16th International Communications Satellite Systems Conference

A
COLLECTION
OF
TECHNICAL
PAPERS

Part 1

Washington, DC
February 25-29, 1996

HYBRID NETWORK MANAGEMENT

John S. Baras, Mike Ball, Ramesh K. Karne,
Steve Kelley, Kap D. Jang, Catherine Plaisant,
Nick Roussopoulos, Kostas Stathatos,
Andrew Vakhutinsky and Jaibharat Valluri

Center for Satellite & Hybrid
Communication Networks
Institute for Systems Research
University of Maryland
College Park, MD 20742
(301) 405-7901

David Whitefield

Hughes Network Systems
11717 Exploration Lane
Germantown, MD 20876
(301) 212-7909

Abstract

We describe our collaborative efforts towards the design and implementation of a next generation integrated network management system for hybrid networks (INMS/HN). We describe the overall software architecture of the system at its current stage of development. This network management system is specifically designed to address issues relevant for complex heterogeneous networks consisting of seamlessly interoperable terrestrial and satellite networks. Network management systems are a key element for interoperability in such networks. We describe the integration of configuration management and performance management. The next step in this integration is fault management. In particular we describe the object model, issues of the Graphical User Interface (GUI), browsing tools and performance data graphical widget displays, management information database (MIB) organization issues. Several components of the system are being commercialized by Hughes Network Systems.

Introduction

Hybrid communication networks provide an economically feasible and technologically efficient means to implement the global information infrastructure. The management of such heterogeneous networks is a critical market differentiator for telecommunications companies and a formidable technical task. In this paper we describe the collaborative effort between the University of Maryland and Hughes Network Systems, under the auspices of the Center for Satellite and Hybrid Communication Networks, to design and implement an integrated network management system for such networks. This paper is a continuation of [1] and provides a description of our progress in the second year of this two-year joint research and development effort. The major accomplishments during the second year were: the extension of the object oriented data model to hybrid networks consisting of satellite networks and terrestrial ATM networks; the extension of the system to hybrid networks with as many as 300,000 nodes; the improvement of the browsing graphical tools so that mesh-connected graph networks (as opposed to tree networks) can be efficiently queried; specially designed graphical widgets for displaying performance data continuously from the network management information database (MIB); extensions of the GUI to distributed operation including appropriate designs for consistency and concurrency between the GUI display and the network MIB; storage and organization issues for performance data in the MIB; integration of configuration management and performance management.

A typical network for which the system developed is intended is shown in Figure 1. In addition to the heterogeneity stemming from the interconnection of a terrestrial ATM network to a satellite LAN network, the system must handle vendor and protocol heterogeneity as well as operate in a distributed interactive(with the operators) environment.

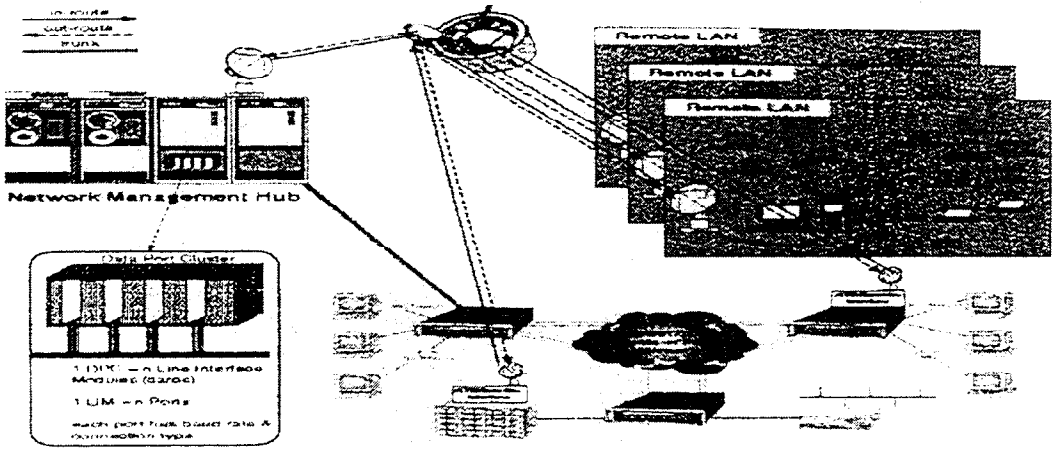

Figure 1. Illustrating a typical hybrid network under investigation

System Architecture

The approach we are following in designing and implementing the INMS/HN is as follows. We first represent the network in a carefully designed Object Oriented data model in an OODB, following the principles of [2]. We develop advanced GUIs linked to this OODB representation of the network including efficient browsing tools which exploit hierarchies in the data model. The OODB is linked to network simulation for comparisons and "what-if" decision assistance. We employ innovative dynamic query techniques which can be invoked from the GUI.

We develop and implement performance objects in the OODB and link them to sophisticated graphical widgets in the GUI for performance monitoring and management. We allow multi-resolution (temporal and in dynamic range) performance data storage for economy of storage and speedy recovery of relevant information. We embed operational and management constraints in the OODB and we embed multi-criteria optimization tools and fast search algorithms in the OODB for fast trade-off analysis and decision assistance. The resulting architecture of the software system is shown in Figure 2.

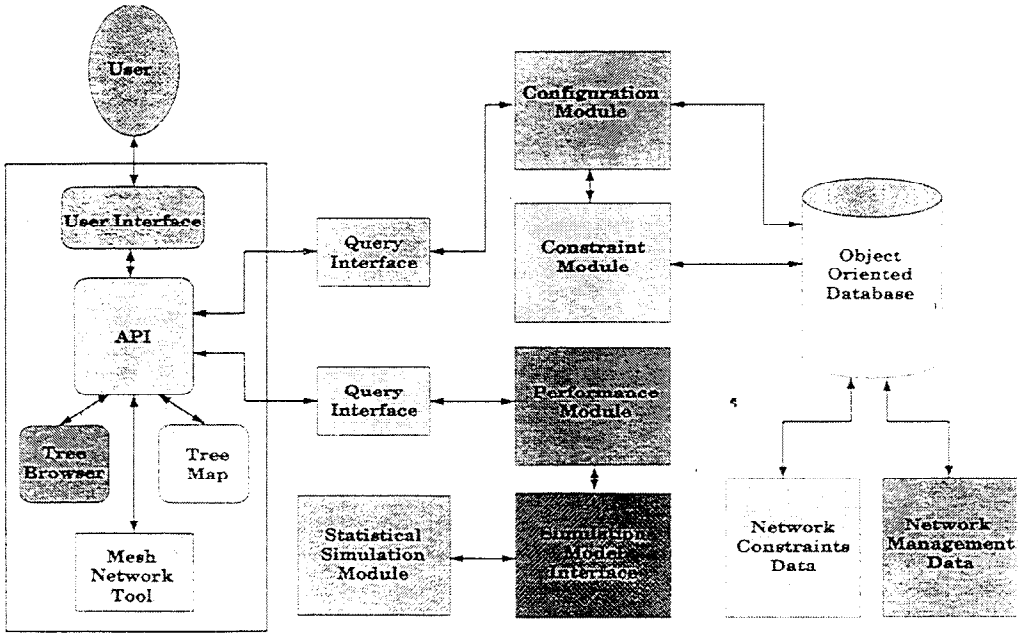

Figure 2. Architecture of the Integrated Network Management System

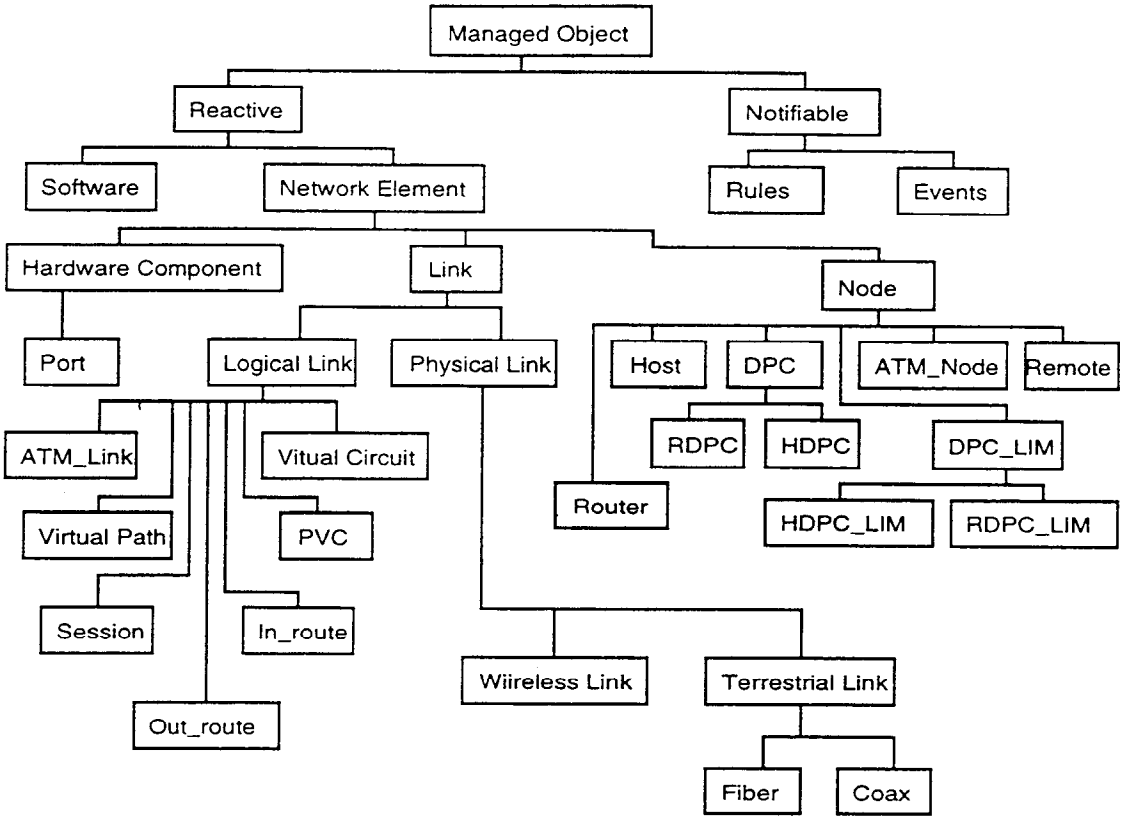

Figure 3. The implemented object class hierarchy for the hybrid network data model

We have currently completed a prototype of integrated configuration and performance management. Our next milestone is the efficient integration of fault management as well. The current implementation has been tested with simulated data of a 300,000 node hybrid network of the type shown in Figure 1.

Extensions/Improvements of the MIB Design

Object Oriented Data Model

We have extended the data model to include ATM networks. We also include frame relay X.25 over ATM in the terrestrial portion of the hybrid network. A hybrid network with an excess of 300,000 nodes and links was created and stored in Object Store. The object model hierarchy is depicted in Figure 3.

The various object classes implemented, and their descriptions are provided below:

Managed Object: This is an object which can be managed by software with respect to Network Management. It is the highest layer of hierarchy in the Network Management System. There are no attributes and functions defined for this object.

Reactive: Reactive object is derived from the Managed Object. In order to implement constraints using the Sentinel method, we need to define this object. Reactive objects are related to rule objects. That is, each rule can subscribe to one or more rule objects. Reactive objects provide a message passing mechanism to rule objects through a notification method.

Network Element: Network element is a reactive object. It has an individual identification, name, type to distinguish from other objects. Network Element may consist of nodes, links, groups and other elements of a communication system.

HW_COMP: Hardware component is a part of the Network Element. There can be one or more HW_COMP in a given example. HW_COMP is associated with a given layer (between 1 and 7). A hardware component is a piece of equipment that is indivisible from a management perspective, i.e., a component may only be managed as a whole and not in terms of its parts.

SW_MOD: Software module is a part of the Network Element. There can be one or more SW_MOD in a given Network Element. SW_MOD depends upon the

functionality that is required in the communication network.

Node: A node is a system that is a source, a sink, or a relay/transformation point of information. Node is a complex object that is composed of smaller units. The smaller units can themselves be subnodes (or nodes), hardware components, or software modules.

ATM Node: This class inherits from the Node class. This server is a model for a generic ATM switch. Every ATM switch has a set of input and output ports, along with an ATM translation table. The entries in the ATM translation table are used to switch Virtual Circuits(VC) or Virtual Paths(VP) depending on whether the switching is done at the VP or VC level. This class can be further *specialized* to model vendor specific ATM switches.

Router: This class inherits from the Node class. A Router is a "network" layer device in the OSI 7 layer model. Each router has a routing table and a set of network interfaces. The routing table entries are used to forward incoming packets to appropriate destinations. Routers are available from various vendors. This class can be further *specialized* to in order to model Routers from specific vendors.

Host: This class models generic Hosts. Workstations, Xterminals, Mainframes, PC's are Hosts found in any of today's LANs. These classes can be further derived from the Host class. The attributes present in this class would be number of applications, number of active sessions etc.

Link: Links are paths of communication between network devices, i.e, the medium through which communication takes place. Links may be physical or logical. Physical links implement a direct communication path between devices. Logical links are communication paths between devices that are composed of potentially several links across several intermediate devices, i.e. virtual circuits or datagrams.

Physical Link: This is a model for a generic Physical Link. The medium could be wireless or wired. It could be a point-to-point link or a multi-drop link.

Terrestrial Link: This class models any generic wired Terrestrial Link.

Fiber: This class models any optical fiber link.

Co-axial cable: This class models co-axial cable links.

Wireless Link: This class models all wireless links. Wireless links could be terrestrial and in different frequency ranges like cellular, PCS, microwave or infrared or they could be satellite links. On the other hand they could be indoor or outdoor. Different kinds of links have different propagation characteristics that need to be modeled based on the environment.

Logical Link: This class is used to model generic Logical Links. Logical links are typically made up of one or many Physical Links. There is a *is-implemented-in-terms-off* relationship between a Logical Links and Physical Links. This is a many-to-many relationship since each Logical Link could have several Physical Links and each Physical Link could be part of several Logical Links.

Virtual Circuit: Virtual circuits are constructs defined in packet switched networks. ATM networks also use this construct. In many ATM references they appear under the name Virtual Channels but they essentially represent the same construct. A Virtual Circuit is a Logical Link defined between two ATM Nodes or two user machines.

Virtual Path: Virtual Paths are a construct defined in ATM networks. An ATM Physical Link could contain several Virtual Paths. Each Virtual Path contains several Virtual Circuits or Virtual Channels. An operator can define certain Virtual Path Connections (VPC) between different ATM Nodes. These are relatively static and are changed by the operator when bandwidth allocations need to be modified in order to provide the desired Quality of Service(QOS). Bandwidth is allocated to Vitual Paths and VPCs by the operator so that connections can be established with smaller delay when a user issues a connection request. Certain Virtual Paths are allocated for carrying signaling and control information.

PVC: A Permanent Virtual Circuit (PVC) is a permanent connection defined between two user machines or two ATM switches. It is a construct defined by the operator of a network in order to facilitate connection establishment.

Performance Data Model and Storage

We have developed and implemented efficient methods for storing and viewing performance information from a large hybrid network. A network simulation was designed and implemented in order to populate the MIB with performance data and related statistics. This simulation can set up Permanent Virtual Circuits (PVC) and vary traffic over them. The simulation periodically reports network traffic, error rate

and cell loss rate which are stored in "Performance Objects" in Object Store.

The system supports two types of user queries:

- Queries on a single object: Typical queries would be
  - Utilization of a particular Link at some specified time.
  - Buffer capacity at a given Node.
  - Delay and Error rate over a specific link.
- Queries across objects: These queries are more complex and involve attributes of more than one object. Typical queries would be of the form
  - The aggregate delay over a specific Virtual Circuit.

Such queries would require computation based on the attributes of different objects.

The operator may want to see the state of the Network at some earlier instant in order to analyze the nature of a fault that may have occurred in some part of the Network. Hence it is critical to store, the state of various Network elements at different instants, along with the instant at which it was recorded for a sufficient period of time. It would be neither practical nor necessary to store all the information gathered over a period of time at the same granularity. A reasonable solution to this would be to reduce the precision of information stored as the information gets older, i.e. for the most recent information we could store every update from the network, for slightly older information we could store an average over 4 periods, for even older information we could store an average over 20 periods etc. At the same time we would like to satisfy the following conditions for every query

- Coherency: If this is satisfied then the information reported together would have been recorded in the network at approximately the same instant. It is essential that this condition be satisfied in order to report a coherent state of the network because values reported together should not have been recorded at two vastly different instants.
- Regency: In addition to the above condition we would like to report information recorded as close in time as possible to the one requested by the user.

Sensors located in different parts of the network would report data at different time instants. This may cause an update to the central server every couple of minutes. The server may not be fast enough to store each update into the database. Hence instead of updating the database each time an update comes in, it would be better to store a few updates in memory and do a block of updates to the database. In case of a crash the values lost can always be retrieved by polling the network elements. This would improve upon the limitations of a centralized server by increasing the throughput.

There are three different processes in our implementation, one for each level of granularity. The high precision process periodically takes values stored in memory and updates the database. The processes at the other two levels periodically take a block of values stored at the previous level of precision, compress them and store them at the next lower level.

We have implemented a performance model that is suited for distributed implementation. Sensors periodically report snapshots of different network elements recorded at certain time instants. These snapshots include all the performance parameters being monitored for each network element. Since all the performance parameters for each time instant are being reported together it would be more efficient if we stored them together rather than storing them in different objects. Another point in favor of this kind of storage is that in cases where the performance of a particular network element is degrading, we might want to see all the parameters for that network element recorded at some time instant. In the case of such queries it would be inefficient to search for these values across several objects.

The three levels of precision are still maintained. There are pointers from the network elements to these objects. Each network element has pointers to three sets of values corresponding to the different levels of precision. Different sets contain snapshots of the network element stored at different levels of granularity.

This structure as mentioned before is suited for distributed implementation since network elements along with their performance parameters can be migrated to other servers.

We have also investigated methods for finding the appropriate precision levels, as well as for efficient storage of time series data (such as those from the network sensors) and their related statistics.

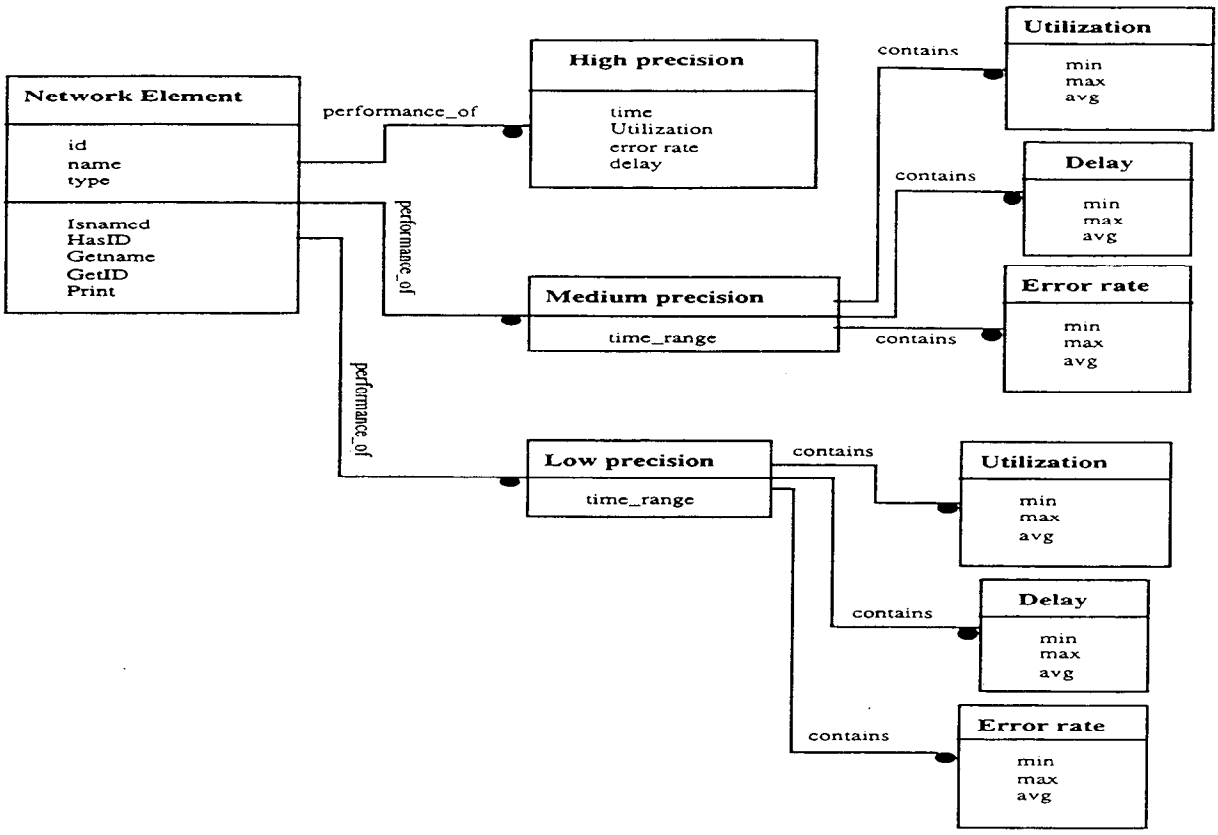

Figure 4. Integrated performance data model

Improvements in GUI Design

Efficient Browser for Mesh Connected Graph Networks

In our previous work [1] we designed and implemented efficient browsers and visualizations of the OODB representing the network, by exploiting the tree structure of the network. In the present extension to hybrid networks which include ATM terrestrial networks, we have to deal with fully mesh connected graph topologies, not just tree topologies. Therefore, there are no unique or obvious hierarchies to drive the browser, like the Tree Map and Tree Browser of [1]. Instead we designed the Mesh Network Browser which explores the various partial orders the operator can create by using subnetworks of the network. This browser can then be used to invoke dynamic queries in the underlying OODB representation of the network, for selectively viewing desired parts of the hybrid network. The display/visualization of the network obtained using the Mesh Network Browser is shown in Figure 5.

We have adopted the Sgraph structure to display large hybrid network configuration stored in an OODB. The configuration data in the database consists of network nodes, links, and connectivity information between nodes. However it does not include x,y coordinates to efficiently display a three dimensional network onto the two dimensional computer screen. The Mesh Network Browse constructs an Sgraph structure corresponding to the network objects displayed. Each network object has an assigned icon, of scaleable size. Then by the layout algorithm employed, it traverses the Sgraph structure to assign x,y coordinates for each network node (object) without allowing nodes to overlap and eliminating unnecessary edge-crossings at the same time. Based on the Sgraph data structure, several layout algorithms are currently available; we have used the Springer Embedder algorithm which emphasizes display of symmetry and isomorphic components.

Nevertheless, a major problem we have addressed is generated from the fact that the area of a computer screen cannot visibly represent large networks including thousands of nodes. We designed our Mesh Network Browser so as to allow the operator to select the subnetworks and components of each subnetwork and even network nodes that he/she wants to query for network information. Our method provides an important innovation and it is a significant departure from current commercial practice, where this browsing

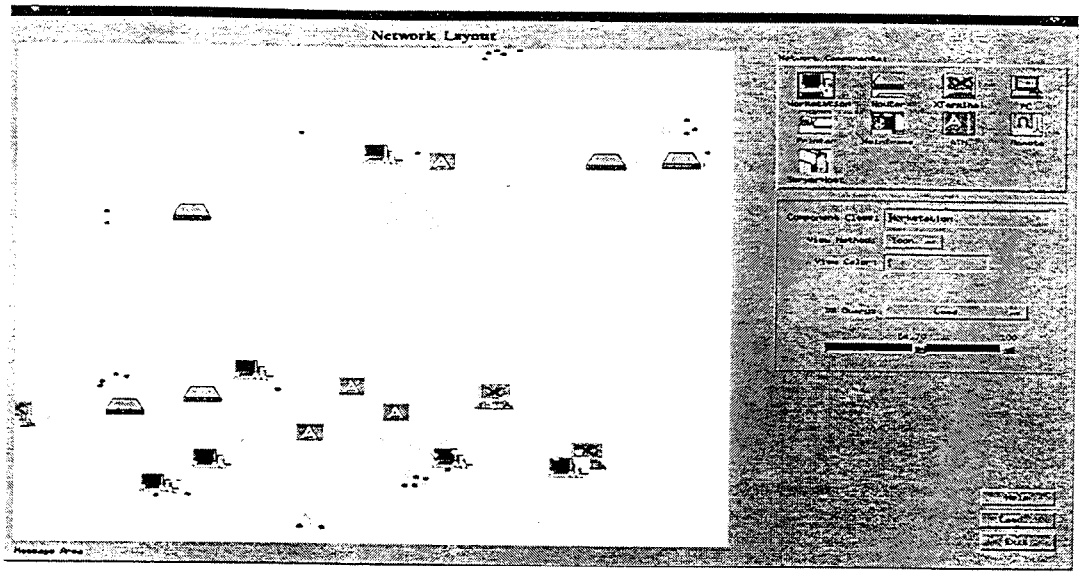

Figure 5. Illustrating the Mesh Network Browser display

is typically done using zooming of subnetworks or nodes. We allow for network component selection and display across a subnetwork or node hierarchy. Our browser allows selection capabilities based on a network element menu, based on subnetworks and also based on filtering of network elements using specific assignable attributes. This is implemented by using the option menu to allow the user to select specific attributes of nodes and by using a Range widget to specify the range of the attribute. For instance, Figure 5, only displays workstations within a certain range of the load attribute. Our browser and display also categorizes network components into several groups. It allows the user to quickly navigate a network group for status and concentrate only on group components which are of interest. Components in the layout are represented as an icon, a colored dot, or are completely hidden by user definition or filtering.

Our browser is directly connected to the OODB representing the network and retrieves data (or objects) using dynamic queries. In this way, it represents an efficient way to capture and display network component status dynamically, for continuous network monitoring and management.

Wheel Widget Performance Data Display

One of the critical requirements for the INMS/HN is to design and implement efficient displays of continuous monitoring of performance data in an integrated fashion; i.e. simultaneous visualization of several attributes or performance metrics. We have designed and implemented one such graphical display/visualization: the "Wheel Widget."

The Wheel Widget displays four numeric values that fall within their upper and lower bounds. The widget allows the user to change those values interactively using a grab & drag mechanism. The widget is also useful and intuitive to use if it is operated with four Range widgets as controllers of its interface. Figure 6 shows how a Wheel Widget can be used with four Range widgets in an application. The widget allows the user to change its data, ranges, colors, size, etc. at run time.

Terminology

spoke : a ball shape object which corresponds to a data having multiple attributes.

ring: circles which have different radius. The ring data of a spoke would be represented by a relative distance from the center of the Wheel.

rim: logical scale along the ring. The rim data of a spoke would be represented by an angle between vertical axis and the spoke.

grab & drag : press and hold on mouse button and move the mouse.

click: press and release a mouse button on.

Range widget: A user defined motif widget which displays a range of numeric values that falls within upper and lower bounds.

In Figure 6, each spoke in the widget represents network performance/monitoring data which has multiple attributes. It might be characterized by four attributes related to the corresponding data. This Wheel widget is used to display large numbers of point to point connections from a single source along with several attributes about each connection at the same time. Each spoke represents the following attributes:

1. Utilization which is represented by a distance from the center of the wheel to the center of a spoke.
2. Virtual Path Identifier which is represented by an angle from the vertical axis clockwise.
3. Capacity which is represented by the size of a spoke.
4. Throughput which is represented by the color of a spoke. The widget provides a spectrum of colors used to represent a range of colors.

The Wheel Widget provides direct data-filtering capabilities on attributes which are related to the ring (distance from the center to spokes) and rim (angle between vertical axis and spokes). To change the maximum (minimum) of the ring attributes, grab & drag the left (middle) mouse button on one of the rings. To change the maximum (minimum) of the rim attributes, grab & drag the left (middle) mouse button on one of the tick marks along with the outmost ring clockwise or courterclockwise. These two capabilities also can be achieved by other widgets such as the Range widgets indirectly. Size and color attributes of the spoke can be used for data-filtering by other widgets indirectly. The Wheel widget provides the exact values of four attributes of the corresponding data if the left mouse button is clicked on a spoke.

The Wheel widget can continuously display network monitoring information and can therefore be incorporated in feedback schemes for automated network management.

Distributed Concurrent Display Of Network Data

Network management systems must be able to operate in a distributed environment. This requirement creates several problems related to the consistency, concurrency and performance of the GUI and its link to the OODB under distributed conditions. We describe here a summary of our results and their implementations for network management. For the full details we refer to [3].

One of the main concerns of application developers is that users are very sensitive to the response time of the system. Lengthy response times are usually detrimental to productivity, increasing user error rates and decreasing satisfaction. Also, users tend to establish expectations of the time required to complete a given task based on past experiments. Unexpected delays usually trouble or frustrate the users. Therefore, high variability in the response time of the user interface should be prevented. So, building a GUI that displays large amounts of information stored and managed by a DBMS can be very challenging. Many potential performance problems exist since the response to a user action may require extensive data processing, a number of network of message exchanges as well as several data retrievals from secondary storage.

Client data caching appears as the best approach to deal with the performance problems of the user interface. Database objects cached in the client's main memory can be directly used for user interface manipulations. This can reduce secondary storage accesses and client-server communication overhead. However, data caching as has been implemented in current systems does not completely address the user interface requirements.

A non-trivial problem for the graphical user interfaces of database applications is presenting a consistent and up-to-date view of the database. This problem is more evident and more difficult in a multi-user environment (as network management) where different users may view and possibly update the same database objects. Obviously, some sort of display synchronization mechanism is required which preserves the consistency of the user interfaces, under the performance requirements mentioned above. Generally, the straightforward approach of periodically refreshing the user interfaces is not considered acceptable since it may cause excessive overhead.

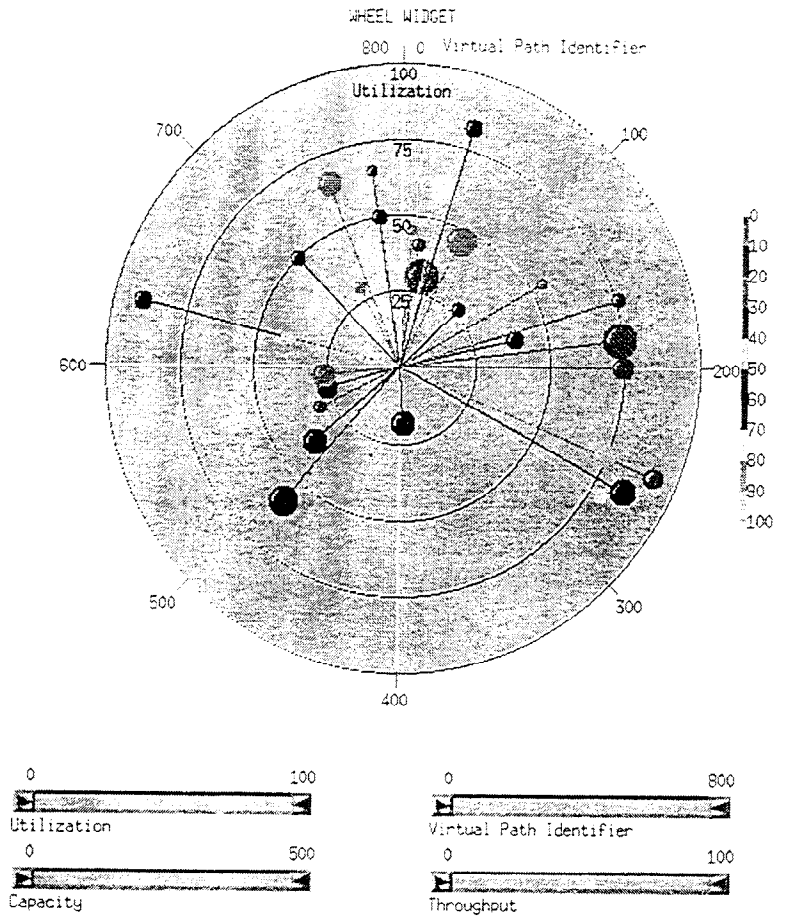

Figure 6: The Wheel Widget visualizing network monitoring data

From the database perspective, the *display consistency* problem is not much different than the client cache coherency problem. GUIs retain graphical representations of database objects much like caches keep copies of these objects. Therefore, the consistency requirements imposed upon the database system by user interfaces are similar to those of client caches.

Displaying some database objects can be considered a kind of long transaction, a *display transaction*, which spans the lifetime of the display. However, traditional transaction semantics cannot be used to preserve GUI consistency since they are much too restrictive.

We propose that, for each interactive application, a proper *external display schema* should be defined over the existing database schema. Such display schemes are composed of *display classes (DCs)* that encapsulate the desired user interface functionality and form inheritance and/or containment hierarchies that better meet GUI requirements (e.g. for screen layout computation, for screen navigation etc.) both in terms of implementation effort and runtime efficiency.

The definition of a DC depends on the database class(es) it represents as well as the user interface context. It should include only attributes and methods that are necessary for the display and manipulation of the corresponding user interface elements. These attributes may be a subset of the database class(es) attributes as well as additional GUI specific attributes (e.g. screen coordinates).

The graphical elements that compose the image displayed by a GUI must be instances of display classes, i.e. *display objects (DOs)*. Display objects are created by copying and/or computing the necessary information from database objects. During their lifetime, they are explicitly associated and kept consistent with those database objects. This association turns the collection of display objects into an active

(updatable) view of the database as opposed to a passive snapshot. We also propose the introduction of *display cache* as an additional level in the memory hierarchy on top of the client's database cache.

Since, the display cache replicates data that may already exist in another part of the same physical memory space, it may appear as an unnecessary overhead. However, it has two major performance advantages over traditional caching [3].

Detection-based protocols, which allow stale copies of data to reside in the client's cache, are not suitable for display objects. Moreover, within the display's lifetime there are no transaction boundaries, thus there are no clear points when data consistency should be validated. The user interface, therefore, needs to be somehow notified on relevant data updates so that any necessary action can be taken (i.e. redraw the updated part of the display). This makes avoidance-based protocols more appropriate since, under such a scheme, data validation is initiated by the server whenever necessary.

However, these protocols are mostly designed to enforce strict transaction correction. For the relaxed correctness requirements of display transactions we propose a non-restrictive form of shared locks, called *display locks*. These are non-restrictive in the sense that display locked database objects can be updated, provided that at any time all lock holders get notified about the updates committed to the database.

The display locking protocol is quite simple and can be easily integrated with a strict avoidance-based protocol. A client requests display locks for all database objects that are associated with display objects. The database lock manager on the server is expected to grant those locks, since display locks are compatible with all types of locks. When a transaction wants to update some data, it does so after obtaining an exclusive lock for that data. When the update is committed to the database, the lock manager releases the exclusive locks and notifies all clients that hold display locks on the updated data. The notified clients refresh the associated display objects (and therefore the display) by reading the new data from the database. We call this protocol *post-commit notify protocol.*

We have demonstrated these ideas in a multiple user, limited functionality version of a network configuration management application. This application employs two different visualization techniques, the *Tree-Map* and the *PDQ Tree-browser*, to display complex hardware hierarchies [1]. ObjectStore, a commercial object-oriented database system, was used to store the network database.

The implementation included three major tasks:

1. Extend the database server with display locking capabilities,
2. Enhance the client applications structural design to incorporate the display locking mechanism, and
3. Design the user interface in terms of defining appropriate display classes for the the tree-map and PDQ tree-browser

The overall system architecture is presented in Figure 7. For more details on the implementation for network management we refer to [3].

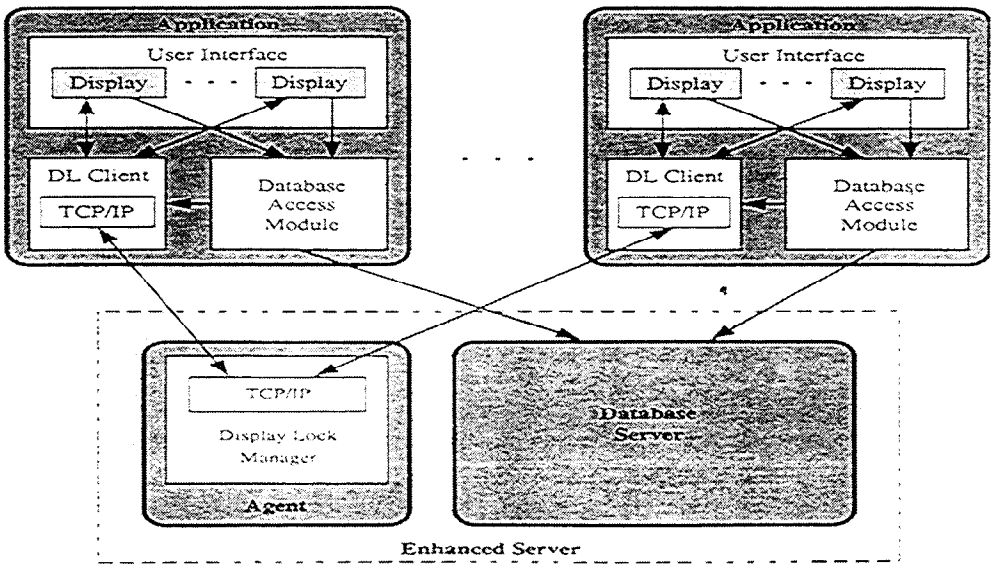

Figure 7. Implementation Architecture

Conclusions

We presented the design and implementation of an integrated network management system which incorporates several advanced technologies predicated by current and future hybrid network management requirements. The next step in our efforts is the integration of Fault Management to the INMS/HN. The prototype implementation has given ample evidence of the advantages offered by our techniques and implementation methods. As a result several of these innovations are being commercialized to HNS commercial products.

Acknowledgments

This work was supported by the Center for Satellite and Hybrid Communication Networks, under NASA contract NAGW-2777, Hughes Network Systems and the State of Maryland under a cooperative Industry-University contract from the Maryland Industrial Partnerships Program (MIPS).

References

Baras, John S., et al., January 1995, "Next Generation Network Management Technology", in AIP Conference Proceedings 325, Conference on NASA Centers for Commercial Development of Space, Albuquerque, NM, pp. 75-82.

Haritsa, J.R., et al, December 1993, "MANDATE: Managing Networks Using Database Technology", IEEE Journal on Selected Areas in Communications, pp. 1360-1372.

Stathatos, K., S. Kelley, N. Roussopoulos and J.S. Baras, "Consistency and Performance of Concurrent Interactive Database Applications", Institute for Systems Research Technical Report TR 95-79.