utilization above 75% or with less than 1Gb of available disk space." For examples of conditions used in several network model creation scenarios see [19]. Experiments with more complex

[1] For a full discussion of the use of URLs and migration facilitators, the reader should consult http://tony-pc.sce.carleton.ca/mct.html

queries, where reasoning with device parameters is required, have also been performed [20]. The Java Expert System Shell (JESS) [21] has been used in these experiments. At each network element, the netlet interacts with the VMC in order to determine where the vendor maintains a properties site for the device in question. A properties site is a device from which the VMR behavior for the device can be obtained. A standard name, mct.vmc.properties, is used for the VMC containing model-related information. The VMC stores the URL that maintains device information for the particular network element just 'discovered'. The subtree of variables stored within the VMC is shown in Figure 7 below. These variable are accessed using names constructed by concatenating one member from each level in the tree in an analogous fashion to SNMP variable identifiers. Referring to the figure below, we might use ifVMC.ifVMCServer.ifVMCNServers, for example.

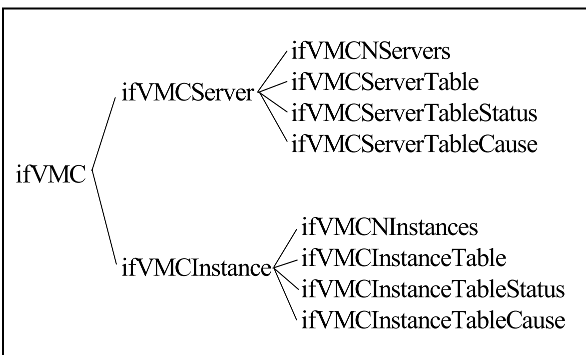

Figure 7: The VMC Variable Subtree

The ifVMCServerTable contains an array of URLs where vendors maintain properties servers. An array is provided in order that, if one is unavailable, another may be used to obtain VMR behavior. The ifVMCServerTableStatus variable contains the status of the table. This variable is used to enable updating of the table to reflect changes in the number of servers and their locations. The ifVMCServerTableCause variable stores information on errors that may occur. The ifVMCInstance subtree stores information related to the VMRs that are constructed for the network element. This subtree is described in more detail in a later section.

For example, while visiting a switch, the VMC would contain the URL of an MCD where the behavior associated with the VMR description for the switch type could be found. A VMR contains information on how to display itself both on textual and graphical displays[2], contains an interface for communication with devices that it represents, and provides an application program interface that can be exploited by network management applications. Once the URL of the vendor server has been established, the 'discovery' netlet spawns a 'configuration' deglet and copies network element state

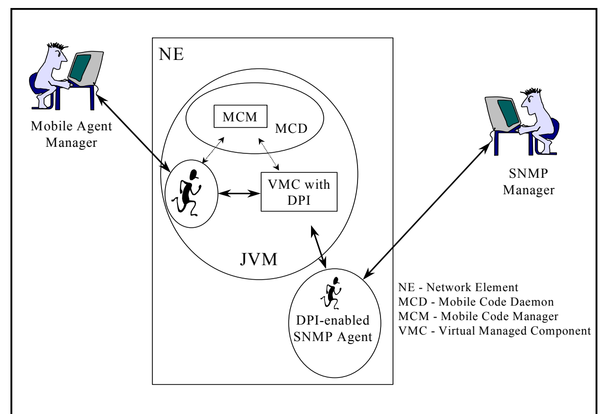

Figure 8: VMC interaction with an SNMP agent

[2] In the future other multi-media support may be added.

information to it.

In the research reported here, network element information has been obtained by querying an SNMP agent on the device. In order to achieve this in a secure manner, a VMC implementing the distributed protocol interface (DPI) [22] has been constructed. The spawned 'configuration' deglet communicates with the SNMP agent through the DPI-enabled VMC in order to acquire information such as device name, model specific information and connectivity with the rest of the network. This interaction is shown in Figure 8. The ‘configuration’ deglet is also given the URL associated with the migration facilitator that is part of the MCD on the workstation requesting the construction of the network model. Enabling VMC-SNMP interaction has also made possible the management of the mobile agent via a conventional management interface. A VMC has been designed that stores information on the number of agents currently resident on the device, the total number of agent arrivals and departures, the average residency time, the average agent bytecode size and several other agent characteristics.

While the 'discovery' netlet continues to explore the rest of the network, a 'configuration' deglet carrying state information from the NE visits the vendor site and requests that a 'model provisioning' deglet be sent to the network management workstation. This is achieved by the having the 'configuration' deglet query a VMC at the vendor site. Once again, a standard name, mct.vmc.vendor, has been chosen for this VMC in order to facilitate its identification. The vendor VMC is queried for the agent bytecodes that are to be used to construct a model agent using the model specific information obtained through the SNMP agent on the network element. The variable name, model.bytecodes.*X*, where *X* is the model name of device for which a VMR is to be constructed, is used to access the bytecodes stored within the vendor VMC. These

```
OnStart() {
  // Get the VMC for the model provisioning agents
  MobileCode mCode = getVMC("mct.vmc.vendor",true);
  if (mCode instanceof VirtualManagedComponent) {
    String model = (String)getParameter("Model");
    String name = "model.byteCodes."+model;
    ByteCode bc = (ByteCode)((VirtualManagedComponent)mCode).get(name);
    // ByteCodes for 'model provisioning' agent obtained, so create an agent container with a
    // unique identifier and instantiate it.
    MCIdentifier id = new MCIdentifier(name);
    MobileCodeContainer mcc = new MobileCodeContainer(id);
    mcc.getJarTable.put(bc);
    instantiate(mcc, getProperties());
}
```

Figure 9: Configuration Agent Code

bytecodes are used to instantiate a new 'model provisioning' agent, passing to it the URL of the workstation where the model is to be created, the name of the model being created and network element specific information. The ‘configuration’ deglet dies on the vendor server. The instantiated 'model provisioning' agent migrates to the workstation where the model is being created immediately on starting, this being achieved with the instantiate call. The properties carried by the 'configuration' agent are transferred to the 'model provisioning' agent using the getProperties() call as shown in the example code for the configuration agent in Figure 9. The onStart() method shown in this figure is the entry point for an agent thread and represents the method where "work" is done by the agent.

On arrival at the NMCP, the 'model provisioning' agent queries the mobile code daemon for the network model VMC using the name passed to it by the 'configuration' agent back at the vendor site. Once found, the 'model provisioning' agent adds a node to the network model using code as shown in Figure 10. The code in Figure 10 identifies the VMC containing the network model, creates a node within the model of the appropriate node class -- found using getParameter("NodeClass") method call - and finally connects the node into the network by establishing links

```
onStart() {
  // First, find the network model name
  String nmName = (String)getParameter("NMname");
  MobileCode mCode = getVMC(nmName,true);
  if (mCode instanceof NetworkModel) {
    NetworkModel = (NetworkModel)mCode;
    // Obtain the domain model (nodes and links)
    DomainModel nm =nm.getBrowser().getDomain();
    // Obtain the interface model: the graphical objects, icons, arcs etc.
    InterfaceModel im = nm.getBrowser().getInterfaceModel();
    // Create the node of the required class and layer in which to install it
    ConfigNode node = nm.createNode((Class)getParameter("NodeClass"), (String)getParameter("Layer"));
    // Install an icon for it
    node.installIcon();
    // Set up the name
    node.setName((String)getParameter("NodeName"));
    // Install entries in its popup menu (as a submenu labeled 'Methods')
    ModelAgentNodeBehaviour nb = new ModelAgentNodeBehaviour("Methods", node, im);
    // Add the links from this node to other nodes
    Enumeration connections = (Enumeration) getParameter("Connections");
    While (connections.hasElements()) {
      String connectedNode = (String)connections.nextElement();
      nm.connectNode(node.getName(), connectedNode);
    }
}
```

Figure 10: Model Agent Provisioning Code

between nodes. A popup menu is associated with the displayed node through the creation of a ModelAgentNodeBehaviour object. It is intended that this popup menu present the network operator with actions that can be used to perform network management activity on the selected node.

Once created, the VMR interacts with the network element, or more precisely the VMC involved in creating it, in order to update the ifVMCInstanceTable that is shown in Figure 7. The table is updated to reflect the fact that a VMR has been created within a network model on the NMCP, the URL of the model being written into the table. In this way, other discovery agents that are looking to create network models on other NMCPs have the option of using the VMR already created or just creating a lightweight, or proxy, VMR that refers to the network model created on another NMCP. This sharing of network models allows NMCP devices with limited resources (e.g. personal digital assistants or PDAs) to have access to network models that require significant memory and processing resources.

To re-iterate, the 'model provisioning' deglet interacts with the VMC on the network management compute platform (or management workstation) for the purpose of constructing a VMR for the part of the network model that is being created. The VMC stores the network model being created; all access to the network model is through an application program interface provided by the VMC. In this way, the network model is a shared resource, available to all visiting agents and applications that run within the MCD. It has also been possible to make the model available to other network management workstations in the network by using Java's Remote Method Invocation (RMI) facility [23]; i.e. by implementing the java.rmi.Remote interface and extending the java.rmi.server.UnicastRemoteObject class. The network model VMC extends the UnicastRemoteObject class and the VMC registers with the local RMI registry when installed. Use of RMI has allowed the network model to be shared between multiple network management workstations. Future research will doubtless implement a similar CORBA facility.

The main work of network model creation is performed by the 'model provisioning' agent. The network 'model provisioning' agent visits the network management workstation and instantiates a virtual network element by interacting with the actual network element in the live network. Since the agent visiting the network management workstation and the VMC on the network element are *both* provided by the vendor, the nature of the protocol used in their communication is unimportant and can be purely proprietary. Only the API presented to applications using the network model needs to be published and, consequently, adhere to de-facto or emerging standards. The API enforced has been designed to be simple and minimal. For example, nodal attributes are stored in a Hashtable, with access via a

getParameter(java.lang.String) method. In this way, a rich structure can be associated with the node without the need for complex documentation of interfaces and it can easily be extended or updated during the lifetime of the node. Another approach, by Knight and Hazemi, [24] has been to use the reflective facilities provided within Java 1.1 to manipulate more complex managed objects; objects which map more closely to the structure of a traditional SNMP MIB. This is a considerably more elegant approach to the attribute addressing problem and we will explore this technology more fully in the near future. Also, unlike traditional network models, the virtual network element provisioning agent installs behavior as well as state for the virtual network element within the network model; hence the use of the term 'intelligent'. Such behavior includes: one or more icons to be used in the display of the node; popup menus associated with the displayed node; forms that allow operator input of parameters that can be used for service provisioning; interfaces to diagnostic agents[3] that can be run on the network element and other network management facilities. This is an example of mobile agents used for the creation of middleware.

While the 'model provisioning' agent creates a VMR for the discovered component, the ‘discovery’ netlet continues to migrate around the network and reports new network elements as they are found. An alternative strategy is to have the ‘discovery’ netlet die when it migrates to a network element that it has already visited and then to inject another netlet at some later time; i.e. probe the network periodically for new or modified components. A further option, and a direction for future research, is to use Jini and JavaSpaces in order to maintain federations of devices and have the NMCP automatically notified of changes in the federation.

A network model constructed in this way has a number of distinct advantages when compared to conventional network model solutions. First, the network model is a managed resource on the management workstation -- accessed through a VMC on that platform -- and is not part of any one network management application. In the authors' research, a Java graphical network display has been constructed that provides multiple layered views of the network model. This is described in the next section. Second, vendor-supplied agents provide VMR behavior and state and so a heterogeneous network can potentially be modeled. This raises the possibility for the demise of the need for the workstation to use any standard protocol for accessing the network element in that all interactions should occur via the VMR and the vendor has implemented both VMR and VMC. In essence, Jini promises this with its publish and subscribe approach to device and service management. Third, the VMR supports the concept of delegation within the object. Making an application object a *delegate* of the VMR may extend the behavior

[3] A fact exploited in an application described in a later section

of the VMR. A delegate object handles messages initially sent to the VMR for which the VMR does not have behavior. This is made possible by use of the reflective capabilities within the Java language; i.e. an object can determine the behavior of a class, and the ability to handle run time exceptions. An exception is handled, and the message passed to a delegate object which does support the interface. In this way, the behavior of VMRs can be modified dynamically by applications. Fourth, by providing the VMRs with communication and display capabilities, element level provisioning and configuration is (potentially) much simplified as the VMRs can access applets, servlets and other utilities maintained on the vendor's server. Finally, VMRs can register for notification of state changes with the device in the network and applications can register with the VMC on the management workstation for notification of changes in the network model (an attribute of the VMC). Hence, fault management applications can be generated for the heterogeneous network.

5. Network Model Visualization

A network model can be used by many applications. However, the visualization of that model is arguably the most important. In traditional network management systems a graphical network browser provides the focal point for network operator interaction with the network. In the research reported here, a Java graphical network browser was created capable of interacting with the mobile agent framework and extendible by mobile agent actions.

Several frameworks were examined during the research reported here. Two frameworks were found to be useful. These frameworks were used as a template and building blocks for the browser framework. The two frameworks are the Graphical Editing Framework (GEF) and the Dynamic Diagram Framework (DDF) [25]. GEF [26] is a framework used for building graphical editing tools. GEF was written in Java and was used as the set of foundation class, on top of which the browser was built. The second framework is the Dynamic Diagram Framework. DDF is a tool written in Smalltalk to create network editors. DDF incorporates the notion of separating the details associated with domain specific information from the details of the user interface. DDF, shown in Figure 11, provides a set of classes for creating application domain models and application interface

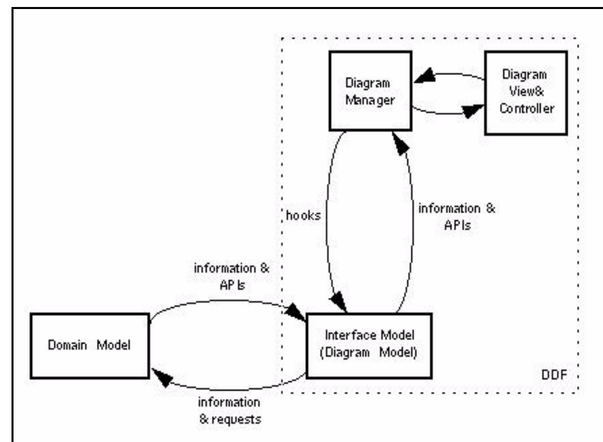

Figure 11: DDF in the GUI Architecture

models. DDF provides hooks for the two models to communicate through. DDF was used as a template in the design of browser framework. Figure 12 summarizes the relationship between the two frameworks.

As Figure 12 shows, the Graphical Network Browser (GNB) is a framework built on top of GEF. The structure of a DDF application is separated into two main models: an *application interface model* (AppModel) and an *application domain model* (DomainModel). In order to provide an interface to the mobile agent system, an *agent interface model* (InterfaceModel) is also present. This latter model hides the implementation details of the graph editing framework from the mobile agent system. The interface model deals with all the graphical aspects

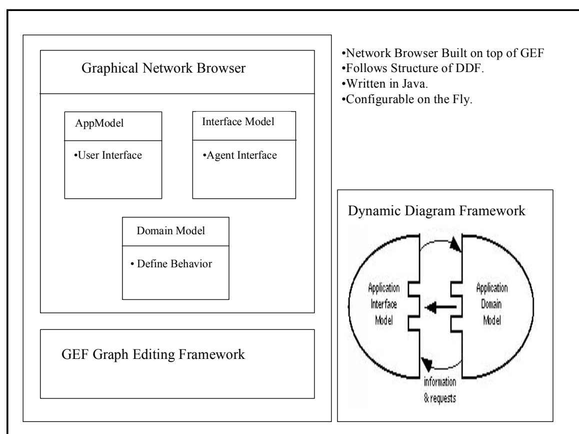

Figure 12: The Graph Editing Framework

of the application. The interface model contains the objects that make up the interface itself. These objects control and define the interface windows, window widgets, the behavior associated with actions on the widgets and any necessary representation of the information displayed. The domain model deals with representing the domain specific information. The domain model contains objects which make up a representation of the data in the application domain itself and the definitions of the behavior that are necessary to manipulate the domain data. The interface model can be separated into smaller blocks, a diagram model, a diagram manager and a view controller. Figure 11 depicts this architecture.

The interface model presents a set of APIs to the domain for manipulating the diagram. The interface model hides the complexities of the diagram manager and the controller from the domain. The interface model and the domain model communicate through a series of requests and notifications. When the diagram is changed, the interface notifies the domain and the domain will respond appropriately. For example, when a node is selected in the diagram, a notification is sent from the interface to the domain to select the domain's representation of the node. The interface may then request additional actions to be carried on the node. The communication between the diagram manager and the interface model is carried out in a similar manner.

The browser framework is organized into three subsystems. The three systems comprise an application model, a domain model and an interface model. The application model is responsible for all the graphical aspects of the tool. The domain model contains the domain specific information. The domain model contains objects that provide the functionality of the tool.

The interface model is responsible for interfacing to the attached applications and visiting agents as well as interfacing to the network. Figure 13 shows a much simplified information model. In this figure, the Apps abstract class refers to mobile agents that link themselves into the browser application at run time. It should be noted that only the principal relationships are represented in this figure for reasons of clarity.

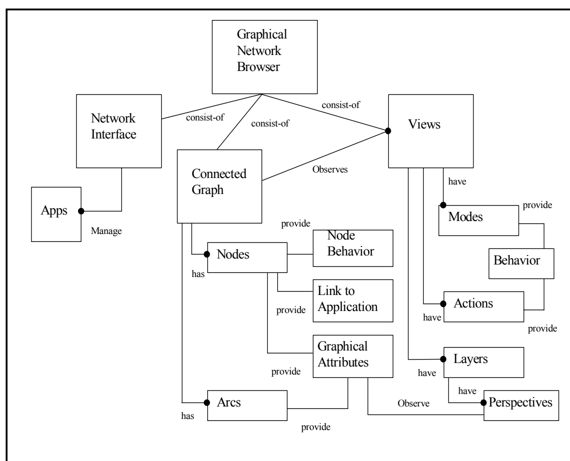

Figure 13: GNB Information Model

6. Applications

Two applications using the mobile agent creation of a network model have been built. The first allows for the management of the mobile agents themselves. The second allows for remote testing of devices. These two applications are described in following two sections.

6.1 Mobile Agent Management

This application installs a popup menu on each node displayed in the GNB that allows the user to view the mobile agents running on that device. When a node is selected, a graphical view of the agents running on the device is displayed and updated as agents arrive and depart from the device. Each agent displayed also has a popup menu that allows a mobile agent manager to suspend, resume, stop, start and destroy the agent.

Behavior for the devices in the network is installed as described in the earlier sections. Once a VMR for a network element is created, the 'model provisioning' agent installs behavior that communicates with the VMC on the device such that the VMC subsequently notifies the network model of changes in agents that are resident on the device. The VMC listens for events related to the local birth or death of agents in order to provide this service to the network model. In order to provide the lifecycle control capability, the 'model provisioning' agent looks up the remote management service in the RMI registry on the device. An example of a graphical display is shown in Figure 14. This display shows a main view -- the network view -- along with a subview consisting of the agents resident on the selected node.

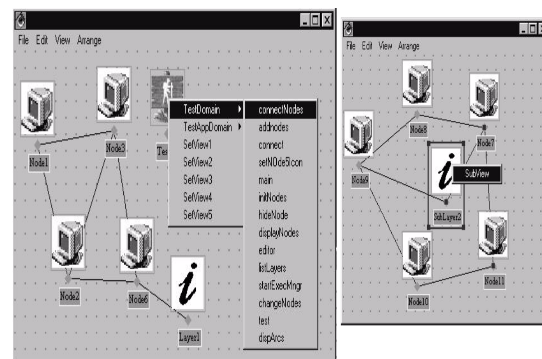

Figure 14: Example GNB Display

6.2 Remote Maintenance

The remote maintenance application allows for the remote execution of tests on devices and the reporting of

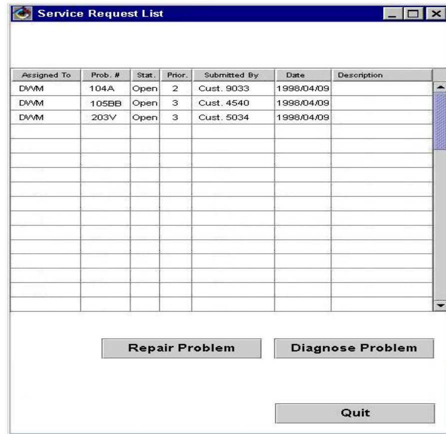

Figure 15: Problem Browser

the results of those tests. In this application, the behavior installed for each VMR is a listing of the device component hierarchy (cpu, disk etc.) and the tests that can be performed on them. When the VMR is created, the 'model provisioning' agent communicates with device VMC in order to register the VMR as being interested in changes to the problem state of the device. Practically, a single point of notification is installed within the network model but conceptually each VMR registers an interest.

An application component -- a problem browser -- can be injected as described earlier and an example of such a browser can be seen in Figure 15.

The Service Request List presents a list of all known problems in the network. From here, the operator can select a problem to address. Most often, the *Diagnose Problem* option will be selected since the operator will not generally know the cause of the problem. Selecting this option causes a list of components to be displayed for the problematic device. This is shown in Figure 16.

The operator can expand the tree of components and select individual test cases to perform by adding them to

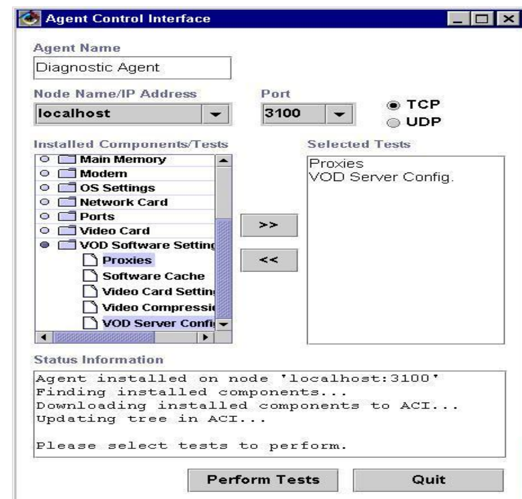

Figure 16: Device Diagnostic Dialog

the list on the right. Once the operator is finished, *Perform Tests* is selected in order to execute the tests on the selected components. An agent is then dispatched to the device in order to execute the tests and collect the results. An agent is sent rather than sending a messaging as a consequence of the complex processing of the test results that is required. In our experiments, the testing agents were each approximately 3K bytes uncompressed.

Upon execution of each test, the agent stores the test results internally. Once all of the tests have been successfully executed, the agent relays the results back

to operator's management platform and displays them in the Data Viewer shown in Figure 17.

Figure 17: Data Viewer

Here, the operator can decide if further tests are needed or if sufficient information on the cause of the problem is available. The operator can select *Repair Node* to correct the problem.

If the problem can be fixed, the network operator can send a repair agent to the node. Repairs can only be done for software related problems using the system. Alternatively, the operator can contact maintenance personnel so they can go to the customer's premise to fix difficult software problems or hardware related failures.

Here, the network operator dispatches a repair agent containing the correct software fixes to the node with the problem. Status information is sent back by the agent while repairs are carried out.

7. Summary

This paper has described how delegation and mobile agents can be used to create and maintain a network model that contains both state and behavior for a heterogeneous network and has provided details of a mobile code environment used in the research reported. Two applications have been briefly described that benefit from the creation of network models that include aspects of the behavior of the device. While Java execution environments are not yet available for network devices, it is the authors' belief that they will appear in the near future and will rapidly be integrated into the network fabric for next generation network management solutions. In fact, the authors' discussions with several equipment vendors have confirmed that Java-enabled devices will soon be commercially available. The availability of Jini, and its implications for plug-and-play devices [27], [28], will doubtless cause others to deploy Java-enabled devices that are not already developing them.

Research work is currently ongoing to create a number of more complex applications using the ideas described in [29] and [30] that make use of the network model and GNB framework for the purpose of fault diagnosis and network service provisioning.

8. References

1. Case, J., Fedor, M., Schoffstall, M. and Davi, J., A Simple Network Management Protocol, Internet Activities Board, RFC 1157, 1990.
2. Stallings, W., SNMP, SNMPv2, and CMIP. Don Mills: Addison-Wesley, 1993.Magedanz, T., Intelligent Agents: State of the Art and Potential Application Areas in Future Telecommunications, In *Proceedings of an International Workshop on High Speed Networking and Open Distributed Platforms*, St. Petersburg, Russia, June 12-15, 1995.
3. Yemini, Y., Goldszmidt, G. and Yemini, S. (1991), Network Management by Delegation. *In The Second*

International Symposium on Integrated Network Management, Washington, DC, April 1991.

4. Magedanz, T., On the Impacts of Intelligent Agent Concepts on Future Telecommunication Environments, in Lecture Notes on Computer Science 998 - "Bringing Telecommunication Services to the People - IS&N'95", pp. 396 - 414, A. Clarke et al. (Eds.), ISBN: 3-540-6 0479-0, Springer Verlag, 1995, *Proceedings of the 3rd Int. Conference on Intelligence in Services and Networks (IS&N)*, Heraclion, Greece, October 16-20, 1995.
5. Magedanz, T., Rothermel, K., Krause, S., Intelligent Agents: An Emerging Technology for Next Generation Telecommunications? In *Proceedings of IEEE INFOCOM* 1996, San Francisco, USA, March 24-28, 1996.
6. Magedanz T., and Eckardt, T., Mobile Software Agents: A new Paradigm for Telecommunications Management, In *Proceedings of IEEE/IFIP Network Operations and Management Symposium (NOMS '96)*, Kyoto, Japan, April 15-19, 1996.
7. Magedanz, T., and Eckardt, T., Mobile Service Agents and their Impacts on IN-based Service Architectures, IEEE Intelligent Network Workshop, Melbourne, Australia, April 21-24, 1996.
8. Krause, S., and Magedanz, T., Mobile Service Agents enabling "Intelligence on Demand" in Telecommunications, in *Proceedings of IEEE Global Telecommunications Conference*, London, United Kingdom, November 18-22, 1996.
9. Magedanz, T., Towards Intelligence on Demand - On the Impacts of Intelligent Agents on IN, in *Proceedings of the 4th International Conference on Intelligent Networks (ICIN)*, Bordeaux, France, December 2-5, 1996.
10. Bieszczad, A. and Pagurek, B. (1998), Network Management Application-Oriented Taxonomy of Mobile Code, in *Proceedings of IEEE/IFIP Network Operations and Management Symposium NOMS'98*, New Orleans, Louisiana, February 1998.
11. Gosling, J. W., and Steele, G., The Java™ Language Specification, Addison-Wesley, ISBN: 0-201-63451-1.
12. Sun Microsystems, Java Management API Specification, http://www.javasoft.com/products/JavaManagement/documents/architecture/html/jmapi-arch.html.
13. Sun Microsystems, Java Dynamic Management Kit Introduction, http://www.sun.com/software/java-dynamic.
14. Sun Microsystems, Jini™, http://java.sun.com/products/jini/index.html.
15. Susilo, G., Bieszczad, A. and Pagurek, B. (1998), Infrastructure for Advanced Network Management based on Mobile Code, in *Proceedings of IEEE/IFIP Network Operations and Management Symposium NOMS '98*, New Orleans, Louisiana, February 1998.
16. Object Management Group, Mobile Agent Services Interoperability Facilities (MASIF) Specification, OMG TC Document orbos/98-03-09.pdf, available at: ftp://ftp.omg.org/pub/docs/orbos/97-10-05.pdf.
17. GRASSHOPPER - The OMG-MASIF conformant Mobile Agent Platform, http://www.ikv.de/products/grasshopper/index.html.
18. Web based Management (1998), http://www.mindspring.com/~jlindsay/webbased.html.
19. Schramm, C., Bieszczad, A. and Pagurek, B. (1998), Application-Oriented Network Modeling with Mobile Agents, in *Proceedings of the IEEE/IFIP Network Operations and Management Symposium NOMS'98*, New Orleans, Louisiana, February 1998.
20. El-Darieby, M., Bieszczad, A. (1999), Intelligent Mobile Agents: Towards Network Fault Management Automation. To be presented at the *Sixth IFIP/IEEE International Symposium on Integrated Network Management*, May, 1999.
21. JESS, the Java Expert System Shell, http://herzberg.ca.sandia.gov/jess/.
22. Wijnen, B., Carpenter, G., Curran, K., Sehgel, A., Waters, G., Simple Network Management Protocol Distributed Protocol Interface Version 2.0, RFC 1592, http://sunsite.auc.dk/RFC/rfc/rfc1592.html.
23. Sun Microsystems, Java Remote Method Invocation Specification, Revision 1.50, JDK 1.2 Beta 4, October, 1998, http://java.sun.com/products/jdk/1.2/docs/guide/rmi/spec/rmiTOC.doc.html.
24. Knight, G. and Hazemi, R. (1998), Mobile Agent based Management in the INSERT project, UCL-CS Research Report RR98/97.
25. Applied Reasoning Systems, Dynamic Diagram Framework, http://www.arscorp.com/ddfWhite.html.
26. GEF (1997), http://www.ics.uci.edu/pub/arch/gef/.
27. Bieszczad, A., White, T., Pagurek, B. (1998), Mobile Agents for Network Management. *In IEEE Communications Surveys*, September, 1998.
28. Raza, S. K., Bieszczad, A. (1999), Network Configuration with Plug and Play Components. To be presented *at the Sixth IFIP/IEEE International Symposium on Integrated Network Management*, May, 1999.
29. White T., and Ross, N. (1996), Fault Diagnosis and Network Entities in a Next Generation Network Management System. In *Proceedings of EXPERSYS-96*, Paris, France, pp. 517-522.
30. White T. and Ross N. (1997), An Architecture for an Alarm Correlation Engine, in *Proceedings of Object Technology 97*, Oxford, 13-16 April, 1997.