# Event Aggregation and Distribution in Web-based Management Systems

***Sakir Yucel*** [1]

Computer and Information Sciences
University of Delaware
Newark, DE 19716
tel: (302) 831-1949 831-8458
e-mail: yucel@cis.udel.edu
http://www.cis.udel.edu/~yucel/

***Nikos Anerousis***

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

**Abstract**

As networks grow in size, managers rely increasingly on asynchronous notifications (events) for most management tasks. With the increasing popularity of the web as a means of providing access to network management services, there is a strong demand for enabling platforms that provide these services in a scalable and cost-effective way. This paper presents the architecture of the event generation and distribution subsystem of the *Marvel* [ANE98d] distributed management environment. The latter allows the network manager to define new event types by aggregating lower level events and make them available to a variety of client applications through the brokerage services of a distributed computing environment. Events are serialized objects that can carry Java byte code to display themselves according to the event consumer's graphical capabilities, or possibly take a corrective action if the event corresponds to a fault condition. Event filtering and aggregation is performed by an external process, operating under the control of the Marvel event management service. The architecture has been applied on an existing management platform based on Marvel for a home broadband access service.

**Keywords**

Event management, event aggregation, event distribution, event notification, event visualization, event broker, Web-based management, Java, Marvel, READY.

## 1. Introduction

Operators of large and fast-expanding networks rely increasingly on asynchronous notifications for most management tasks. The polling paradigm for detecting irregular conditions in a managed network has many disadvantages, especially with regard to the amount of attention that a management application has to devote in monitoring the state of network elements. Even an event-based model has limitations when it comes to receiving and processing events from a very large number of components. A common problem is that these (elementary) events are often correlated spatially (one fault appears as multiple events from many sources), and temporally (alarms are often repetitive until the error condition is cleared explicitly by the manager). Managing large networks requires the capability to reduce the number of events reaching the management system to the ones that are essential to detecting irregular conditions.

---

[1] Work performed when the author was a summer intern at AT&T Labs Research.

In our previous work [ANE98c,d] we described a distributed computing model for providing network management services to lightweight clients using Java and the world-wide web. In this environment, clients are not aware of the structure of management information and the management services provided at network management agents (servers), but rather download all the necessary code to access these services from a server. Java and the world wide web have become in the recent years the ubiquitous technologies supporting this type of functionality. Java provides all the necessary features for loading code dynamically and accessing distributed object services, while world-wide web (WWW) browsers have become the de-facto client software for information retrieval.

Web-based network management has increased in popularity for the same reasons. Currently, there exist a number of efforts for providing web-based network management services such as JMAPI [SUN97], WBEM [THO98] and Marvel [ANE98d].Marvel is a distributed computing environment that allows the creation of scalable management services using mobile code technology and the world-wide web. It 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.

One of the most important functions of a web-based management system is the handling of asynchronous notifications (events) emitted from managed elements and network management agents. This work attempts to complement the Marvel framework with an event generation, aggregation, and distribution model. While a number of products are available today for intelligent filtering of events, the system that we are developing has many advantages: It integrates the different phases of event processing, from event collection to event aggregation, to event distribution, and event visualization. It provides both temporal and spatial aggregation of events, and further allows the user complete control of how these aggregations are performed. And finally, it addresses the intricacies of Web-based management environments, which in their simplest form have very limited capabilities for receiving and displaying events.

This paper describes how events can be collected from network elements, aggregated through a series of spatial and temporal filters to represent a higher-level view of the managed network, compatible with the Marvel framework, and finally be advertised for consumption by clients through an event broker. Furthermore, we present a client architecture suitable for Web browsers that allows the coordination of Java applets in registering, receiving, displaying and automatically responding to events.

This paper is organized as follows: Section 2 describes our architecture for providing event services. Section 3 addresses the design and implementation issues of the architecture. An application of the event architecture on the SAIL broadband access network can be found in section 4. Section 5 presents related work, and Section 6 our conclusions and directions for further study.

## 2. Architecture

The Marvel framework [ANE98c,d] proposes an object-oriented information model in which objects represent computed views of management information. Computed views allow the network manager to interact with the managed system at a much higher level of abstraction compared to standards-based network management protocols such as SNMP and CMIP and, further, allow for a more scalable architecture. Every computed view has the following components:

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

To define a view, the network manager provides an aggregation rule used to compute an attribute's value. 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 value. Marvel objects are also referred to as *Aggregation Managed Objects* (AMOs), since their attributes represent aggregations of lower-level management information.

### *2.1. Event Architecture Overview*

Many web-based management systems do not provide satisfactory support for events, particularly when they rely on CGI interactions with a web server to retrieve and control management information. The reason is that the HTTP protocol has been primarily designed to retrieve files containing HTML text. It is not possible, for example, to maintain an open connection with the server and change on the fly the layout on a HTML page based on new information received from the server. However, support for events becomes much easier if, in addition to the HTTP/HTML interaction, a distributed computing environment is used to push events from the server to the client. This is possible when the client supports a flexible execution environment (e.g., a Java virtual machine) that can execute programs (applets) that receive events from a management server and present them through a graphical user interface. However, event collection and presentation is only a small part of the functionality required to support events in a web-based management system.

The event architecture presented here consists of the following functional components: event collection, event aggregation, event notification and event visualization as well as the management of the above. In order to provide the above functions, our event architecture is organized in four layers, as shown in Figure 1. The lowest layer is comprised of network elements and element management agents (EMAs). The EMAs are the primary event sources. The events generated by the EMAs are collected at the second layer by the READY system [GRU97], an event collection, filtering and correlation engine. The filtered events are then routed to the appropriate AMOs in the Marvel server. Every AMO adds the generated event to a local event log facility and further sends a copy of the event to the event services unit. The latter then sends the event to a set of registered consumers (clients). In addition, it provides navigation services for Marvel clients to browse through the available list of events and subscribe to events of particular interest.

The Marvel event architecture distinguishes between the following physical components:

- Event producers, event sources, and event suppliers are terms used interchangeably for the elements that generate events. These can be element management agents or other Marvel objects.
- Event consumers or event listeners are the elements that can register to receive events. These can be Marvel clients or Marvel objects (AMOs).
- Event servers are components that offer access to event services. The event broker for example is an event server offering event registration services. AMOs are also event servers offering an event log browsing capability.
- Event clients or event subscribers are elements that use the services of an event server.

The event system supports both synchronous and asynchronous event services. Synchronous services for example, include browsing and data mining functions on the event log kept by every Marvel object. In the asynchronous case, event consumers first register with the event broker to receive an event. When an instance of that event type is generated, an event object is pushed to the client. Thus, asynchronous event notification requires a registration step, which is not required in the synchronous case. Upon reception of an event, the consumer executes an action routine to handle the event. The action routine can be intrinsic to the client (e.g. statically compiled), or dynamically downloaded from a server, in which case the event is handled in a "pre-programmed" fashion. The event object may even carry the code in itself to bypass the code download step, which may delay processing of the event.

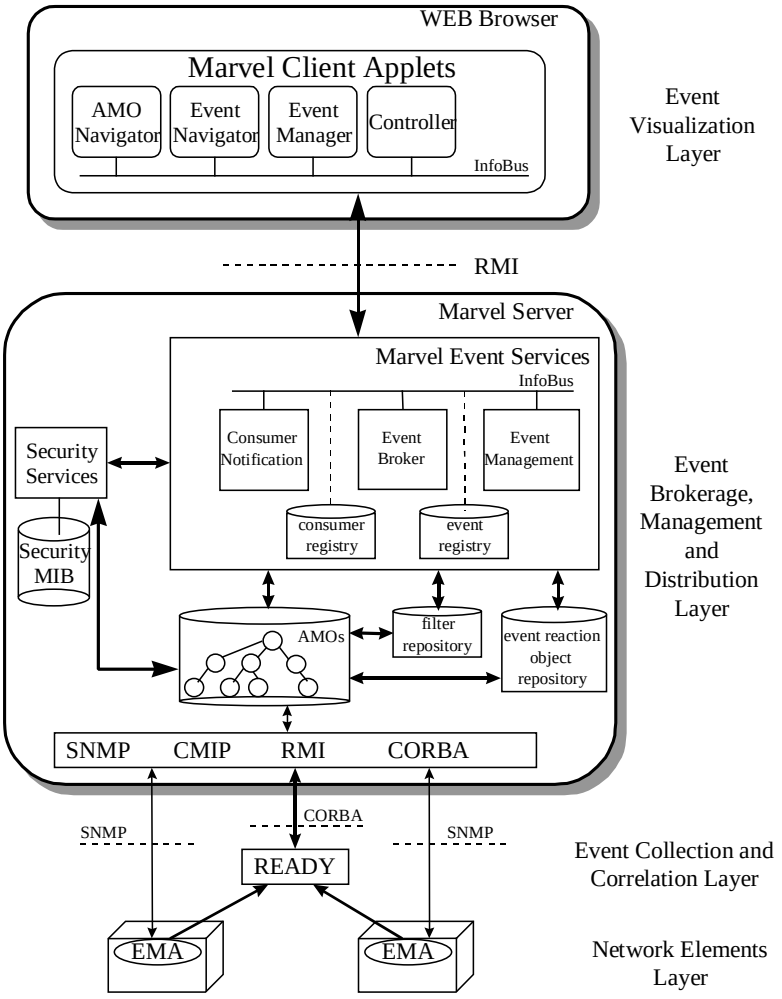


Figure 1. The Event Aggregation and Notification Architecture

### *2.2. Event Generation*

The primary sources of events are element management agents that have been configured to send notifications using a management protocol such as SNMP and CMIP to a Marvel server. The READY process associated with the server collects a series of elementary events and produces a higher-level event corresponding to a particular management view. Events can also be generated internally within a

Marvel server as a result of an attribute changing value, or another internal event such as a timer expiration, etc.

### 2.3. Event Collection and Aggregation

Every event generated by Marvel usually depends on the occurrence of a series of lower-level events. These events are collected by a READY process, filtered, and passed to the appropriate Marvel object for further distribution. READY [GRU97] is an event notification service developed by AT&T Labs. It provides both filtering of events and asynchronous and decoupled communication of event notifications.

The event view associated with every Marvel object specifies the types of events supported by the object and the conditions that generate them. When an event requires that a series of other lower-level events have occurred, an aggregation rule must be specified to compute the attributes of this event. Such a rule can be specified either declaratively, or explicitly:

- In the declarative specification case, the AMO notifies the READY process of its interest in producing the complex event. The AMO supplies the event sources and a filter specification that will be used to generate the event, and registers itself as an event consumer of the complex event type. In addition, it may also specify a set of delivery policies by describing how often it wants to receive the event. The filter specification is carried out in the READY event specification language, capable of expressing from simple attribute aggregations to more complex rule-based event correlation algorithms. The READY process, in turn, registers as a consumer to the specified event sources, and starts receiving the above events. When all conditions for generating a complex event have been satisfied, the complex event is sent to the AMO for further processing. The benefit of this approach is that event filtering is external to the Marvel system, allowing the use of different event filtering engines in the future. Marvel contains only the functions necessary to specify the conditions generating the complex events and the interface that Marvel clients must use to register as event consumers. Further, the event collection, filtering and aggregation tasks are delegated to a specialized event processing engine, which takes this burden off the Marvel server. This improves the overall response time for serving clients.

- In the explicit specification case, the AMO registers as a consumer to the event sources and starts collecting the events on its own. In this case, the programmer of the object must supply the code to collect the necessary events and trigger the generation of the composite event, bypassing completely the READY process. This becomes useful when the network operator is willing to implement a customized or complex aggregation rule, which cannot be specified in the READY language. The network operator can choose among already compiled and available aggregation rules, or, can write entirely new code. The code is then compiled at the Marvel server, dynamically loaded and used as the aggregation rule for the rest of the execution. Although this case provides greater flexibility, it certainly introduces computational overhead within the Marvel server and concerns regarding the reliable processing of captured events (when, for example, the server is busy serving clients).

In both cases, filter specifications are stored in the filter repository, either in the form of READY language source files or as compiled Java class files. AMOs keep information on the events that they can generate and handles to the filter specifications that will be used to produce these events. Every event generated by an AMO is first stored in a local log and then forwarded to the event services unit for further distribution to clients that have registered as consumers of that particular event type. As a further optimization, AMOs can send the generated events to the event services unit only if event notification is activated for that particular event type, and there is a registered listener for that event.

### 2.4. Marvel Event Services

The Marvel Event Service unit is the core of the event architecture. It consists of three subsystems: The Event Broker, the Event Management component and the Consumer Notification component. These subsystems are organized around an inter-component communication channel to share event information with each other. The following sections discuss in detail the functionality of each subsystem.

#### 2.4.1. Event Broker

The Event Broker acts as an intermediary between the Marvel event producers and the Marvel event listeners. It maintains an event repository to store information about the available events and their sources. The event broker offers navigation and introspection services that allow clients to discover the events available from that server and their semantics. Every new event generated by an object must be registered with the event broker. Clients also have the capability to receive a *meta-event* that signals the availability of a new event type from the broker.

When a Marvel client contacts a Marvel server for the first time, it receives a bootstrap code that allows browsing through the available events. This code further has the capability to automatically register the client as an event consumer for some default events related to the operation of the particular Marvel server. For example, if the server contains objects representing customer profiles in a customer network management application, the client could be automatically registered to receive events representing outages in customer premises equipment. Automatic registration allows the server to convey immediately to the clients events that are important in the context of the server's intended management function. Automatic event registration can also occur when a client accesses a particular AMO. This capability is very useful in the sense that the clients are not required to find and register to individual event sources.

The bootstrap code may further contain a graphical navigation interface (e.g. a Java applet) that allows the user can examine the available events and subscribe to them through the event broker's registration API.

Once an event consumer has been registered, event notification follows the push model, that is, the producer AMOs generate events, which are then passed to the consumers using a callback function that the consumer has supplied when it first registered for the event. This information is stored in the consumer registry. The pull model can also be employed, for example, to perform data mining operations on event logs stored in the server.

#### 2.4.2. Event Notification

Upon receiving a new event from an AMO, the Consumer Notification subsystem scans the consumer registry to identify clients that have registered to receive the event. It then makes a callback to each registered consumer through a distributed computing environment, such as Java RMI (Remote Method Invocation) [SUN97] or CORBA. Depending on the consumer's interest, the Consumer Notification unit sends to the consumer, either the event information only, or the event information as well as the code to handle the event within the client environment. In the case that an automatic reaction is desired for the event type, this unit invokes a local *event reaction object*, in addition to notifying the registered consumers. Event reaction objects are mobile autonomous agents that can be uploaded and executed in other network components to help recover from the condition that produced the event. These objects reside in the *event reaction object repository*.

The motivation behind the Consumer Notification subsystem is to assign the event distribution functionality to a specialized component rather than requiring every event source to handle the distribution of events that it generates. This allows more design flexibility. The only trade-off is that the Consumer

Notification subsystem has to perform a search on the list of registered consumers for every event that occurs. However standard database indexing techniques can make this task a lightweight operation.

### *2.4.3. Managing the Event Service*

The Event Management subsystem implements the functionality necessary to manage the event services available through the broker. These include: suspending and resuming consumer notifications, enabling and disabling an event of a particular class, navigating through the consumer registry, adding and removing event consumers, changing the access control list of event clients, managing the repositories by adding and removing new event types, event filters and event reaction objects. The event service enforces a security model that requires appropriate authentication before invoking these primitives.

One event management feature that we found particularly useful is the capability to enable or disable the distribution of events with different granularities. For example, the manager has the capability to suspend distribution of events of a particular class regardless of their source, or allow event notification only from a particular group of objects, etc. When an event notification is disabled, Marvel objects still generate and record that event in their logs, however, they do not send it to the Consumer Notification unit.

### *2.5. Marvel Event Clients*

A Marvel event client is any entity that uses the event services of a Marvel server, either synchronously or asynchronously. It can be an applet, a standalone application, or any other object within a distributed computing environment. Figure 1 shows some types of event clients that may exist in a typical web management environment. The AMO Navigator assists the operator in navigating through the event repository and registering to receive the events of interest. It is further capable of visualizing received events, triggering an event reaction object to a take corrective action, or browse through event logs of individual Marvel objects (AMOs). The event management applet allows the operator to perform secure event administration.

## 3. Design and Implementation

This section describes the implementation of the upper two layers of the architecture. The lower two layers (event generation and event aggregation) are already provided through element management agents and READY. We follow an object-oriented methodology; however, our design process is different from classical design patterns such as the *model-view separation pattern* [LAR98]. The latter draws a separation between the model (which, in our case, corresponds to a Marvel event ) and the view (that corresponds to a Marvel client), and dictates that the model should not contain any code related to user interfaces. The view is responsible for displaying the model, maintaining the knowledge of its structure. Unfortunately, this is hardly the case in a distributed computing environment, in which different types of applications consume Marvel events. We cannot assume that every such application would have prior knowledge of the availability and type of events in a Marvel server. In fact, new Marvel event types can be constructed at run-time by a user. We adopted an approach that uses mobile code technology to download the necessary code to register and handle a particular event (e.g., visualize, or take an automatic corrective action). The code can be loaded beforehand (e.g. can be statically compiled into the client or loaded when the client subscribes to the event), or sent together with the message indicating the occurrence of the event. The advantage of this approach is that event consumers do not need to be aware of an event's semantics to convey the correct information to the user

In the current implementation, the event subsystem is written entirely in Java. In addition to benefits such as code portability, availability in web environments and the object-oriented features of the Java

language, we further take advantage of code mobility, built-in thread and exception handling support, dynamic class loading and execution, and the distributed computing facilities provided by Java RMI.

### 3.1. Event class inheritance

Marvel event types follow an object-oriented model to benefit from code reuse and inheritance. The class *MarvelEvent* is the root of the inheritance tree, and contains attributes common to all derived classes: an event identifier, the event class type, the event source (the object that produced it), a time-stamp, a time-to-live (after which the event can be discarded), and severity. Additional attributes may include information about the cause of event, the sequence of lower-level events that may have produced it, possible consequences of the event, and a collection of event reaction object handles. By subclassing from the root class it is possible to define new event types to suit a more specific management function. The model allows to populate the attributes of such event classes by performing computations on the attributes of the lower-level (component) events.

The object-oriented specification of the event model allows converting some standard event formats such as SNMP traps, CMIP event reports, CORBA event services, etc., into the Marvel event structure. All Marvel event classes overload a visualize() method that can be invoked remotely by an event consumer. The visualize() method displays the event according to the graphical capabilities of the client environment.

### 3.2. Event Visualization Model

When clients first connect to a Marvel server, they receive general information about object and event browsing services. We call this a *global* view of the server. When the client invokes a service of a particular Marvel object (e.g. the visualization service), it enters a local or *specific* view. At any time, a client may have one global and zero or more specific views open. The global view can notify the user of an event being produced from within the server regardless of its exact source (object). This feature is useful in attracting the attention of the manager to a particular object. To link a particular event with the global view, the client specifies an action that will be taken upon reception of the event. The default action supported in the Marvel system is an audio-visual notification of the user by expanding the object navigation applet to highlight the object that produced the event. In addition, it is possible to automatically link the occurrence of an event to retrieving the specific view of its source (the AMO that produced it) and/or executing the visualize() function of the event. This method is overloaded for different visual domains, e.g., text-based applications or Java-enabled web browsers. For every domain, the visualize() method first displays the common event information (e.g., type, source, time-stamp, etc.), and for each additional attribute, it calls the display() method of the attribute, exactly in the same way as Marvel objects (AMOs) present themselves in the client environment [ANE98d]. The only difference is that the visualize() method is called locally at the client for an event object, and remotely (at the server), for an AMO.

Web-based management poses unique problems in receiving and visualizing events because clients must obtain all event information from a web server. Simple interactions using CGI scripts and HTML pages allow only for very simple handling of asynchronous notifications from the server to the client. The use of a distributed computing environment through a Java applet allows much more flexibility. The applet in our environment communicates with the server through Java RMI. Figure 2 shows a screen shot of Event View window while browsing through the event log of a user profile object. The first part displays the common part of each event including the type, source, date and time and the severity. The second part shows the attributes of the events in the form of the html generated code by the visualize() method in an html viewer.

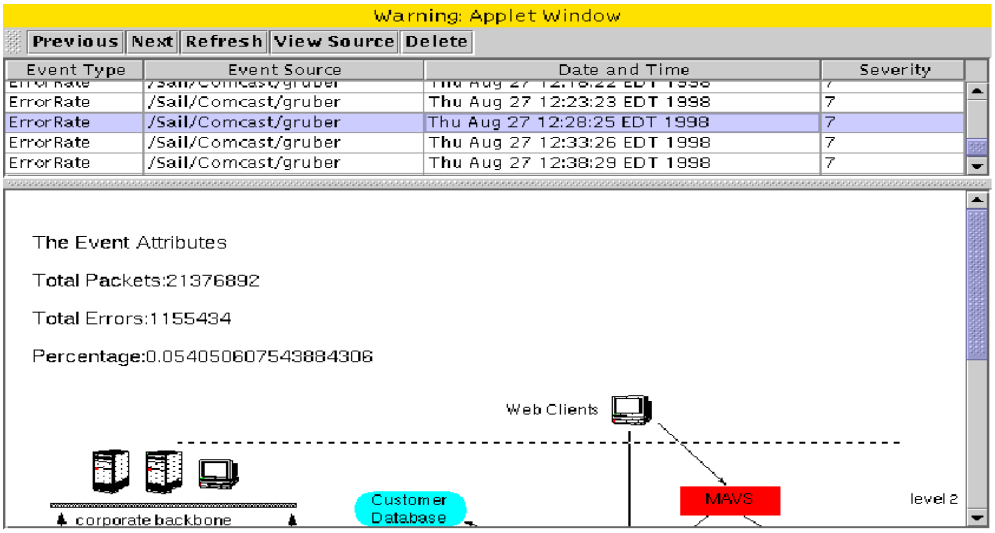


Figure 2. Event Visualization Example

### 3.3. Event Registration

Defining and using interfaces between event sources, event brokers, event listeners, event filters, etc., reduces the coupling between software components. It also makes the system more extensible with respect to adding new objects and services. This section discusses the *MarvelEventRegistration* interface for registering to receive:

- a specific type of event, regardless of its event source.
- one or more event types of a particular *AMO class*, for example, one or more types of events generated by a terminal server, etc.,
- all the events generated by an AMO,
- one or more events generated by a group of AMOs. The group can be defined by the client and passed as a parameter to the server. The server then stores this group for later references. Alternatively, the client can refer to a group already defined in a group directory. The group name is then expanded in the server into a list of objects (AMOs).

The client can specify through the interface several event delivery options, such as, reliable delivery, priority based delivery, secure delivery, and confirmed delivery. The client can also specify if it is interested in downloading server-side code to handle the event, or simply receive the serialized event object. The registration API further specifies offers methods for de-registering from listening to default or other explicitly registered events.

### 3.4. Use of Component Technology

Component technology is another area that network management can benefit from, as has been the case when CORBA and the Web were introduced a few years ago. Component technology allows easy customization of a network management application. Furthermore, component-based software allows code reuse, easy integration with other architectures and rapid prototyping. Our implementation follows the Java Beans model, where every element in the architecture is a “Bean” object that can be, integrated, composed and replaced with other beans to create customized components [BEA98], [BDK98]. Users can specify new event types, event filters, and even event broker components through a graphical interface. Component technology further facilitates the seamless integration of customer environments, database servers, even other applications that follow another component model using an interface gateway.

We use the Java *Infobus* [SUN98] to link together components that need to communicate asynchronously. One infobus is used within the Event Broker subsystem. Marvel event clients use a separate local infobus to receive events from the server and distribute them to local consumer components (e.g., Java applets). Instead of using introspection to learn about each other at run-time, beans components (1) agree on the data items to be exchanged, (2) implement the InfoBus interfaces for producing and consuming the data items and (3) follow the basic InfoBus communication protocol. In this way, a structured and efficient inter-component interaction scheme is accomplished between the cooperating components.

### 3.5. Use of Mobile Code Technology

Mobile code technology allows the offering of complex management services using a small client footprint. The capability to load new code can dynamically decentralize management functionality by delegating some tasks into clients. Marvel uses mobile code technology as the foundation for providing graphical user interfaces to lightweight clients. Further, the Marvel event subsystem uses this technology to activate event reaction objects, which are similar in functionality to mobile agents that travel to the source of a problem and apply corrective actions.

### 3.6. Repositories

Repositories are persistent collections of objects. The Event Services subsystem of a Marvel server contains two repositories: the consumer registry which keeps track of clients currently registered to receive events, and the event registry, which contains all available events to which clients can register as listeners. In addition, every Marvel server contains a filter repository, which acts as a database of event filter definitions (Figure 1). The event reaction object repository contains Java byte codes for event reaction objects. The latter can be also centralized. The advantage of this approach is that the repository can be decoupled from a particular Marvel server, allowing the definition of new objects or the improvement of existing ones from a single location. The trade-off is the additional delay needed to retrieve such an object from a centralized repository.

### 3.7. Scalability

Modern networks tend to grow at a very fast pace and introduce new services. Consequently, the scalability properties of the management system are of major concern. Traditional centralized management architectures are becoming insufficient handle this explosion of complexity.

Marvel addresses some scalability issues in the following ways:

(1) The Marvel framework creates a hierarchy of servers and distributes the task of event filtering and aggregation, as shown in Figure 3. AMOs in a Marvel server can register as event listeners with the event broker of another Marvel server, thus creating multiple levels of event aggregation. In addition, a READY process (shown at level 3 in Figure 3), can register with the event brokers of Marvel servers at level 2, aggregate their events with the events of other sources and pass the results to an AMO at level 3. These levels may represent views of the network of different granularities. In general, multiple event clients and event service components can be instantiated and distributed at different levels of the hierarchy for scalability purposes, reflecting the need for managing large networks using several levels of abstraction. A Marvel server in a multilevel configuration assumes a dual role, operating as a manager to Marvel servers at a lower level, and, as an agent for other servers at higher levels of the hierarchy.

(2) Events can be collected and aggregated by one or more external READY processes. In this case, only filtered events arrive to a manager (Marvel server. In addition to improving the scalability of the particular configuration, this approach has many other advantages: Managers are no longer required to continuously monitor managed elements for irregular conditions. This reduces the in-

tervention required from a network operator and provides a foundation for self-healing and self-managing networks.

(3) By offering management services over the web, heavyweight clients are no longer necessary to access management services. The latter are now available to a wider user audience

(4) Mobile code technology enhances scalability since clients and servers have a simpler software design and rely on incremental code downloads to expand their functionality, support thin clients, fault diagnostics, recovery, etc. Further, Mobile code technology helps significantly distribute the processing load among the network elements.

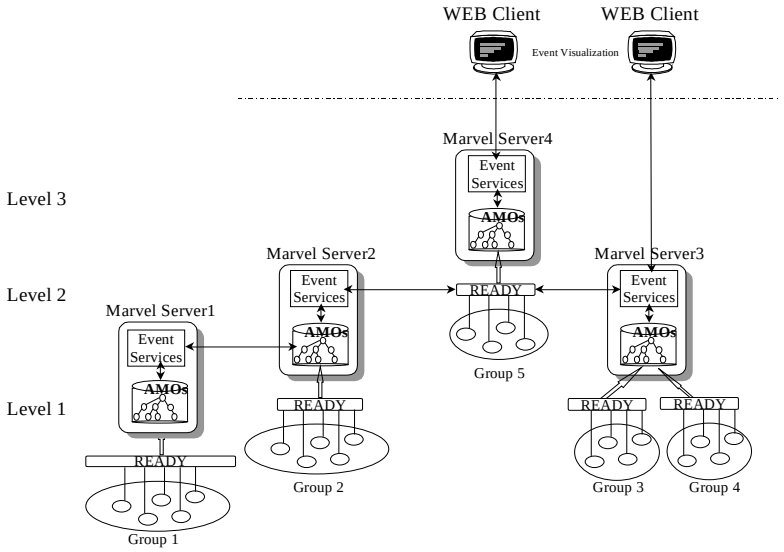


Figure 3. Example of event aggregation in multiple levels.

## 4. Events in the SAIL Home Access Network

The Marvel system is currently employed to manage the SAIL experimental home access network. SAIL (Speedy Asymmetric Internet Link) is an AT&T Labs home broadband access trial that brings a 10Mbps data channel to users homes through a downstream CATV channel, and uses a telephone 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. Home access networks have exactly the large scale properties that can benefit from the Marvel architecture to provide summarization of performance data, bulk control actions and aggregated event notifications.

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 AMOs. After each poll, the AMOs in the Marvel server check for error conditions such as a degradation in the user-perceived QoS due to high error rates at the physical layer. The object representing the user's profile can then produce an event indicating the problem.

Figure 3 shows a part of the event class inheritance tree used in the SAIL management system. The object-oriented event model of Marvel offers the flexibility of creating new event hierarchies and

modifying the existing ones by the network operator. We have defined some event aggregation rules both declaratively and explicitly using the events shown in Figure 4. One such rule, for example, generates a BadQoS event if ErrorInMedia and BufferOverflow events occur five times within an hour. In the SAIL management system, if such an aggregation rule is associated with a cable modem, the composite event signals a perceived degradation in the QoS of a user session, since there is one cable modem per user. If this rule is associated with a terminal server, the generated event signals the overall degradation in a CATV distribution tree. This example demonstrates that the Marvel event information model provides powerful event aggregations that complement the basic information aggregation model [ANE98c].

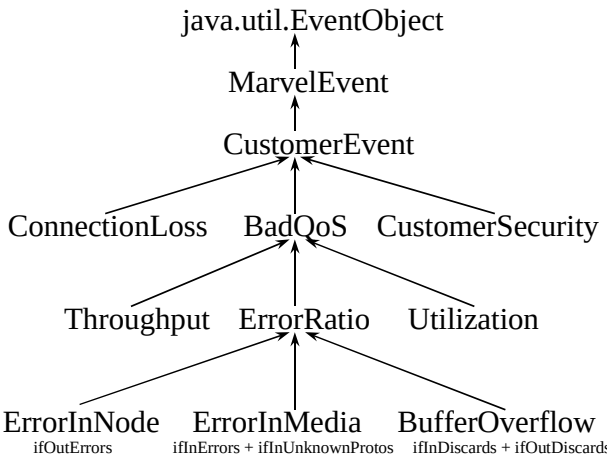

Figure 4. Example of the event inheritance tree for SAIL.

## 5. Related Work

Web-based network management has become a topic of increasing interest in the recent years and a detailed survey of Web technologies for network management can be found in [HON97]. At the same time, many researchers are investigating the benefits of intelligent/mobile agent technology as a more generic framework for performing distributed management tasks [MAG96, BAL97].

The need for event filtering and correlation has been identified in a number of previous works [YEM96], [XEM97], [GRU97], [KAL96]. However, to the best of our knowledge, little of no work has been performed for consuming and visualizing events in web-based management environments. [YEM96] describes a distributed event management architecture, InCharge, for the root cause analysis procedure. This procedure is based on an event model which serves as the knowledge-base for event abstractions and on a computation model which is a reasoning algorithm based on Codebook correlation [MAY97]. [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. This spreadsheet is capable of generating event reports by evaluating system or user defined predicates containing relational expressions. The X/Open event management service [XEM97] addresses the necessary data structures, components and interfaces for a distributed event management service. It specifies a comprehensive set of interfaces from registering to event servers, to managing event services. Our system implements many of these interfaces and some others while following an object-oriented model and using the transparency and abstraction facilities of a distributed computing environment.

Parallel to our work is the development of the Java Management API (JMAPI) toolkit from Sunsoft [SUN96]. The JMAPI event management service (EMS) provides the basic asynchronous event notification between managed objects and management applications. 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. The Java Message Service (JMS) provides enterprise messaging services for the Java applications [SUN98a]. However, it lacks features such as event repositories, event brokerage, and event management services. The Java Dynamic Management Kit [JDM98] has been introduced recently to provide the tools and services for creating autonomous and mobile SNMP agents. 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. It defines an object-oriented model for events, for event filters, for event consumers and for the bindings of event filters to event consumers. The event consumers register to the event producers and obtain the notifications (indications) through the HMMP protocol. The model allows to define the cost of delivery of events and provides a number of event delivery mechanisms. However, the event architecture does not provide event trading and event management services for the WBEM servers, yet leave the event producers and consumers coupled too much to each other.

## 6. Conclusions, Contributions, Future Directions

Web based management has gained significant popularity in the recent years as a cost-effective means of accessing complex network management services. In addition, the introduction of distributed object computing in network management allows building more scalable management systems. The Marvel project makes use of these technologies to offer computed views of management information representing different levels of abstraction of network information. The Marvel event services architecture operates in 4 layers: At the first layer, events are collected from their sources and routed to an external event aggregation process (the 2$^{nd}$ layer). Aggregated events are then sent to the appropriate objects within a Marvel server, where they are logged and further forwarded to an Event Services component (3$^{rd}$ layer). The latter then distributes them to event consumers (4$^{th}$ layer).

The Marvel event architecture uses mobile code technology to provide event clients with the necessary code to display an event or take a corrective action. Clients are not required to know in advance the types and semantics of the event services provided. Rather, by using the introspection capabilities of the event service, they can dynamically discover and register to receive particular events. In addition, the Marvel server offers an automated event reaction capability that can be triggered upon the occurrence of an event to dispatch a mobile agent that can diagnose and repair the fault. The architecture is complemented with en event management interface capable of configuring new event types, instantiating new aggregation rules, managing the flow of events to and from the server, etc. Moreover, by using public domain component technology (i.e. web clients, Java, RMI, InfoBus, etc.) we believe that our event aggregation architecture can be easily applied to other web-based management environments. We are currently applying the system to a management application for a home access network and demonstrated examples where event aggregation can be useful.

The integration of our event architecture with the READY system is currently under way. Integration with other event processing engines will be studied in the future. We are also in the process of investigating a model for designing event reaction objects and a methodology for specifying event recovery policies. The event services management interface will also be converted into a general purpose configuration utility using JNDI (the Java Naming and Directory Interface). In addition, we are studying the implementation of security services to enhance the Marvel framework with a capability-based access interface to object and event services.

## References


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

[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.

[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.

[BEA98] Sun Microsystems Corporation, “JavaBeans API”, http://java.sun.com/beans/index.html.

[BDK98] Sun Microsystems Corporation, “The JavaBeans Development Kit”, http://java.sun.com/beans/software/bdk_download.html

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

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

[JDM98] Sun Microsystems Corporation, “Java Dynamic Management Kit”, http://www.sun.com/software/java-dynamic/

[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.

[LAR98] Craig Larman “Applying UML and Patterns” Prentice Hall, 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.

[MAY97] A. Mayer, S Kliger, D. Ohsie and S. Yemini, “Event Modeling with the Model Language”, in Proceedings of the 1997 IEEE Integrated Management, San Diego, CA, 1997.

[GRU97] R. E. Gruber, B. Krishnamurthy, E. Panagos, “READY: A Notification Service for ATLAS Work Project”, AT&T Labs-Research Technical Memorandum, HA6163000-970905-07TM, September 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/.

[SUN98a] Sun Microsystems Corporation, “Java Message Service”, http://java.sun.com/products/jms/jms-092-spec.pdf.

[SUN98b] Sun Microsystems Corporation, “Java Dynamic Management Kit”, http://www.sun.com/software/java-dynamic/

[SUN98c] Sun Microsystems, “Java Infobus 1.1 specification”, http://java.sun.com/beans/infobus/spec/index.html

[THO98] J.P. Thompson, “Web-based Enterprise Management Architecture”, IEEE Communications Magazine, March 1998.

[XEM97] Systems Management; Event Management Service (XEMS), X/Open Document Number: P437, at http://www.opengroup.org/onlinepubs/8356299/toc.htm.

[YEM96] S. Yemini, E. Moses, Y. Yemini and D. Ohsie, “High Speed and Robust Event Correlation”, IEEE Communications Magazine, May 1996.