SCALABLE MANAGEMENT SERVICES USING JAVA AND THE WORLD WIDE WEB

Nikolaos Anerousis

AT&T Labs Research
180 Park Avenue, Bldg. 103
Florham Park, NJ 07932-0971
tel: (973) 360-8767, fax: (973) 360-8871
e-mail: nikos@research.att.com
http://www.research.att.com/~nikos/

Abstract

Marvel is a distributed computing environment that allows the creation of scalable management services using intelligent agents and the world-wide web. Marvel is based on an information model that generates computed views of management information and a distributed computing model that makes this information available to a variety of client applications. 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 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 a) support thin clients by uploading the necessary code to access Marvel services and b) extend its functionality dynamically by downloading code that incorporates new objects and services. A prototype implementation in Java is presented together with results from its first application on a broadband home access network using cable modems.

Keywords: Network management, web-based management, distributed computing, information aggregation, Marvel.

1. Introduction

Java and the world wide web have received significant attention recently as a means of providing low cost and easy to use management services. Customer network management is an example of an application that has benefited significantly from the web-based management model. Traditionally, customer network management services have required the installation at the customer premises of high-end network management applications such as HP Openview to access fairly low level management information using SNMP or log files that are copied and examined off-line. By incorporating this information into web pages, service providers have a more cost effective way of providing management services to their customers. Moreover, customers have come to appreciate the easy-to-use interface and the absence of a need to maintain a separate service management system.

Customer network management is only one successful application of web technology in network management. Its success reveals however an important trend: that the emphasis on providing network management services is not on how these services are provided (i.e., the underlying information modeling and protocol technology such as SNMP or CMIP), but rather on the *functionality* that these services provide to the user. Web technology requires a provider to define both the level of abstraction at which management information is presented and the graphical user interface to interact with these services. By hiding the details of the underlying management technology under an easy to use and access interface, network management becomes

a useful commodity rather than a privilege of the network operator. Today, the bundling of management capabilities with a service is proving to be a product differentiator of increasing importance.

On the other hand, web technology has its limitations and cannot be used (yet) to replace every management application, particularly the high-end network operation centers. There are several reasons contributing to this fact:

1. The management systems of large networks require a large display area to project maps of different logical views of the network. Even if the web-based interface is replicated, there is little support in current web technologies for coordinating individual displays with each other.
2. Web interfaces require the client to retrieve large amounts of code and state information from a server. As a result, the rate at which information can be consumed by the client's display is very limited compared to the capabilities of a traditional centralized network management system.
3. Web-based interfaces that are limited to HTTP interactions have a very restricted control interface with the server (hypertext links, post operations, CGI scripts, etc.)
4. Java-enriched interfaces are affected by the performance problems of Java and the reliability of its graphical user interface.
5. The web computing model does not allow clients to use many local resources (e.g. the file system, network connections to other servers, etc.), and as a result significant bandwidth and computing resources are wasted in transferring information from and to the server.
6. Web interfaces require that an additional layer of processing be introduced in the management architecture to convert management information into a web-based form. This may impact the freshness of information and the response time of the management system to control actions.
7. Finally, the power behind web-based management can sometimes be its most significant limitation: since management services and the way management information is presented at the client are defined at the server, there is no capability for the client to further process this information to create, for example, more detailed management views or correlate the information with other prior knowledge.

The Marvel project at AT&T Labs Research is an attempt to investigate the limitations of web technologies in network management and propose an architecture that can both scale and rival conventional management systems in terms of performance and expandability. Marvel (which stands for Management Aggregation and Visualization Environment) is trying to address the following issues:

- Expand the web-based views of management information well beyond customer network management and network element management applications. Marvel uses a framework for information aggregation that allows the dynamic construction of arbitrary views of management information that can also be beneficial to network operators [ANE98c].
- Propose a distributed computing model for accessing management information that can be easily incorporated into web clients. Standards such as CORBA and Java remote method invocation (RMI) [JAV97] are now widely accepted for network management applications. Marvel uses a distributed computing environment to reduce the cost related to deploying centralized hardware and software and at the same time hide all details of this distribution from its users.
- Use concepts from intelligent agent and mobile code technology to enhance the functionality of the web-based interface and improve the availability and maintainability of management services.

Marvel is not limited to the web-based interaction model. Rather, any type of client applications can be built around its distributed computing services, from traditional applications with their own user interface, to server-driven graphical interfaces. The emphasis, however, of this paper is on presenting the aspects of Marvel that can be most beneficial in deploying web-based management services.

This paper is organized as follows: Section 2 presents the architecture of Marvel; Section 3 describes the current implementation work. Experiences from using the Marvel system on a production network can be

found in Section 4. Section 5 presents related work in the field, and Section 6 our conclusions and directions for further study.

2. Marvel Architecture

2.1 Information model

Marvel follows an object-oriented model to store management information. Marvel objects are computed reflections (views) of management information that reside in existing network management agents or other repositories of information in the network. We also refer to those agents as element management agents (EMAs), since they are usually associated with a particular network element and follow one of the network management standards such as SNMP, CMIP or DMI.

Marvel objects are stored in the database of a special management agent (the Marvel server - MS), and are sometimes referred to as *aggregated managed objects* (AMO) since they represent the result of a filtering process on information collected from element management agents and/or other servers. An object implementation does not follow a particular standard; its structure however resembles the OSI structure of management information [ISO91]: every Marvel object contains a list of attributes, and each attribute represents a computed view of lower-level management information. However, in contrast with the OSI model that does not specify how attribute values are computed, Marvel follows some very specific guidelines that allow the manager to link each attribute value to a complex filtering process. More details on how this information model operates are presented in [ANE98c]. In this way, Marvel attributes can be specified in a declarative fashion during the specification of the object and save much programming effort which would be usually required in a similar CMIP agent implementation.

For example, one Marvel object may represent a customer profile in a customer network management system. It may contain attributes that represent the customer's identity and billing information, the services that the customer is subscribing and performance data acquired from the service usage logs. In the case of an ATM virtual network service, performance data can be presented for every virtual path that composes the virtual network, or as an aggregate over all component virtual paths. For example, the customer may be interested in seeing the total capacity usage of the virtual network, a quantity that can be easily computed by adding the corresponding performance figures of the component virtual paths. Marvel allows to specify how these attribute values are computed in a declarative fashion: attributes are linked to groups of information components and to a filtering function that computes the final value. Optionally, the declarative specification can be bypassed to cope with information abstractions that are not easily specified in a declarative fashion [ANE98c].

2.2 Computation model

Every Marvel system is composed of a hierarchy of servers. We chose a distributed architecture for several reasons:

- Computed views of management information require significant processing resources, especially when they must be maintained continuously up-to-date. By distributing the computing task, our architecture can be more scalable.
- Computations can take place closer to the sources of management information such as element management agents, thereby reducing the amount of management traffic to a centralized network management facility.
- A distributed architecture is more reliable since the failure of one server affects only the availability of information computed within that server. In addition, server maintenance and upgrade tasks can be performed without affecting the operation of the entire system.
- And finally, a distributed architecture can be implemented by putting together components of lower cost and smaller footprint.

Every server stores its objects in a persistent database. Since computed views of management information usually include valuable historical data that cannot be easily reconstructed, object persistence ensures the availability of this information through server failures.

Objects in Marvel provide two tiers of services: *Basic Access*, which are mandatory for all objects, and *Extended* that are implemented optionally. The latter can be used to provide a richer customized interface to the object for performing more complex operations related to its intended management function. There are three types of basic services:

1. **Attribute access** services are used to set and retrieve attribute values and control several aspects of every 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 objects), etc.
2. **Visualization** services are used to provide clients with the necessary information to setup graphical user interfaces (GUIs) to access the object's basic and extended services. The benefit of this approach is that clients do not need to be aware of an object's internal structure to provide a user-friendly interface. In essence, the GUI is "programmed" as part of the object and is transferred to the client when it first accesses the object. The object may provide more than one visualization services depending on the type of clients that are supported by the Marvel system (Section 2.3 contains a detailed description of the visualization model).
3. **Event** services are used to subscribe internal and external consumers to receive event notifications generated by the object, and control the event flow. Events in Marvel are usually aggregations of lower-level events corresponding to the management view portrayed by the object. The Marvel event system is described in detail in [YUC98].

The Marvel object 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 visualization functions can be overridden to set up a user interface that accesses some of the object's extended services.

In addition, every Marvel server provides a set of high level services that can be used by client applications to navigate and examine the database:

1. **Navigation** services are used to navigate through the server database and examine its contents. Current Marvel implementations store objects in a tree, and include functions like *getRoot* (retrieves the root object), *getParent* (retrieves the parent of an object), *getSubordinates* (retrieves the object's children), *getPath* (retrieves the path from the root), etc.
2. **Registration** services are used to examine the structure and capabilities of every object in the database. In this way, clients can dynamically browse through the services provided by the object and invoke a service with the appropriate parameters. This introspection capability does not require clients to be previously aware of the services provided by every object. Rather, services are "discovered" in real-time and invoked after loading the appropriate stub code at the client. Objects must register themselves when they are created and provide information on the attributes they contain, the extended services they support and the stubs that must be loaded to invoke these services.
3. **Object management** services are used to instantiate, upgrade or delete objects while the server is running. The manager provides the name of the object to be instantiated, its location in the database and a pointer to the code that can be used to instantiate the object. The server then dynamically loads the code and generates a new object instance. Objects can be upgraded, in which case, the state of the object is frozen, the old code is purged from the agent, the new code is loaded and the captured state is passed to the new object.

The above services can be implemented using industry-standard platforms such as CORBA [OMG93] and Java [SUN97] which are currently being used in many network management applications. Java's remote method invocation (RMI) is a package that provides distributed computing primitives tightly integrated with

the language, and is extremely easy to use and integrate into Java applications. CORBA is a more widely accepted standard but requires more heavyweight implementations. Our intention is to provide the same object services under both frameworks: Java RMI is more suitable for web (and other lightweight) clients, while CORBA for more demanding applications that require the widest possible interoperability.

2.3 Object visualization model

Marvel does not rely on an existing standard to store management information or a particular protocol to access it. The architecture assumes that all interaction between a client and the Marvel system occurs at a very high level, provided by a user interface that is dynamically loaded into the client. The interface hides the details of how objects are defined and what services they provide.

Marvel was designed under the assumption that the majority of its clients will have no prior knowledge of the information stored in Marvel servers and the methods used to access it. Clients rely on standard features provided by the distributed computing platform to download all the necessary code to navigate through the database and generate a graphical user interface to interact with Marvel objects. The Marvel framework requires that every object be able to "visualize" itself by generating a user interface. In fact, there may be several ways of visualizing an object, depending on the capabilities of the client. For this reason, Marvel supports a small number of *visual domains*. For every supported domain, a Marvel object must implement a visualization function capable of displaying the object in that domain. For example, a Gopher system would require that the object be converted into a textual representation before it can be displayed. A web-based system would require that every object be converted into an HTML page[1], and any control actions for the object be implemented through HTML post operations and a CGI interface [MAS97]. A Java enriched web browser can download Java applets to provide a more interactive interface and use directly distributed computing facilities such as CORBA and Java RMI to access the object's services. As web and remote visualization standards evolve, new visual domains such as XML can be added in the future.

The latter visualization technique is of particular interest to our architecture due to the wide acceptance of the world wide web and Java. Object conversion to Java-enriched HTML is a mandatory service that must be provided by every Marvel object. The technique works as follows:

First, the client invokes the object's toJeHTML() method in one of the following ways:

- by directly invoking the object's method through the distributed computing environment, or,
- by indirectly making an HTTP get request supplying the object's name and address. The get request is then translated by the HTTP server to a call to the object's toJeHTML() method, and the results are returned through the HTTP reply (Figure 1).

When the toJeHTML() method is called, the object generates an HTML page that can be viewed by the browser. It does so by generating a default layout for the page, on which the values of the attributes will be displayed. Then, each attribute is instructed to convert itself into a Java-enriched HTML form. Simple data types such as strings and integers need only convert themselves into simple text. More complex data types (especially the ones representing computed views of management information such as tables and time-series graphs) may choose to invoke a Java applet (by inserting the <applet> primitive). The same holds for attributes that represent the object's control capabilities. When the applet is used purely for monitoring purposes, it is possible to supply all the necessary information inside the applet specification block through the <param> primitive. It is also possible however to pass to the applet the name and address of the object, in which case the applet can interact with the object directly. This is required for applets that need to perform control operations on the object, or to refresh the displayed information after the page has been loaded.

When the web browser encounters the <applet> block within the html page, it attempts to load the applet's code and any other Java classes needed for the applet's operation. Java classes are always loaded from an HTTP server.

---

1. The Hypertext Markup Language (HTML) is the language used to format web pages [GRA97].

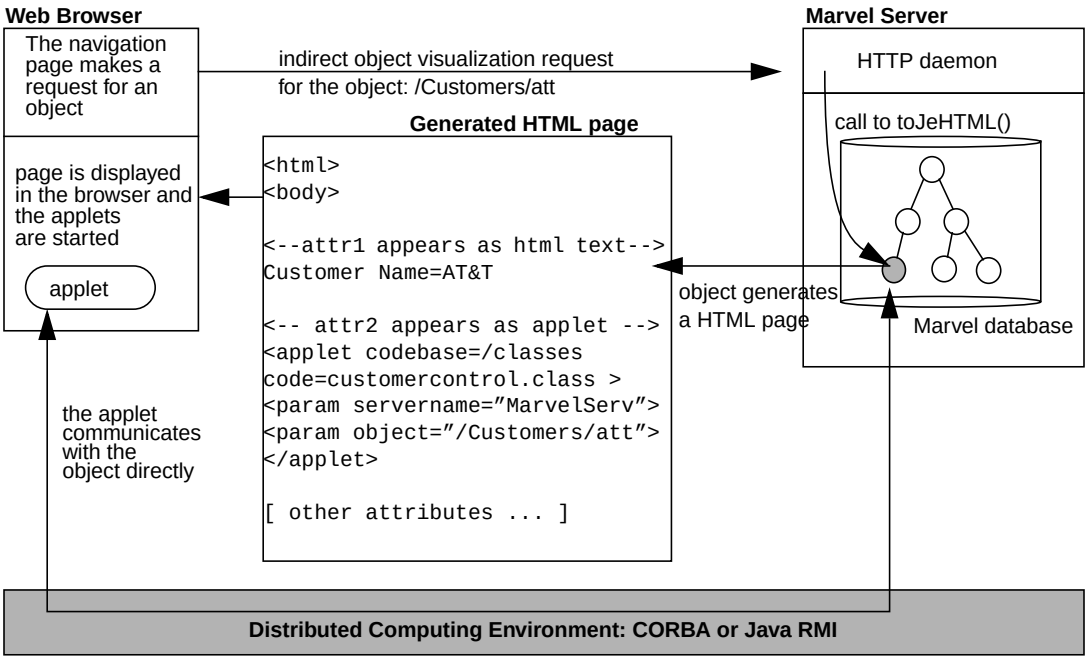

Figure 1: Example of an indirect object visualization procedure through an HTTP server

2.4 Object distribution model

Marvel objects are distributed across a hierarchy of Marvel servers. The number and location of these servers is determined by the system designer, taking into account the amount of management data that needs to be processed, the storage and computing capacity of the servers, etc. Servers that perform frequent aggregation functions on management information should be located as close to the sources as possible. Servers are typically organized in a hierarchical structure for convenience. For example, at the lowest level, there exist servers that process data collected from element management agents using one of the standard management protocols. These servers can generate views that correspond to the TMN subnetwork layer. A server at the level immediately above can generate management views for the network and service layer, etc. (Figure 2).

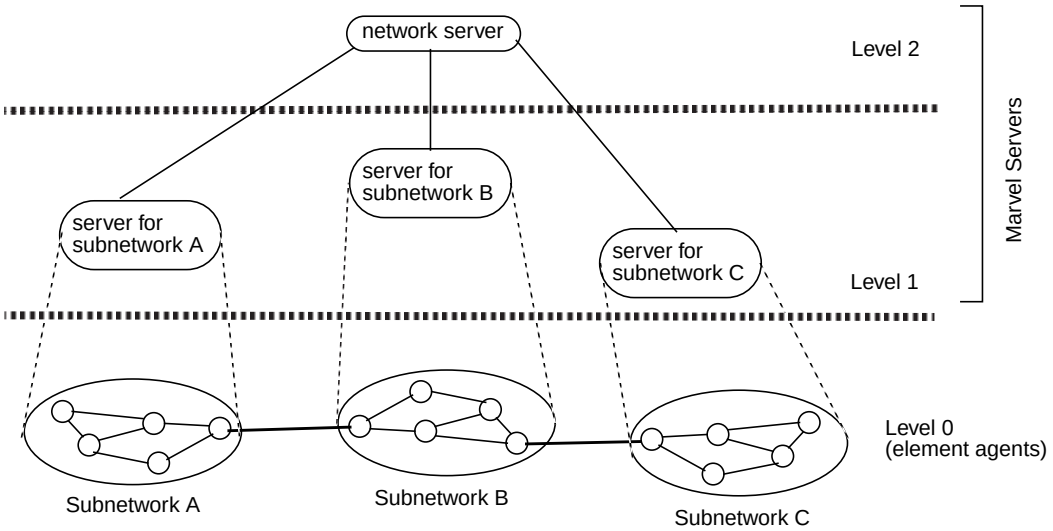

Figure 2: Example of Marvel server distribution

In order to make navigation easier through this hierarchy of databases and hide the details of object distribution, Marvel uses a redirection primitive as part of the object navigation services. An external configuration directory contains information about the hierarchy of Marvel servers. Every server can determine its parent, siblings and descendants in the hierarchy by supplying its identity to the configuration database. The server then creates pseudo-objects that, when accessed through the navigation interface, return a pointer to the navigation service of the corresponding server. In this way, the manager is able to navigate through the hierarchy of Marvel servers and gain access to their objects without knowing the details of the hierarchy or address or physical location of these servers. Navigation in this distributed database resembles very much the navigation through a Unix file system with NFS mount points.

3. Marvel Implementation

We selected Java to implement the Marvel architecture because it has a number of useful features:

- Distributed object services (Java RMI) are well integrated with the language.
- Ability to load and execute code dynamically.
- Graphical User Interfaces are a core part of the language.
- Compile once - run everywhere capability.
- Is supported from practically every computer on the Internet with a Web browser.

In the current implementation, Marvel clients are web browsers with the capability to run Java. In future implementations, we expect a wide variety of clients, from text based tools to customized applications that make more efficient use of the distributed computing environment. Marvel servers are fully implemented in Java to take advantage of code portability and dynamic execution features of the language.

3.1 The Marvel server

Every Marvel server is composed of a collection of subsystems (Figure 3). This section provides a general description of each subsystem. We have omitted most implementation details and API specifications due to lack of space.

Marvel objects are defined directly in Java and stored persistently [ODI97]. The root of the inheritance tree provides a template for implementing an object's basic and extended services, and a facility for registering with the local server. Object services are currently provided using Java RMI. The database is implemented as a tree of objects. It is usually convenient to follow a natural containment relationship for placing objects in the tree. For example, an object representing a summarization of performance parameters from a set of users could be placed as the parent of the objects that contain performance parameters for individual users. Every object has a unique string identifier (the name of the object) that distinguishes it from its siblings. Path names in the Marvel database are constructed by following a path from the root to the object and using '/' as the path separator, exactly as in the Unix file system. For example "/Customers/att" refers to an object named "att" located under a container object called "Customers". Every object implementation has the capability to initialize threads that can be used for background computations, event generation, etc.

A set of basic attribute classes are also included in the Marvel core package, to implement simple data types, tables and time series. More complex data types can be easily derived from these base classes. The constructor function for every object 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 object's basic and extended services. When the attribute class library needs to be augmented 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 server since the appropriate viewer/controller will be loaded automatically.

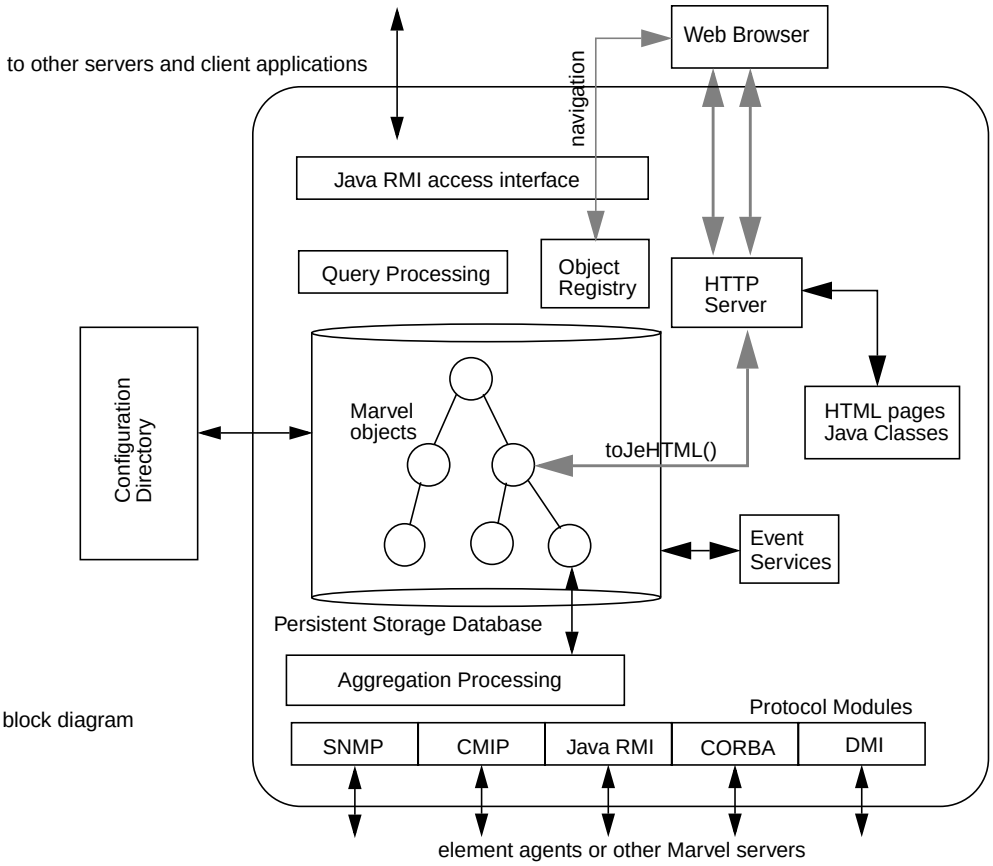

Figure 3: Server block diagram

Operating closely with the database is the aggregation processing engine. This engine is responsible for computing every object's attribute values. Value computations can either be manual (the object programmer is responsible for collecting the necessary information, or automatic (in which case the attribute's declarative specification is used [ANE98c]). In the latter case, the value computation proceeds as follows: First the information components are identified. These can be objects within an SNMP agent, attribute values within a CMIP agent, attribute values within another Marvel server, etc. Then, using the appropriate protocol, the required values are collected, and a filter function is applied that produces the final value. This computation can occur in a synchronous fashion (when the manager requests the attribute value) or asynchronously (by a separate thread executing in the background). The latter is preferable in cases where the attribute is requested very frequently or changes in its value must be monitored for generating potential event reports.

An external configuration directory is used to store information about the entire Marvel system (the hierarchy of servers, group definitions used in attribute specifications, dynamically loaded code, etc.). The directory is the only centralized piece of our architecture and is consulted frequently during the operation of each server. We have chosen a centralized directory to avoid information replication issues between the servers, since the configuration information is not static: Marvel servers can be added into the system at any time, group definitions can change to incorporate new elements in the network, etc. Currently, we are using an X.500 directory with LDAP as the directory access protocol. We are following closely the directory enabled networks (DEN) initiative [JUD98] to eventually align the Marvel directory architecture with this work.

An HTTP server is integrated in the Marvel server for performance reasons. Incoming http requests for Marvel objects are converted into a call to the object's toJeHTML() function and the result is printed directly to the http response stream. This technique does not buffer the resulting HTML text (as is the case for CGI and fast-CGI calls) and allows for very fast rendering of these pages in the browser.

Two additional modules are currently under development: The event processing module is responsible for aggregating lower-level events, generating event notifications, registering event consumers and creating event channels over which events are transmitted [YUC98]. Also, many times a manager may wish to generate a computed view of management information using an unusual filtering predicate. 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, the Marvel architecture allows for a query processing module. This module 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.

3.2 The Marvel client

The minimum requirement for a Marvel client is to have a bootstrap capability to load the client-side code from a Marvel server. The bootstrap code will then enable access to Marvel services. The current architecture favors clients that support display of HTML pages and Java, such as the most popular web browsers. Every server maintains a "home page" which, when loaded by a client, starts a navigation applet that is used to examine the server's database and invoke an object's visualization function. More demanding applications can access directly Marvel services using the Java RMI environment.

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 Labs broadband 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 architecture to provide summarizations of performance data and bulk control actions. The architecture is shown in Figure 4.

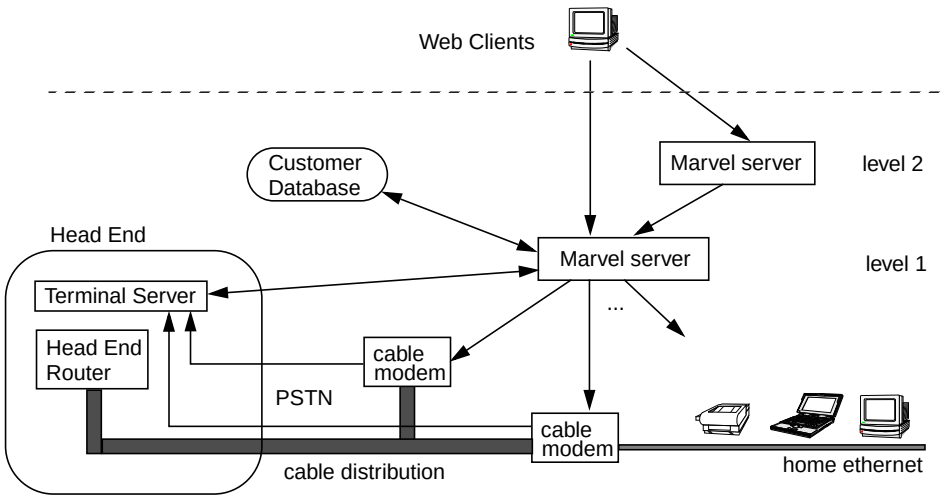

Figure 4: Managing the SAIL Network

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 can 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 a group that represents the entire network. We have found that the Marvel system has many advantages over commercial network management applications because performance aggregations can be stored and accessed at any time. In order to do so when using one of the current commercial network management tools, a new application needs to be written that accesses off-line the centrally collected data from the cable modems using a proprietary database access interface.

In addition to the performance summarization features we also support customer views through objects that represent “user profiles”. 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 a user can access to view its 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.

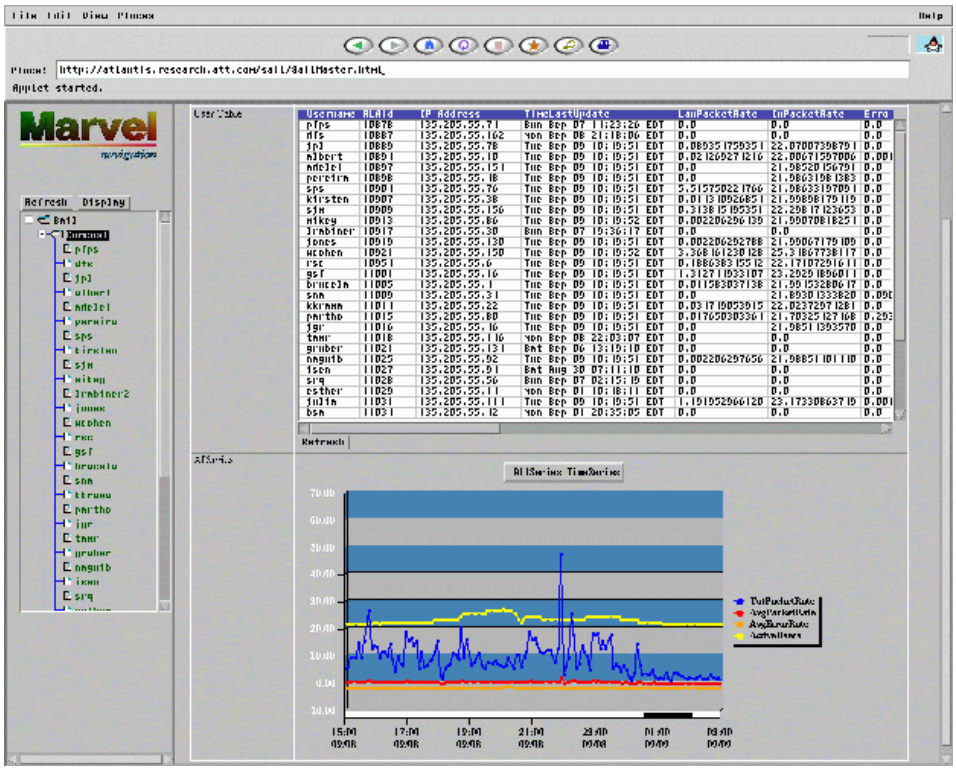

Figure 5: A snapshot of the cable tree group object

In the current architecture we use one Marvel server for every CATV distribution tree (about 50 modems). The server periodically obtains information from the cable modems through a low level monitoring and control protocol and updates the appropriate objects. 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 (see Figure 5). 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. The second level server contains performance and control aggregations for the entire network and allows software distribution to the entire set of cable modems.

5. Related Work

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). More recently, many researchers are investigating the benefits of intelligent/mobile agent technology as a more generic framework for performing distributed management tasks [MAG96, BAL97].

In the visualization area, [CRU93] first proposed an immersive environment that enables the manager to navigate through the network state and perform high level control operations through the exchange and processing of multimedia information. Early work on Web-based management can be found in [BAR97] and [REE97]. The first presents an HTTP proxy to SNMP and CMIP resources and further describes a framework for remote execution of management tasks written in Java. The latter describes the implementation of a Web-based management console for the IBM Netfinity PC management application. [MAS97] highlights the advantages of using the Web as a network management tool and presents the advantages and disadvantages of different Web technologies (CGI scripts, HTML forms and Java) in element, network and service management applications.

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. Marvel has several advantages over JMAPI, namely the capability to define automatically computed views of management information, protocol adapters to a large variety of managed elements, a unified object naming space, and support for a large variety of thin clients.

Similar in functionality to JMAPI is the WBEM project [THO98]. WBEM relies on the HyperMedia Management Protocol (HMMP) to access management information, allowing management solutions to be platform independent and physically distributed across an enterprise. A more detailed survey of web technologies for network management can be found in [HON98].

6. Conclusions and Future Work

Marvel is a framework that enables the development of scalable network management services. Scalability is achieved by a) supporting computed views of lower level management information that convey the essence of the network operating state rather than the details, b) distributing the view computation task to a hierarchy of processors (servers) and, c) supporting large numbers of clients allowing ubiquitous access to network management information.

Marvel objects represent aggregations of management information which can be obtained through any combination of standard management protocols such as SNMP, CMIP, DMI or any other non-standard solution. The manager is responsible for specifying the policy with which these aggregations are computed. Information stored in Marvel objects is then made available in a variety of visual forms using the concept of visual domains and object self-visualization. A distributed computing environment such as Java RMI or CORBA provides remote object access services. We have given particular attention to the web visual interface; web browsers can navigate through the Marvel distributed database and visualize objects of arbitrary complexity by dynamically loading the appropriate viewers/controllers.

The SAIL broadband home access network is currently serving as the platform for testing the view computation framework and evaluating several aspects of the Marvel system such as security, client/server interaction performance, etc. Work is also under way to add event support, enhance the attribute library with additional data types and design an improved user interface that integrates data mining with event notifica-

tion capabilities. Security is also a major issue that requires further study. We are exploring an access control list based model to restrict access to the services provided by Marvel objects.

References

[ANE98c] N. Anerousis, "An Information Model for Generating Computed Views of Management Information", *Proceedings of the 9th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, Newark, DE, October 1998.

[BAL97] M. Baldi, S. Gai and G.P. Pico, "Exploiting Code Mobility in Decentralized and Flexible Network Management", *Proceedings of the First Intl. Workshop on Mobile Agents*, Berlin, Germany, April 1997.

[BAR97] F. Barillaud, L. Deri and M. Feridun, "Network Management using Internet Technologies", in *Proceedings of the 1997 IEEE Integrated Management*, San Diego, CA, 1997.

[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.

[GRA97] I.S. Graham, "HTML Sourcebook", John Wiley and Sons, 1997.

[HON97] J. Hong et. el., "Web-based Intranet Services and Network Management", IEEE Communications Magazine, October 1997.

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

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

[MAG96] T. Magedanz, K. Rothermel and S. Krause, "Intelligent Agents: An Emerging Technology for Next Generation Telecommunications?", in *Proceedings of the 1996 INFOCOM*, San Francisco, CA, 1996.

[MAS97] M.C. Maston, "Using the World Wide Web and Java for Network Service Management", in *Proceedings of the 1997 IEEE Integrated Management*, San Diego, CA, 1997.

[ODI97] Object Design Inc., "Objectstore Persistent Storage Engine for Java", URL: http://www.odi.com/content/products/pse/doc_120/doc/index.html.

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

[REE97] B. Reed, M. Peercy and E. Robinson, "Distributed Systems Management on the Web", in *Proceedings of the 1997 IEEE Integrated Management*, San Diego, CA, 1997.

[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/.

[THO98] J.P. Thompson, "Web-Based Enterprise Management Architecture", IEEE Communications Magazine, March 1998.

[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.

[YUC98] S. Yucel and N. Anerousis, "Event Aggregation and Distribution for Web-based Management Systems", under submission.