Old Dominion University Research Foundation

Object-oriented Approach to High-level Network Monitoring and Management

Final Report
For
Research Performed Under
NASA LaRC Grant NAG-1-2143

Principal Investigator:
Ravi Mukkamala
Department of Computer Science
Old Dominion University
Norfolk, Virginia 23529-0162
Mukka@cs.odu.edu

Object-oriented Approach to High-level Network Monitoring and Management

Final Report
For
Research Performed Under
NASA LaRC Grant NAG-1-2143

Principal Investigator:
Ravi Mukkamala
Department of Computer Science
Old Dominion University
Norfolk, Virginia 23529-0162
Mukka@cs.odu.edu

!

Summary

In this report, we summarize the results from our research conducted under NASA LaRC grant NAG-1-2143. As part of this research, we have investigated methods to build high-level network monitoring systems on top of low-level monitoring tools. In particular, we have used UML (Unified Modeling Language) to model several aspects of the system, including the top-level design, the subsystem design, and the user interactions. Modeling, using UML, was found to be extremely useful in not only expressing the high-level user requirements but also low-level design details. In addition, we have designed and implemented a system to evaluate the availability of different monitored systems in the NASA LaRC environment. The system is implemented using the HP Openview software. It monitors the up/down exception messages from the system regarding the status of different devices, and computes the availability of different devices being monitored. The results from our research are published in two conference papers (attached with this report).

1. Ravi Mukkamala, Eli Siman-Tov, and Louis Galland, "An object-oriented monitoring system for management of large networks," Proc. 1999 ASEM Conference, Oct. 21-23, 1999, pp. 225-233.
2. Ravi Mukkamala, Eli Siman-Tov, and Louis Galland, "An object-oriented modeling and prototyping of large-scale computer networks for performance monitoring," Proc. World Multi-conference on Systems, Cybernetics, and Informatics, July 23-26, 2000, pp. 302-307.

Introduction

An absolute prerequisite for the management of large computer networks is the ability to measure their performance. Unless we monitor a system, we cannot hope to manage and control its performance. Since most of the networks in large organizations are always evolving with changes in available technology and the organization's own growth, these systems are often heterogeneous in nature. In addition, several independent monitoring systems (i.e., legacy systems) may already be in place which collect certain performance data. However, this also poses problems. For example, the definition of a specific measure may not be consistent among all the tools. But the users and system administrators would like to be presented with a consistent view of the system. Another example of a problem is the low-level measures often provided by the tools. The system managers and users often need high-level measures. For example, when a user complains that he/she is unable to connect to a specific machine, it is important to identify where the bottleneck(s) is in this path. Most systems let you only check whether or not a particular component is currently functioning. Thus, the responding service personnel need to do several queries prior to identifying the cause.

Under this grant, we investigated methods to build high-level monitoring systems that are built on top of existing monitoring tools. Prior to building such a system, we have undertaken the task of modeling the current system features as well as the requirements of the users. Due to the complexity of the system as well as the dynamically changing requirements, we use an object-oriented approach for the modeling. First, we use UML (Unified Modeling Language) to model users' requirements. Second, we identify the existing capabilities of the underlying monitoring system. Third, we try to map the former with the latter. During this mapping, we arrive at definitions (or expressions) that may implement the user defined performance metrics in terms of the system-collected metrics. Finally, our modeling methods should be flexible enough so that minor changes in users' requirements do not result in major changes to the models. The models are then prototyped using HP Openview.

The report is organized as follows. Section 2 summarizes background information used in our research. Section 3 describes the network configuration for which we have developed the current system. Section 4 describes the performance model developed for the network. Section 5 describes the prototyping efforts. Finally, Section 6 summarizes the report

Background

In this section, we give a brief review of different concepts used in the current research effort. We describe the UML, a modeling language, and SNMP, a standard object-oriented protocol that is used for network performance monitoring.

Unified Modeling language (UML) -- A Modeling Tool

Unified Modeling language (UML) is a standard language for writing software blueprints [2]. It may be used to specify different activities of a software development effort: visualize, specify, construct, and document the activities. For example, during the first phase of our project in which we developed the user-interface specification, we used UML. The Use case constructs of UML helped us capture and specify the intended behavior of the system. The use case diagrams then became a common ground for discussion among the development team as well as with the user and system community. In addition to the use cases, UML supports classes, relationships, packages, activities, and interactions. The concepts are mainly expressed in terms of diagrams. For example, the class diagram in Figure 7 models a bridge, a network infrastructure component. It defines the two attributes of a bridge and also defines its relationships with two other components, a network interface and an IP network device. The functions supported by these are also shown in the diagram.

Figure 8 is an example of a use-case diagram. It shows the actors and their interactions with the system. Within the system, the functions accessed by the actors are denoted as ellipses. Similarly, higher level models are shown in diagrams such as Figures 1-5.

Simple Network management Protocol (SNMP)

The SNMP (Simple Network Management Protocol) model of a managed network consists of four components: managed nodes, management stations, management information, and a management protocol. The managed nodes can be hosts, routers, bridges, printers, or any other devices capable of communicating their status to the outside. An SNMP agent residing on the component achieves this communication to and from the outside world. The network management is done from management stations. These stations communicate with the agents to get the status information. To facilitate the coordination between different devices and the management station, the information maintained by the agents is standardized. In SNMP terminology, each agent maintains a set of objects (variables). The collection of all possible objects (variables) in a network is given in a data structure called the MIB or Management Information Base. [4]. It also defines a hierarchical structure for naming the variables. This type of structure has enabled to be easily incorporated in several networking tools such as HP Openview

Let us look at the MIB variable structure in the system that we are currently modeling. Most components maintain the system-up-time indicating the time (in units of hundredths of a second) the component has been up since the last recovery. In the MIB-2 standards it is designated as .iso.org.dod.internet.mgmt.mib-2.system.sysUpTime. Similarly .iso.org.dod.internet.mgmt.mib-2.udp.udpInDatagrams specifies the total number of UDP datagrams delivered to UDP users. In addition to their textual name, each of these variables also has a numeric ID called object ID that is unique. For example, the

number of udpINDatagrams has an object ID of .1.3.6.1..2.1.7.1. The object IDs could be used in communicating with the network managers at each component.

In addition to the standard variables, each network component could maintain vendor specific variables. For example, the variable .iso.org.dod.internet.enterprises.hp.nm.interface.serial.serialConfigTable.serialConfigEntry.
serialTimeout is specific to Hewlett-Packard's (HP) serial interfaces. This timeout value is used when the management station has initiated a conversation over the serial link and represents the number of seconds of inactivity allowed before terminating the connection on this serial interface. Clearly, while there is no homogeneity in the time units across different measures, the standards make it easier for a developer to have a prior knowledge of the measures and their units for individual components.

From the brief description of UML and SNMP, it is clear that these tools support object-oriented design models that we are pursuing in our current effort.

LaRCNET: The Target Network of Modeling

LaRCNET is the NASA Langley Research Center's Local Area computer network connecting over 10,000 end-user devices together. Initially developed in 1985 to provide local researchers with better access to Langley's Supercomputers, it has grown from a 20-computer network spanning three buildings to its current size connecting 10,000 devices across more than 100 buildings. It is composed of three main Fiber Distributed Data Interface(FDDI) rings: the Isolation LAN, the Direct Attached Network and the main backbone---each running at 100 million bits per second (100 Mb/sec). The rings interconnect through routers.

The Isolation LAN serves to connect LaRCNET to the outside world. It connects to special purpose networks (AERONet, EOS, NHGS, NREN, NSI, etc.) and to the world wide Internet. A firewall between this ring and Langley's two internal rings provides a measure of protection from external infiltration. Connections range in speed from 1.5Mb/sec to 155Mb/sec. No user devices are connected, however. Traffic on this network, the majority of which goes through the Internet connection, averages 2Mb/sec with peaks at 10 Mb/sec (10% of available bandwidth).

The Direct Attached Ring connects approximately 700 high performance computers directly to the FDDI ring via concentrators - providing 100Mb/sec to the desktop. The second largest of the rings, it contains 65 infrastructure devices spanning 15 buildings using 20 miles of fiber. Typically, hosts are connected through a concentrator directly to the FDDI or via Ethernet switches. Traffic averages 9% of available bandwidth (9Mb/sec) with peaks of 60%.

The main campus backbone is the largest of the rings connecting the majority of Langley's computers to the network at 10Mb/sec. FDDI bridges connect 31 buildings directly to the backbone. These buildings in turn connect to 80 additional buildings. Approximately 550 infrastructure devices (bridges, switches, hubs, and repeaters) interconnect 220 Ethernet segments. Typically a bridge will connect from three to six Ethernet segments containing one or more hubs. Computers attach to these hubs via common telephone wiring. Traffic on the backbone ring averages 15% of available bandwidth with 45% peaks (45Mb/sec) [5].

Network performance Model

In this section, we describe the network performance model that we developed for monitoring the network.

Top-level View of the Model

Figure 1 describes a top-down view of the system model. At the highest level the system is modeled in terms of five modules or components. This type of modularization provides the freedom to choose different tools to implement each of the modules and to be concerned of developing a single tool that provides all the services. This strategy works well when the objective is to incorporate as many of the legacy systems as possible in the new system.

The *Automated Monitoring System (*AMS) is a module that is the foundation for the system. It has several functions including establishing the events in the network that need to be monitored, determining the criticality level of the events, receiving events from the underlying system, and triggering alarms. The AMS will communicate with the SNMP module in order to collect the needed data. Typically, the data is collected from the infrastructure components such as the bridges, routers, and switches. The AMS will communicate with the Processing System module to translate low-level device data into higher-level network performance measurements. For example, if the AMS has been collecting information about the system-up-time from a component, then it interacts with the Processing System module to calculate the availability of the component. It will interface with the File System module to query and update the Objects attributes such as status, thresholds, and device types. The AMS will interface with the User Interface subsystem by updating the status of a particular object and sending users alerts via an appropriate mechanism such as dialog box pop-ups, email, or paging.

The **SNMP System** (SNMPS) (Figure 2) is the module that will establish the data needed for collection from the network infrastructure devices, collect the data from the network infrastructure hardware, and distribute the data received from the hardware to the appropriate subsystem for processing and storage. The SNMPS will interface with the File subsystem to determine Object attributes such as community strings, hostnames, and device types. It will also interface with the File subsystem to save long term trending data. It will interface with the Processing subsystem to provide low-level data needed in the high-level network performance equations. It interfaces with the User Interface subsystem by receiving information about the Objects that have been selected by the user. This may include ObjectIDs and MIB variable OIDs, and then returning the data gathered from the hardware represented by the selected Objects. The SNMPS will primarily interface with the Processing subsystem to continuously provide data to be used in the network performance calculations.

The **User Interface System** (UIS) (Figure 3) is the module that will provide the user a means to interact with the network performance system via mechanisms such as dialog boxes, toolbars, menus, symbols and maps. Systems such as HP Openview use symbols to represent the instantiated objects within the system and provides many mechanisms for users to use the symbols in a variety of ways. For instance, the symbol color can change based on the status of an object; actions such as menu items can be

enabled or disabled based on the attributes of the underlying object. Clicking the symbol causes a selection that enables programs, using the software development kit, to receive that object as a variable, which can then be used within the program. The UIS primarily interacts with the File system. Using pre-defined files and directories, the UIS menus, toolbars, maps, and symbols can be customized for particular network performance monitoring requirements. The UIS interacts with the AMS, Processing, and SNMP modules by providing the data associated with user input and receiving data associated with user output.

The **Processing System** (PS) (Figure 4) is the module that will establish the network performance attributes of interest in the LaRCNet system, provide the equations needed for determining the performance of the network devices, and provide the processing requested by specific user requests. Many of the interactions with the other modules have already been described earlier.

The **File System** (FS) (Figure 5) is the module that provides customization of the user interface, definition of bitmaps and symbols, definition of object classes and the repository of instantiated objects, and storage of long-term trending data.

Bottom-up View of the Model

In the bottom-level view of the hierarchic model, we find the class definitions for each of the network components. For example, Figure 6 defines the network interface class diagram.. In the network domain, the lowest element of any network device is the Network Interface. All network devices, regardless of network topology or physical interface type, share this common class. Each network device is assigned a unique Media Access Control Address which is ultimately used by all protocols to identify a particular device on the network. The next level that we define network devices on the network is via the protocol that it is using. In Figure 6, we define three different protocol device classes: IP, AppleTalk, and IPX. Thus, the model was able to abstract out the common features of the network devices at the same time enabling us to represent the differences in them. The figure shows parent-child relationships often referred in object-oriented methodologies. For example, the IP Network Device class inherit attributes MAC Address and the method Get_MAC_Address from the Network Interface class.

Similarly, the model for a Bridge is shown in Figure 7. The Bridge is composed using an aggregation of one and only one IP Network device class and one or more of the Network Interface Class per every Bridge Class. Within the various device types, a variety of manufacturer devices are used such as DEC bridges for bridging standard Ethernet and HP bridges for bridging AppleTalk. Even within a particular device type and manufacturer there are various models. Some model differences can be handled by simply having an attribute within the class to contain different values that represent the differences. For instance, a DEC 620 bridge has three Ethernet interfaces while a DEC 900 bridge has 7 Ethernet interfaces. This difference can be represented within one class by having a Number_Of_Interfaces attribute that would contain the value 3 for a 620 model and 7 for a 900 model. Some differences in models may not lend themselves to an equivalent attribute, but have enough similarities that a parent-child class relationship can represent them.

Modeling User Interactions

As mentioned in the introduction, our network monitoring system model not only include the network components but also the interactions with the users. We use the use-case diagrams (as defined in UML) for this. The purpose of a use case is to define a piece of behavior of an entity (e.g., system, subsystem, module, or component). Each use case specifies a service the entity provides to its users. This is expressed in terms of the interaction between users and entities, as well as the responses performed by the entity. The interactions only describe the communications between the users and the entity. The internal behavior or implementation details are hidden. Here, we describe one of the use-cases of our system, the monitor use-case.

Monitor is an entity that monitors network status and performance. We describe our monitor use-case in Figure 8. As shown here, the monitor has the capability to check the availability (of system or infrastructure components), check their utilization, and check the presence of broadcast storms (a transmission is said to be a broadcast when it is to be delivered to multiple machines, typically all, across the network). In addition, it has an option to locate a device within the network topology. It can also check for network configuration errors, and trigger alerts when certain undesired conditions have occurred. In this diagram, one can also observe other roles such as the administrator and the analyst that interact with the monitor through the underlying system functions. Of course, network devices and data collection files are also responsible for offering the proposed services to the monitor role.

High-level Monitoring Functions

In our model, we have included several high-level performance which may or may not be supported by a commercial monitoring tool. For example, consider the availability function defined over infrastructure components. Clearly, a tool such as the HP OpenView system provides sysUpTime or the time since the last recovery of the component. It also has the ability to plot the sysUpTime as a function of time. However, the underlying system does not provide device availability which is the fraction of the time the device under consideration is available to the rest of the system. While it can be computed from the plots, it is a tedious process for a high-level user. In addition, there are other complications such as determining whether a device is actually down or it is simply not accessible as one of the bridges or routers to which it is connected (directly or indirectly) is simply down.

Similarly, whenever a device is inaccessible administrators would like to know the cause for its inaccessibility. Im most of the existing toolkits is not possible to know directly to which infrastructure component and to which port is the device connected in the network. As before, while the information can be obtained through several queries, it cannot be known directly by a user. We have provided functions through which given a device name or its IP address, the system can provide complete information as to how it is connected in a network. We model path availability. In addition, some exceptional conditions such as the occurrence of broadcast storms in the system can also be detected and reported to the users.

Prototyping the Model using HP Openview

In order to check the validity and completeness of our model, we have prototyped it using HP OpenView (HPOV)---a system to monitor and control networking environments [1]. It provides tools for discovery and mapping of the entire network so network changes can be instantly identified and network problems rapidly discovered. In addition, to meet the increasing demands for dependable network performance, it also provides performance management and reporting tools to help network administrators. The tools help managers to proactively identify changing traffic patterns and plan for increasing network needs. Allocating and controlling network bandwidth helps assure network service level objectives are continually met and network simulation enable proposed network changes to be tested before they are implemented.

The data about the occurrence of events in the system and collecting data from the infrastructure components is done through the HP Openview. In other words, we have prototyped our model on top of the HPOV. In addition, we have used its database to store the performance measures computed by the model We found that the HPOV system along with the HPOV software development kit Application Programming Interfaces (APIs) would be sufficient to prototype our model representing the NASA Langley network (LaRCNET) infrastructure system.

The HPOV database is created from a network discovery process. The process involves queries to the various network components in the system including routers, hubs, concentrators, bridges, and hosts that make up the whole LaRCNET system. Each of the components is represented as an object in the object database. The "attributes" of an object are initially determined by the object's "capabilities." Some of the capabilities supported by the system are isBridge, isHub, isIP, siPrinter, isPC, and isSNMPSupported. So if a device on the network was determined to have the capability of isIP=True, then that object would be assigned attributes associated with that IP capability (e.g., . "IP Address", "IP Hostname", "IP Netmask"). Of particular interest is when a device is found that has the isSNMPSupported =True capability. This defines the device as **manageable** and further information is acquired from the device using SNMP queries to standard Management Information Base (MIB). When an object is manageable, then HPOV periodically polls its status and produces events within the system based on the object's status. Also, when a user interacts with objects, via symbols on maps, or the menu and toolbar systems within HPOV, additional events are triggered and can be used for individual processing and customization. Standard network management methods are given to us via HPOV's predefined "user interface", but this user interface can be altered and customized to fit the needs of our OOD. These features of HPOV will also produce an event driven aspect to the system that must be modeled within the design.

Once HPOV has completed the discovery process, its database contains all the "instantiated" objects within the LaRCNet infrastructure. All infrastructure type devices support the SNMP protocol and therefor have the capability of isSNMPSupported=True. We can now use the "SNMP sysObjectID" attribute given to each instantiated object to help us in defining its class represented in the design. Based on this information, we can assign additional attributes (HPOV database fields) to all the objects that have the same

value of "SNMP sysObjectID". Using this process, we can create a set of capabilities and/or attributes that reflect the class hierarchy of our design.

Although the objects in our database will not have "methods" attached to them as shown in the class diagrams, there is a way to cause the same effect from within HPOV. Customized programming within HPOV come in the form of "Actions", "Action Callbacks", and "Event Callbacks". The Action Callbacks and Event Callbacks have the same properties as actions. Actions are invoked by user interaction via the menus, map symbols, and toolbars. These menus, map symbols, toolbars and even actions can be prohibited from being available for execution based on a set of capabilities. For instance, we could create a toolbar button or menu item that executes a "ping" action for an object represented by a selected symbol and then only enable those items on objects that have the isIP=True capability. Therefore, by using attributes and capabilities representing our class hierarchy, we can enable actions based on those attributes that would then reflect the hierarchy of methods of our design.

Due to the object-oriented nature of SNMP, UML, and HP Openview, we are able to model and prototype the system in an incremental manner and carry out testing

Conclusion

In this report, we have summarized our current efforts to build an object-oriented monitoring system for management of large networks. First, we have described the model that we developed for the monitoring system. We used UML for this purpose. We then described how the model was prototyped using HP Openview System. We have considered the network infrastructure components at NASA LaRC to prototype our model. Much of the model has been completed. We started prototyping several of the modules in the model. Due to the object-oriented nature of the design methodology as well as the prototyping system, we are able to take an incremental approach to this problem.

References

1. HP Openview: http://www.openview.hp.com

2. Grady Booch, Ivar Jacobson, James Rumbaugh, "The Unified Modeling Language User Guide," The Addison-Wesley Object Technology Series, October 1998.

3. Grady Booch, "Object-Oriented Analysis and Design With Applications," Addison-Wesley Object Technology Series, February 1994.

4. William Stallings, "Snmp, Snmpv2, and Rmon : Practical Network Management," Addison-Wesley Publishing, July 1996.

5. David B. Yeager and Michael D. Bray, LaRCNET 1997 -- A status Report, NASA Langley Research Center, 1997.

Figure 1. Top-level Subsystem Diagram

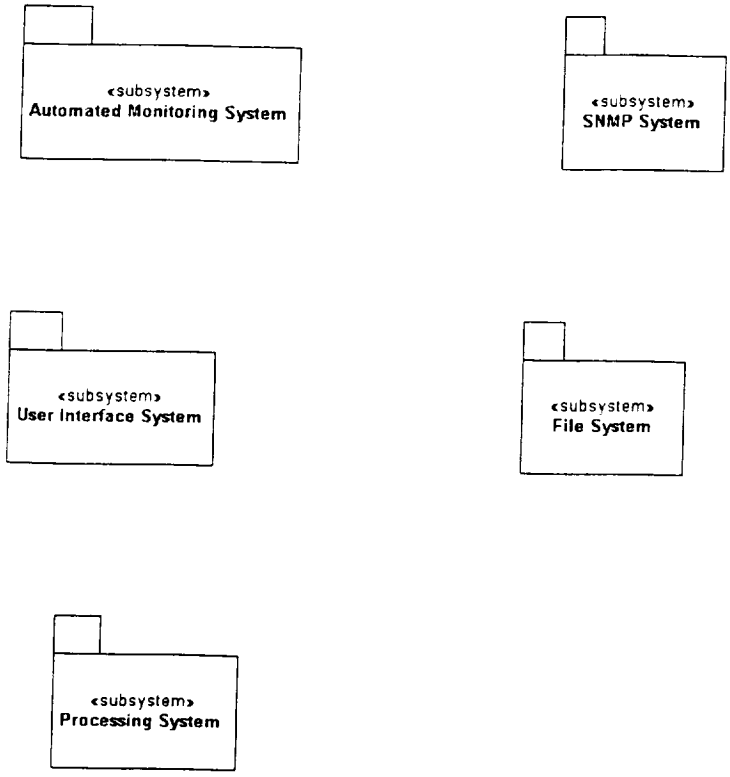


Figure 2: SNMP System Diagram

Figure 3. The User Interface System

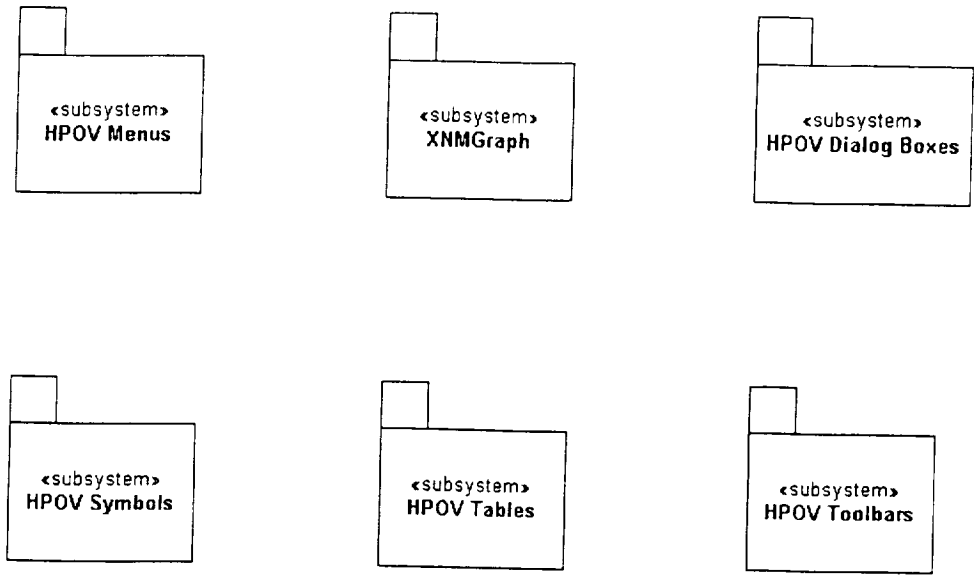


Figure 4. The Processing System

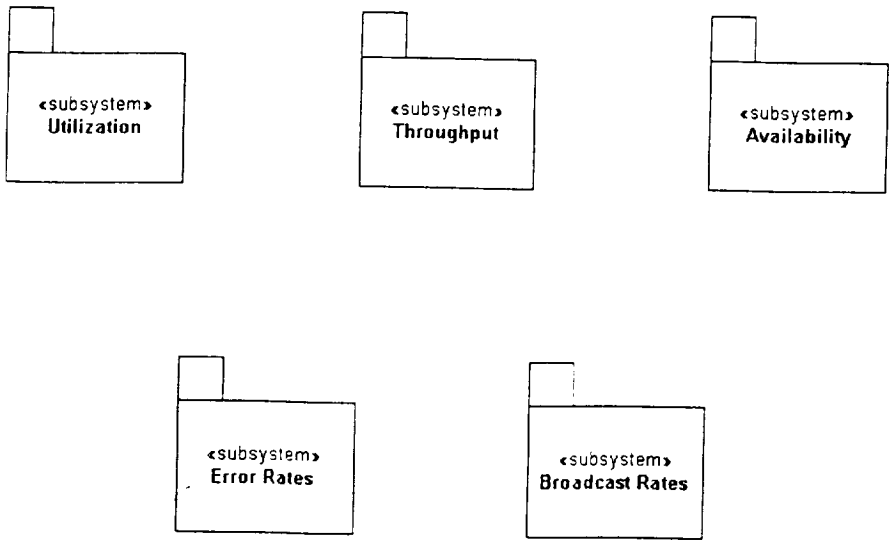


Figure 5. The File System

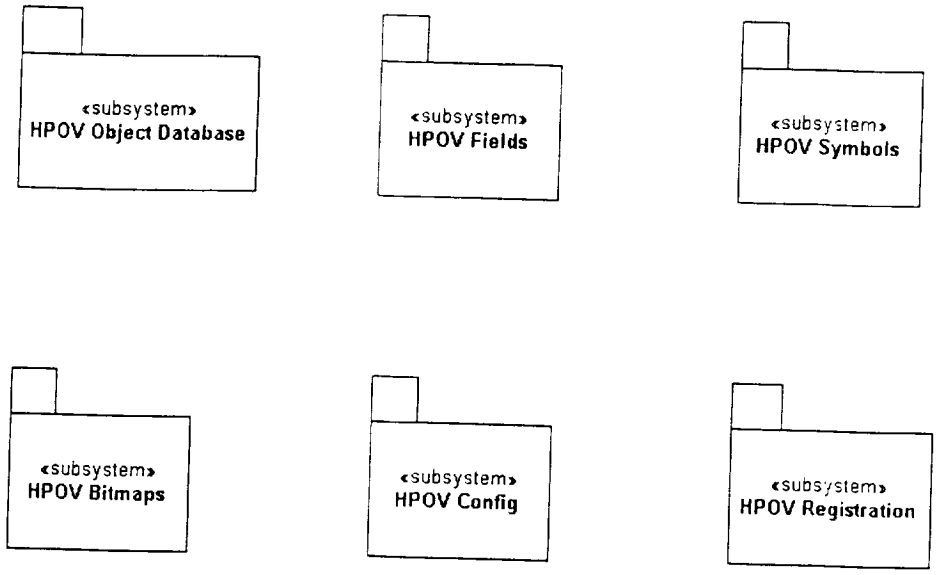


Figure 6. Network Interface Class Diagram

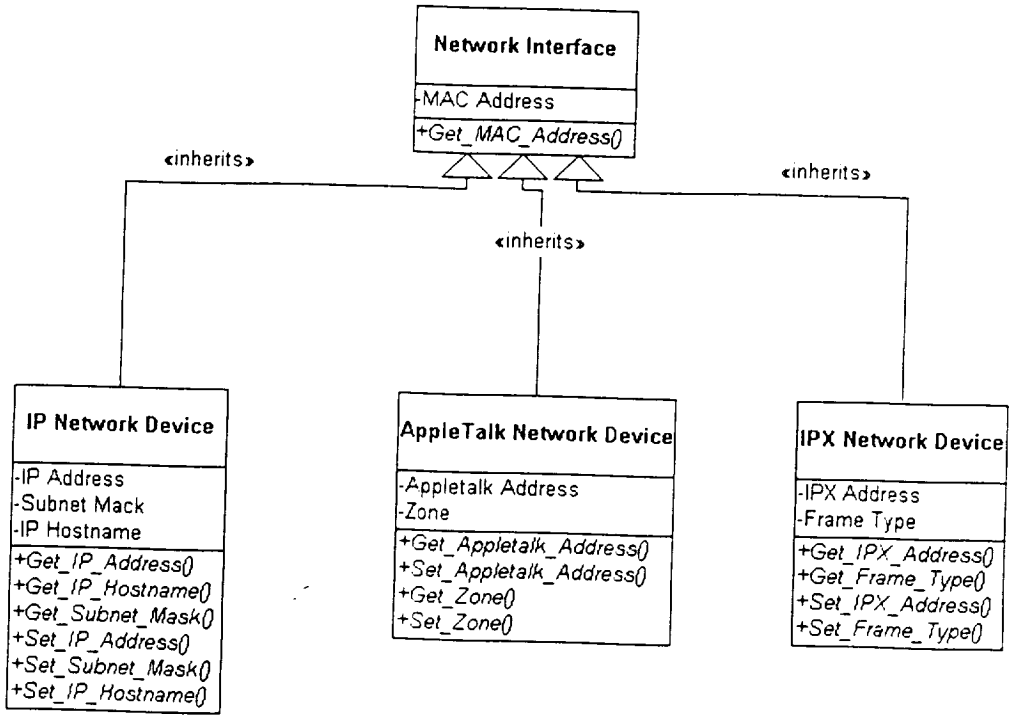


Figure 7. Bridge Infrastructure

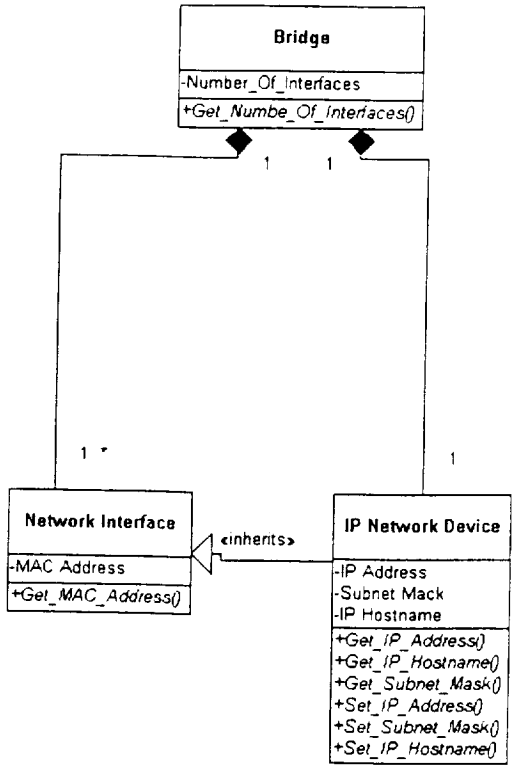
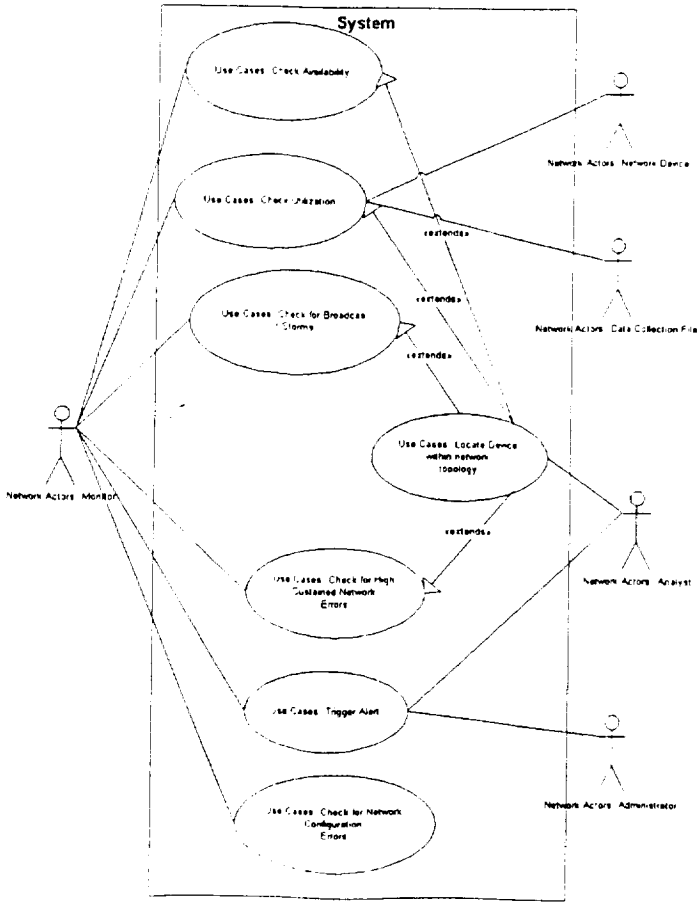


Figure 8. Monitor Use-case Diagram



World Multiconference on Systemics, Cybernetics and Informatics

July 23-26, 2000
Orlando, Florida, USA

PROCEEDINGS

Volume IV
Comunications Systems and Networks

Organized by IIIS

International Institute of Informatics and Systemics

Member of the International Federation of Systems Research

IFSR

Co-organized by IEEE Computer Society

EDITORS
Belkis Sanchez
Dong Feng Yuan
Germinal Isern
Kainam Thomas Wong

An Object-Oriented Modeling And Prototyping Of Large-Scale Computer Networks For Performance Monitoring

Ravi Mukkamala
Department of Computer Science
Old Dominion University
Norfolk, Virginia 23529-0162, USA

and

Eli Siman-Tov and Louis D. Galland
NASA Langley Research Center
Mail Stop 124
Hampton, Virginia 23681-2199, USA

ABSTRACT

An absolute prerequisite for the management of large-scale computer networks is the ability to monitor their performance. Prior to developing a performance monitoring system for such networks, it is important to model the networks and test the model using either simulation or prototypes. In this paper, we describe our efforts to develop such a model for the networks at NASA Langley Research Center. Keeping in mind the complexity of the task and the required flexibility for future changes, we use an object-oriented design methodology. In particular, we use the Unified Modeling language (UML) for modeling. The model is being prototyped using the APIs offered by the HP OpenView system.

Keywords: Computer networks, HP Openview, modeling, object-oriented design, performance monitoring, prototyping, SNMP, UML.

1. INTRODUCTION

An absolute prerequisite for the management of large computer networks is the ability to measure their performance. Unless we monitor a system, we cannot hope to manage and control its performance. Since most of the networks in large organizations are always evolving with changes in available technology and the organization's own growth, these systems are often heterogeneous in nature. In addition, several independent monitoring systems (i.e., legacy systems) may already be in place which collect certain performance data. However, this also poses problems. For example, the definition of a specific measure may not be consistent among all the tools. But the users and system administrators would like to be presented with a consistent view of the system. Another example of a problem is the low-level measures often provided by the tools. The system managers and users often need high-level measures. For example, when a user complains that he/she is unable to connect to a specific machine, it is important to identify where the bottleneck(s) is in this path. Most systems let you only check whether or not a particular component is currently functioning. Thus, the responding service personnel need to do several queries prior to identifying the cause.

In order to solve these problems, we are investigating methods to build high-level monitoring systems that are built on top of existing monitoring tools. Prior to building such a system, we have undertaken the task of modeling the current system features as well as the requirements of the users. Due to the complexity of the system as well as the dynamically changing requirements, we use an object-oriented approach for the modeling. First, we use UML (Unified Modeling Language) to model users' requirements. Second, we identify the existing capabilities of the underlying monitoring system. Third, we try to map the former with the latter. During this mapping, we arrive at definitions (or expressions) that may implement the user defined performance metrics in terms of the system-collected metrics. Finally, our modeling methods should be flexible enough so that minor changes in users' requirements do not result in major changes to the models. The models are then prototyped using HP Openview.

The paper is organized as follows. Section 2 describes LaRCNET, the network in focus for our work. Section 3 summarizes the problem. Section 4 describes the performance model developed for the network. Section 5 describes the prototyping efforts to evaluate the availability measures. Finally, Section 6 summarizes the paper.

2. LaRCNET

LaRCNET is the NASA Langley Research Center's Local Area computer network connecting over 10,000 end-user devices together. Initially developed in 1985 to provide local researchers with better access to Langley's Supercomputers, it has grown from a 20-computer network spanning three buildings to its current size connecting 10,000 devices across more than 100 buildings. It is composed of three main Fiber Distributed Data Interface(FDDI) rings: the Isolation LAN, the Direct Attached Network and the main backbone---each running at 100 million bits per second (100 Mb/sec). The rings interconnect through routers.

The Isolation LAN serves to connect LaRCNET to the outside world. It connects to special purpose networks (AERONet, EOS, NHGS, NREN, NSI, etc.) and to the world wide Internet. A firewall between this ring and Langley's two internal rings provides a measure of protection from external infiltration. Connections range in speed from 1.5Mb/sec to 155Mb/sec. No user devices are connected, however. Traffic on this network, the majority of which goes through the Internet connection, averages 2Mb/sec with peaks at 10 Mb/sec (10% of available bandwidth).

The Direct Attached Ring connects approximately 700 high performance computers directly to the FDDI ring via concentrators - providing 100Mb/sec to the desktop. The second largest of the rings, it contains 65 infrastructure devices spanning 15 buildings using 20 miles of fiber. Typically, hosts are connected through a concentrator directly to the FDDI or via Ethernet switches. Traffic averages 9% of available bandwidth (9Mb/sec) with peaks of 60%.

The main campus backbone is the largest of the rings connecting the majority of Langley's computers to the network at 10Mb/sec. FDDI bridges connect 31 buildings directly to the backbone. These buildings in turn connect to 80 additional buildings. Approximately 550 infrastructure devices (bridges, switches, hubs, and repeaters) interconnect 220 Ethernet segments. Typically a bridge will connect from three to six Ethernet segments containing one or more hubs. Computers attach to these hubs via common telephone wiring. Traffic on the backbone ring averages 15% of available bandwidth with 45% peaks (45Mb/sec) [3].

3. PROBLEM STAEMENT

In the Langley Research Center network environment, the method for determining availability is using a script called netmon. This script will simply read a file containing infrastructure devices that need to be monitored. The script will then periodically ping each device in the list and receive a response from the device, indicating that the device is still working. If a device does not respond, then an alarm is sounded on the screen alerting personnel in the Network Operations Center that there is a problem with that device. The error would also be entered into a log file for archiving.

There are several problems associated with using netmon as a means of tracking availability.

1. The first problem is that it is unknown whether the device is really down. The device could be behind another device that is down and therefore the ping can not traverse the network topology to get to that particular device. The device could just be too busy to deal with a low priority task such as a ping. Another possibility is that the machine doing the pinging is down or unable to reach the network.
2. The second problem is that there is no mechanism to determine trends in a device's availability (although the logs were used to give us a "rough estimate" of the device's availability). The errors are merely logged to a file that is archived at a periodic interval to an archive storage area. No corrections are made regarding the first issue and no analysis is done to extract long-term trend data.
3. The third problem is that the file used to generate the pinging of devices must be manually edited to add or delete any device in the network system.
4. Finally, the fourth problem is that devices are replaced without notation in the logs, so the hostname and IP address contained in the file may represent many different devices in the logs. A separate system of trouble tickets is kept, but this system is not correlated with logs files in anyway that could automate the determination of which log entries correspond with any particular device.

4. NETWORK PERFORMANCE MODEL

In this section, we describe the network performance model that we developed for monitoring the network.

Top-level View of the Model

At the highest level the system is modeled in terms of five modules or components. This type of modularization provides the freedom to choose different tools to implement each of the modules and to be concerned of developing a single tool that provides all the services. This strategy works well when the objective is to incorporate as many of the legacy systems as possible in the new system. The individual modules are described below.

The *Automated Monitoring System* (AMS) is a module that is the foundation for the system. It has several functions including establishing the events in the network that need to be monitored, determining the criticality level of the events, receiving events from the underlying system, and triggering alarms. The AMS will communicate with the SNMP module in order to collect the needed data. Typically, the data is collected from the infrastructure components such as the bridges, routers, and switches. The AMS will communicate with the Processing System module to translate low-level device data into higher-level network performance measurements. For example, if the AMS has been collecting information about the system-up-time from a component, then it interacts with the Processing System module to calculate the availability of the component. It will interface with the File System module to query and update the Objects attributes such as status, thresholds, and device types. The AMS will interface with the User Interface subsystem by updating the status of a particular object and sending users alerts via an appropriate mechanism such as dialog box pop-ups, email, or paging.

The *SNMP System (SNMPS)* is the module that will establish the data needed for collection from the network infrastructure devices, collect the data from the network infrastructure hardware, and distribute the data received from the hardware to the appropriate subsystem for processing and storage. The SNMPS will interface with the File subsystem to determine Object attributes such as community strings, hostnames, and device types. It will also interface with the File subsystem to save long term trending data. It will interface with the Processing subsystem to provide low-level data needed in the high-level network performance equations. It interfaces with the User Interface subsystem by receiving information about the Objects that have been selected by the user. This may include ObjectIDs and MIB variable OIDs, and then returning the data gathered from the hardware represented by the selected Objects. The SNMPS will primarily interface with the Processing subsystem to continuously provide data to be used in the network performance calculations.

The *User Interface System (UIS)* is the module that will provide the user a means to interact with the network performance system via mechanisms such as dialog boxes, toolbars, menus, symbols and maps. Systems such as HP Openview use symbols to represent the instantiated objects within the system and provides many mechanisms for users to use the symbols in a variety of ways. For instance, the symbol color can change based on the status of an object; actions such as menu items can be enabled or disabled based on the attributes of the underlying object. Clicking the symbol causes a selection that enables programs, using the software development kit, to receive that object as a variable, which can then be used within the program. The UIS primarily interacts with the File system. Using pre-defined files and directories, the UIS menus, toolbars, maps, and symbols can be customized for particular network performance monitoring requirements. The UIS interacts with the AMS, Processing, and SNMP modules by providing the data associated with user input and receiving data associated with user output.

The *Processing System (PS)* is the module that will establish the network performance attributes of interest in the LaRCNet system, provide the equations needed for determining the performance of the network devices, and provide the processing requested by specific user requests. Many of the interactions with the other modules have already been described earlier.

The *File System (FS)* is the module that provides customization of the user interface, definition of bitmaps and symbols, definition of object classes and the repository of instantiated objects, and storage of long-term trending data.

Bottom-up View of the Model

In the bottom-level view of the hierarchic model, we find the class definitions for each of the network components. In the network domain, the lowest element of any network device is the Network Interface (Figure 1). All network devices, regardless of network topology or physical interface type, share this common class. Each network device is assigned a unique Media Access Control Address which is ultimately used by all protocols to identify a particular device on the network. The next level that we define network devices on the network is via the protocol that it is using. Thus, the model was able to abstract out the common features of the network devices at the same time enabling us to represent the differences in them. The figure shows parent-child relationships often referred in object-oriented methodologies. For example, the IP Network Device class inherit attributes MAC Address and the method Get_MAC_Address from the Network Interface class.

Similarly, the Bridge is composed using an aggregation of one and only one IP Network device class and one or more of the Network Interface Class per every Bridge Class. Within the various device types, a variety of manufacturer devices are used such as DEC bridges for bridging standard Ethernet and HP bridges for bridging AppleTalk. Even within a particular device type and manufacturer there are various models. Some

model differences can be handled by simply having an attribute within the class to contain different values that represent the differences. For instance, a DEC 620 bridge has three Ethernet interfaces while a DEC 900 bridge has 7 Ethernet interfaces. This difference can be represented within one class by having a Number_Of_Interfaces attribute that would contain the value 3 for a 620 model and 7 for a 900 model. Some differences in models may not lend themselves to an equivalent attribute, but have enough similarities that a parent-child class relationship can represent them.

Modeling User Interactions

As mentioned in the introduction, our network monitoring system model not only include the network components but also the interactions with the users. We use the use-case diagrams (as defined in UML) for this. The purpose of a use case is to define a piece of behavior of an entity (e.g., system, subsystem, module, or component). Each use case specifies a service the entity provides to its users. This is expressed in terms of the interaction between users and entities, as well as the responses performed by the entity. The interactions only describe the communications between the users and the entity. The internal behavior or implementation details are hidden. Here, we describe one of the use-cases of our system, the monitor use-case.

Monitor is an entity that monitors network status and performance. We describe our monitor use-case in Figure 2. As shown here, the monitor has the capability to check the availability (of system or infrastructure components), check their utilization, and check the presence of broadcast storms (a transmission is said to be a broadcast when it is to be delivered to multiple machines, typically all, across the network). In addition, it has an option to locate a device within the network topology. It can also check for network configuration errors, and trigger alerts when certain undesired conditions have occurred. In this diagram, one can also observe other roles such as the administrator and the analyst that interact with the monitor through the underlying system functions. Of course, network devices and data collection files are also responsible for offering the proposed services to the monitor role.

5. PROTOTYPING AVAILABILITY

The first thing done at LaRC was to use the logs being kept by netmon (along with its input files) to make a "rough estimate" of the long term trends in availability for each Hostname/IP combination. This was used to refine the UML design and HP OpenView (HPOV) implementation of a final solution to address all the problems lacking in the current solution. Next, using the UML design, a prototype program for calculating and presenting availability information for LaRCNet's infrastructure devices was generated. This program was generated within the HPOV's development environment using the HPOV software development kit. This SDK allows access to HPOV's internal function calls including the user interface, events, SNMP communications, and tasking structure. The prototype deals with the problems raised above in the following ways.

To address the first problem, the infrastructure device itself keeps track of how long it has been in operation since the last time it was powered up. This is a MIB-II OID called sysUpTime. By querying this variable on a device we can solve the problems of decided whether the device was really down, or it was actually up but could not be reached for a period of time by the network management machine. Also, when a device is tagged as "managed" within HPOV, then HPOV will query the device on a periodic basis. By using HPOV's event system, the prototype can detect when a device goes up or down. Therefor, the prototype will "listen" to HPOV's event system for a Device Up or Device Down event and extract the Hostname, IP, and Time information from the event call. If the event is a Device Down, then the program will tag the device as being in an Unknown state, because at this point the device could be the victim of the situations stated in the first problem. Once an Event for Device Up is received for this device, then the prototype will query the device's sysUpTime parameter and compare it to how long the device was supposedly down. If the sysUpTime time is greater, then the device was never down, otherwise the prototype will calculate the down time and write the information to a file for later presentation processing.

To address the second problem, a new file structure could be created to contain all the data needed to analyze a particular device. By storing information about an infrastructure device's status at various periodic intervals, trend graphs can be easily produced. The prototype program will log all device activity into a file. Periodically, the program will create additional files that contain different "views" of the device. Once the user requests availability on a device, the program will present the availability in the following "views"; 1) Day View, showing the last 24 hour period in 5 minute resolution, 2) Weekly View, showing the last week period using 30 minute resolution, 3) Monthly View, show the last month with a 2 hour resolution and, 4) Yearly View, shows the last year with a year resolution. Finally, if the user is willing to wait for the additional file processing time, the user may request a particular date range with a particular accuracy rating.

To address the third problem, information is available from HPOV's discovery process that will indicate when devices are added/modified/deleted from the network system. This information could be captured and used to automatically update the list of devices that would be tracked. The prototype program will "listen" to the event log (the same way as when it was listening for the devices Up/Down status) for a Device Added or Device Deleted events. The device being deleted from the network will not affect the program, other than to close out an Unknown state for that device. A device added event will cause the prototype program to add that device to the file of machines being monitored and begin creation of the different "views" file. Other aspects of the program (such as the user interface) are automatically engaged via the HPOV attributes that are applied to the other infrastructure devices.

To address the fourth problem, again information is available from HPOV's network monitoring process that will indicate when a Hostname/IP/MAC address combination has changed on the network. This information could be captured and used to indicate that when the MAC address for an Hostname/IP has changed, then it can be assumed to be a new device, that needs separate metrics on its availability. This part of the program is currently not implemented within the prototype program. A new file would need to be maintained to track the devices Hostname/IP/MAC address combination. The program could "listen" for an event from HPOV or the program would check this file whenever a device's Up event occurred, compare it to the current MAC address of the device, and determine if it is the same device as when it last went down. If the device was different, then the program would need to "archive" the information from the files for historical purposes and start to maintain new data on the new device.

6. CONCLUSION

In this paper, we have summarized our current efforts to build an object-oriented monitoring system for management of large networks. First, we have described the model that we developed for the monitoring system. We used UML for this purpose. We then described how the model was prototyped using HP Openview System. We have considered the network infrastructure components at NASA LaRC to prototype our model. Much of the model has been completed. We started prototyping several of the modules in the model. Due to the object-oriented nature of the design methodology as well as the prototyping system, we are able to take an incremental approach to this problem.

7. ACKNOWLEDGEMENTS

The work is supported in part by a grant (NAG-1-2143) from NASA Langley Research Center, Hampton, Virginia.

8. REFERENCES

1. Grady Booch, "Object-Oriented Analysis and Design With Applications," Addison-Wesley Object Technology Series, February 1994.
2. William Stallings, "Snmp, Snmpv2, and Rmon : Practical Network Management," Addison-Wesley Publishing, July 1996.
3. David B. Yeager and Michael D. Bray, LaRCNET 1997 -- A status Report, NASA Langley Research Center, 1997.

Figure 1. Network Interface Class Diagram

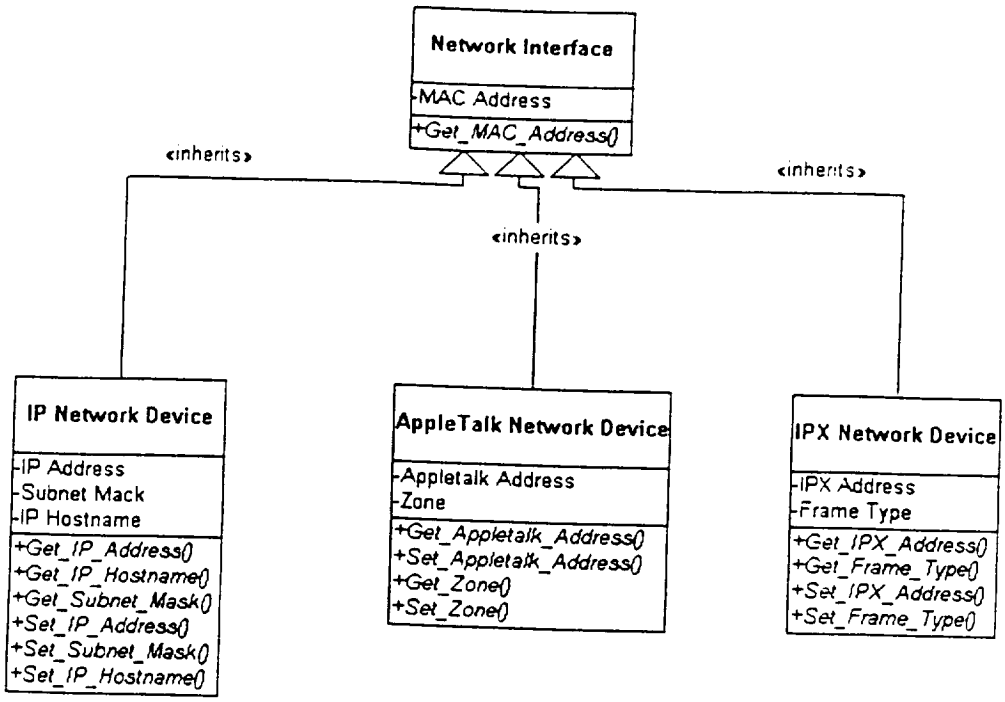
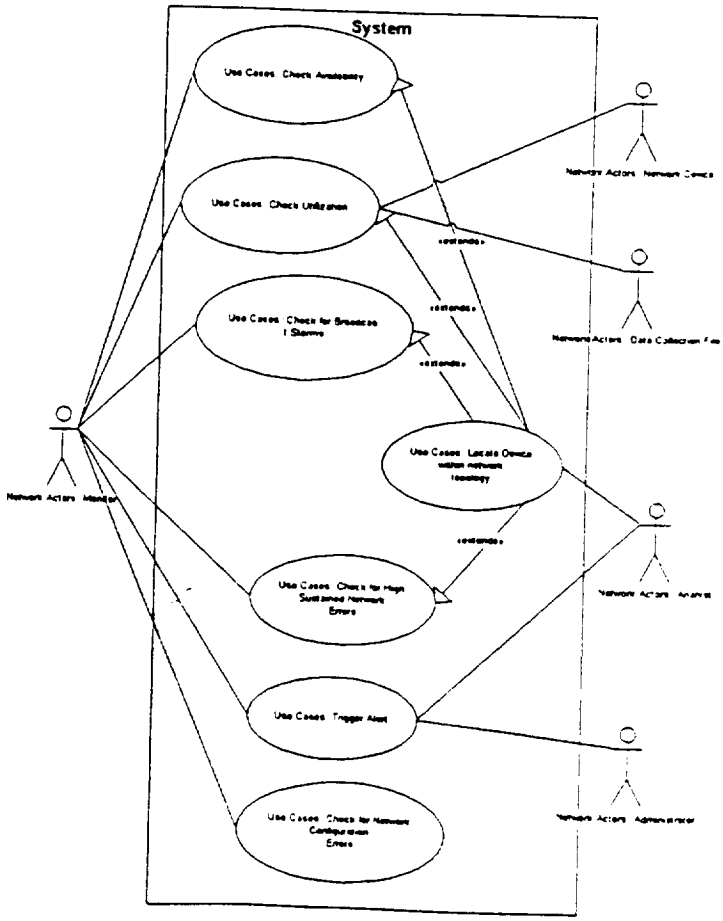
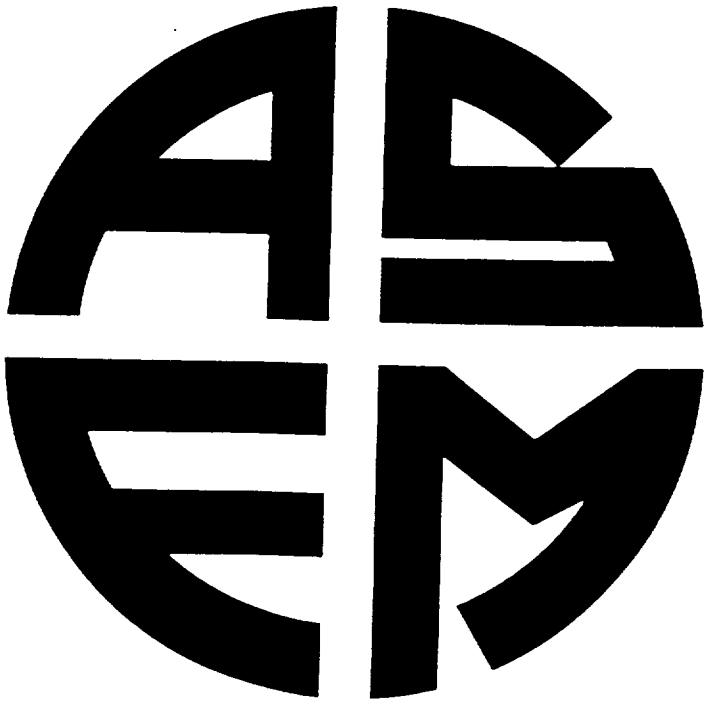


Figure 2. Monitor Use-case Diagram



AMERICAN SOCIETY FOR ENGINEERING MANAGEMENT



"Pathways to Technological Leadership"

PROCEEDINGS

from the
1999 ASEM National Conference
October 21-23, 1999
Sheraton Inn Ocean Front
Virginia Beach, VA

AN OBJECT-ORIENTED MONITORING SYSTEM FOR MANAGEMENT OF LARGE NETWORKS

Ravi Mukkamala
Department of Computer Science
Old Dominion University
Norfolk, Virginia 23529-0162
mukka@cs.odu.edu

Eli Siman-Tov and Louis D. Galland
NASA Langley Research Center
Mail Stop 124
Hampton, Virginia 23681-2199
{e.siman-tov, l.d.galland@larc.nasa.gov

Abstract

An absolute prerequisite for the management of large computer networks is the ability to measure their performance. Unless we monitor a system, we cannot hope to manage and control its performance. In this paper, we describe a network monitoring system that we are currently designing and implementing. Keeping in mind the complexity of the task and the required flexibility for future changes, we use an object-oriented design methodology. The system is built using the APIs offered by the HP OpenView[1] system. The implementation work is currently in progress at NASA Langley Research Center.

Introduction

An absolute prerequisite for the management of large computer networks is the ability to measure their performance. Unless we monitor a system, we cannot hope to manage and control its performance. Due to the heterogeneous nature of networks and the monitoring tools that are currently being used, one of the main challenges in monitoring large distributed networks is the choice of the measures. Some of the problems that are faced by system administration professionals in the network management are as follows:

- Too many measures (or indicators) in use by different tools,
- ambiguities in the definitions of the measures,
- differences in measures supported by different vendors, and
- compute intensive measures.

In addition, the measures provided by the existing tools are often low-level. For example, there are measures for the availability of a host or a bridge. But often, system managers need higher-level measures such as path availability (i.e., availability of a path between a pair of hosts with certain bandwidth).

[1] HP OpenView is a product of Hewlett-Packard.

In order to solve these problems, we are investigating methods to build high-level monitoring systems that are built on top of existing monitoring tools. Due to the heterogeneous nature of the underlying systems at NASA Langley Research Center, we use an object-oriented approach for the design. First, we use UML (Unified Modeling Language) to model users' requirements. Second, we identify the existing capabilities of the underlying monitoring system. Third, we try to map the former with the latter. During this mapping, we arrive at definitions (or expressions) that may implement the user defined performance metrics in terms of the system-collected metrics. Finally, our design methodology should be flexible enough so that minor changes in users' requirements do not result in major changes to the system.

The object-oriented approach to the network management problems enables us to address the heterogeneous issues of networks and the changing user requirements in an efficient manner.

The paper is organized as follows. First, we summarize different fundamental concepts and tools that we use in our research. Second, we describe the network configuration for which we have developed the current system. Third, we describe our current approach to object-oriented design and integrating it with HP Openview system. Finally, we give a summary of our efforts.

Background

Unified Modeling language (UML)

Unified Modeling language (UML) is a standard language for writing software blueprints [2]. It may be used to specify different activities of a software development effort: visualize, specify, construct, and document the activities. For example, during the first phase of our project in which we developed the user-interface specification, we used UML. The Use case constructs of UML helped us capture and specify the

intended behavior of the system. The use case diagrams then became a common ground for discussion among the development team as well as with the user and system community. In addition to the use cases, UML supports classes, relationships, packages, activities, and interactions. The concepts are mainly expressed in terms of diagrams. For example, the class diagram in Figure 4 models a bridge, a network infrastructure component. It defines the two attributes of a bridge and also defines its relationships with two other components, a network interface and an IP network device. The functions supported by these are also shown in the diagram.

Figure 5 is an example of a use-case diagram. It shows the actors and their interactions with the system. Within the system, the functions accessed by the actors are denoted as ellipses. Similarly, higher level models are shown in diagrams such as Figures 1-2.

HP OpenView Network Management Solution

HP OpenView (HPOV) Network Management Solution is a standards-based solution that provides IT organization and networks administrators the tools and processes they need to take control of their networking environment [1]. It provides tools for discovery and mapping of the entire network so network changes can be instantly identified and network problems rapidly discovered, correlated, and often automatically resolved. One of the key components of HPOV that we build our system on is the network node management or NNM. This component has the ability to have a single point control of the entire network. In other words, using NNM enables a user to monitor a network from a single workstation. It provides a graphical representation of the entire network. So a user simply has to click on the right part of the map to know the status of a node. The network map is itself built as a hierarchical structure so the system can work for networks with a few nodes to hundreds of nodes. Its graphical interface enables selected performance results to be represented graphically. The system is based on events and it is possible to capture the selected network events by user programs outside the system. This is achieved through the callback interface provided by the system. In addition to the fields provided by the system for each object in the network, users can add new fields. Thus, the management system may be customized depending individual user's needs.

Object-oriented design and development

The term object-oriented design indicates an approach that suggests modular approach to design [3]. For example, using bottom-up-design method, we first specify the basic building blocks, then using these blocks, build higher level modules, and so on. For each such block or module, we specify the interface (or methods) that it offers to the users of the module. In other words, the internal details of implementation of the methods are hidden from the users. In implementing a module, an effort is first made to identify needed functionality that may already be offered by other modules. In this case, instead of repeating the code, calls are simply made to the existing modules. Thus, a system of network of modules is developed.

One may wonder how such an approach is useful in a network management environment. First, most of the current network monitoring and management software is based on SNMP (Simple Network Management Protocol). This protocol is designed with object-orientation and standards in mind [4]. Accordingly, it defines standard or MIB (Management Information Base) variables for each node in the network. It also defines a hierarchical structure for naming the variables. This type of structure has enabled to be easily incorporated in several networking tools such as HP Openview. The network component (e.g., host, bridge, switch, and hub) manufacturers also include software (also referred to as an SNMP agent) in their hardware that support the MIB standards. The system also provides for manufacturer-specific variables to be supported by a component in addition to the standard variables. As mentioned earlier, the strict naming standards for all the variables enable building software to heterogeneous component management much easier.

Let us look at the MIB variable structure in our system. Most components maintain the system-up-time indicating the time (in units of hundredths of a second) the component has been up since the last reset. In the MIB-2 standards it is designated as .iso.org.dod.internet.mgmt.mib-2.system.sysUpTime. Similarly .iso.org.dod.internet.mgmt.mib-2.udp.udpInDatagrams specifies the total number of UDP datagrams delivered to UDP users. In addition to their textual name, each of these variables also has a numeric ID called object ID that is unique. For example, the number of udpINDatagrams has an object ID of .1.3.6.1..2.1.7.1. The object IDs could be used in communicating with the network managers at each component.

In addition to the standard variables, each network component could maintain vendor specific variables. For example, the variable .iso.org.dod.internet.enterprises.hp.nm.interface.serial.serialConfigTable.serialConfigEntry.serialTimeout is specific to Hewlett-Packard's (HP) serial interfaces. This timeout value is used when the management station has initiated a conversation over the serial link and represents the number of seconds of inactivity

allowed before terminating the connection on this serial interface. Clearly, while there is no homogeneity in the time units across different measures, the standards make it easier for a developer to have a prior knowledge of the measures and their units for individual components.

Network Configuration

LaRCNET is the NASA Langley Research Center's Local Area computer network connecting over 10,000 end-user devices together. Initially developed in 1985 to provide local researchers with better access to Langley's Supercomputers, it has grown from a 20-computer network spanning three buildings to its current size connecting 10,000 devices across more than 100 buildings. It is composed of three main Fiber Distributed Data Interface(FDDI) rings: the Isolation LAN, the Direct Attached Network and the main Backbone---each running at 100 million bits per second (100 Mb/sec). The rings interconnect through routers.

The Isolation LAN serves to connect LaRCNET to the outside world. It connects to special purpose networks (AERONet, EOS, NHGS, NREN, NSI, etc.) and to the world wide Internet. A firewall between this ring and Langley's two internal rings provides a measure of protection from external infiltration. Connections range in speed from 1.5Mb/sec to 155Mb/sec. No user devices are connected, however. Traffic on this network, the majority of which goes through the Internet connection, averages 2Mb/sec with peaks at 10 Mb/sec (10% of available bandwidth).

The Direct Attached Ring connects approximately 700 high performance computers directly to the FDDI ring via concentrators - providing 100Mb/sec to the desktop. The second largest of the rings, it contains 65 infrastructure devices spanning 15 buildings using 20 miles of fiber. Typically, hosts are connected through a concentrator directly to the FDDI or via Ethernet switches. Traffic averages 9% of available bandwidth (9Mb/sec) with peaks of 60%.

The main campus backbone is the largest of the rings connecting the majority of Langley's computers to the network at 10Mb/sec. Ethernet-FDDI bridges connect 31 buildings directly to the backbone. These buildings in turn connect to 80 additional buildings. Approximately 550 infrastructure devices (bridges, switches, hubs, and repeaters) interconnect 220 Ethernet segments. Typically a bridge will connect from three to six Ethernet segments containing one or more hubs. Computers attach to these hubs via common telephone wiring. Traffic on the backbone ring averages 15% of available bandwidth with 45% peaks (45Mb/sec) [5].

Overlaying an Object-oriented Design (OOD) onto HP Openview

When deciding how to implement new objects in our system, we needed to choose between creating our own database system or using the existing HPOV database system. We found that the HPOV system along with the HPOV software development kit Application Programming Interfaces (APIs) would be sufficient to implement an object structure representing the NASA Langley network (LaRCNET) infrastructure system.

The HPOV database is created from a network discovery process. The process involves queries to the various network components in the system including routers, hubs, concentrators, bridges, and hosts that make up the whole LaRCNET system. Each of the components is represented as an object in the object database. The "attributes" of an object are initially determined by the object's "capabilities". Some of the capabilities supported by the system are isBridge, isHub, isIP, isPrinter, isPC, and isSNMPSupported. So if a device on the network was determined to have the capability of isIP=True, then that object would be assigned attributes associated with that IP capability (e.g., . "IP Address", "IP Hostname", "IP Netmask"). Of particular interest is when a device is found that has the isSNMPSupported =True capability. This defines the device as **manageable** and further information is acquired from the device using SNMP queries to standard Management Information Base (MIB) tables. When an object is manageable, then HPOV periodically polls its status and produces events within the system based on the object's status. Also, when a user interacts with objects, via symbols on maps, or the menu and toolbar systems within HPOV, additional events are triggered and can be used for individual processing and customization. Standard network management methods are given to us via HPOV's predefined "user interface", but this user interface can be altered and customized to fit the needs of our OOD. These features of HPOV will also produce an event driven aspect to the system that must be modeled within the design.

Once HPOV has completed the discovery process, its database contains all the "instantiated" objects within the LaRCNET infrastructure. All infrastructure type devices support the SNMP protocol and therefore have the capability of isSNMPSupported=True. We can now use the "SNMP sysObjectID" attribute given to each instantiated object to help us in defining its class represented in the design. Based on this information, we can assign

additional attributes (HPOV database fields) to all the objects that have the same value of "SNMP sysObjectID". Using this process, we can create a set of capabilities and/or attributes that reflect the class hierarchy of our design.

Although the objects in our database will not have "methods" attached to them as shown in the class diagrams, there is a way to cause the same effect from within HPOV. Customized programming within HPOV come in the form of "Actions", "Action Callbacks", and "Event Callbacks". The Action Callbacks and Event Callbacks have the same properties as actions. Actions are invoked by user interaction via the menus, map symbols, and toolbars. These menus, map symbols, toolbars and even actions can be prohibited from being available for execution based on a set of capabilities. For instance, we could create a toolbar button or menu item that executes a "ping" action for an object represented by a selected symbol and then only enable those items on objects that have the isIP=True capability. Therefore, by using attributes and capabilities representing our class hierarchy, we can enable actions based on those attributes that would then reflect the hierarchy of methods of our design.

Top-level Subsystem

The top-down view of the system shows a hierarchy of subsystems (Figure 1). This provides us with a means of differentiating, at a high-level, the functionality that is provided by commercial tools like the HP Openview or third party tools that can be integrated within the HPOV environment and the functionality needed to be provided by code written within the HPOV software development kit. All subsystems provided by HP Openview or third party tools that can be integrated within the HPOV environment are labeled with the prefix HPOV.

The **Automated Monitoring System** (AMS) is the subsystem that will establish the events in the network that need to be monitored, determine the criticality level of the events, receive events from the HPOV Event subsystem, and trigger alarms. The AMS will communicate with the SNMP subsystem in order to collect the needed data, from the infrastructure devices, to determine when a threshold has been met. The AMS will communicate with the Processing subsystem to translate low-level device data into higher-level network performance measurements. The AMS will interface with the File subsystem to query and update the Objects attributes such as status, thresholds, and device types. The AMS will also interface with the File subsystem to trigger automated responses using Object methods defined by the Actions in external programs or HPOV registration files. The AMS will interface with the User Interface subsystem by updating the status of a particular object, sending events to HPOV's Event Categories dialog box, or sending users alerts via dialog box pop-ups or an appropriate notification such as email or paging.

The **SNMP System** (SNMPS) is the subsystem that will establish the data needed for collection from the network infrastructure devices, collect the data from the network infrastructure hardware, and distribute the data received from the hardware to the appropriate subsystem for processing and storage. The SNMPS will interface with the File subsystem to determine Object attributes such as community strings, hostnames, and device types. It will also interface with the File subsystem to save long term trending data. It will interface with the Processing subsystem to provide low-level data needed in the high-level network performance equations. It interfaces with the User Interface subsystem by receiving information about the Objects that have been selected by the user, such as ObjectIDs and MIB variable OIDs, and then returning the data gathered from the hardware represented by the selected Objects. The SNMPS will primarily interface with the Processing subsystem to continuously provide data to be used in the network performance calculations.

The **User Interface System** (UIS) is the subsystem that will provide the user a means to interact with the network performance system via dialog boxes, toolbars, menus, HPOV symbols and maps. HPOV uses symbols to represent the instantiated objects within the system and provides many mechanisms for users to use the symbols in a variety of ways. For instance, the symbol color can change based on the status of an object; actions such as menu items can be enabled or disabled based on the attributes of the underlying object. Clicking the symbol causes a selection that enables programs, using the software development kit, to receive that object as a variable, which can then be used within the program. The UIS primarily interacts with the File system. Using pre-defined files and directories, the UIS menus, toolbars, maps, and symbols can be customized for particular network performance monitoring requirements. The UIS interacts with the AMS, Processing, and SNMP subsystems by providing data associated with User Input and receiving data associated with User Output.

The **File System** (FS) is the subsystem that provides customization of the user interface, definition of bitmaps and symbols, definition of object classes and the repository of instantiated objects, and storage of long-term trending data.

The **Processing System** (PS) (Figure 2) is the subsystem that will establish the network performance attributes of interest in the LaRCNET system, provide the equations needed for determining the performance of the network devices, and provide the processing

requested by specific user requests. Many of the interactions with the other subsystems have already been described within the description of those subsystems, therefore they will not be reiterated here.

The breakdown of the PS is shown in Figure 2. The Utilization subsystem is used to calculate the utilization of various device types. This is a good example of how to use OOD to keep a concept abstract and not burden the user with details, because utilization is defined in abstract terms as the percentage of the theoretical capacity of a resource that is being used. Utilization in a bridge may be a measurement of the bits per second flowing through each interface since the bridge doesn't do much work on a per packet basis, while utilization in a router may be packets per second since it's capacity is most limited by this. The user or program will simply ask for the utilization of a device and the proper measurement will be returned. The Throughput subsystem is used for the measurement of the time at which transmission of a packet will take to traverse the network infrastructure. The throughput measurement is most useful for the network infrastructure in terms of getting long-term trending data. The Availability subsystem determines whether a device is currently available, availability for a particular time frame, and total availability since the device was placed on the network. The Error Rates subsystem is used to locate devices that are jabbering (clogging the network with bad packets), find areas of the network that are congested, and give a breakdown of error types. The Broadcast Rates subsystem is used to detect broadcast storms in the network, identify the source of the storm, and alert network administrators when the situation reaches critical thresholds.

Bottom-up Representation of the System

The class hierarchy (Figure 3) defines the bottom-up representation of the system. In the network domain, the lowest element of any network device is the Network Interface. All network devices, regardless of network topology or physical interface type, share this common class. Each network device is assigned a unique Media Access Control Address (MAC) which is ultimately used by all protocols to identify a particular device on the network. The next level by which we define network devices on the network is via the protocol that it is using. In Figure 3, we define three different protocol device classes that each inherits the attributes and methods of the Network Interface class. This shows the building of parent-child relationships.

Network Infrastructure

The final step of defining the physical objects in our system is to "build" the devices based on the classes defined in our abstract class hierarchy of network interfaces. As explained earlier, LaRCNET is composed of infrastructure devices of various types such as bridges, hubs, routers, firewalls, concentrators, and switches. Figure 4 shows the "composition" of a bridge. The Bridge Class is composed using an aggregation of one and only one IP Network device class and one or more of the Network Interface Class per every Bridge Class. Within the various device types, a variety of manufacturer devices are used such as DEC bridges for bridging standard Ethernet and HP bridges for bridging AppleTalk. Even within a particular device type and manufacturer there are various models. Some model differences can be handled by simply having an attribute within the class to contain different values that represent the differences. For instance, a DEC 620 bridge has three Ethernet interfaces while a DEC 900 bridge has six Ethernet interfaces. This difference can be represented within one class by having a Number_Of_Interfaces attribute that would contain the value 3 for a 620 model and 6 for a 900 model. Some differences in models may not lend themselves to an equivalent attribute, but have enough similarities that a parent-child class relationship can represent them.

Use-case Diagrams

The use-case diagrams (in UML) describe the interactions of the system with the user. In other words, describe the user interface of the system. Here, we describe one of the use-cases of our system, the monitor use-case. Monitor is a user entity that monitors network status and performance. We describe our monitor use-case in Figure 5. As shown here, the monitor has the capability to check the availability (of system or infrastructure components), check their utilization, and check the presence of broadcast storms (a transmission is said to be a broadcast when it is to be delivered to multiple machines, typically all, across the network). In addition, it has an option to locate a device within the network topology. It can also check for network configuration errors, and trigger alerts when certain undesired conditions have occurred. In this diagram, one can also observe other roles such as the administrator and the analyst that interact with the monitor through the underlying system functions. Of course, network devices and data collection files are also responsible for offering the proposed services to the monitor role.

High-level Monitoring Functions

The system is designed to support several high-level performance metrics which are otherwise not supported by the current HP OpenView system. For example, consider the availability function defined over infrastructure components. Clearly, the underlying HP OpenView system provides sysUpTime or the time since the last recovery of the component. It also has the

ability to plot the sysUpTime as a function of time. However, the underlying system does not provide device availability which is the fraction of the time the device under consideration is available to the rest of the system. While it can be computed from the plots, it is a tedious process for a high-level user. In addition, there are other complications such as determining whether a device is actually down or it is simply not accessible as one of the bridges or routers to which it is connected (directly or indirectly) is simply down.

Similarly, whenever a device is inaccessible administrators would like to know the cause for its inaccessibility. Currently, it is not possible to know directly to which infrastructure component and to which port is the device connected in the network. As before, while the information can be obtained through several queries, it cannot be known directly by a user. We have provided functions through which given a device name, MAC address, or its IP address, the system can provide complete information as to how it is connected in a network.

We also propose to provide some information about path availability, throughput, utilization, error rates, and broadcast rates.

Due to the object-oriented nature of the underlying system (HP OpenView) as well as our design approach, we are able to build the system incrementally and test it.

Summary

In this paper, we have summarized our current efforts to build an object-oriented monitoring system for management of large networks. In particular, we have considered the network infrastructure components at NASA LaRC to design and implement our system. We have used the Unified Modeling language (UML) for high-level design of the system. The system is built on top of HP OpenView. It implements several high-level functions such as component availability analysis. It also provides a good interface through which users including analysts and network administrators can interact and obtain system information.

Acknowledgements

The work is supported in part by a grant (NAG-1-2143) from NASA Langley Research Center, Hampton, Virginia.

References

1. HP OpenView: http://www.openview.hp.com

2. Grady Booch, Ivar Jacobson, James Rumbaugh, "The Unified Modeling Language User Guide," The Addison-Wesley Object Technology Series, October 1998.

3. Grady Booch, "Object-Oriented Analysis and Design With Applications," Addison-Wesley Object Technology Series, February 1994.

4. William Stallings, "Snmp, Snmpv2, and Rmon : Practical Network Management," Addison-Wesley Publishing, July 1996.

5. David B. Yeager and Michael D. Bray, LaRCNET 1997 -- A status Report, NASA Langley Research Center, 1997.

About the Authors

Ravi Mukkamala received his Ph.D. degree from the University of Iowa in 1987 and M.B.A. from the Old Dominion University in 1993. Since 1987, he has been with the Department of Computer Science at the Old Dominion University, Norfolk, Virginia, where is currently an Associate Professor. Dr. Mukkamala's research interests include distributed systems, real-time systems, networks, data security, and performance analysis. His research has been sponsored by NRL, DARPA, and NASA.

Eli Siman-Tov received his Electrical and Computer Engineering Masters degree from Old Dominion University of Norfolk, VA in 1995 and Electrical and Computer Engineering Bachelors degree from University of Tennessee of Knoxville, TN in 1989. Since 1986 he has been with NASA Langley Research Center, Hampton, VA where he is currently a Computer Engineer with the Computer and Communications Systems Branch. Mr. Siman-Tov's research interests include all aspects of digital design with special interest in RISC processor design, real-time mission critical space-flight software; Object Oriented Analysis, Design, and Programming; and Network Communications. He has analyzed, designed, and produced embedded software for the CERES and MIDAS space-flight instruments as well as the Hyper-X supersonic airplane project.

Louis Galland received a BS in Computer Science from Christopher Newport University in 1996. He has worked in communications and networking at NASA Langley since 1989 where he serves a Computer Engineer in the Communications and Computer Systems Branch. His professional interests include World Wide Web development, networking and graphics.

Figure 1. Top-level Subsystem Diagram

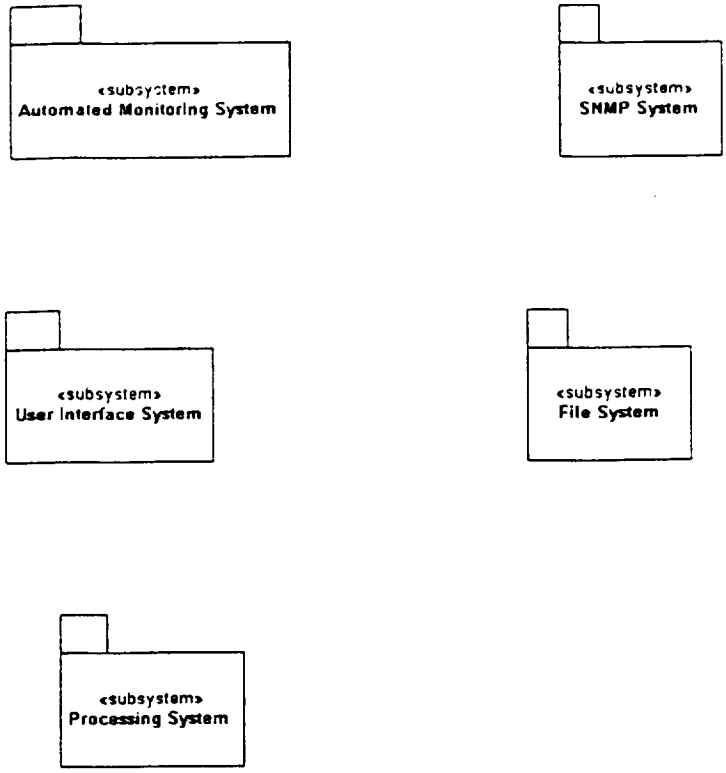


Figure 2. The Processing System

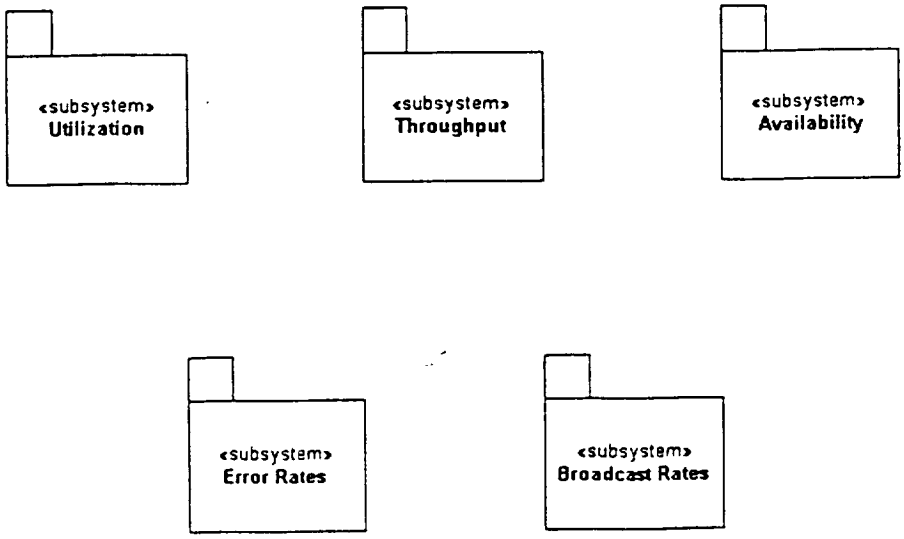


Figure 3. Network Interface Class Diagram

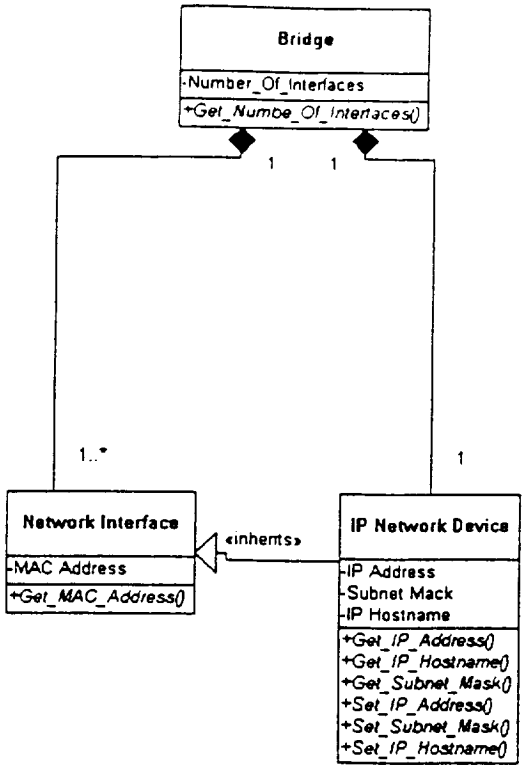


Figure 4. Bridge Infrastructure

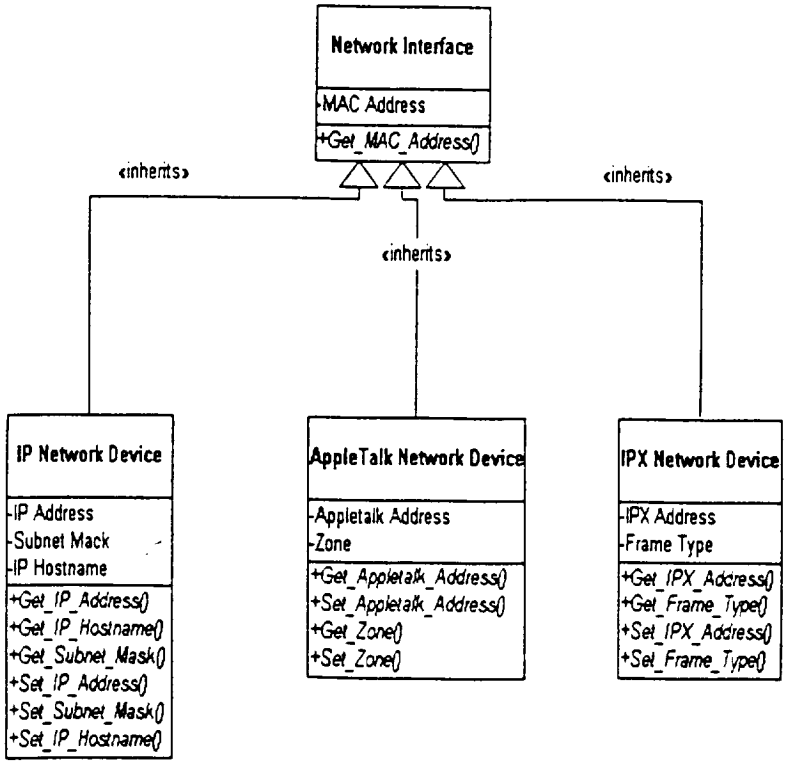


Figure 5. Monitor Use-case Diagram