AN INFORMATION MODEL FOR GENERATING COMPUTED VIEWS OF MANAGEMENT INFORMATION

Nikolaos Anerousis

AT&T Labs Research
180 Park Avenue, Bldg. 103
Florham Park, NJ 07932-0971
e-mail: nikos@research.att.com
http://www.research.att.com/~nikos/

Abstract

We present an information that model allows the generation of computed views of management information. Computed views are used in combination with a distributed computing environment to provide higher-level management services, a necessary component in building scalable management systems. Computed views consist of monitoring, control and event views of information collected from network elements and aggregated using spatial and temporal filters. The model handles management information available in any standard form such as SNMP or CMIS and can be easily adapted to any proprietary management protocol. The model is being implemented in the Marvel distributed management environment and tested on a broadband home access network to provide customer views of network services and performance summarization functions.

1. Introduction

In the recent years, one can observe a trend in the network management market that shifts the emphasis from the underlying technology that the network management system employs to the functionality that the system provides. Finally people have come to agree that the choice of a network management protocol (e.g. SNMP, CMIP or other) is only a minor detail in the overall management framework. Rather, emphasis is now given on the system's ease of integration with other components, expandability, availability, ease of use (web-based management systems are increasing steadily their presence), and reduced maintenance costs (including the time to train people, install and configure software, etc.). A common pitfall of most management applications today is that they still operate at a very low level, many times exposing the details of communication with network elements using a low level management protocol.

The main challenges in providing powerful network management services for today's integrated networks lie in two main areas: how to provide support for heterogeneity (components of different types from different manufacturers) and scalability (large numbers 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 protocol stacks and networks (MIB-2 and RMON), ATM switches, etc.

There are however no widely established methods however for dealing with large numbers of network elements. Managing large networks requires powerful abstractions that capture the essentials of the state of the network rather than the details. 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. As networks grow larger and integrate an ever increasing number of components and services, the existence of a scalable network management architecture becomes critical. We believe that the key behind offering truly scalable management services will be a new management information model that expresses naturally the most common forms of aggregation of network state. This model must further be application-independent, to ensure that it can be applied to a variety of management problems.

The new information model will allow managers to “customize” the way that the network appears to them for monitoring and control purposes, and will be key element behind management systems that scale as networks grow in size. The model that we propose provides high level constructs to hide elements behind groups, define managed objects that represent computed views of management information and provide higher level monitoring and control primitives that allow the manager to interact with the network management system at a *service* level as opposed to the element level that has been the common practice so far. It has been designed as part of the *Marvel* framework, an extensible distributed computing environment for creating web-based management services [ANE98d].

This paper is organized as follows: Section 2 presents the information model for generating views of management information. Section 3 presents a first application of the information model on a broadband home access network. Section 4 contains a review of related work in the field, and Section 5 our conclusions.

2. Information Model

Management information in a large network today is usually distributed between the MIBs of network elements and, as a consequence, represents small aspects of the configuration or operation of those elements rather than of the network as a whole. Management applications today require access to a much higher level of management information and services. Our framework proposes an object-oriented information model where the value of an object’s attribute can be defined as an arbitrary computation over other attribute values. The latter can be information residing inside element management agents or other computed attributes. The emphasis of our model is in providing a technology-independent specification framework in which these computations can be described. Using this model, the network manager can define new managed objects that represent *computed views* of management information. Computed views can represent a summary of lower level configuration and performance information, or a more detailed view of a particular management parameter.

Objects representing computed views of management information can be regarded as implementing a “middleware management services” layer (Figure 1). This layer extracts information from managed elements using a standards-based management protocol, processes this information according to the computed view

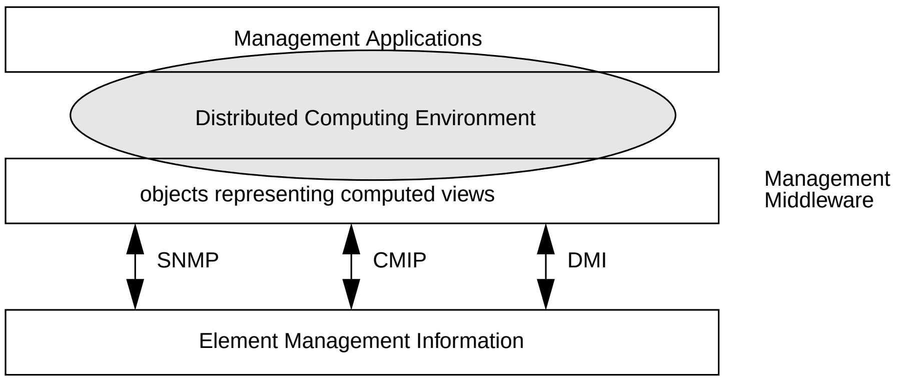
Management Applications
Distributed Computing Environment
objects representing computed views
Management Middleware
SNMP
CMIP
DMI
Element Management Information

Figure 1: 3-level architecture for generating computed views of management information

specification and makes it available to management applications using a distributed computing environment. Objects within the management middleware layer can follow the SNMP or OSI structure for management information (in which case they are accessed using the corresponding management protocol), or a proprietary format that exports management services to a legacy distributed computing environment such as CORBA or Java. Our model only specifies the way that an attribute value is computed from a set of components and for this reason it can be used as an extension to standards-based models for structuring management information such as SNMP SMI and GDMO [ISO91]. However, we believe that it fits better in a distributed computing environment such as CORBA [OMG93], since the notion of computed views for network management is closely related to the notion of higher level management services that can be more efficiently implemented in this framework.

2.1 Computed Views of Management Information

Computed views are constructed through an information aggregation process applied to management information collected from network elements. Every computed view in our framework is stored in an object and has one or more of the following components:

1. A monitoring, view which contains information that has been collected from the network and processed to represent a higher-level view of network state.
2. A control view, which represents a control interface to higher-level network management services.
3. An event view, which represents notifications that are generated by the object following the occurrence of a series of other (elementary) events.

The object's attributes represent the network state corresponding to the monitoring and control views provided by the object. When the view is defined, the network manager provides an aggregation rule with which every attribute value is computed. The rule can be specified declaratively, in which case a description of the aggregation is provided in a structured language, or explicitly, in which case the manager provides a piece of code that will be executed to compute the attribute's value. Aggregations can be *spatial* or *temporal*.

2.1.1 Spatial aggregations

Spatial aggregations try to reduce management information that is distributed among a number of network elements by presenting information about *groups* of elements. By defining the appropriate groups, the manager can obtain a high-level view of network configuration, performance, etc. For example, the ingress traffic to a network region can be computed by processing traffic information collected from switches at the border of the region.

In order to describe spatial aggregations efficiently, network elements (NEs) must be organized into groups. 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. Groups can be defined easily when there exists a network configuration database. In this case, group membership can be determined by identifying the network elements whose properties match a query predicate. The directory enabled networks (DEN) proposal is a recent effort to standardize the schemata required for such a configuration database. In fact, the DEN specification also identifies the need to model network services and groups to support more complex management functions.

It is easy to generalize and define groups with other groups as members. 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 element management agents (EMA). 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 EMAs. 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. Sometimes it is convenient to refer to a group of level 0 which, however, really implies an element management agent.

Figure 2 presents an example where network elements are first organized into groups G1, G2 and G3 that compose the first level of aggregation. Group G4 is defined as the union of G1 and G2, and similarly G5 as the union of G2 and G3. Once the group hierarchy has been defined, the manager can define higher-level

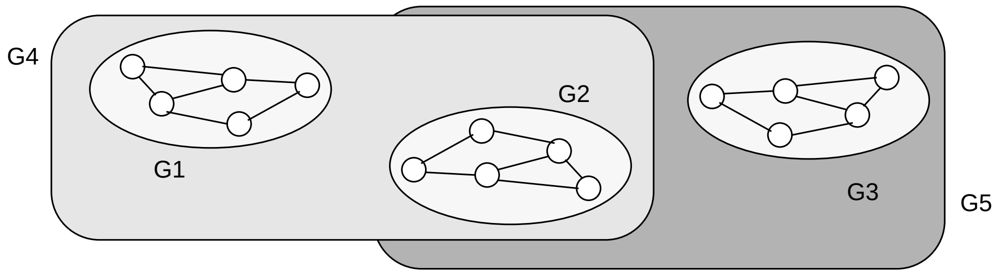

Figure 2: Example for grouping network elements

management views and services by referring to groups rather than network elements. As the simplest example, assume that the attribute `ErrorCount` is defined on some EMAs, representing the number of unrecognized packets arriving at the corresponding network element. A new attribute `ErrorCount` can be defined in a managed object representing G1 to count 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 G1.

In general in order to compute the value of an attribute that refers to a group, the group must be resolved to a set of management agents that contain the required information. The resolution is performed recursively in a number of steps equal to the order of the group. Once the information components have been identified, their values are retrieved and a new value for the attribute is computed. This procedure is described in detail in Section 2.2. Similarly, control operations on an attribute representing a group of level *n* are mapped to a series of (elementary) control operations on the components of the group.

2.1.2 Temporal aggregations

Temporal aggregations expose the time-varying properties of management information. In order to compute a temporal aggregation, information is collected periodically and provided as input to an aggregation function. This procedure can be used, for example, to form a time series of various granularities (minutes, hours, etc.) or, to provide an autocorrelation or cross-correlation measurement.

2.2 Declarative specification of attributes

As mentioned previously, attributes may be specified declaratively. In this case, the attribute is associated with a list of groups that determine the information components over which the attribute value is computed. In order to compute the value of the attribute, the list of groups is further expanded into a list of other objects representing computed views or pointers to information within element management agents. When the appropriate attribute value from each one of these objects is retrieved, a *filter function* is applied to calculate the final value. 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 complex filter functions may, for example, compute statistical properties such as the mean and standard deviation of a distributed data set, extract topological information to generate a topology map, etc.

More formally, every attribute can be expressed using the following formula:

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

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 managed objects (MOs) within the group from which the attribute value will be collected. Depending on the underlying management protocol, the object and attribute selection predicates may be specified in a different syntax to comply with the particular structure of management information (e.g. SNMP or CMIP) within an element management agent. Note that an attribute need not be computed exclusively from components of the same type.

Figure 3 demonstrates an example of the attribute value computation procedure. In this example, the at-

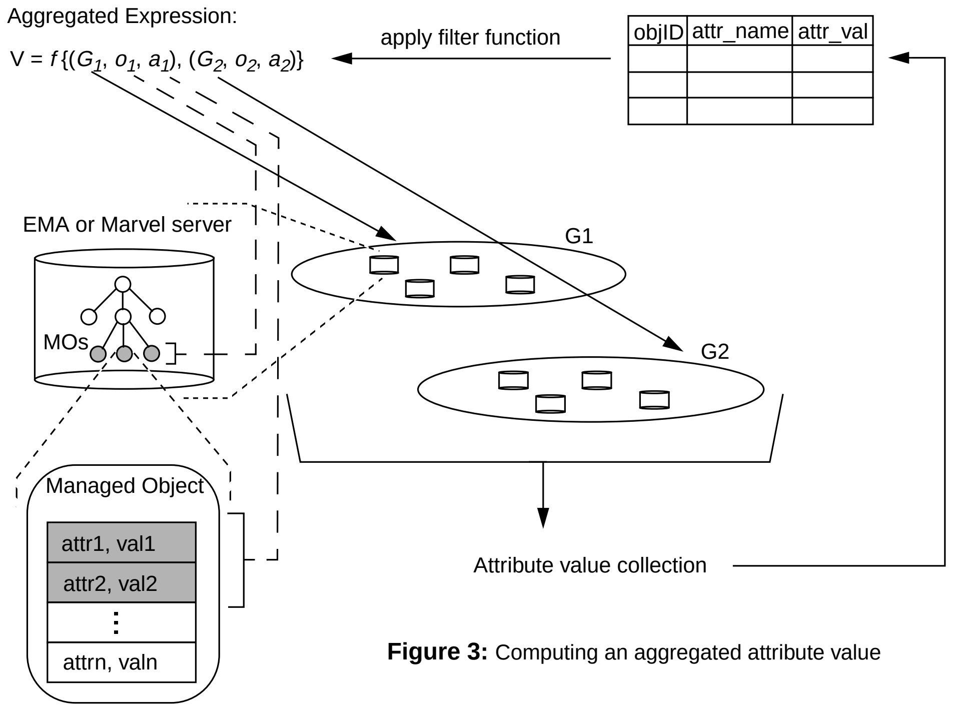

Figure 3: Computing an aggregated attribute value

tribute value *V* is computed from information components in groups G_1 and G_2. First, G_1 and G_2 are resolved into a list of management agents. For each agent in group G_1, the object selection predicate o_1 identifies the managed objects that contain the required information. From each such object, we obtain only the values of attributes that correspond to the attribute selection predicate a_1. The group G_2 is processed similarly. The result of the collection process from all the agents of G_1 and G_2 is stored in a temporary table that contains the origin of every attribute, its type and its value. The table is then used as input to the filter function which calculates the new value *V*.

Filter functions can be specified by *reference*, in which case the required code segment is loaded dynamically from an external function directory. The benefit of this approach is that filter functions need not be integrated with the server, 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 control operations that require setting the same value to a group of devices, such as firmware upgrades, enabling or disabling features, etc. More complex mapping functions may use additional information from the environment to determine the exact value that every component attribute must receive. For example, in an ATM virtual private network (VPN), the manager may decide to increase its total allocated capacity by 10 Mbps. The mapping function in this case will determine the details of how the new capacity will be distributed between individual virtual paths.

A *refresh policy* specifies how an attribute value is computed: A synchronous policy implies that the value is computed dynamically upon a *get* operation on the attribute. In the asynchronous case, an internal or external condition triggers the computation and storage of the value. The trigger condition can be a time interval, in which case the value is computed periodically by "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. An eager policy would recompute the attribute's value each time any of its components change. The choice of the update condition must be made with great care: Infrequent 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.

2.3 Accessing Element Management Agents through SNMP or CMIP

In many cases, the computed view is generated after collecting information from element management agents (EMAs) using a standards-based network management protocol such as SNMP or CMIP. When attributes are specified declaratively, additional information is usually needed to identify which MIB objects inside the EMAs should be involved in the computation. For this reason, we provide mappings for both the SNMP and CMIP domains. The following rules apply:

First, group definitions must always be resolved into a set of application-layer addresses and other parameters of EMAs that represent the members of the group. In the SNMP domain for example this implies that every EMA is represented by the combination of an IP address, port number and any security information such as community strings that may be required. In the OSI domain, every CMIP agent must be described by an application layer address to which an association can be established, and additional security information.

For every SNMP agent within the group, the object selection predicate is a set of object identifiers (OIDs). If the OID corresponds to a column of a table, the entire column is retrieved using a series of get-next operations.

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 agents. The object selection predicate contains the *objectClass* and *objectInstance* parameters to specify the class and distinguishing name of a base object. In addition, a scoping and filtering predicate further identify a subtree of managed objects in the OSI containment tree rooted at the base object. The attribute selection predicate identifies the attributes within the selected objects whose value will be retrieved.

2.4 Event Views

Traditionally, event filtering and generation of higher level (composite) events has been done by a separate centralized event processing engine (usually an expert system). There are many advantages in decentralizing the event filtering process:

- A higher degree of integration is achieved between the computed view and the events associated with the view. Most management systems today store events in a single event database. Now, a computed view can store all events associated with the view inside the corresponding object and provide a customized event browsing/monitoring tool as part of its monitoring interface.

- A distributed architecture for event filtering and generation can be implemented at significantly less cost compared to configuring and installing centralized event filtering systems.

There has been a significant amount of work in the field of composite event generation [NYG95, HAY96, MAN97]. Our event architecture uses the services of *READY,* an event processing and notification system [GRU97] to generate composite events. This section describes the overall model for producing and consuming events in our environment and the structure of event objects.

We distinguish between 4 entities: Event producers, consumers, clients and servers. Every view object can assume any or all of the above roles. Event producers are the sources where events are generated. These can be element management agents, or other view objects that generate composite events. An event consumer is an entity that has subscribed to receive events from a producer. An event server consumes events, processes them and stores the result in a local database. Finally, an event client contacts an event server and performs operations on the stored event information. For example, event servers can be simple event log objects and clients browsing tools that examine these logs.

Events are defined using an object-oriented model. The *EventRoot* class contains the attributes that are common to all event classes. Further specialization from this class allows the definition of new types with additional attributes or functionality. The root class contains the following information: event producer, type (class), timestamp, severity and data (in a list of attributes). The event producer field indicates a network element known to the network configuration directory (the same directory used to generate the groups of Section 2.1.1).

When an object acts as an event producer, it must register with a READY process a specification of the composite event(s) that it is capable of generating. The latter is expressed as a function of a number of component events in the READY language and installed in the READY server when the producer object is initialized. Then, the READY server collects all the required event components and fires an instance of the composite event when all necessary conditions have been satisfied. The composite event is then sent to the corresponding view object for further distribution. Not all events produced by an object need to be generated at an external READY server. For example, events that correspond to state changes within the object are generated locally and then distributed to consumers.

In order to specify an event generation condition in the READY language, the manager must supply the following information:

1. The type (class) and severity of the event.
2. A time window during which the component events must have occurred,
3. The list of component events and a matching filter. The former are specified by supplying the type and source (producer object) of each event. The filter is a value assertion on the other attributes of the event.
4. An ordering condition for the events. The ordering condition may require that events have occurred in a certain order in order to be satisfied.
5. A trigger condition, which identifies when the computation to generate the event will take place. The trigger can be periodic (upon the expiration of a timer) or event-based, in which case it is possible to trigger the computation upon the reception of a component event.
6. Rules that will be used to populate the data fields of the new event.

The filter of item 3) above allows to narrow down the events that will be used during the event generation process. It is possible to filter by source (groups can also be used, in which case the event must originate in one component of the group), severity, or an assertion on the event's other attributes. By using an object-oriented model, it is also possible to filter events based on their class, in which case, all descendents of the class in the event inheritance tree will satisfy the filter.

The ordering condition can impose a partial or total order in which events have occurred. It is possible to omit the ordering specification completely and simply indicate that the event is generated only when all or one (or none) of the component events have occurred in any order. The ordering condition can also handle temporal repetitions of the same event (it is possible to require that the component event has happened a certain number of times within the time window), or spatial repetitions (the manager may indicate for example that a sufficient condition to generate the new event is its occurrence in at least five elements of a particular group), or a combination of the two.

The event model that we propose has a number of advantages. First, it is object-oriented, allowing the abstract specification of event type matching filters. Second, it allows the specification of both spatial and temporal ordering conditions, a capability which has proven to be very useful in large networks where events associated with element groups convey valuable information to network managers. Third, the distribution of the event filtering task between objects allows for a more scalable filtering architecture.

2.5 Using a distributed computing environment

To summarize, computed views of management information are stored in objects. Every object contains a list of attributes that represent the monitoring and control views, and can generate a number of notifications that represent the event view. So far, we have not made any particular assumptions about the management information model or the management protocol used to implement the information aggregation architecture. In theory, it is possible to transparently implement this functionality within existing management modeling standards such as SNMP, CMIS GDMO [ISO91] or CIM [DMT98]. In CMIS for example, the attributes of the object may be accompanied by a declarative specification (Section 2.2) that will be used to implement the object's information aggregation functionality. Since this is related only to the object's implementation, it can be completely transparent to the management protocol. Similarly, event views can be implemented through the CMIS event notification service. The implementation of the object is responsible for registering with all component events and performing the filtering procedure of Section 2.4. The higher-level event is then generated as a CMIS notification and filtered through the CMIS event forwarding discriminator layer.

However, the power of this information aggregation framework can be fully exploited when a more generic object services model is employed, such as the one provided by CORBA. The CORBA framework allows the object to define an arbitrary set of services that complement the intended management function of the computed view. For example, if the computed view represents a virtual private network service, the corresponding object may include higher-level services to dynamically manage the allocated capacity of the VPN, retrieve graphs of usage information, configure event notifications whenever a particular usage pattern is observed, generate monthly billing information, provide predictions of service usage to help the manager provision for additional capacity, etc. In contrast to the declarative specification of an attribute's value, the additional services provided by the computed view cannot be generalized. They must be specified and implemented separately for every view. Some of these services may include functions that display the computed view in a particular graphical interface environment. For example, the Marvel framework [ANE98d] uses the concept of *visual domains* to represent computed views in different graphical forms (conventional user interfaces such as Motif, Java-enhanced web pages, etc.).

2.6 Dynamic Query Processing

The realization of computed views and especially monitoring views requires substantial processing power, especially when they need to be updated frequently. For many years, the argument against computed views has been that they are needed infrequently and, by consequence, should only be computed on demand. It is not clear however that this approach has proven to be more efficient. Element management agents usually store only dynamic information, and as a result, this information needs to be transferred periodically to large data warehouses for off-line processing. A typical example is the generation of usage logs for customer network management services. The network operator collects usage statistics for every customer and generates a usage report at regular intervals. The problem with off-line computation is that the computed view can be significantly out of date, and further, the needs for storing such information from a very large network can easily surpass the cost for computing it on-line. We believe that processing power today is cheap, and when combined with a distributed computing environment, the advantages of on-line view computation outweigh the cost. Not only can the physical location of view computation be completely hidden from the manager, but it can be enhanced with a server-driven graphical user interface, as is the case in web-based management systems. The result is an easy-to-use higher-level management service that provides significant added value to its users.

There are many cases however that the manager wishes to obtain highly customized views. One could regard those as the database equivalent of a query with unusual selection predicates. For example, one may wish to create a sorted table of customers spending more than $20 every month. It would be probably inefficient to dedicate a separate object to represent the above quantity since this type of query may be encountered very infrequently, or with a different predicate every time. For this reason, we intend to extend model with a query processing capability that receives queries in a structured language and generates transient objects that represent the appropriate computed views. The objects are deleted once the results of the query have been returned to the client application.

3. Applications

Most of the functionality of the information model presented in the previous section is currently being incorporated into the *Marvel* system, a distributed management framework developed in AT&T Labs Research to generate computed views using Java and the world wide web [ANE98d]. Marvel does not replace existing element management agents but rather builds on top of them a hierarchy of servers that aggregate the underlying information in a synchronous or asynchronous fashion into view objects and present it in the form of Java-enriched web pages. It uses a distributed database to reduce the cost associated with centralized network management systems and mobile agent technology to support thin clients by uploading the necessary code to access Marvel services and extend its functionality dynamically by downloading code that incorporates new objects and services.

We are currently using the Marvel system to manage the SAIL experimental home access network. SAIL (Speedy Asymmetric Internet Link) is an AT&T Labs 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 an information aggregation model to provide summarizations of performance data and bulk control actions. The architecture is shown in Figure 4. We use one Marvel server for every group of users served by the same cable operator (i.e. that share the same head-end). These servers operate at level 1. At the second level of the hierarchy, another Marvel server provides global monitoring statistics for the entire Sail network.

Computing performance views 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 may also help in planning the system capacity and the 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 location of the problem.

In terms of control operations, cable modems occasionally require updates in their operating software. Marvel 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.

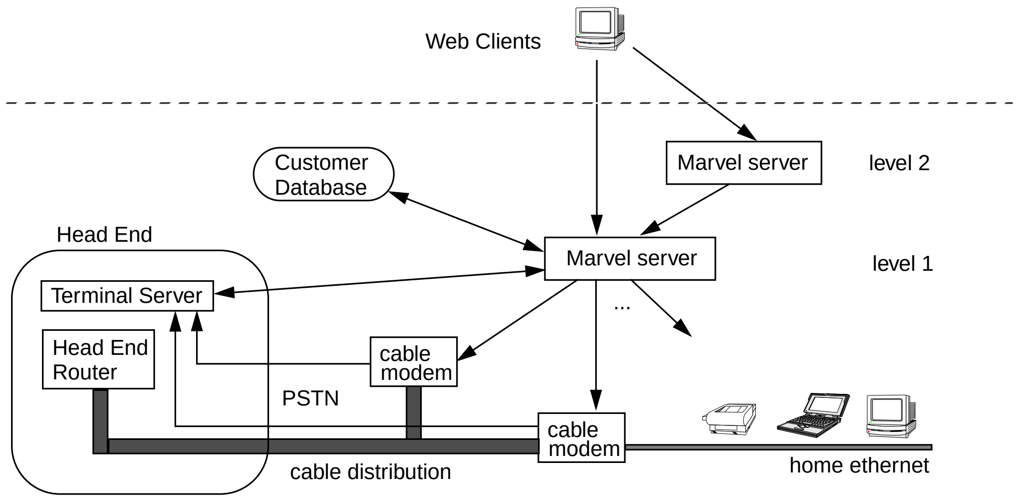

Figure 4: Managing the SAIL Network

In addition to the above performance summarization features we also support customer views of the service through “user profile” objects. These objects combine account information from the user registration database with per-user temporally aggregated performance data. Every such object 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.

4. 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 [ITU91].

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:

Domains [SLO89] was the first framework to explore division of management responsibility by defining groups of managed objects on which a common management policy is applied. Groups in Marvel do not represent managed objects but rather collections of network elements and servers in which managed objects are stored. Marvel assumes that if a new view (or management policy) needs to be defined on a collection of objects, it will be provided by a new object and hence the concept of grouping is used only as a means of reference from the new object to its information components.

[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 [ISO91].

[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 most coordinated effort in the area of network configuration directories is the one from the directory enabled networks (DEN) group. The DEN proposal [JUD98] integrates the storage of information about network components, services and users in an X.500 directory. It defines 8 base classes and an extensible schema based on inheritance and aggregation for modeling application-specific properties and information. The information model allows to describe both the static structure and temporal relationships (behavior) of network elements and services, and further allows for the definition of groups. We believe that the DEN effort will provide an excellent support framework for the work presented in the paper. The definition of the group hierarchy required by the information model that we have presented here can be stored into this directory and accessed during the execution of data aggregation functions. In addition, the directory can be used to store common aggregation function definitions together with their executable code. One could also control the declarative specification of an attribute representing an aggregated quantity through the directory.

Finally, CIM [DMT98] aspires to define a common information model based on object-oriented constructs that can be used across management applications. We are currently exploring ways in which CIM can be expanded with features from our information model to provide automated information aggregation functionality.

Overall, we believe that our work constitutes a valuable addition to previous work in the field by proposing an information aggregation model and integrating monitoring, control and event views inside a distributed computing framework. Our model can expand naturally other object-oriented information models to include this functionality, since it affects only an object's implementation and not its programmatic interface or modeling constructs.

5. Conclusions

A key feature of future network management systems will be scalability: the ability to deploy them in networks of very large sizes and configure them to present to the manager the important aspects of network state rather than the details. We have presented an information model that can be used in such systems and potentially coexist as an extension to existing modeling standards such as SNMP, GDMO and CIM. The model allows the network manager to define computed views of network information and integrate them in a distributed management system. Computed views can be of three forms: monitoring, control and event views. We presented an object-oriented model capable of implementing these views and the concepts with which aggregations of information can be realized.

Most of the work presented in this paper is currently under implementation as part of the Marvel project at AT&T Labs Research, an extensible distributed computing environment that provides higher-level management services and web-based management interfaces. Our recent work in this area investigates a modeling language for describing information aggregations and a more formal procedure for expressing them in an existing management information model.

References

[ANE98d] N. Anerousis, “Scalable Management Services using Java and the World wide web”, in *Proceedings of the 9th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, Newark, DE, October 1998.

[DMT98] Desktop Management Task Force, “Common Information Model (CIM) Specification Version 2.0”, March 1998.

[GOL96] German Goldszmidt, “Network Management Views using Delegated Agents”, in *Proceedings of the 6th IBM/CAS Conference*, Toronto, Canada, November 1996.

[GRU97] R.E. Gruber, B. Krishanmurthy, E. Panagos, “READY: A Notification service for ATLAS”, AT&T Labs Technical Memorandum HA6163000-970905-07TM, September 1997.

[HAY96] R. Hayton, J. Bacon, J. Bates and K. Moody, “Using Events to Build Large Scale Distributed Applications”, in Proceedings of the 1996 SIGOPS European Workshop, Connemara, Ireland, Sept. 1996.

[ISO91] Information Processing Systems - Open Systems Interconnection, “Structure of Management Information - Part 1: Management Information Model”, July 1991. International Standard 10165-1.

[ITU91] ITU-T Recommendation M.3010, “Principles for a Telecommunication Management Network”, Geneva, November 1991.

[JUD98] S. Judd and J. Strassner, “Directory-enabled Networks - Information Model and Base Schema”, preliminary draft, February 1998.

[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.

[MAN97] M. Mansouri-Samani and M. Sloman, “GEM, A Generalized Event Monitoring Language for Distributed Systems”, IEE/BCS/IOP Distributed Systems Engineering Journal, 4:2, June 1997, pp 96-108.

[NYG95] N.Y. Nygate, “Event Correlation using Rule and Object Based Techniques”, in Integrated Network Management IV, Chapman and Hall, 1995.

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

[SLO89] M. Sloman and J. Moffett, “Domain Management for Distributed systems”, in *Integrated Management I*, pp. 505-516, North Holland, 1989.

[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.

[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.