A Framework for Spatial, Temporal and Functional Aggregation of Management Information

Nikolaos Anerousis ***Alexandros Biliris***

AT&T Labs Research
180 Park Ave., PO Box 971
Florham Park, NJ 07932-0971
tel: (973) 360-8767, fax: (973) 360-8050
e-mail: {nikos, biliris}@research.att.com

Abstract

We present a framework that allows the rapid development of scalable network management services with a particular emphasis on data aggregation and visualization. A new information model allows the aggregation of management information in spatial, temporal, and functional forms. The generated abstractions are stored in a special management agent which provides persistent storage facilities, event correlation services and access modules to information that is only available through one of the standard network management protocols such as SNMP and CMIP. A query processing subsystem handles operations on network abstractions that cannot be expressed in any of the above forms of aggregation. Simple network management clients such as web browsers can navigate through the different levels of aggregation and visualize managed objects of arbitrary complexity by dynamically loading the appropriate viewers/controllers. The system is being evaluated on the SAIL experimental home access network.

Keywords: Network Management, Information Aggregation, MARVEL.

Corresponding Author: Nikolaos Anerousis.

1. Introduction

The main challenges in providing comprehensive network management services for today's integrated networks lie in two main areas: how to handle heterogeneity (components of different types from different manufacturers), and scaleability (large number of network elements). The heterogeneity problem is being addressed through standardization efforts within organizations such as the IETF and the Network Management Forum. Today, a number of specifications are available for managing TCP/IP stacks (MIB-2 and RMON), ATM switches, etc.

There are no widely established methods however for dealing with large numbers of network elements. Most approaches for reducing state and event information in commercially available network management (NM) platforms are ad-hoc and usually customized for a particular management problem or network. Management of large networks requires powerful abstractions that capture the essentials of the state of the network rather than the details.

The first generation of network management tools such as HP Openview, Sun Net Manager, IBM Netview, etc., rely on the point-to-point management model. According to this model, a network management application (NM client) connects to a management agent (NM server) using one of the standard protocols for management such as SNMP or CMIP. The agent contains information about a network element or a group of elements. The manager retrieves or controls this information by issuing get and set operations. Especially in SNMP systems that do not support rich data types, this exchange of management information is at a very low level. As a result, all the intelligence for providing more complex NM services should reside within the client. First generation tools are characterized exactly by complex and expensive clients. These clients have the capability to maintain a hierarchical topology map and thereby provide easier navigation through a possibly large network. The manager however still has to use a low level management protocol with every network element. First generation systems offer few capabilities to customize the available management services beyond the functionality offered by the underlying NM protocol such as SNMP or CMIP.

Second generation NM systems have tried to address some limitations of the first generation by providing better customization options and automation of routine management applications. Tools such as Tivoli's TME 10, CA's Unicenter TNG and NetExpert are targeted at the corporate IT infrastructure and offer a richer set of management services including end-to-end application management, network services management, software distribution etc.

Although these tools simplify to a large extent the effort required to manage large numbers of network elements and applications, we still feel that they are customized to work with specific products and network protocols. Fundamentally, the network manager has little control over what management services are presented to him/her and how information is aggregated, stored and visualized.

Our work aspires to provide an environment for building scalable network management services, that addresses many limitations of currently available platforms. The foundation of our framework is a management information model that allows the aggregation of management information to reduce complexity and a distributed object services model that allows the definition of rich data types and management services. The framework implements the management system as a hierarchical database of aggregated information. Tools are provided for easy navigation and visualization.

This paper is organized as follows: Section 2 presents the information model for aggregation and service provisioning. Section 3 describes the MARVEL environment which was developed to implement the information model and provide a platform for deploying scalable management applications. Current applications of MARVEL are presented in Section 4. Section 5 outlines related work in the field and Section 6 presents our conclusions and directions for further study.

2. Architecture

Large network management systems collect a tremendous amount of information from network elements and make it available to network operators in a myriad of formats. In order for this information to convey the essence rather than the details of network state, it must be organized, summarized and simplified as much as possible. Symmetrically the network manager needs mech-

anisms that aggregate the control of a large number of network elements into simpler interfaces. We distinguish between the following methods of aggregation:

Spatial aggregation, where information is collected from a number of components and a summarization filter is applied. For example, the ingress traffic to a network region can be computed by summing traffic information collected from switches at the border of the region.

Temporal aggregation, where information is collected periodically to form a time series of various granularities (minutes, hours, etc.) or provide an autocorrelation measurement.

Functional aggregation, where information from different functional areas of a management system is combined to construct a functional view of a network element or service. A subscriber's profile that contains the subscriber billing information, CPE hardware configuration, performance measurements, etc., is an example of functionally aggregated information.

2.1 The Network Element Grouping Model

The main difficulty behind creating spatial aggregations is the need to specify and maintain a (possibly) long list of components over which the aggregation is computed. Sometimes different aggregations are computed over the same group of components (which share a set of commonalities such as location, functionality, etc.) and for this reason a network element grouping model can be beneficial to reducing the overall amount of information required to specify spatial aggregations.

According to our model, the network consists of a number of interconnected *network elements (NEs)*. Network elements are physical devices such as routers, printers, workstations, etc. Usually, the manufacturer of each of these elements provides an *element management agent* (EMA) - a rudimentary facility to manage (monitor and control) one or more instances of their equipment in a network.

On top of network elements we define *groups* as collections of network elements. Users can dynamically define groups based on any factor that makes sense. For example, groups can be formed according to geographical criteria (location) - a group of all NEs in a building, campus, state, etc., or functionality (a group of all ATM switches) or some combination such as all ATM switches in New York City area.

It's easy to generalize and have groups of groups. Thus, every network grouping hierarchy forms a tree with leaves being network elements. Groups are not necessarily disjoint. Every group is characterized by a level indicator that corresponds to the depth of the tree where the particular group belongs. Level 0 is reserved for the leaves of a hierarchy which (from a network management standpoint) represent the element management agents. A group at the first level corresponds to a set of element management agents. A group at the second level may contain groups of level one and perhaps one or more agents. In general, a group at level *n* is allowed to contain groups of any level lower than *n*, i.e., level *n*-1, *n*-2, ..., level 0 inclusive. Some times it's convenient to refer to a group of level 0 which, however, really implies an element management agent.

A group can be a point of aggregation of information about its subordinates. As the simplest example, assume that a variable `ErrorCount` is defined on some agents, representing the number of unrecognized packets arriving at the corresponding network element. A new variable `ErrorCount` can be defined on G to represent the total number of errored packets received within the group. The latter can be computed by summing the `ErrorCount` variables retrieved from every member of G.

Subsequently we will describe how such mapping functions can be defined within the group hierarchy that satisfy requirements on the “staleness” of the generated data.

In general, attributes defined at a group of level *n* are computed based on attributes defined over groups at the levels below. Control functions defined over spatial abstractions operate in a similar way: a control operation on a higher-level attribute propagates down the group hierarchy until it is eventually translated in to a set of control operations on attributes within the element management agents.

Figure 1 demonstrates an example of a network with 8 managed elements A-H. Each one of these elements contains an Element Management Agent. We create two groups of four elements each,

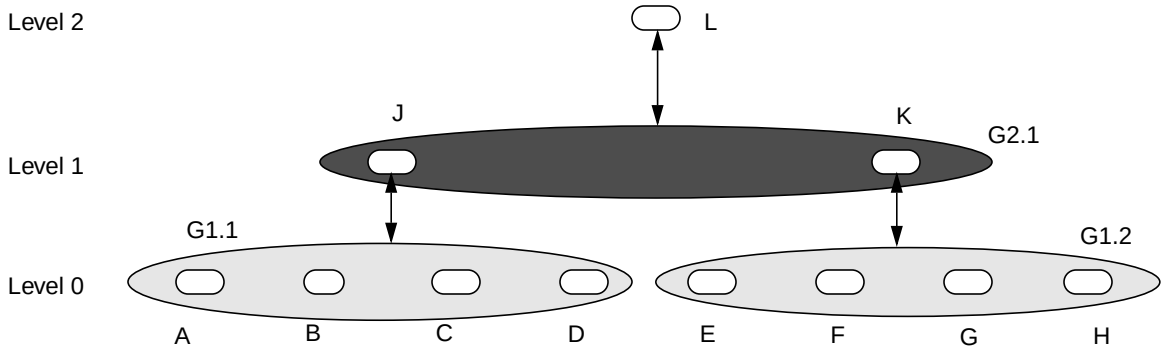

Figure 1: Example of a group hierarchy

G1.1 and G1.2. Information for these groups is stored in objects J and K. Furthermore, we create a second level group G2.1 which consists of groups G1.1 and G1.2. Information about this group is stored in object L. L’ s attributes are computed from the attributes of J and K, which in turn, are computed from attributes in A-D and E-H.

2.2 Information Model

We follow the object-oriented model to store aggregated management information. We believe that this model captures in a natural way the types of management abstractions that need to be created and the inter-relationships with their logical components.

Aggregations of management information are stored in *Aggregation Managed Objects* (AMOs) - in contrast to regular Managed Objects (MOs) which reside in the agents of level 0. AMOs are stored in the database of a special management agent (the MAVS - Management Aggregation and Visualization Server). AMOs however do not abide to a specific network management standard either in terms of structure or the protocol for accessing them. AMOs are distributed between many MAVS for convenience reasons (usually to reduce communication overhead between the AMOs that are heavily interdependent). The distribution of AMOs to MAVS can be independent of the grouping hierarchy.

Aggregated management information is contained within every AMO in a list of attributes. Every attribute is associated with a list of groups to determine the information components over which the attribute value is computed. This list of groups can then resolved into a list of AMOs or pointers to information within element management agents. The appropriate attribute value from each such

object is then retrieved and a *filter function* is applied. The filter function operates on the collected attribute values and stores the result as the current value of the attribute. For example, the operation SUM sums all the retrieved values and stores the result as the new attribute value. The operation NULL stores all the retrieved values in an array indexed by each retrieved attribute. More formally, the attribute value can be expressed using the following formula:

$$V = f\{(G_1, o_1, a_1), (G_2, o_2, a_2), \ldots, (G_n, o_n, a_n)\}, \quad \textbf{(EQ 1)}$$

where f is the filter function, G_i is a group, a_i is the component attribute's name and o_i is an object selection predicate. The latter is used to select the AMOs or MOs within the group from which the attribute value will be collected. Note that in this framework, an attribute need not be computed exclusively from components of the same type. Figure 2 demonstrates an example of the attribute value computation procedure.

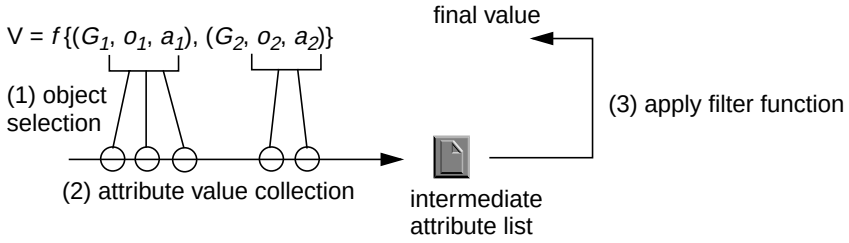

Figure 2: Flow chart for computing an aggregated attribute value

Filter functions can be specified by *reference*, in which case the required code segment is loaded dynamically from an external function library. The benefit of this approach is that library functions need not be integrated with the MAVS, which allows the definition of new functions or the improvement of existing ones without disrupting the operation of the management system.

Temporal aggregations of an attribute can be accomplished by using special filter functions. For example, a sliding window filter can store a collected attribute value as a time-series. It is also possible to define new attributes using filter functions that operate on the stored time-series such as delta functions, cross-correlation functions, etc.

For a settable attribute, there also exists a *mapping function* that describes how the value set by the manager is to be propagated to the underlying components. The simplest mapping function is the one that distributes the same value to all of its component attributes. It can be used for simple on-off operations or control operations that require setting the same value to a group of devices.

A *Refresh Policy* specifies how the attribute value is computed: A synchronous policy implies that the value is computed dynamically upon a *get* operation on the attribute's value. In the asynchronous case, the value is computed and stored according to an update condition. The latter can be a time interval, in which case the value is computed by periodically "pulling" information from the component objects. It is also possible to link the computation of an attribute's value with the occurrence of an event. For example, an event could be an indication that one of the component attributes has changed its value. In an eager policy, the attribute's value is recomputed each time any of its components change. The choice of the update condition must be made with great care: Infre-

quent updates introduce the danger that the computed information is out of date. On the other hand, an eager policy may trigger very frequent computations of an attribute's value, some of which may not even be necessary (if the value is accessed at slower time scales). The manager sets the update condition taking into consideration the sensitivity of management applications that use this information with regard to its accuracy and the complexity involved in computing its value.

The information kept in an AMO is reduced for the following reasons: Firstly, AMO attributes refer to groups of network elements rather than the NEs themselves. Secondly, values are computed by applying a filter function to the collected information. And thirdly, AMOs have the capability of evaluating their attribute values synchronously upon a query, thereby eliminating the need for storage.

2.3 Object Types

AMOs appear in two forms: Generic AMOs and Group AMOs. The former have already been described in Section 2.2. Group AMOs are a special case of AMOs that are permanently tied to a group definition in the grouping hierarchy (we also refer to them as "group objects"). Their attribute list contains only properties associated with the group. The restriction that we impose on these objects is that they contain only attributes that are also defined in their component groups (down to level 1). Because of this property, the value of an attribute can be computed by applying the filter function on the same attribute of the component group objects (i.e., no object selection predicate or subordinate attribute name needs to be specified). There can be only one group object associated with every group. Group objects are used to represent network properties that have meaning through several grouping levels while keeping the object specification simple. For example, a performance attribute of interest may be the amount of traffic carried through individual network nodes, network regions and finally the entire network. The traffic measurement at every group level can be computed by summing the measurements of the groups below.

On the other hand, Generic AMOs are more complex to specify but allow for describing more elaborate network abstractions. Unlike Group AMOs, their attributes are not necessarily tied to the same group(s) and for this reason they can be used to create abstractions that combine properties from different functional areas of the managed network (an example was given early in Section 2).

Summarizing, our model achieves spatial aggregations through grouping and filtering, temporal aggregations through the use of sliding window filter functions, and functional aggregations by combining attributes whose logical components are defined independently.

2.4 Object Services

We are using a distributed object services model to provide client access to AMOs. Platforms based on industry standards such as CORBA [OMG93] and Java-RMI [SUN97] are gaining momentum within the network management services industry and can be used to support our framework. AMOs provide network management services through a set of advertised interfaces. Clients obtain access to these services by contacting a service broker. The broker returns a reference to the AMO that offers the requested service. The client then invokes operations on the AMO directly and receives the results.

AMOs provide two tiers of services: Basic access services, which are mandatory for all AMOs and Extended Services that are implemented optionally. The latter can be used to provide a richer cus-

tomized interface to the object for performing more complex operations related to its intended management function. There are three categories of generic access services:

1. **Navigation** services are used to navigate through the MAVS database and retrieve AMOs based on a key or a selection predicate. The navigation API depends on the particular implementation of the database. For example, if AMOs are stored in a tree, the API would include functions like *getParent* (retrieves the parent of the current AMO), *getSubordinates* (retrieves the AMO' s children), *getPath* (retrieves the path from the root), etc.
2. **Attribute access** services are used to set and retrieve attribute values and control several aspects of each attribute's operation. These functions include *get* (retrieves an attribute value as an opaque object), *set*, *action* (dynamically downloads control logic that operates on one or more attributes or other AMOs), *setFilterFunction*, *setRefreshPolicy*, etc.
3. **Visualization** services are used to provide clients with the necessary information to setup graphical user interfaces (GUIs) that access the object's basic and extended services. The benefit of this approach is that clients do not need to be aware of an AMO's internal structure to provide a user-friendly interface. In a sense, the GUI is already programmed in the AMO and is transferred to the client when it first accesses the object. The *visualize* function for example scans the list of attributes and for each attribute generates code which can be used at the client to visualize the attribute's value and perhaps provide a control interface (for a settable attribute).

The AMO designer is responsible for providing an implementation for all basic and extended services. Access to the latter can sometimes be provided indirectly through the basic services. For example, the *visualize* function can be overloaded to return GUI code for some application specific services.

2.5 Accessing Element Management Agents

We have assumed so far that agents of level 0 follow the AMO structure and access conventions. However, this is hardly the case in a real environment where level 0 agents are provided by hardware manufacturers and communicate through one of the standardized network management protocols. To overcome this problem we allow the tuples of (EQ 1) to take different forms (a property known as *allomorphism*), depending on the underlying management protocol.

In the SNMP domain for example, every tuple contains a group definition (which describes a group of SNMP agents and an attribute object identifier. The OID can also be a prefix, in which case all attributes matching the prefix are selected.

The OSI framework requires additional parameters to specify a target attribute, namely object class, object instance, scope, filter and attribute name. Again, The group G refers to a set of CMIP compatible agents. The *objectClass* and *objectInstance* parameters specify the class and distinguishing name of a reference object. The *scope* identifies a subtree of managed objects in the OSI containment tree rooted at the reference object that contain an attribute of *attrName*. When these objects are selected, a *filter* will be applied to narrow down the search to the objects that satisfy the filter condition. Once these objects have been identified, the requested attribute values can be returned.

2.6 Dynamic Query Processing

In some cases, the model that we have just described is not sufficient to provide certain forms of aggregation. For example, consider the case that we want to determine the currently reserved bandwidth on all network links from a group of nodes G1 to another G2. We could do this by defining a new group G that contains G1 and G2 and an object selection predicate that will identify the links with a source node in G1 and a destination node in G2. This type of queries however is quite infrequent and it might not be efficient to create a persistent AMO that captures these abstractions.

For this reason we use a query processing engine to handle uncommon queries. The engine is capable of retrieving the required information by directly identifying the component objects and applying the appropriate filter function. The identification phase might require the generation of some transient groups and object selection predicates. Query processing gives the additional capability of retrieving information based on unusual user-defined predicates without the requirement of static storage that most network management architectures require. In addition, all processing and filtering is done at the server, which frees up computation and communication resources at the client.

3. The MARVEL Management Environment

In order to put the model for information aggregation to work, a supporting framework for developing network management services was required. The resulting toolkit was named MARVEL (Management AggRegation and Visualization Environment). MARVEL is built using the Java language. We selected Java because it is an industry standard and has the following important features:

- Distributed object services are well integrated with the language (the Java Remote Method Invocation - RMI package).
- Ability to load code from the server to the client and vice versa.
- Graphical User Interfaces are a core part of the language and can be dynamically downloaded into clients.
- Compile once - run everywhere capability.
- Is supported from practically every computer on the Internet with a Web browser.

The MARVEL architecture consists of lightweight clients and a hierarchy of Management Aggregation and Visualization Servers (MAVS).

3.1 The MARVEL client

The minimum requirement for a MARVEL client is to have a Java Runtime Environment (JRE). All the necessary code to access management services provided by AMOs can be downloaded in real-time from the MAVS. In addition, if the client has the capability to display HTML (the Hypertext Markup Language) it can use the visualization features provided by the MAVS that aggregate attribute specific user interfaces (applets) on HTML pages. This is why Web browsers are the ideal MARVEL clients. In addition, the MARVEL architecture benefits from the fact that Web browsers are very widespread. By making the minimum number of assumptions for the client, MARVEL provides network management services of arbitrary complexity to practically every user on the Internet.

3.2 The Management Aggregation and Visualization Server

The MAVS is a management agent designed to handle aggregations of management information. Every AMO must be instantiated within a MAVS. For this reason a MAVS has a number of subsystems designed exclusively to support AMO features. Its main components are:

1. The aggregation processing engine. This engine is responsible for implementing an attribute's update policy by computing its value from a set of components; it initially resolves group references into target objects, invokes the appropriate protocols to collect the necessary information, and finally applies the filter function to compute the final value(s). For control operations the last two tasks are reversed.
2. The persistent storage engine. By default, the state of every AMO is made persistent to survive failures of the MAVS. Persistence is necessary when the stored aggregated information cannot be reconstructed from the current contents of element management agents (a time series attribute is a good example of an object that must be persistent).
3. The AMO service registry. The registry logs every AMO on the MAVS together with its exported service interfaces (basic and extended). Clients first contact the registry to obtain a handle to their AMO of interest.
4. A transaction processing system. This system allows coordinated access to all AMO services in order to avoid consistency problems when multiple clients are operating on the same objects.
5. An HTTP server, that provides access to HTML documents generated by some AMO's visualization functions. It also serves to download Java class byte code to clients and provides an initial navigation page to the contents of the MAVS object database.
6. The event processing subsystem, that registers event subscriptions and performs event collections and correlations.
7. The query processing engine, that supports operations on AMOs with user-supplied predicates and filters.
8. The protocol translation module, which allows AMOs to access management services using a different protocol, such as SNMP, CMIP or CORBA.

A management system based on MARVEL may contain many MAVS. Although it is not required to follow a particular structure, it is usually convenient to structure all MAVS similar to the grouping hierarchy for easier administration, and to minimize communication overheads. So, in a MARVEL system one would expect to have separate MAVS containing the aggregation objects for a particular subnetwork, a parent MAVS containing aggregations about a network region (consisting of several subnetworks), and finally a top level MAVS containing the aggregations for the entire network. In reality however, the management system designer can expand or collapse these levels as is necessary. A single MAVS may in fact support any number of levels of aggregation.

AMOs within a MAVS represent aggregations of levels 1 and above. The MAVS itself acts in a dual role: It acts as a server (agent) for its clients and as a client (manager) when accessing the services of other MAVS or EMAs.

In the current implementation, every MAVS contains a number of AMOs in a tree containment structure, very much alike the OSI management model [ISO91a] (the tree structure was selected

purely for implementation convenience). AMOs are placed in the containment tree in any fashion that the manager wishes. It is appropriate however to enforce a natural containment relationship. For example, an AMO representing a summarization of performance parameters from a set of users would be placed as the parent of the AMOs that contain performance parameters of individual users. Note that the structure of the containment tree expresses an arbitrary containment relationship between the AMOs and is not necessarily related to the grouping hierarchy.

Since group definitions and filter functions can be shared between many MAVS, they can be stored in an external directory server. In a way, this directory acts as a central network configuration database. By separating the fairly static configuration information from the MAVS, we avoid synchronization issues when group definitions change. The penalty however is that a directory access is necessary every time a group is resolved into its components.

AMOs are programmed directly in Java. A core class provides the basic management services described in Section 2.4. Every AMO is then subclassed from the parent class and inherits automatically the basic access interface. In addition, the designer can implement the optional extended services by adding new service interfaces. We believe that writing AMOs directly in a programming language has the following advantages:

- The development environment is much simpler since no AMO schema compilers are required (in contrast with the OSI model that uses a GDMO compiler for generating managed objects).
- Some of the default object's functions can be overridden by the programmer to implement, for example, specific data collection and aggregation policies.
- Objects can be extended to provide customized high level services in addition to the fundamental get/set operations on their attributes.

A set of basic attribute classes are also included in the MARVEL core class package, to implement simple data types, tables and time series. More complex data types and filter functions can be easily implemented from these base classes. The constructor for every new AMO class is responsible for instantiating every attribute. An important property of the MARVEL system is that attributes are also Java objects and are therefore permitted to export their own service interfaces. Therefore, the designer has the option of implementing attribute-specific services as a complement to the AMO's basic and extended services.

Also included with the MARVEL system are a set of classes that provide interfaces to standard NM protocols such as SNMP, CMIP (only SNMP is currently functional) and other distributed object service environments such as CORBA.

The event processing subsystem is still under development. It will consist of a filtering mechanism that operates both in the space and time domains. In the space domain, it collects events from many sources and generates a notification if a logical predicate is satisfied. A time domain filter also takes into account the temporal order with which elementary events occur. An event subscription interface allows AMOs to register events of interest and be notified upon their occurrence. Events can also be used to trigger attribute value updates, as explained in Section 2.2.

Special attention has been given to the visualization interface. Every AMO can generate an HTML page which can be viewed from any Web browser by calling the object's *visualize* method. This works as follows: When the function is invoked, the AMO scans its list of attributes. The *display*

method is called for each attribute. *display* then inserts some HTML code that produces an accurate visualization of the attribute. For simple data types such as numbers and text, a regular string is sufficient. For more complex data types such as time series, the display function inserts an applet capable of displaying the time series. When the user needs to augment the attribute class library with a new data type, a display function must also be provided for the new attribute. In this way visualization becomes an integral part of data type definition, and so, client applications need not be aware of the data types supported within each MAVS since the appropriate applet viewer/controller will be automatically loaded. Finally, the default *visualize* function can be overridden to include code for visual interfaces (applets) that access the object's extended services. The MAVS structure is shown in Figure 3.

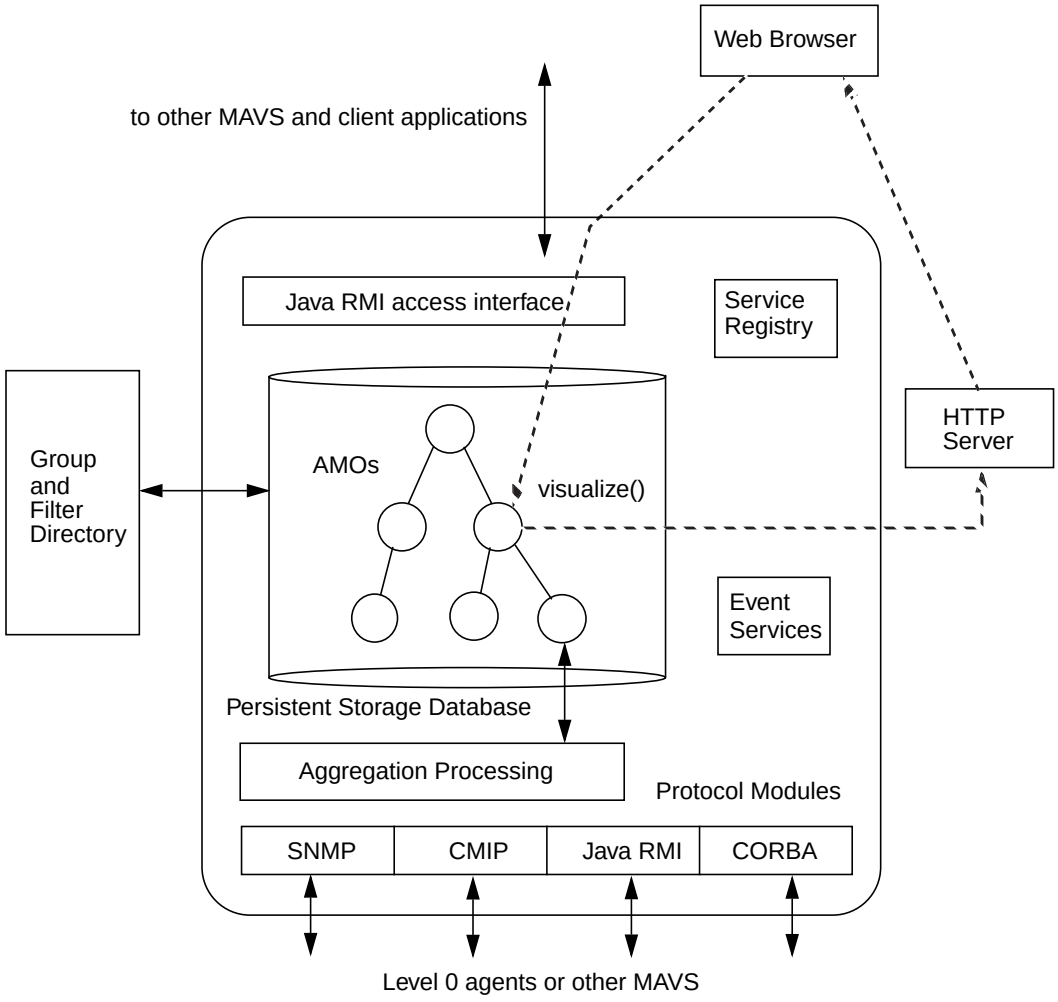

Figure 3: MAVS Structure

4. MARVEL Applications

4.1 The SAIL network

We are currently using the MARVEL system to manage the SAIL experimental home access network. SAIL (Speedy Asymmetric Internet Link) is an AT&T home access trial that brings a 10 Mbps data channel to users homes through a downstream CATV channel, and uses a 28.8 modem for the return path. Sail consists of a head-end router which multiplexes all user traffic on the CATV channel a terminal server, and cable modems (one per user) that terminate the upstream and downstream channels and route the collected packets onto a local Ethernet on which the user has connected a number of PCs or workstations. The SAIL network currently supports about 150 users and will grow to about 400. The home access architecture however has been engineered to handle many cable distribution head-ends, each one of them providing service for several hundred users. Home access networks have exactly the large scale properties that can benefit from the MARVEL environment to provide summarizations of performance data and bulk control actions. The architecture is shown in Figure 4.

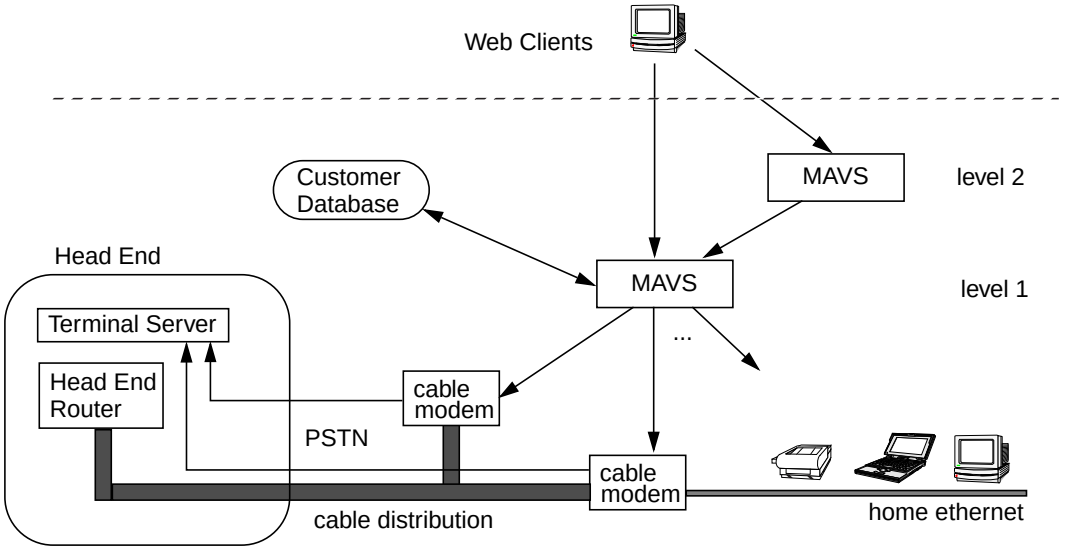

Figure 4: Managing the SAIL Network

Summarizing performance such as bandwidth usage in this environment is very attractive because not only does it give statistics for larger sets of the user population, but also helps in planning better the capacity and number of served users at every branch of the cable distribution system. In addition measuring the error rates for different groups of users can help to identify areas with transmission problems and sometimes pinpoint the exact location of the problem.

In terms of control operations, cable modems occasionally require updates in their operating software. The MARVEL environment is also valuable in this case since it is able to perform this distribution with a single control action on the group that represents the entire network. We have found that the MARVEL system has many advantages over commercial management clients such as HP Openview because performance aggregations can be stored and accessed at any time. By contrast,

in the Openview framework aggregations can be created only by off-line processing the centrally collected data from the cable modems and can be performed only by clients that are compatible with Openview's database access interface.

In addition to the above performance summarization features we also support functional aggregations on a user-by-user basis through AMOs that represent "user profiles". These objects combine account information from the user registration database with per-user temporally aggregated performance data. Every such AMO generates a Web page that users can access to view their account status together with a time series of their bandwidth usage and observed quality of service (transmission error rates in the downstream channel). Time series are visualized using a special Java applet that displays a chart and allows scrolling and zooming for more careful examination of the data (see Figure 5).

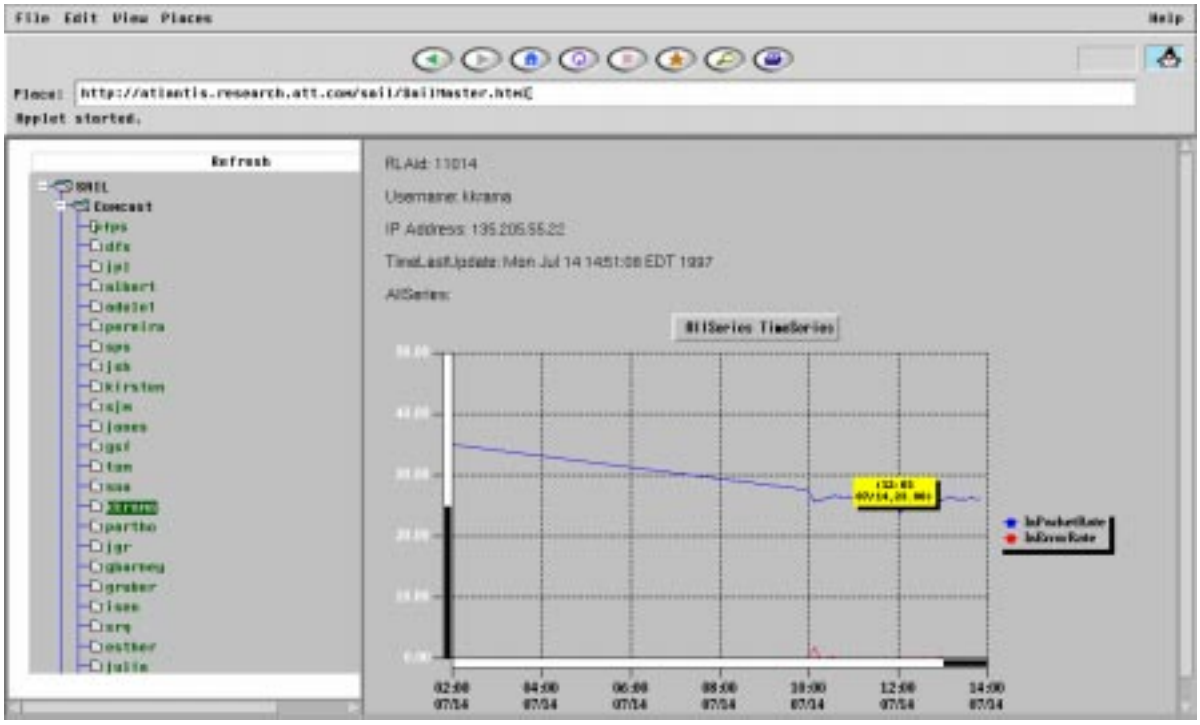

Figure 5: A user profile displayed on a MARVEL client

In the current architecture we use one MAVS for every CATV distribution tree (about 50 modems). The MAVS periodically obtains information from the cable modems through a low level monitoring and control protocol and updates the appropriate AMOs. A group object contains aggregated performance information for the group and also a table attribute which is used to sort individual user information based on usage or error rates. Finally, a time-line chart indicates the times that every user has been active. These two attributes can be used to identify quickly users with high usage or error rates or online activity as well as compare individual users. A second level MAVS contains performance and control aggregations for the entire network and allows software distribution to the entire set of cable modems.

5. Related Work

The need for aggregations in order to manage large networks has been identified in a number of previous works in this field. One of the most organized efforts to provide management capabilities for large networks is the Telecommunications Network Management (TMN) working group within the ITU. The TMN standards have been developed to ensure interoperability between heterogeneous networking equipment and to provide a conceptual layering of administration functions in 4 levels: the network element level, the network level, the service level and the business level [ITU91a,b,c].

At the same time, network management researchers have started to acknowledge the necessity of introducing new abstraction mechanisms in network management to cope with issues of scale [TAK95]. There have been several pieces of work in this area addressing one or more aspects of the problem:

[STA96] presents a language that can be used to provide high-level management abstractions. The language can be used to define the mapping between abstract models and the underlying management information provided by agents and management platforms. The language is based on the OSI GDMO, and for this reason is specific to GDMO-defined object classes for basic management information [ISO91b].

[KAL96] presents a special agent proxy that acts as a front end for one or more nodes that need to be managed as a group. It uses a spreadsheet paradigm to process the underlying management information and provide a table of computed attributes. The spreadsheet language supports all arithmetic, logical and relational operators. Except from computed attributes, the spreadsheet is also capable of generating event reports by evaluating predicates containing relational expressions.

[GOL96] proposes extensions to the SNMP-SMI that allow the creation of MIB views within an agent. Views are created by defining operations on SNMP tables using a language similar to SQL. Supported operations include joins, filtering, table snapshots, etc. However, the main drawback of this approach is that views are restricted to the information contained in one MIB only.

The idea of loading code into a management agent to perform locally some management tasks was proposed for the first time in [YEM91]. The initial approach was to download management scripts that were compiled and executed at the agent. The arrival of Java has made this task significantly simpler, and also allows the reverse operation (code downloaded from the agent to the client).

In the visualization area, [CRU93] proposed an immersive environment that enables the manager to navigate through the network state and perform high level control operations. In this environment, the network is managed through the exchange and processing of multimedia information. Its main characteristics are the following:

- The manager can actively navigate through the network, and directly observe the state of the system at many levels of abstraction. Information is structured so that the manager is automatically guided through the hierarchy of network states. The manager can access any piece of information at any time.
- Control actions are effected directly through commands with a semantic level that match very closely their intended purpose.

- The user interface to the network naturally represents the system being managed and allows management operations to be effected rapidly and easily.

Parallel to our work is the development of the Java Management API (JMAPI) toolkit from Sunsoft [SUN96]. JMAPI builds on the idea of downloading java byte code to web browsers that allows them to access server-defined management services. In fact, MARVEL uses some of JMAPI's GUI components. JMAPI however is designed to provide more of a front end to already available SNMP services as opposed to providing a scalable architecture for aggregating management information. A direct comparison of the two architectures is not possible at the time being, pending the release of more complete documentation on JMAPI's internals and applications from Sunsoft.

6. Summary and Future Work

We presented a framework that allows the rapid development of scalable network management services with a particular emphasis on data aggregation and visualization. A new information model allows aggregation in spatial, temporal and functional forms. An event processing subsystem currently under development allows spatial and temporal correlation of events. Simple clients such as web browsers can navigate through the different levels of aggregation and visualize objects of arbitrary complexity by dynamically loading the appropriate viewers/controllers.

The SAIL network is currently serving as the experimental testbed for testing the aggregation framework and providing advanced NM services for user administration. We are also working to enhance the attribute library with additional data types and the filter function library.

Using the leverage of the World Wide Web in terms of client applications (Web browsers) and network connectivity, we achieved our goal of providing very simple access to arbitrarily complex management services. Eventually, the quality of network management services will become an important differentiator for many telecommunication service offerings. In addition, we demonstrated that standards based management is not the only solution for providing comprehensive management services. Since users interact with a NM system through a graphical user interface, it is logical to assume that they are more interested in the functionality provided to them and less in the nature of the underlying management protocol. Of course interoperability issues are critical and for this reason it is worthwhile to consider how aggregations and higher level management services could be standardized.

The authors would like to thank H.V. Jagadish and Chuck Kalmanek for participating in the discussions of the MARVEL architecture and providing their valuable feedback.

References

[CRU93] L. Crutcher and A.A. Lazar, "Management and Control for Giant Gigabit Networks", IEEE Network Magazine, vol. 7, no. 6, pp. 62-71, November 1993.

[GOL96] German Goldszmidt and Yechiam Yemini, "Computing MIB Views via Delegated Agents", unpublished draft, 1996.

[ISO91a] Information Processing Systems - Open Systems Interconnection, “Structure of Management Information - Part 1: Management Information Model”, July 1991. International Standard 10165-1.

[ISO91b] Information Processing Systems - Open Systems Interconnection, “Structure of Management Information - Part 4: Guidelines for Definition of Managed Objects,” July 1991. International Standard 10165-4.

[ITU91a] ITU-T Recommendation M.3010, “Principles for a Telecommunication Management Network”, Geneva, November 1991.

[ITU91b] ITU-T Recommendation M.3020, “TMN Interface Specification Methodology”, Geneva, November 1991.

[ITU91c] ITU-T Recommendation M.3200, “TMN Management Services: Overview”, Geneva, November 1991.

[KAL96] Pramod Kalyanasundaram, Adarshpal S. Sethi and Christopher M. Sherwin, “Design of a Spreadsheet Paradigm for Network Management”, in *Proceedings of the 1996 DSOM: Distributed Systems Operations and Management*, L’ Aquila, Italy, October 28-30, 1996.

[OMG93] Object Management Group, “The Common Object Request Broker Architecture and Specification”, Rev. 1.2, Dec. 1993.

[STA96] Michael Stadler, “Mapping Management Information to Service Interfaces Supporting High-Level Network Management Applications”, *Proceedings of the 1996 DSOM: Distributed Systems Operations and Management*, L’ Aquila, Italy, October 28-30, 1996.

[SUN97] Sun Microsystems Corporation, “Java RMI Specification”, ftp://ftp.javasoft.com/docs/jdk1.1/rmi-spec.pdf.

[SUN96] Sun Microsystems Corporation, “Java Management API Architecture”, http://java.sun.com/products/JavaManagement/.

[TAK95] Makoto Takano and Katsumi Fujita, “Multilevel Network Management by means of System Identification”, in “*Proceedings of the 1995 INFOCOM”*, pp. 538-545, Boston MA, April 1995.

[YEM91] Y. Yemini, G. Goldszmidt and S. Yemini, “Network Management by Delegation”, in Second International Symposium on Integrated Network Management, pp. 95-107, Washington DC, April 1991.