Open Architecture for Multi-Vendor Compatibility of Element Management and Status Monitoring

Kiran Babu
Design Engineer
Philips Broadband Networks Inc.

Abstract

This paper addresses the software architecture for an Element Management System which can be developed keeping protocol dependency, device specifics and management platform dependency transparent. The unique aspect of this architecture is that it provides the seamless integration of any kind of device from any manufacturer and any kind of communication at the Element Management System. This paper provides an Object Oriented Design technique of such an Element Management System which can be easily migrated to Distributed Object Oriented Network Management of the future.

Introduction

Element management and status monitoring products currently offered by manufacturers are exclusively for the devices and networks developed by those manufacturers. Desirable management systems will be flexible enough to monitor products from many manufacturers with various communications protocols. As networks become more complicated, element management software based on an open architecture makes the most sense.

History

Initially, cable monitoring systems merely verified that a device was operating. Gradually, monitoring systems evolved to report on the status of key operating parameters. A typical status monitoring application had four components for each class of device monitored: user interface, communication protocol, performance analysis, and database storage.

To monitor the operational status of their equipment, manufacturers developed proprietary status monitoring applications that used unique protocols to communicate with each type or class of device. User interfaces, performance analysis, and database storage components were also unique to each device class. Developing specialized status monitoring applications was not only time consuming, but also resulted in a limited, single-manufacturer monitoring system.

Proprietary communications protocols are the major obstacle to developing a flexible system that is capable of monitoring the devices of various manufacturers. This problem was identified not only in the cable industry, but also in the data communications and telecommunications industries.

Migration to Better Concepts

As networks grew in size and capability, simple status monitoring evolved into hierarchical network management. This network management hierarchy, shown in Figure 1, has four layers, each with a specific function.

The **element management layer** accepts information from all devices (or elements) in a network, translates the information into a form understood by all other layers, and passes it along to the network management layer. This layer consists of a third-party open platform running multiple manufacturer-specific applications.

The **network management layer** manages multiple element management layers.

The **service management layer** manages the services of multiple networks.

The **business management layer** manages the services layer below it and implements business strategies.

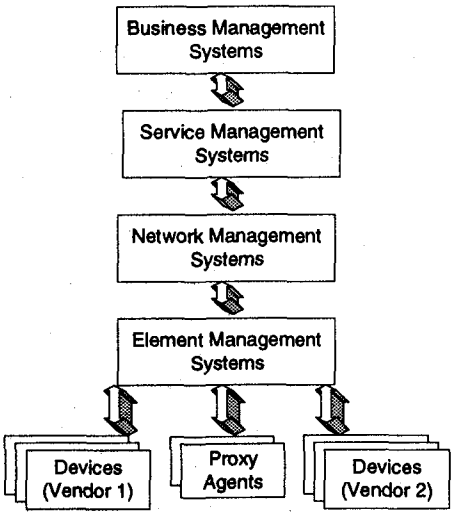

Figure 1.
Network management hierarchy has four layers

To achieve interoperability, the devices need to have a standard interface protocol instead of proprietary protocols. Specifications have been developed for a standard management protocol and a standard way of describing the management information of the devices.

Although new device development will be based on these standards, legacy devices still present the problems of multiple protocols. The element management layer plays a critical role in solving these problems. Proxy agents in the element management layer act as translators between the proprietary protocols used by the legacy devices and the standard management protocols used by the other layers in the hierarchy. Once this translation has occurred in the element management layer, the other layers need not address this issue.

This paper focuses on the element management layer, where the protocol translation and multi-vendor interoperability takes place. The three higher layers are generic for any organization and can be handled by third-party vendors.

Third-Party Open Platform for Element Management Applications

Third-party developers provided an open platform with basic features. An element management platform is open when any number of manufacturers can develop the applications specific to their devices under the same platform. Basic features include creating networks, triggering device-specific user interfaces, managing databases, and communicating with agents through standard protocols.

To manage a manufacturer's device from this open platform, a separate interface called a manufacturer's specific application must be created. These manufacturer-specific applications should be integrated with the platform's core software using application protocol interfaces and may include such things as device-specific user interfaces, databases, and performance analyses. This integration of software will be transparent to the user.

The following figure shows the relationship between the open management platform and a manufacturer-specific application.

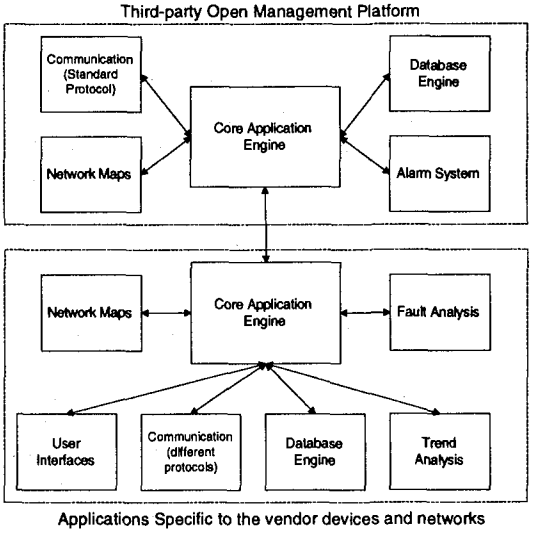

Figure 2.
Open platform supports manufacturer-specific applications

Problems and Solutions

Most third-party open element management platforms assume that the manufacturer-specific applications use the basic features provided. This would allow manufacturers to concentrate on the portions specific to their devices. Most manufacturer-specific applications, however, bypass the third-party open platform and are developed from scratch for the following reasons.

- **Problem:** No open management platform in the market supports all of the standard and proprietary communications protocols. The open management platform may provide communication support for one standard protocol, but there are many standards (for example, SNMP, CMIP, CORBA, and TL1) and more being developed.

  Solution: This problem can be resolved if a third-party platform can be customized to handle any communication protocol.

- **Problem:** Network map creation is not standard, and different management applications use different methods. Most of the open management platforms support a geographical organization of devices; however, there is no standard defined in this area.

  Solution: It is possible to define standards for network map creation common to cable, data communication, and telecommunication networks. For example, headend or central office devices can be grouped in racks and shelves and field devices can be grouped geographically in different regions.

- **Problem:** Open management platforms provide limited support for device-specific user interfaces because information and graphical representation of any type of device is unique. Although third-party open management platforms can provide some support for vendors to describe and graphically represent their specific devices, device-specific user interfaces are often created from scratch.

  Solution: Third-party, open platforms can be customized to include any device from any manufacturer.

- **Problem:** Open platforms provide very limited support for fault analysis. Once the manufacturer-specific application receives a fault notification from the device, it performs a root cause analysis using device dependency information to track the fault. The open platform applications do not recognize device dependencies.

  Solution: The open platform can be customized to include device-dependency information. This is possible only if the devices are maintained as generic objects with dependencies defined for their parameters.

- **Problem:** Not only do open platforms provide limited support for fault analysis among products from the same manufacturer, but they also cannot track faults among products from different manufacturers. Each manufacturer-specific application is maintained as a separate entity in open management platforms. A device managed by one application may be dependent upon a device managed by another application. These dependencies must be communicated between applications to track trends and resolve faults.

  Solution: Open platforms must be expanded to keep track of the dependencies of the parameters within a device, between different device types from the same manufacturer, and between devices from multiple manufacturers. This would allow the open platform to handle root cause fault analysis for all manufacturers' devices.

- **Problem:** Third-party open platforms have limited access to information about the devices being managed by manufacturer-specific applications. Therefore, database storage, reporting, trend analysis, and trouble ticketing are implemented by the manufacturer-specific application.

  Solution: If the device information is maintained by the open platform, it is possible to have a generic implementation

of reporting, trend analysis, and trouble ticketing functions.

Resolving these problems will lead to an element management system that takes maximum advantage of the features provided by an open platform. The complete element management system requires an open platform which can be customized to allow element management of devices from multiple manufacturers.

An Object-Oriented Solution

Managing many devices from various manufacturers can be accomplished when the element management system uses an object-oriented approach. An object-oriented approach uses a generic object created in the element management system to represent an actual device in the network. The user communicates with the object, and the object communicates with its counterpart in the network to get and receive information. For this to occur, the object must know about the device it represents, as shown in Figure 3.

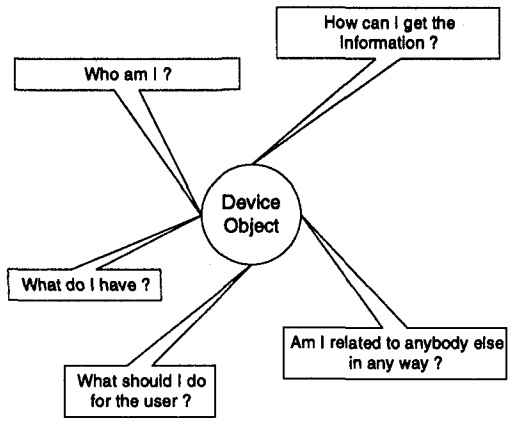

Figure 3.
The object must know about the device it represents

Each object is identified by its device class or type. Then, it is further distinguished from others in its class by key parameters. Some parameters are static, while others are dynamic. Dynamic parameters can be automatically updated by the application or manually updated by the user.

Thus, to manage a device, an object representing the actual device has to be created in the element management system. Through an interface for each device class, the user will interact with the object to initialize, update, monitor, detect and analyze faults, and analyze trends.

Initializing Objects

The user initializes the object by defining the device it represents. This definition specifies the type of device and values for key parameters which uniquely identify the device in its class. To do this, the object has to know what interface to display and the key parameters that require user input. Once the object gets enough information from the user to identify the actual device it represents, the object can communicate with the actual device to get values for additional parameters.

Updating Parameters

The user may want to change some device parameters. To allow such changes, the object must know which interface to display and which parameter values the user can change. The user communicates the changes to the object, and the object then communicates them to the actual device.

Monitoring Parameters

An object must know which parameters to monitor and their acceptable ranges. In addition, it must know which user interface displays the parameter values. Then, the object gets the parameter values from the actual device and shows them to the user.

Detecting Faults

If a parameter value falls outside of the acceptable range, the user is notified of a *critical condition* by either the actual device or the object. The user defines the acceptable range for the object, and the object communicates the range to the actual device.

Objects regularly poll actual devices to verify that monitored parameters fall within the acceptable range. If an actual device or its

agent has intelligence to check the parameters being monitored, it can detect critical conditions and notify the element management system directly.

Analyzing Faults

In a network, a fault in one device may depend on a fault in another device. This may be because both devices are on the same signal path. For example, an abnormal parameter value in device A may depend on an abnormal parameter value in device B. Similarly, device B may be dependent upon device C, and so on. These dependencies may be within the parameters of a device, within devices of the same class, and even within devices from different classes. The list of dependencies in a network is called the inference engine. Using the object, an inference engine can be defined for each device in the network .

Individual objects can be formed into a network representing the actual network being monitored. If an object has an inference engine, it knows the other objects in the network on which it depends for a fault. Hence, a fault for an object can be analyzed and the root cause of the actual device's fault can be identified. This method of analyzing faults is generic, but the inference engine for each device is specific.

Analyzing Trends

To analyze trends an object compares its performance to a history database of parameter values. Each object class can have templates defined by the user to support trend analysis. These templates can define the database storage, report, and real-time graphic structures for each device class. Trend analysis can be a separate process analyzing the performance of devices and networks. This process can be totally independent of the objects, since it is based on information already stored.

The catch is that the trend analysis process learns about the actual devices being managed from the database. To do this the database storage must be generic, so that reporting functions like trend analysis can learn about the devices from the database itself. This is feasible because objects are generic and learn about themselves from templates.

Communicating between Objects and Actual Devices

Communication between the object and the actual device it represents is not generic because the actual device may use any protocol, standard or proprietary. But a generic approach can be developed to a certain level by using specific protocol handlers, also called communication handlers.

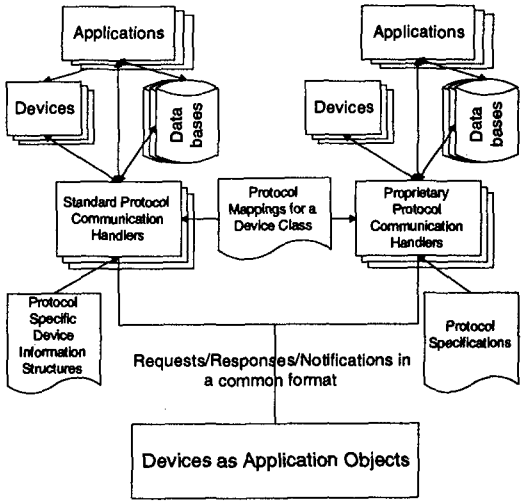

Figure 4.
Specific protocol handlers aid in communication between objects and actual devices

A device with a standard protocol maintains the device information in a certain format. A device with proprietary protocols also maintains the device information in a certain format. A communication handler for each protocol extracts the device information structure and maintains it in a common format. The communication handlers for proprietary protocols should have protocol specifications. A common protocol should be defined between objects and communication handlers.

Typical command structures between the objects and communication handlers to get or set a parameter value in the actual device or collect responses and notifications are shown below.

<Set/Get><DeviceClass><Key Parameters to Identify Device Instance><Request Parameter ID>[Value]

<Response/Notification><DeviceClass><Key Parameters to Identify Device Instance><Request Parameter ID><Value>

Communication handlers will map the requests to protocol-specific requests and handle the communications for each object. Each device class has to be associated with a corresponding communication handler.

The Total Solution

Adding a new device class to the element management system requires that device management information, dependencies, user interfaces, and communication be customized.

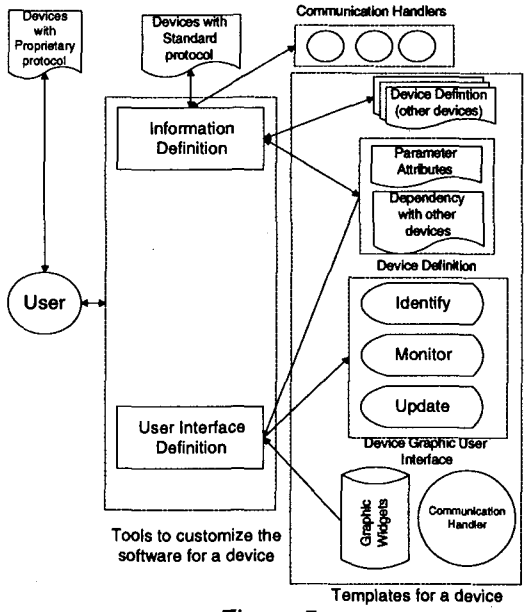

Figure 5.
Customizing to add a new device class

Figure 5 shows how to customize using tools in the element management system.

The information definition tool displays information about the other device classes, so the user can assign dependencies. These dependencies will be defined for each class. This tool should understand the device information specifications written for different standard protocols like SNMP, CMIP, CORBA, TL1, and SQL. Device information structures defined with proprietary protocols can be manually entered by the user or read from files with an intermediate standard defined for proprietary protocols. This tool can prepare a common structure for device definition, which describes the device class identification, protocol identification, and parameter attributes. In addition, the information definition tool should identify a specific protocol communication handler for a device class.

The user interface definition tool provides the user interface screen definitions for each device class. With this tool, the user defines graphic widgets for each parameter of the device class. The object associates parameters to these graphic widgets. The user interacts with the graphic widgets to monitor and update parameter values. User interfaces defined for a device class are shared by all objects in that class.

These tools help the user create a template for each device class. An object will be created from this template whenever the user adds a device to the network map. Specifics of the device class will be read from this template. Implementation of the object for any class will be almost generic and can be applied to any device from any vendor. In addition, whenever a object is added to the network, a corresponding communication handler is defined in the template and attached to the new object.

The graphical organization of the managed devices is a network map. Each device can be represented by a specific icon, and the icons can be arranged to represent the actual network topology. The network map should be created using passive elements like icons for each device, geographical maps, and racks and shelves with different configurations. These elements help the user create a graphical representation of the network with devices as groups.

Figure 6 shows a run-time version of an element management system.

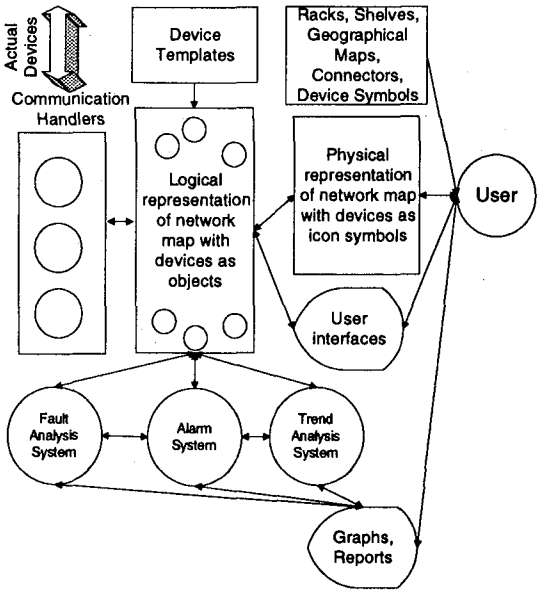

Figure 6.
A run-time element management system

Since the manufacturer-specific information can be customized and retrieved from templates, the whole element management system can be generic, and an open platform can accommodate multi-vendor interoperability.

This type of architecture will save users a significant amount of time, since adding new devices can be done easily with tools provided by the element management system.

However, a major effort is required to implement communication handlers for each protocol. This effort could be avoided if the industry agrees upon one standard management protocol.

Distributed Object-Oriented Management

The architecture proposed so far is based on object modeling of a device. For an object to communicate with its counterpart in the network, huge amounts of time, effort, and money are being devoted to implementing a protocol wrap-around. If the management system object is implemented as a client object and the corresponding server object is implemented as an actual device, a time-consuming effort can be avoided. This is possible with CORBA (Common Object Request Broker Architecture), and it is ideal for network management of massive networks like cable which contain many elements.

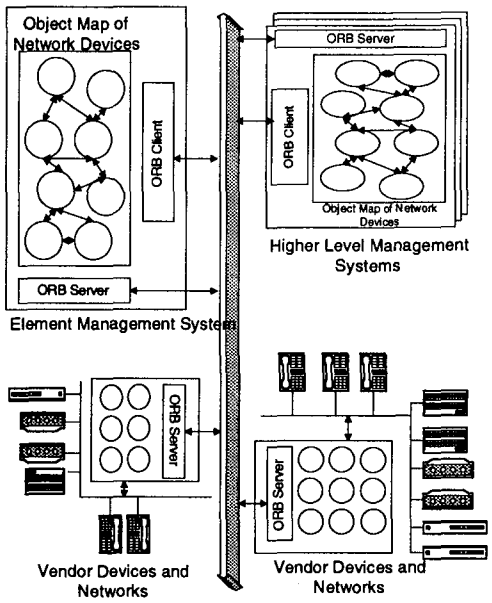

Figure 7.
Distributed object-oriented management using CORBA

With CORBA, the object exists in the device or agent itself. A mirror image is maintained at the management system. The communication wrap around is invisible to the developer. This approach has a significant impact because objects can be distributed to any level of management. As a result, any level of management can be applied at any level. If a distributed, object-oriented protocol is used, the architecture proposed in this paper will plug and play for any device a manufacturer can come up with.

Conclusion

Designing an element management system based on an open platform, as proposed in this paper, will allow operators to customize the system to accommodate any new device from one or multiple manufacturers. A management system based on this open platform architecture allows the operator to add new devices without developing new device-specific applications. Instead, the operator takes advantage of the open platform and customizes it to incorporate management information, fault patterns, user interfaces, and communications protocols of the new devices.