HEWLETT-PACKARD
JOURNAL

October 1996

HEWLETT-PACKARD

table of contents

October 1996,
Volume 47, Issue 5

Articles

A Platform for Building Integrated Telecommunications Network Management Applications

Telecommunications companies today are faced with rapid technological change, large heterogeneous environments, and a greater need to provide customers with products that ensure reliable, cost-effective network service. This means that these companies need a platform that has a visionary strategy that enables them to develop standards-compliant network management solutions for a continually changing environment.

by Prabha G. Chadayammuri

The telecommunications industry is going through phenomenal growth and change. This growth has made telecommunications networks essential to the daily activities of the enterprise and individuals. It has also given rise to the need for better ways to manage and maintain heterogeneous and multivendor networks.

Network management includes the operations, administration, maintenance, and provisioning (OAM&P) functions required to monitor, interpret, and control a network and the services it provides. When networks started to be used beyond the academic community and before deregulation and privatization of the telephone industry, there were fewer vendors, thus fewer multivendor management issues. Also, the rate of introduction of new network technologies was much slower. These conditions meant that network management could be ad hoc and vendor-specific. Today, issues such as multivendor networks and equipment, the need to automate certain network management tasks, and the rapid integration of new technologies have driven the need to standardize telecommunications network management.

Since the early 1980s, the standardization bodies have been developing and specifying a collection of standards for managing telecommunications networks. A portion of these standards, dealing with open systems, is contained in the X.7xx series of standards defined by the ITU-T (International Telecommunications Union—Telecommunications). Another series of standards, the M.3xxx series from ITU-T, defines a model known as the Telecommunications Management Network (TMN).[1]

TMN is based on the Open Systems Interconnection (OSI) systems management model, which is set of standards that define the rules for processing and transferring data over networks.[2] Such systems are called *open systems*. Although not intrinsically part of TMN, OSI systems management standards were developed jointly by the ISO and ITU standards bodies.

All of these standards, no matter how worthy, are simply collections of well-written guidelines without a platform and tools to build network management solutions. Choosing a network management platform is a critical strategic decision that has long-term implications. The development of large-scale telecommunications management systems requires a significant investment of resources. Solutions, once deployed, will be supported for many years.

For equipment manufacturers and systems integrators, the network management foundation must enable rapid development of applications that can differentiate and add value to their products. For telecommunications service providers, the network management foundation must enable rapid deployment of new services that improve competitiveness and new operations that increase efficiency.

HP OpenView products provide the platform and enabling technologies required for network management solutions for today's telecommunications environment.

HP OpenView DM

The HP OpenView Distributed Management (DM) platform is a software platform for designing portable, standards-based systems for telecommunications management (see Fig. 1). HP OpenView DM products are focused on meeting the reliability, performance, distribution, and standards needs of telecommunications equipment manufacturers, service providers, and system integrators. The HP OpenView DM platform offers the following features for developing TMN applications.

Standards Support. The HP OpenView DM products support protocol, object, and service specifications defined by ITU, OSI, X/Open®, the Internet Engineering Task Force (IETF) for SNMP (Simple Network Management Protocol), and the Network Management Forum (NMF).[3]

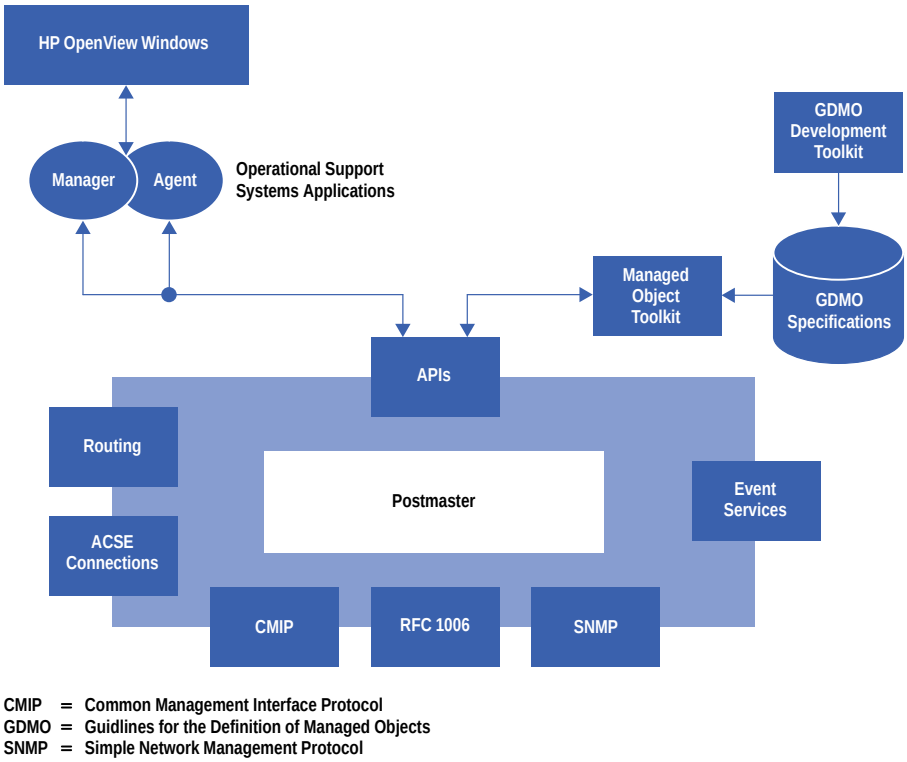


Fig. 1. *The main components of the HP OpenView Distributed Management Platform.*

There is also full support for network management protocols CMIP (Common Management Information Protocol), RFC 1006 (TCP/IP), and SNMP.[4,5]

Open Systems. The HP OpenView DM platform is built on an open systems architecture, enabling solutions to run on a variety of hardware platforms. Native support is implemented for HP 9000 workstations and servers running the HP-UX* operating system and Sun SPARC workstations running the Solaris and SunOS operating systems. Support for HP OpenView is also provided on other hardware and software platforms.

Postmaster. The postmaster serves as the integration point for management protocol stacks such as CMIP and SNMP, management APIs, and related facilities (e.g., routing, events, and association control). The postmaster provides distributed message routing and access to applications and services through standard management protocols. Finally, the postmaster reliably creates and manages associations (connections), maps objects to network addresses and protocol stacks, and routes requests from manager systems and responses from managed systems (agents).

Event Services. HP OpenView DM provides a set of services that management applications can use to control event and alarm messages from diverse network elements and systems. It includes a mediation service that collects, stores, filters, and extracts messages and an alarm management service that displays and correlates alarm messages and invokes external applications based on alarm data. Alarm management and event correlation services are described in ***Articles 3*** and ***4***, respectively.

HP Distributed Processing Environment (DPE). The HP DPE provides an Information Networking Architecture (INA) compliant platform for telecommunications services and operations systems. Trader services and an API framework simplify the development and deployment of distributed telecommunications applications. HP DPE is described in ***Article 2***.

Graphical User Interface. The HP OpenView windows graphical user interface (GUI) provides network operators and administrators with a consistent view of the managed environment and seamless integration of management functions, regardless of vendor or managed object type. HP OpenView windows provides a common interface that simplifies the development and use of management applications. Finally, the HP OpenView windows GUI is the key integration point for HP OpenView applications.

Modeling Toolset. The HP OpenView GDMO (Guidelines for the Definition of Managed Objects)[6] Modeling Toolset is an integrated suite of software tools for designing and analyzing objects used in network management applications. GDMO is an ISO standard that describes a consistent methodology for specifying managed objects in TMN applications.

The HP OpenView GDMO Modeling Toolset has a forms-based GUI that enables developers to create GDMO specifications and export them as ASCII files for use by the next application in the tool chain, the Managed Object Toolkit. The HP OpenView GDMO Modeling Toolset is described in ***Article 5***.

Managed Object Toolkit. The HP OpenView Managed Object Toolkit is a C++ code generator that accelerates the development of GDMO-based manager and agent applications (described below). The managed object toolkit includes an infrastructure that provides a collection of reusable objects that handle CMIS operations such as GET, SET, and ACTION.

Agent application development is improved because the Managed Object Toolkit takes the GDMO ASCII file and automatically converts the GDMO specification into an OSI-conformant, executable agent. The Managed Object Toolkit is described in *Article 6*.

TMN Applications and HP OpenView

HP OpenView products have been adopted by many prominent equipment manufacturers and telecommunications service providers to implement a variety of TMN solutions. Some of the areas in which TMN applications can be built upon the HP OpenView foundation include:

- Services management for broadband networks including Synchronous Optical Network (SONET), Synchronous Digital Hierarchy (SDH), Asynchronous Transfer Mode (ATM), and residential services such as video-on-demand
- Provisioning and monitoring applications for broadband networks
- Network monitoring for outsourced customer networks managed by telecommunications service providers
- Customer gateways into public networks for real-time monitoring and data management
- Integration with other management platforms for TMN compatibility and a single view from a multivendor environment
- Element management systems for new equipment and new data communications services.

The HP OpenView DM platform has traditionally supported the OSI systems management model to provide TMN solutions. However, in recent years the Common Object Request Broker Architecture (CORBA)[7] from the Object Management Group (OMG) has attracted interest as a general model for distributed application development.

The combination of the CORBA and OSI models is an extremely powerful solution for TMN application development. Thus, HP OpenView DM platform development is moving in that direction.

The rest of this article will discuss various aspects of the TMN architecture and the OSI model and their relationship to the existing OSI-based HP OpenView DM platform and the evolving CORBA-based platform.

TMN Architecture

Fig. 2 shows the business, service, network, and element management layers of the TMN model and the interaction between applications in these different management layers. The functionality of applications in each of these layers is defined in ITU-T Recommendation M.3010.[1]

Network Element Layer. Functionality at this layer is provided by the network elements (e.g., switches, multiplexers, repeaters, hubs, terminals, etc.). These functions include operations such as performance data collection, alarm collection, protocol conversion, and so on. Applications at this level are responsible for managing network elements.

Element Management Layer. Functions at this layer are responsible for managing a subset of network elements, performing as a gateway to network elements in the upper layers, and keeping statistical and historical information about network elements.

Network Management Layer. Network management functions are used to support TMN applications that require a global view of the network. Data for this global view is collected from data summarized by the network element management layer. This layer is also responsible for the technical provision of services requested by the service management layer.

Service Management Layer. This layer is responsible for managing the services provided to customers. It provides the point of contact with customers for all service transactions, including billing, quality-of-service (QoS) data, service contracts, and so on.

Business Management Layer. This layer contains functions that are responsible for the whole enterprise. These functions include goal setting and budgeting, product planning and definitions, and agreements between jurisdictions.

Operation Systems and the Manager/Agent Model. The operations systems shown in Fig. 2 are integrated telecommunication management applications that implement the network management functions in the TMN layers. The operations systems are based on an agent/manager model. This model resembles the client/server paradigm in which applications in the manager role are clients and applications acting as agents would be servers. The agent/ manager model is also called a managed system (agent) and managing system (manager) architecture in TMN terminology. The agent/manager model is based on using objects to model the system being managed. Each object can have attributes that represent its state or relationship with other objects, its specialized behaviors (called actions), and notifications it issues to signal some event. Thus, an object encompasses a device's behavior as well as its physical characteristics. An agent resides in an object and reports the object's status to the manager. The manager, equipped with the capability to have a global view of the network, manages the agents and handles the notifications from agents.

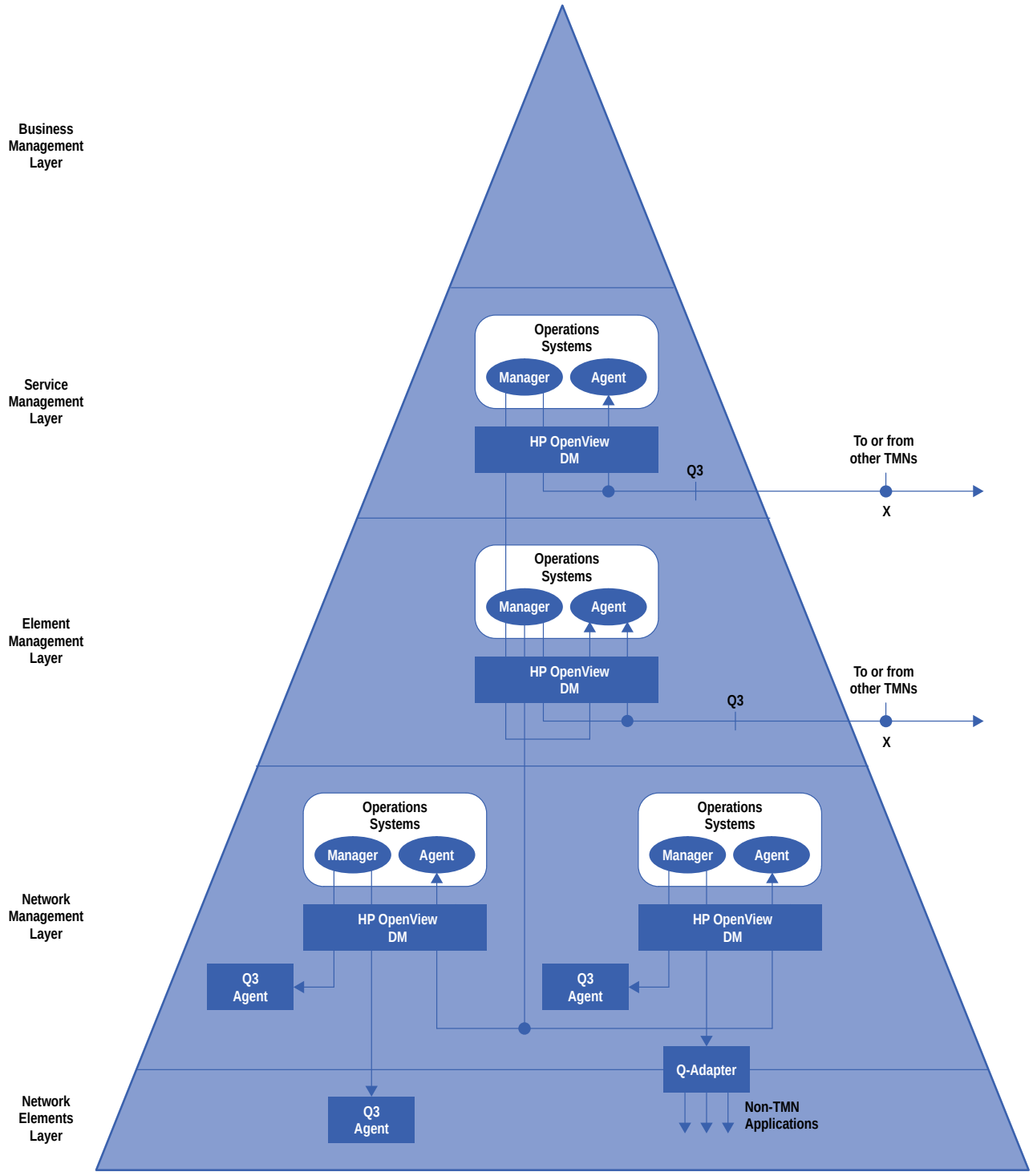


Fig. 2. *TMN architecture showing the network management layers and various TMN elements in each layer.*

Q3 Interfaces. Operations systems within and between TMN layers are required to use a set of standard interfaces called *Q3 interfaces* for the exchange of management information.[8,9] Q3 interfaces are responsible for connecting an operations system to a network element, an operations system to a Q adapter, an operations system to a mediation device, or two operations systems in the same TMN. Q3 specifications use the Common Management Information Service Element (CMISE) protocol[10] for management and the file transfer access and management (ftam) protocol for bulk transfer.

The standard way to convert a non-TMN function into a TMN function is called a Q adapter. Loosely stated, Q adaption is a translation between Q3 and the non-Q3 models at run time. Translation to a level less than Q3 requires a mediation device to raise the adaption to Q3 levels. The X reference points in Fig. 2 also perform an interface function. They provide an interface for communications with operations systems belonging to other TMNs or between TMN operations systems and non-TMN

operations systems on other TMNs that support TMN-like interfaces. Q3 interfaces are generally regarded as appropriate for the X reference point.

The HP OpenView DM platform supports the APIs and protocols necessary for TMN applications. The HP OpenView DM platform provides the Q3 interfaces via the X/Open management XOM/XMP APIs and the C++ classes generated by the Managed Object Toolkit described in ***Article 6***. Faster APIs like the BER (Basic Encoding Rules) Management Protocol (BMP) and the generic data type dictionary APIs are available on the platform.[11] Application developers can build OSI applications using the APIs or the Managed Object Toolkit. The Managed Object Toolkit generates a complete application skeleton that can be customized by adding user-defined behaviors.

The OSI Model

As mentioned earlier, TMN is based on the OSI model and the HP OpenView DM platform supports the OSI model. In OSI system management, managed object classes are defined using GDMO (Guidelines for the Definition of Managed Objects). A managed object class has its state and relationships with other objects represented in its attributes, which can be accessed by GET and SET methods. The managed object class definition can have complex interfaces called actions and can specify notifications, which are emitted signal events associated with the object.

Abstract Syntax Notation One (ASN.1),[12] a data definition language, is used to describe the syntax of management data exchanged between objects. Behavior templates are used to define the semantics of operations on attributes and objects and are commonly expressed in natural languages. As a result, there is no standard way of parsing the behavior templates. The agent developer is allowed to implement the behaviors appropriately.

A managed object can be created or deleted by external commands if allowed by the object's GDMO specification. GDMO allows multiple inheritance, in which a given object can inherit all the operations, notifications, and behaviors of other objects. References 13, 14, and 15 provide many of the widely used objects, attributes, and notifications used in network management.

When defining new objects, these standard definitions are expected to be reused whenever possible. This is one of the challenging aspects of OSI object modeling. The GDMO Modeling Toolset, available on the HP OpenView DM platform, makes this task much easier. ***Article 5*** describes the GDMO Modeling Toolset.

Management Interactions

Fig. 3 shows the seven-layer OSI reference model.[2] Each layer has a clearly defined role in the transfer of information over a network. For systems management, the application layer is of the greatest interest. Applications interoperate with each other using application service elements (ASEs), which are defined by the application layer. The Common Management Information Service Element (CMISE), the Remote Operations Service Element (ROSE), and the Association Control Service Element (ACSE) are the most important ASEs used for systems management. The protocols used to implement these service elements are also defined as part of the ISO standard specifications.

OSI systems management operates like the agent/manager model described above. An application issuing management operations and receiving notifications is said to be acting in the manager role, and an application performing management operations and emitting notifications on behalf of managed objects is said to be acting in the agent role. An open system is made up of managed objects and the various processes involved in processing and transferring information.

A manager is expected to establish an association with an agent using the ACSE before attempting any management interaction. If the association goes down, both parties can detect it. When the association is set up, the manager and the agent exchange management information about their respective capabilities, including authentication schemes, encoding schemes, maximum data sizes, multiple object selection capabilities, and so on. These capabilities are called functional units. The HP OpenView DM platform supports both direct user control over association management and the automatic connection management mode in which the user does not have to be directly involved in the association management.

Once the association is set up between a manager and an agent, management information can be exchanged. The manager is allowed to perform CREATE, DELETE, and ACTION operations on the managed objects and GET and SET operations on their attributes as defined in their GDMO specification. The agent performs the operations on the managed objects on behalf of the manager and sends replies back to the manager.

The managed objects emit notifications (events) specified in their GDMO specifications. Notifications usually signify something of interest happening at the object, like its creation, deletion, or attribute value change. The agents deliver the notifications either directly to the manager or indirectly through event forwarding discriminators, which are managed objects that filter events coming from agents. This filtering ensures that only events of interest are received by the manager. The OSI-based HP OpenView DM platform supports the most generic form of event discrimination available today.

Another important aspect of OSI system management is that it is based on an asynchronous message passing model, as are most other network management protocols in use today. All operations can be classified into four primitives (or types): requests, replies, confirms, and indicates. These primitives are used in the following way:

1. To perform an operation, a manager sends a request message.

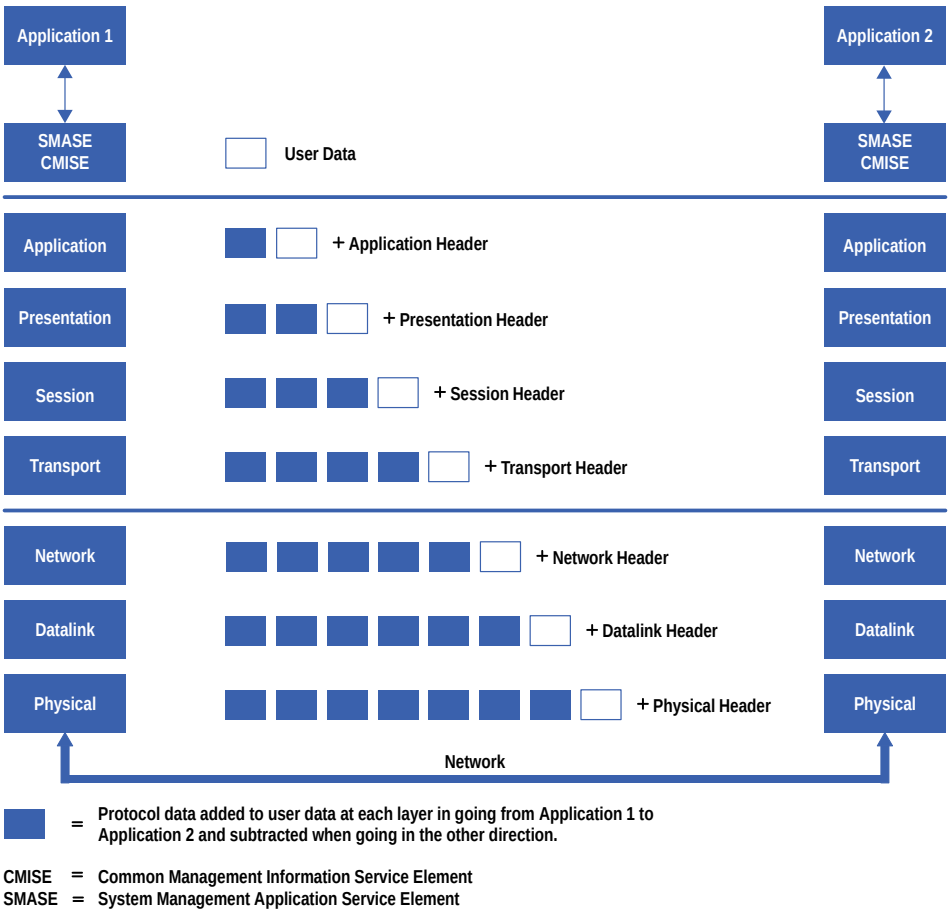


Fig. 3. *OSI stacks showing the significance of the OSI system management standards.*

2. When the message shows up at the agent, it is received as an indicate message.
3. Later, the agent may send a reply message.
4. The reply message is received at the manager as a confirm message.

The agent sends a reply message if the original request required a confirmation. The CMISE GET, CANCEL-GET, CREATE, and DELETE operations are always confirmed, whereas the SET, EVENT-REPORT, and ACTION operations can either be confirmed or unconfirmed. Request and reply messages are always directed outward from the application and indicate and confirm messages are always directed inward.

The Open Protocol Interface Architecture

Fig. 4 shows the HP OpenView DM postmaster with the attached API stacks, protocol stacks,[4,5] and intermediate stacks.[16,17,18] The postmaster is at the heart of the OSI-based HP OpenView DM platform. Applications bind to the postmaster processes running on different nodes. Postmasters on the different nodes coordinate interactions between applications bound to them.

The HP OpenView DM postmaster is built on an architecture known as the Open Protocol Interface, which is based on the OSI messaging model described above.

Messages flow into the postmaster either through the API stacks or through the protocol stacks. The processed messages that are sent out and need confirmations are kept on a sent queue awaiting confirmations. When the confirm messages come in they are matched with the corresponding request messages. This store-and-forward mechanism allows greater reliability in the message delivery. Flow control mechanisms are implemented to address congestion problems.

The X/Open management APIs (XOM/XMP) and the BER Management Protocol (BMP) are the API stacks. These and the CMIP, SNMP (RFC 1157), and RFC 1006 protocols are all supported by the OSI-based HP OpenView DM platform. This support, along with the requirements of association control and routing, provide the full complement of OSI conformance.

User-defined API stacks and protocol stacks can be added easily to the HP OpenView DM platform using the Open Protocol Interface architecture. New API and protocol stacks continue to be added by HP OpenView DM users. This flexibility allows easy integration of existing legacy applications into the management framework.

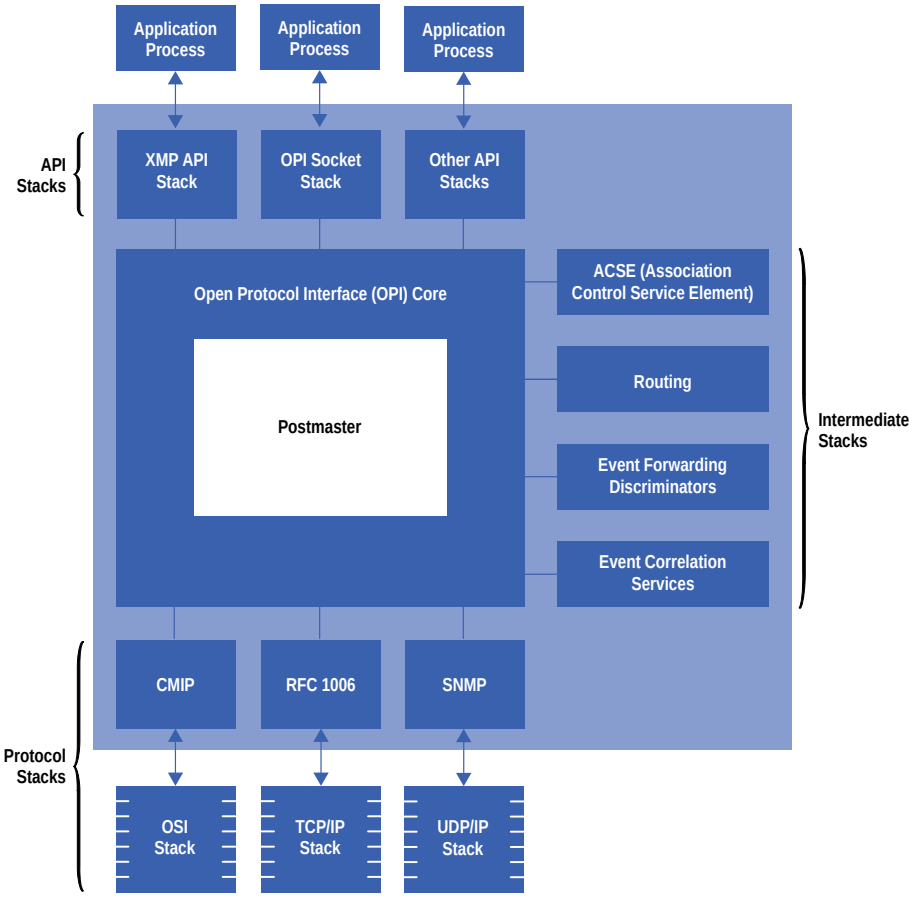


Fig. 4. *The HP OpenView DM postmaster showing the open protocol interface core and the attached API, protocol, and intermediate stacks.*

The intermediate stacks on the OSI-based HP OpenView DM platform are used to set up associations, determine routes, perform event forwarding discrimination, and so on. Each message is passed through its configured set of intermediate stacks for processing.

The intermediate stacks can also be used for data concentration or other similar purposes. This makes the Open Protocol Interface architecture ideal for building TMN mediation devices. For instance, the Event Correlation Service stack on the postmaster performs event correlation for events that pass through the stack.

Adding intermediate stacks is relatively trivial. This allows extreme flexibility in customizing the platform for specific needs. Consider, for instance, how user-defined security might be added to the platform. The Open Protocol Interface architecture presently allows security information (authentication token and authorization data) to be specified in each message. Today such information is regarded as opaque and is not interpreted by the stacks. If a user-defined intermediate security stack were added to the platform, the security information in the messages could be intercepted. The user stack could interpret the information and accept or reject the message, implementing user-specific behaviors.

The Open Protocol Interface development kit is separately available as an HP product.

Naming and Containment

To perform the operations and actions described above, there has to be a way of addressing the object instances. In OSI system management, each object instance has to have a unique name, known as the object's *distinguished name*. The uniqueness of the name is guaranteed by naming all objects with respect to a containing object or its parent instance. The only (virtual) object not contained in another is called the root. The relationship between the parent (superior) object and the child (subordinate) object is called containment.

Since every object instance (except root) is contained in its parent instance, an acyclic, hierarchical tree of object instances can be constructed. This is known as a *containment tree*. The idea of collecting objects based on containment is particularly useful in defining operations that apply to multiple objects. Such operations are called scoped operations in OSI systems management terminology. The CMISE GET, SET, DELETE, and ACTION operations can be scoped and result in the operations being applied to all objects that fall within the specified scope. Multiple object selection and actions make the OSI system management model far more powerful (and complex) than simpler models like SNMP.

The object instance name is required in all CMISE transactions. When automatic connection management is used, the HP OpenView DM postmaster uses a service called the *object registration service* to identify the target application for each request from its instance name. The object registration service allows users to configure the object location externally, enabling one to build highly scalable systems that provide complete location transparency.

Naming and containment are described in more detail in ***Article 6***.

CORBA-Based Application Development

So far, we have gone over the standards support and other features of the OSI-based HP OpenView DM Platform. Since most telecommunication resources today are modeled using a GDMO specified object, these applications tend to be in the element management layer of the Telecommunications Management Network.

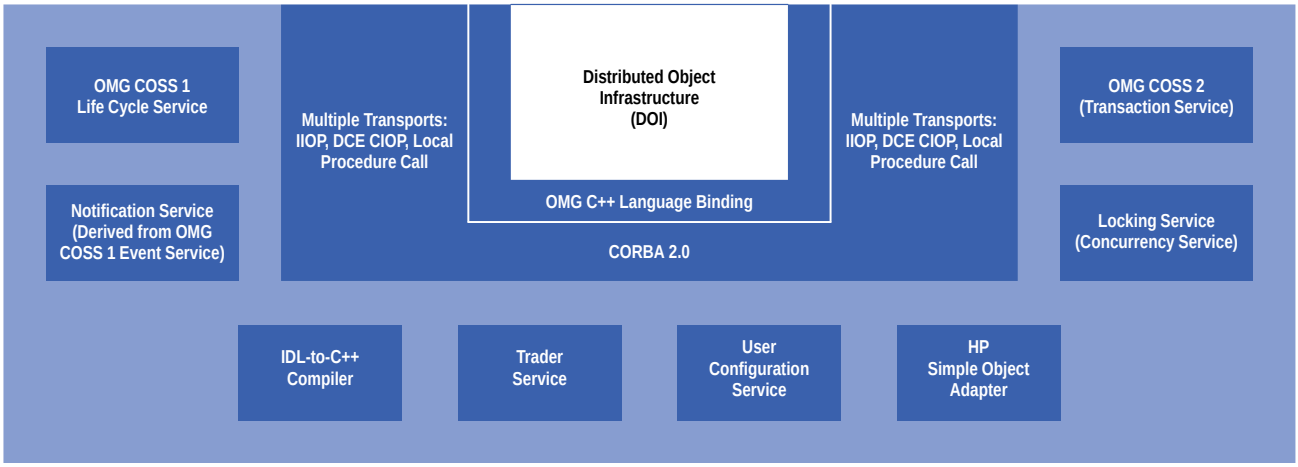
As we move up the TMN hierarchy (Fig. 1), the need for greater distribution, reliability, database access, and user interface access become obvious. TMN standards do not constrain the internal structure of applications. As a result, several nonstandard models are in use that need to be integrated into a single model to reduce costs.

The new HP OpenView telecom management platform addresses these specific issues with the use of the Common Object Request Broker Architecture (CORBA) from the Object Management Group (OMG). CORBA provides a highly scalable distributed object model. The OMG has a large industry participation and addresses all aspects of object modeling.

The new HP OpenView telecommunication management platform uses the HP ORBPlus distribution backplane for application interactions. HP ORBPlus supports the standard IIOP and DCE CIOP transports as well as a highly optimized local procedure call mechanism.

The OMG Common Object Service Specifications (COSS)[19] define a basic event service. Even though this service is implemented on the HP OpenView platform, it is not sufficiently robust for telecommunications management applications. HP OpenView, therefore, has developed a CORBA-based notification service,[20] which allows users to register with a notification manager for events filterable on multiple attributes.

The CORBA-based HP OpenView telecom platform also comes with the OMG naming and life cycle services, the OMG standard transaction service, and a location service, called the *trader service*. The collection of CORBA components and services, known as the HP OpenView distributed object infrastructure, is shown in Fig. 5.



OMG = Object Management Group
CORBA = Common Object Request Broker Architecture
COSS = Common Operational Support Services
IDL = Interface Definition Language

Fig. 5. *The HP OpenView distributed object infrastructure showing the various standard services supported.*

The notification service for the distributed object infrastructure provides the same value in the CORBA-based platform as the OSI event-forwarding discriminator does in the OSI-based platform. The event-forwarding discriminators implemented on the HP OpenView DM platform are more suitable for Q3 notifications.

The CORBA-based OpenView platform also provides a more scalable version of the relationship service known as the topology service and a database strategy based on the industry standard ODBC (Open Database Connectivity) interfaces. The topology service enables the developer to define relationships between topological entities, which are the abstract objects corresponding to the elements in a network. The ODBC interface is a transparency layer that the X/Open Consortium developed to allow access to relational databases. This API allows a great degree of independence from specific databases, with a trivial performance loss.

With the availability of a CORBA-based platform, application development is made considerably easier. Object modeling is done in IDL, the OMG's Interface Definition Language. IDL has the same capabilities as GDMO and ASN.1 combined.[21] Also, the CORBA-based platform allows operations on remote objects to be just as easy as operations on local objects, although local object access would be faster.

In the model shown in Fig. 6, CORBA-based applications access Q3 and other objects using adapters, or HP OpenView DM APIs. Q3 adapters can be generic adapters that provide a CMIP interface in IDL,[22] adapters that follow the mappings from the X/Open Joint Interdomain Management (JIDM) task force,[23] or class-specific adapters that expose modified JIDM interfaces.[24]

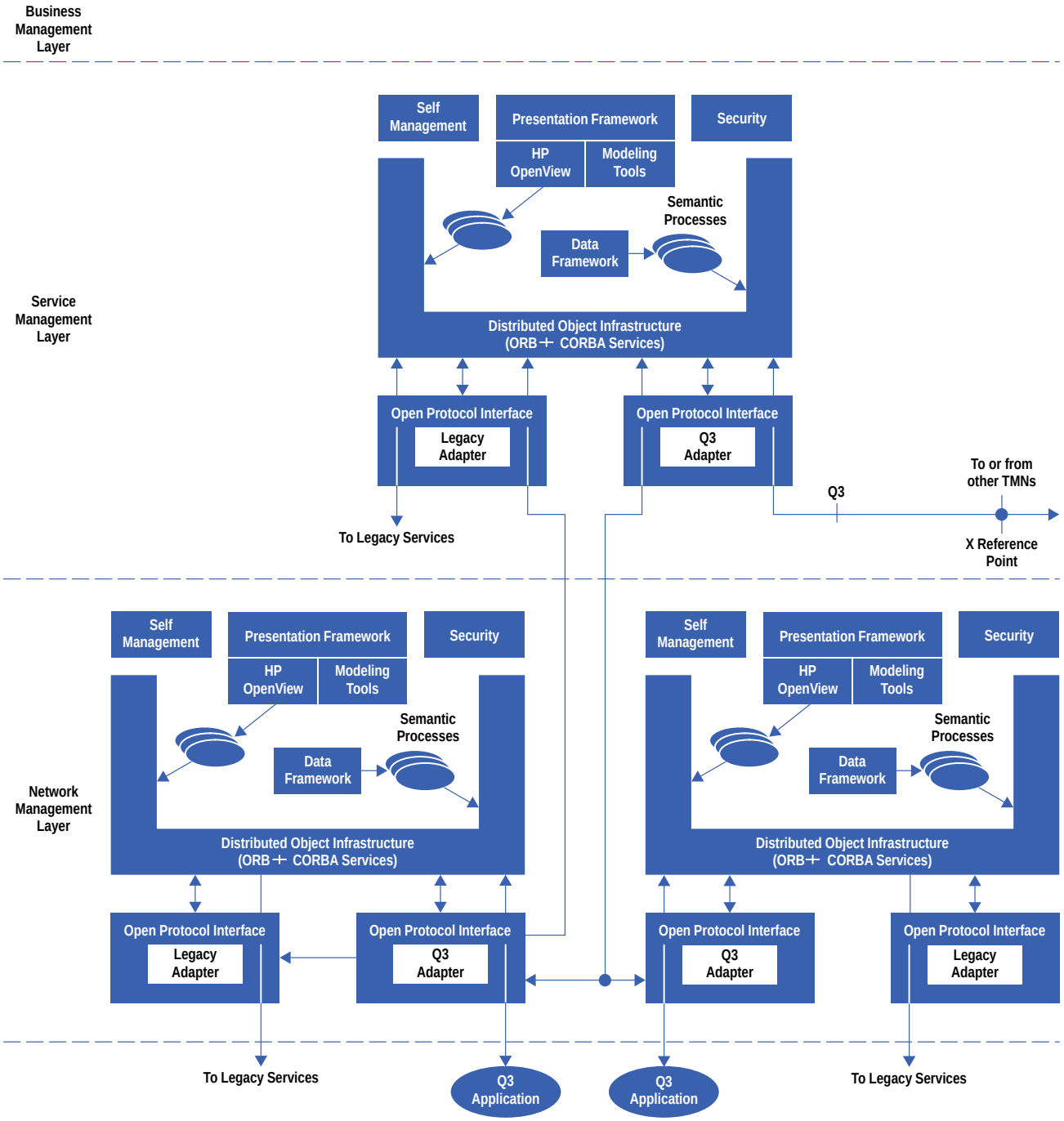


Fig. 6. *TMN applications built with CORBA accessing Q3 and other object models.*

The JIDM adapters can be static or dynamic. The static adapters are built for specific GDMO Management Information Bases (MIBs) and are expected to offer better performance. The dynamic adapters use the generic facilities of the CORBA architecture (DII and DSI), which are more flexible than the static interfaces. The JIDM activity produces mappings for CORBA-Q3 interaction and CORBA-SNMP interaction. The CORBA-based HP OpenView platform will supply adapters after

the standards in this area have stabilized. The Open Protocol Interface architecture discussed before is ideally suited for building Q3 and SNMP adapters to CORBA.

With the use of adapters, all other object models appear to be CORBA objects to the application developer. Applications use CORBA to gain distribution, standard language mappings, and common object services for portability and the topology services for data integration. The ODBC layer supports transparent access to multiple databases. In addition, a suite of enterprise management tools are available from other HP organizations that greatly enhance CORBA-based application development.

Summary

The new HP OpenView telecom platform combines the power of the CORBA model with the support for OSI management standards. As SNMP-based management gains acceptance in the telecom industry, the HP OpenView SNMP-based management platform will be integrated into the above model.

For pure Q3 access, developers today are encouraged to use the OSI-based HP OpenView DM platform. Q adapters, mediation devices, Q3 manager applications, and Q3 agents usually found in the TMN element management and network element layers fall into this group.

For highly scalable distributed applications requiring transaction processing, user interfaces, database access, and greater control over the quality of service usually found in the TMN network and service layers, developers should use the CORBA-based HP OpenView telecom platform.

See the ***Glossary*** for definitions of telecommunications terminology.

References

1. *Principles for a Telecommunications Management Network*, ITU-T Recommendation M.3010 (see also Recommendations M.3200 and M.3400), 1992.
2. *Open Systems Interconnection* (*Basic Reference Model*), ITU-T Recommendation X.200, 1994.
3. *Network Management Forum*, Forum 04, 1990.
4. *Simple Network Management Protocol*—RFC 1157, May 1990.
5. *Common Management Information Protocol Specification*, ITU-T Recommendation X.711, 1991.
6. *Guidelines for the Definition of Managed Objects*, ITU-T Recommendation X.722, 1992.
7. *The Common Object Request Broker—Architecture and Specification*, OMG Document 95-03-04, July 1995.
8. *Lower-Layer Protocol Profiles for the Q3 Interface*, ITU Recommendation Q.811, March 1993.
9. *Upper-layer Protocol Profiles for the Q3 Interface*, ITU Recommendation Q.812, March 1994.
10. *Common Management Information Service Definition*, ITU-T Recommendation X.710, 1991.
11. *BER Management Protocol*, HP OpenView 4.1 Users Manual, 1996.
12. *Specification of Abstract Syntax Notation One*, ITU-T Recommendation X.208, 1993.
13. *Definition of Management Information* , ITU-T Recommendation X.721, 1992.
14. *Generic Management Information*, ITU-T Recommendation X.723, 1993.
15. *Generic Information Model*, ITU-T Recommendation M.3100, 1992.
16. *Service Definition for Association Control Service Element*, ITU-T Recommendation X.217, 1992.
17. *Event Report Management Function*, ITU-T Recommendation X.734, 1993.
18. K.A. Harrison, *A Novel Approach to Event Correlation*, HP Laboratories, Bristol, England.
19. *Common Object Services Specification*, OMG Document 94-01-02, 1994.
20. *Event Notification Service*, COSS-1, OMG Document 94-01-02, 1994.
21. *Comparison of Object Models*, OMG Document 94-03-07, 1994.
22. E. Shen, M. Shan, and M. Robinson, *CMIP Interface—TC'95 Model*, HP Laboratories Research Report, 1995.
23. *Joint Interdomain Management Specification Translation*, X/Open Company, 1996.
24. *Class-adapter—Blanca Architecture Paper*, preliminary version, 1996.

Glossary

This glossary contains definitions of some of the telecommunication terminology and acronyms used in many of the telecommunications articles in this issue.

ACSE (Association Control Service Element). In the OSI model this is an application-layer protocol that is used to establish and terminate an association between applications on the same system or on different systems.

Agent/Manager Model. This model defines the basic architecture for network management of distributed systems. (This model is also called the managed system/managing system model.) The agent/manager system manages devices called *managed objects,* which represent a conceptual view of network resources that need to be monitored or controlled. The manager's role is to maintain a global view of the network and to control, coordinate, and monitor network activity. The manager also issues requests for operations to be performed by the agent and then receives notifications emitted by the managed objects and sent by the agent. The agent's role is to maintain its portion of the MIB, receive and execute requests sent from the manager, and send notifications to the manager when necessary (see Fig. 1).

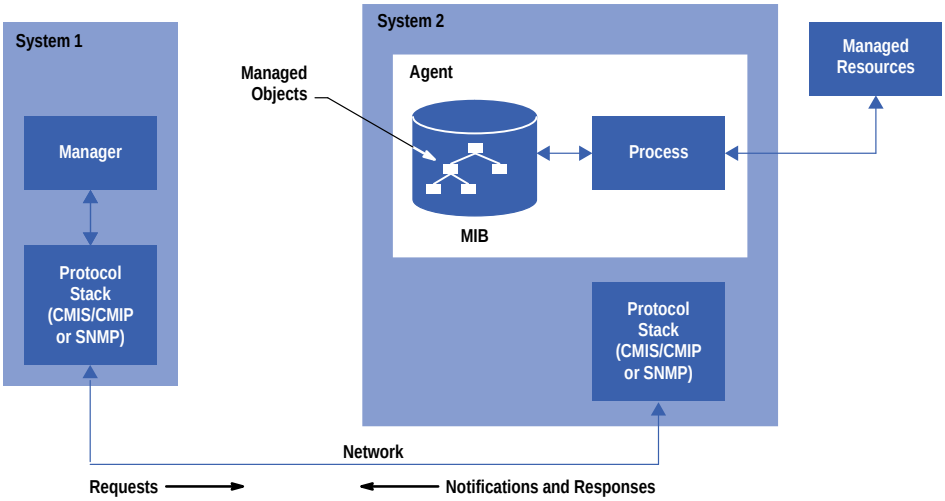


Fig. 1. *The agent/manager model.*

ASN.1 (Abstract Syntax Notation One, or ITU standard X.208). This is a description language used to define the data types exchanged between systems.

BER (Basic Encoding Rules). A method for encoding data in the OSI environment.

CMIP (Common Management Interface Protocol). This is half of the OSI's systems management protocol (the other half is CMIS). CMIP uses the agent/manager paradigm to communicate management information between systems. This protocol differs from SNMP in that it is more rigorous, is designed for open systems, and is an association-oriented protocol, requiring the two communicating CMIP processes to establish an association before sending any management messages. This association is governed by ROSE and ACSE. See ***Article 6*** for more about CMIP.

CMIS (Common Management Information Service). This is the part of the OSI systems management protocol that enables management applications to communicate in the OSI environment. CMIS offers a set of services that provide for management operation, retrieval of information, and notification of network events (see also CMIP). See ***Article 6*** for more about CMIS.

Containment. In an object-oriented hierarchy, containment defines the relationship between a parent object and a child object.

Contracts. In the context of the Distributed Processing Environment (DPE), contracts are the way in which objects in one building block (a software package containing several objects) describe their interfaces to objects in other building blocks. See ***Article 2*** for more about contracts and DPE.

CORBA (Common Object Request Broker Architecture). This is an implementation of the Object Management Group's specification of an object request broker. An object request broker provides the services that enable objects to make and receive requests and responses in an object-oriented distributed environment.

Distributed Processing Environment. This is a platform for managing and controlling distributed computing in a TMN network.

GDMO (Guidelines for the Definition of Managed Objects). These guidelines define how network objects and their behavior are specified. For example, GDMO can be used to specify how a certain system command (software object) should behave when executed. See ***Article 5*** for more about GDMO.

Managed Object. This is a conceptual view of a logical or physical resource that needs to be monitored and controlled to avoid network failure and performance degradation. A managed object is defined in terms of its attributes, operations that can be performed on it, notifications it may emit, and its relationship with other objects.

MIB (Management Information Base). This is a structured collection of managed object instances and their attributes. See ***Article 7*** for more about the MIB.

Mediation Device. This element of the TMN architecture is responsible for protocol conversion, information conversion and storage, data buffering, and filtering. This is probably the most vaguely defined element in TMN and its functions are sometimes implemented in a Q adapter.

Network Elements (NE). These elements represent the devices that make up a telecommunications network. It is assumed that an NE is "intelligent" enough to have the possibility of generating and transmitting some kind of information useful for network management (alarms, status, etc.). All NEs produce for external use some sort of internal alarms, both urgent and nonurgent. These alarms are representative of internal faults. Urgent alarms indicate a need for immediate maintenance. Network elements play the role of managed objects in the agent/manager model. The ***Article 1*** contains more about network elements.

OAM&P (Operation, Administration, Maintenance, and Provisioning). These are the functions required to solve the complex problem of providing telecommunications network management.

OMG (Object Management Group). This is a nonprofit international corporation made up of a team of dedicated computer industry professionals from different corporations working on the development of industry guidelines and object management specifications to provide a common framework for distributed application development.

Operations Systems (OS). These are the applications where network management takes place. They can be thought of as supervisory or control systems that receive a large amount of data from the network and provide for its elaboration and for the generation of data useful for management purposes. ***Article 1*** contains more about operations systems.

Q Adapter. This is a TMN element that is used to connect a TMN system to a non-TMN system. ***Article 1*** contains more about Q adapters.

Q3 Interfaces. These are a set of interfaces used within and between layers in the TMN architecture to exchange management information. Q3 interfaces are responsible for connecting an operations system to a network element, an operations system to a Q-adapter, an operations system to a mediation device, or two operations systems in the same TMN. ***Article 1*** contains more about Q3 interfaces.

ROSE (Remote Operation Service Element). This is a generic OSI service that allows applications to invoke request and reply interactions with applications on remote systems. ***Article 1*** contains more about ROSE.

SNMP (Simple Network Management Protocol). This is TCP/IP protocol that defines how to manage a network. SNMP uses the agent/manager model to monitor and administer the network. SNMP is based on a connectionless protocol, which requires no established connection between manager and agent before transmission.

Trader Service. This is a matchmaking service for clients and servers in a Distributed Processing Environment. A server registers its capabilities in the form of a contract with an entity called a trader, and when a client needs a capability in a certain contract type, it uses the trader service to find the server that has the particular capability. See ***Article 2*** for more.

Telecommunications Management Network (TMN). TMN, which is defined in ITU-T Recommendation M.3010, is a management communications concept that defines the relationships between basic network building blocks (network elements, different network protocols, and operations systems) in terms of standard interfaces. See ***Article 1*** for more about TMN.

XMP (X/Open Management Protocol). This protocol provides the TMN application developer with a C-language interface to the underlying CMIS/CMIP and SNMP protocol services. XMP APIs use XOM objects as parameters. See ***Article 6*** for more.

XOM (X/Open OSI Abstract Data Manipulation). A C-language interface designed for use with application-specific APIs that provide OSI services, such as X.400 and CMIS. XOM APIs provide functions for accessing managed objects and shield programmers from the complexities of the ASN.1 data types in the MIB. See ***Article 6*** for more.

Distributed Processing Environment: A Platform for Distributed Telecommunications Applications

Vendors developing applications for a heterogeneous, distributed environment need to be able to build towards a platform that integrates all the management and control functions of distributed computing into a unified software architecture that allows their applications to be available from any point in the network regardless of the system or geographic location.

by Frank Leong, Satya P. Mylavarabhata, Trong Nguyen, and Frank Quemada

The HP Distributed Processing Environment (DPE) provides infrastructure services that facilitate the rapid development, deployment, and management of distributed applications in the telecommunications arena. DPE is a key component of the Telecommunications Information Networking Architecture (TINA), an architecture for multimedia networks that emphasizes distribution and interoperability of telecommunications applications. TINA is an evolving architecture and is governed by the TINA Consortium (TINA-C), which is a project sponsored by 40 leading telecommunications and computing companies. The project's aim is to find a way to integrate all telecommunications management and control functions into a unified logical software architecture supported by a single distributed computing platform.

This paper describes the architecture and components that make up HP DPE, a product that is compatible with (and will evolve with) the TINA specifications.

INA, TINA, and DPE

HP DPE and TINA have a common root in the Information Networking Architecture (INA), which was originally developed at Bellcore. TINA's architecture specifies a distributed processing environment based on the original INA DPE specifications. HP DPE provides key infrastructure services for INA and TINA.

INA defines a methodology and framework for developing, providing, and maintaining highly distributed systems, characteristic of the next generation of communications environments. INA leverages and combines the efforts of multiple standards bodies, research organizations, development organizations, and consortia (e.g., TMN, OSCA, OSF/DCE, OMG CORBA, OSI/NMF, etc.). Fig. 1 shows the relationship between INA DPE and the TMN (Telecommunications Management Network) model. TMN is described in ***Article 1*** and the DPE services are described later in this article.

INA applications and services are deployed as software modules called *building blocks*. A building block is made up of several objects and can be installed and modified independently of other building blocks in the network. Building blocks interact with one another via interfaces called *contracts*. Contracts are the exposed interfaces of an object in that they are used for communication between building blocks. They are also backward compatible to ensure interoperability between software objects contained in multivendor building blocks. Contracts are subject to authentication and access control checks.

A building block can be a server or a client or both. A server must offer one or more contracts to allow clients to interface and make use of its services. In the DPE architecture (described below), applications are modeled as building blocks. The DPE itself is made up of server building blocks (e.g., contract trader, repository, etc.) which offer contract interfaces to application client building blocks.

The INA structure enables distributed software building blocks from multiple suppliers to interoperate. This distributed object computing results in faster software development since there is greater software reuse and modularity in design.

In summary, INA is a framework for interoperability, portability, and network resource management. The following goals have been established for INA:

- Rapid and flexible introduction of new services
- Reuse of software modules
- Use of general-purpose solutions
- Multivendor hardware and software solutions
- Independence of applications from the transport implementation technology

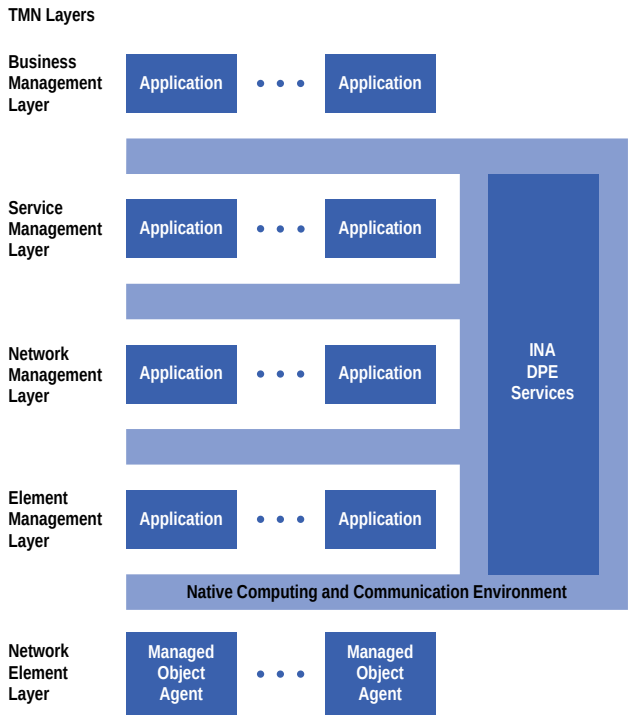


Fig. 1. *The INA DPE architecture applied to the Telecommunications Management Network (TMN).*

- Separate transport technologies from higher-level control and OAM&P (operation, administration, maintenance, and provisioning)
- Allowance of customer access to OAM&P services
- Seamless integration of services
- Network and element management.

DPE Architecture

Fig. 2 shows the components and services that make up the DPE architecture.

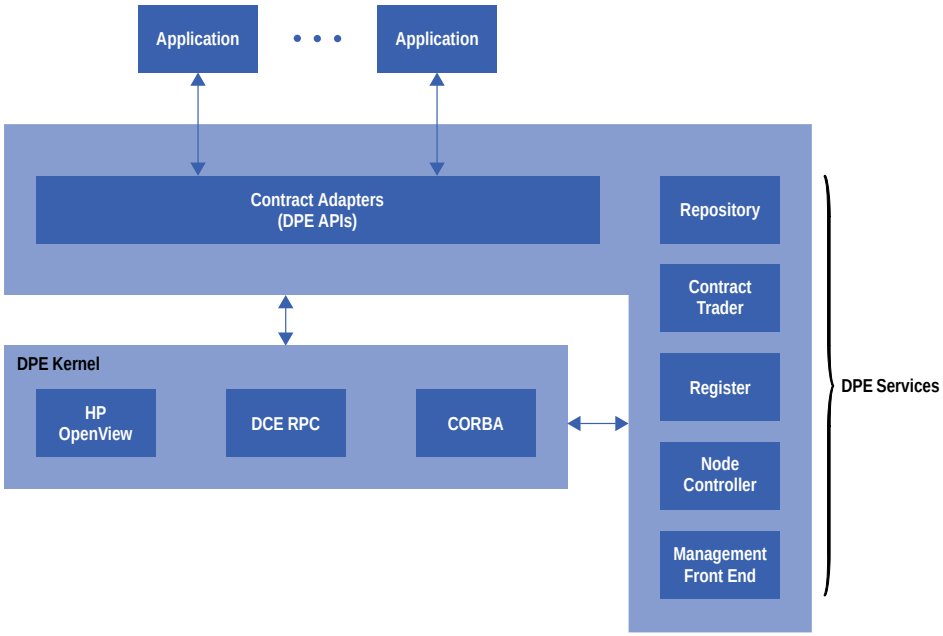


Fig. 2. *Components of the DPE architecture.*

DPE Kernel. The DPE kernel provides the foundation for building block interaction and execution services. To implement these services, the DPE kernel uses the services provided by the underlying native computing and communications environment, which include:

- DCE: threads, security, RPC, and IDL compiler
- CORBA: HP ORB+ with IIO and DCE CIO protocols and C-IDL compiler
- HP OpenView components: XMP API, pmd (postmaster daemon), orsd (object registration service), and ovead daemon (event sieve agent).

The DPE kernel is resident in every node of a distributed system. Building blocks and other DPE components at a node cannot access the DPE kernel at other nodes directly. Access to the DPE kernel services at a remote node is accomplished using the interprocess communication facilities of the native computing environment of the node.

Contract Adapter. A contract adapter is an application programming interface that provides all the transparencies required by a client or server building block. It also provides an API for accessing either application-level services or services provided by DPE. Contract adapters are kept as library modules which can be linked with building blocks before or during execution.

The inclusion of adapters as components of DPE implies that the components of DPE increase over time as new applications are deployed in a network. When a contract type is specified and registered for some application-level service, adapters for these contract types can be automatically generated and made a part of DPE.

DPE Services. Each DPE service is a building block and access to its functions is only through contracts offered by the DPE service. A node may have zero or more DPE services installed. Since access to a function provided by a DPE service is available only through a contract, a building block or a DPE service in a node can use the functions provided by a DPE service in a remote node. Thus, DPE services depend on the communication and execution services provided by the DPE kernel. References to contracts of some of the DPE services, such as the trader, can be passed to a building block when it is activated.

Although both DPE services and applications are built using the concepts of building blocks and contracts, there is a fundamental difference between the two. DPE services do not provide network resource management functions, nor do they provide telecommunications services to network customers. These functions are provided only by applications.

Fig. 3 shows the interactions among the DPE services shown in Fig. 2. An arrow directed from one service to another indicates that the source service provides services to the destination service.

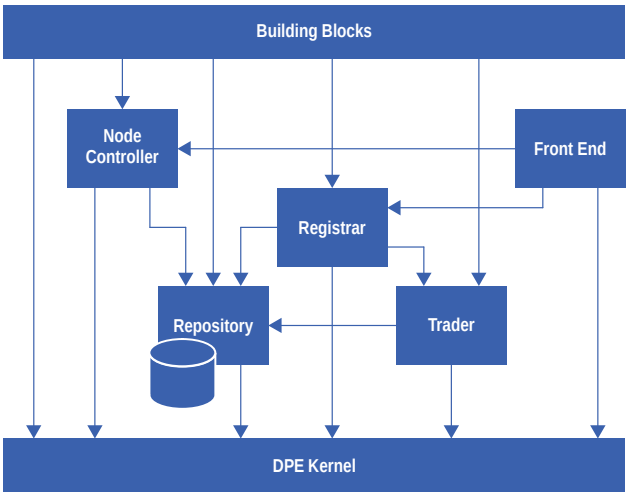


Fig. 3. *Interrelationships between different components in the DPE services.*

Contract Trader. This DPE service provides a discovery service for client and server building blocks. It is the key service for providing location transparency in a distributed network. When a building block offers a contract, information about this contract is conveyed to the DPE kernel. This information includes the name of the corresponding contract type and the value of the service attributes provided by the contract. DPE stores this information in the repository.

When a client wishes to invoke an operation defined in a specified contract type, it queries the DPE trader for one or more references to contracts that match the specified type and whose service attribute values satisfy a constraint expression supplied by the client. Regardless of where the server is physically located, the client can discover servers at run time, based on the latest contract information recorded in the repository database. The DPE trader provides two types of contract trading: attribute-based trading and resource-based trading.

The attribute-based form of contract discovery is based on the specified contract type and a constraint expression involving any number of the service attributes. The constraint expression used by HP DPE is modeled after the ANSAware 2.0

constraint language. This language supports relational operators on attributes and maximum, minimum, and logical operators. This provides a great deal of flexibility in how a client discovers a server.

An example of a constraint expression might be a request to find one or more print servers that can print in color, provide A4 size paper, and use PostScript™ fonts. The constraint language would express this request as: attribute_list = color, A4, postscript. If we need a certain capacity and speed for the printer, we might add a request for faster than six pages per minute: attribute_list > 6.

A resource-based form of contract discovery is an extension of attribute-based trading and is used by resource management applications. In resource management applications, it is typical to provide service over a domain of resources. This domain may be dynamic. An example would be a connection management application that is responsible for providing connection management services to all clients whose phone numbers (domain) begin with area code 408 and have the exchange number 447. This application may offer contracts over a domain that may vary in size depending on how many phone numbers are actually assigned (e.g., all the numbers following 447). This type of trading requires the client to supply a contract type name, a constraint expression, and the name of the resource. With this information the HP DPE trader can locate a server offering a contract of the appropriate type that satisfies not only the search constraint expression, but also the specified resources.

Repository Server. This DPE service maintains persistent information for the operation of DPE. It stores specifications of trading attributes, contracts, building blocks, and configuration information. The repository server provides operations for the creation, retrieval, update, and withdrawal of DPE-persistent objects. These reference objects are used to initialize, activate, deactivate, and withdraw contract and building block instances using a generic front-end administrative tool. This server is implemented using the ObjectStore 4.0 OODBMS from Object Design Inc.

The information stored in the repository can be used for several purposes. The DPE front end can traverse repository information to help application developers locate potential reusable attribute types, contract types, and building-block type specifications. It also provides type information that allows the DPE controller to check for valid operation parameter types at run time. The following three kinds of information are stored in the repository:

- Specification information. This consists of information contained in contract type specification templates and building-block type specification templates registered with the DPE repository.
- Configuration information. This consists of information contained in the building-block configuration templates, contract configuration templates, and node configuration templates registered with the DPE repository. This means that the repository contains information needed for managing building-block instantiating operations or startup operations.
- Trading information. This consists of information that supports trading operations, specifically contract types and contract instances.

Registrar. This DPE service provides registration and withdrawal services for the various templates used in the operability services, including specification templates, installation templates, and configuration templates. Its function is to parse and verify the correctness of the specification templates before invoking the registration operation of the repository server.

Node Controller. The node controller at each node provides activation, deactivation, monitor, and restart functions for building blocks configured in that node. It receives notifications when a building block is started and deactivated, and continuously monitors the "liveness" of all building blocks executing in the node. Since the implementation of these functions is dependent on the native computing environment's facilities, one instance of the node controller building block is required in each node.

Management Front End. HP DPE provides a graphical front end and a command line interface to DPE system administration, building-block management, repository browser functionality, and DPE shutdown and restart functions. This user interface offers a generic and uniform way of managing the whole DPE domain from any node. DPE objects present in the GUI are organized in a hierarchical structure similar to the renowned Smalltalk browser. This structure is organized as nodes, building block types and instances, and contract types and instances (see Fig. 4). The DPE front-end interface provides the following functions:

- Contract building-block type registration
- Activation, shutdown, and withdrawal of building-block instances
- Activation, shutdown, and withdrawal of contract instances
- Setup and modification of contract trading attributes
- Browser for DPE objects.

With the command line interface, routine DPE administrative tasks can be automated using shell script languages.

DPE Telecommunications Examples

This section provides two examples of the use of HP DPE in the design and deployment of telecommunications services and applications. The steps illustrated in these examples present a high-level view of the communications that occur. The actual designs are much more complex. Also, to reduce the complexity of the figures, three assumptions have been made:

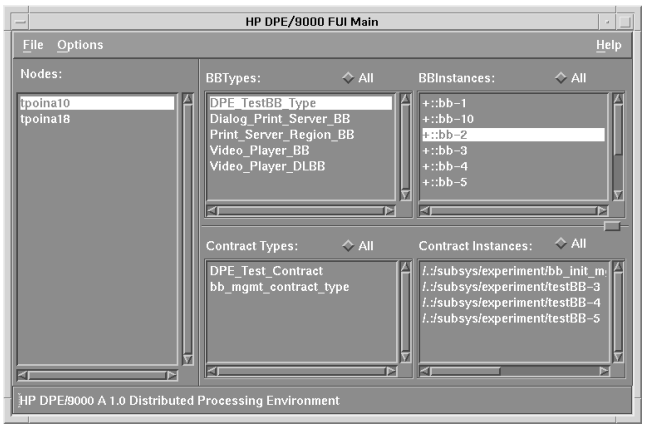


Fig. 4. *The DPE graphical user interface.*

- All interfaces that are used have already been registered with the DPE registrar, and binding information for each interface is available from the DPE repository.
- All communication with the DPE trading service is done via an RPC mechanism.
- Most applications will either trade at initialization time to obtain binding handles or simply use a well-known address to maximize throughput. Trading during execution will most likely be reserved for those occasions that dictate the need for dynamic binding. For illustrative purposes, however, the examples show trading occurring for each initial communication between any two modules.

Example 1: Permanent Virtual Circuit Service

The most basic connection service provided by broadband networks is a permanent virtual circuit (PVC) service. This service provides the capability of setting up a connection between two or more points with given bandwidth and quality-of-service (QoS) parameters. Typically PVCs are long-term connections used to interconnect LANs or provide long-term video service between distant points. Fig. 5 illustrates how a simple PVC service might be designed using an architecture based on INA. Each of the following steps corresponds to a number in Fig. 5.

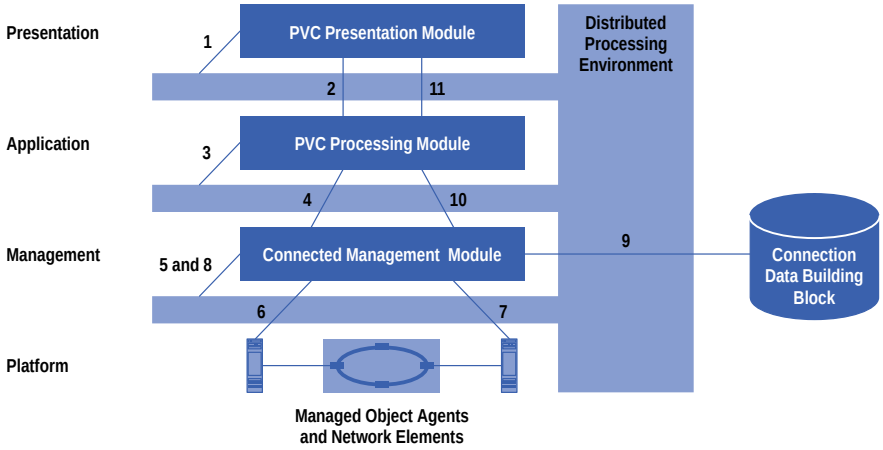


Fig. 5. *The architecture for a permanent virtual circuit service.*

1. The PVC presentation module consults with the DPE trading service for the location of the PVC processing module applications. This communication is done via an RPC (remote procedure call) interface.
2. The PVC presentation module provides the PVC processing module with the user input parameters that define the PVC being requested. This communication is done via an RPC interface.
3. The PVC processing module consults with the DPE trader to locate the connection management application server that controls the switch servicing the originating end of the PVC. This is done via an RPC interface.
4. The PVC processing module uses the DPE RPC mechanism to access the connection management application. If the connection requires more than one switch, the connection manager will trade for and bind to another connection manager to move the connection towards the termination point (this is not shown in Fig. 5).

5. The connection manager trades for the binding handle of the managed object agent that services the originating (and terminating if local) points. For performance reasons, in most designs this step is done at system initialization time.
6. The connection manager instructs the managed object agent to connect the originating end using the DPE system management protocol CMISE (Common Management Information Service Element).
7. The connection manager instructs the managed object agent to connect the terminating point using the CMISE protocol.
8. The connection manager uses RPC to request a binding handle from the connection data building block.
9. The connection manager requests the connection data building block to update its data store to reflect the addition of the new PVC connection. The communication is done via RPC.
10. The connection manager reports the establishment of a connection back to the PVC processing module via an RPC.
11. The PVC processing module returns the status of the connection establishment back to the PVC presentation module for display to the user.

Example 2: Switched Virtual Circuit Service

This example shows that the modularity and code reuse capability of the DPE architecture can be used to add new features. The switched virtual circuit implementation shown in Fig. 6 provides users with the capability to establish or reconfigure existing connection sessions at any time, much like voice telephony service. As shown in Fig. 6 the connection management, data building block, and managed object agents are all being reused. Only the top two modules need to be replaced with new code.

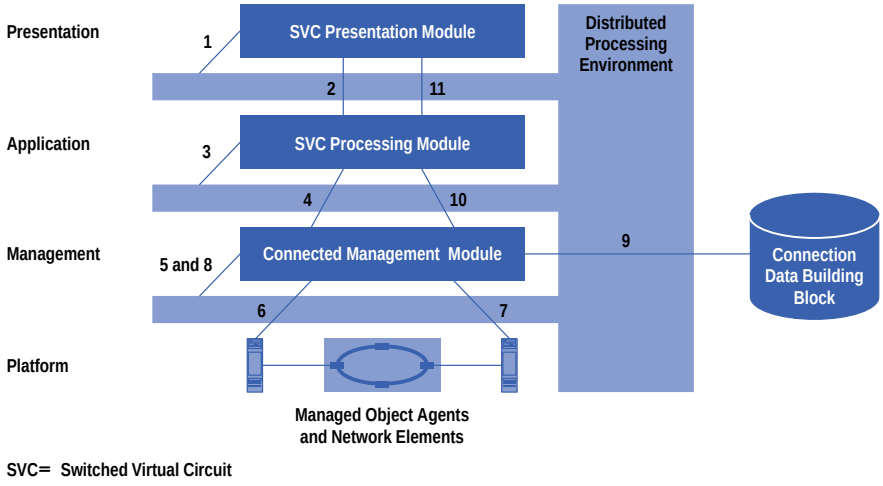


Fig. 6. *The architecture for a switched virtual circuit service.*

Summary

This paper has presented an overview of the HP DPE implementation. DPE plays a key role within the Telecommunications Information Networking Architecture (TINA). HP DPE offers a development environment to develop distribution transparency for both RPC-based and CMIP-based INA-compliant applications. This paper has also detailed the services provided by HP DPE and described the implementation of the contract trading servers and contract adapters, the key components providing distribution transparency.

Acknowledgments

The authors would like to acknowledge other members of the development and product team: Joel Fleck, Bruce Greenwood, Hai-Wen Liang, David Wathen, and Chris Liou.

HP OEMF: Alarm Management in Telecommunications Networks

This article explains the HP OpenView Element Management Framework concept, which is based on the HP OpenView Fault Management Platform (FMP) and complements the functionality of the FMP to provide an integrated network management solution. This article also explains the FMP, which facilitates efficient management of alarms in a telecommunications network, and the open APIs provided in the FMP, which allow seamless integration with other applications.

by Sujai Hajela

There has been an unprecedented growth in the telecommunications industry around the globe. The rapid evolution of new technologies, the offering of a broad spectrum of data services, and the need to have fast access to information are some of the factors that have contributed to a tremendous increase in the number of subscribers to telecom services. This has imposed great demands on the telecommunications networks of both public and private operators. To keep up with the demand, telecom operators are expanding their existing infrastructure at a hectic pace. Furthermore, deregulation of the telecommunications industry has led to the emergence of a number of private service providers, and this has created keen competition within the industry. A good quality of service at an economical price has become a key factor for service providers to increase their customer bases.

Telecommunications Management Network

Offering a high quality of telecom services and at the same time generating high revenues requires efficient management of telecommunications networks by the service providers. The Telecommunications Management Network (TMN) defines activities that aid in managing a telecommunications network. According to ITU-T Recommendation M.3010, a TMN is intended to support a wide variety of management areas including planning, installation, operations, administration, maintenance, and provisioning of telecommunications networks and services. The following five functional areas have been identified in TMN (ITU-T Recommendation M.3400):

- Fault management
- Configuration management
- Performance management
- Security management
- Accounting management.

Fig. 1 shows the TMN functional blocks and components. The TMN architecture consists of the functional architecture, the information architecture, and the physical architecture. The TMN functional architecture defines the following blocks:

- Operations systems function (OSF)
- Mediation function (MDF)
- Network element function (NEF)
- Workstation function (WSF)
- Q adapter function (QAF).

The TMN information architecture defines the information exchanged between these functional blocks.

The TMN physical architecture provides a means to transport and process information. The physical architecture is made up of the following types of physical components:

- Operations system (OS). Performs OSF.
- Mediation device (MD). Performs MDF.
- Q adapter (QA). Performs QAF, that is, connects network elements and operations systems with noncompatible interfaces to OSI Qx and Q3 interfaces.
- Data communications network (DCN). Performs data communications function (DCF), which is used by the TMN functional blocks to exchange information.

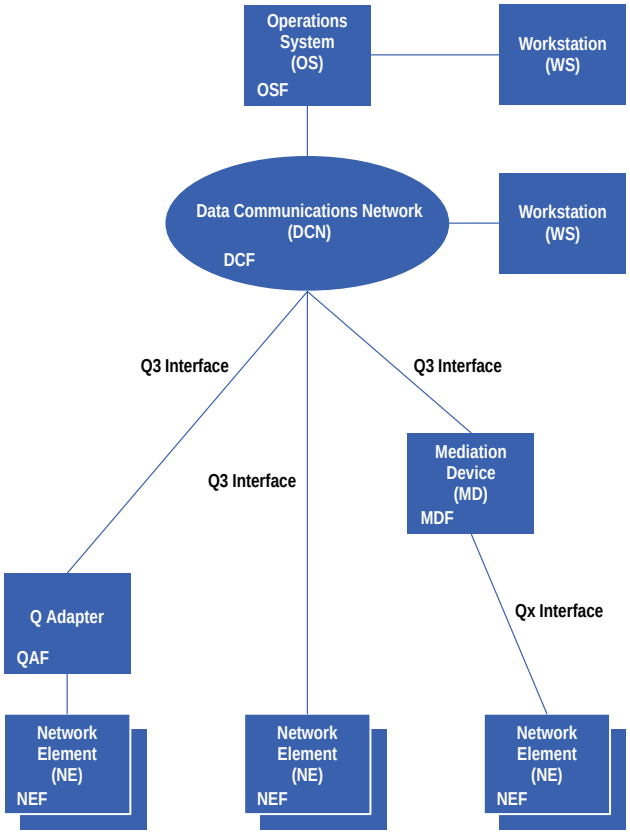


Fig. 1. *Telecommunications Management Network (TMN) functional blocks and components. (F = function, e.g., OSF = operations system function.)*

- Network element (NE). Performs NEF.
- Workstation (WS). Performs WSF.

OpenView Element Management Framework

The HP OpenView Element Management Framework (OEMF) aims to provide a set of management activities defined in ITU-T Recommendation M.3400 to facilitate efficient management of a telecommunications network. The functional areas covered within the OEMF are fault management (including trouble management), performance management, and other third-party applications to complement the existing set of applications under the OEMF umbrella—for example, configuration management and asset management.

OEMF is an open system that makes possible the detection, isolation, and correction of abnormal operation of the telecommunications network. OEMF consists of the HP OpenView Fault Management Platform (FMP) integrated with the Trouble Ticketing System provided by Remedy and the Performance Management System from Metrica. Other third-party applications for inventory, asset, and configuration management have also been integrated. Integration with test and measurement products like HP AcceSS7 further enhances the OEMF functionality.

Fig. 2 illustrates the physical architecture of the OEMF. OEMF has a distributed architecture in which different management activities can reside on different servers or on the same server. OEMF offers application availability, that is, if one of the management activities ceases to function, the operator can still execute the functionality provided by the other applications.

In the TMN hierarchy, OEMF resides between the network management level and the element management level (Fig. 3). It can manage the network elements directly or can be interfaced to an existing element manager to manage the network. Providing this flexibility to OEMF are a rich mediation service and APIs (application programming interfaces) for integrating with customer-specific data collection mechanisms.

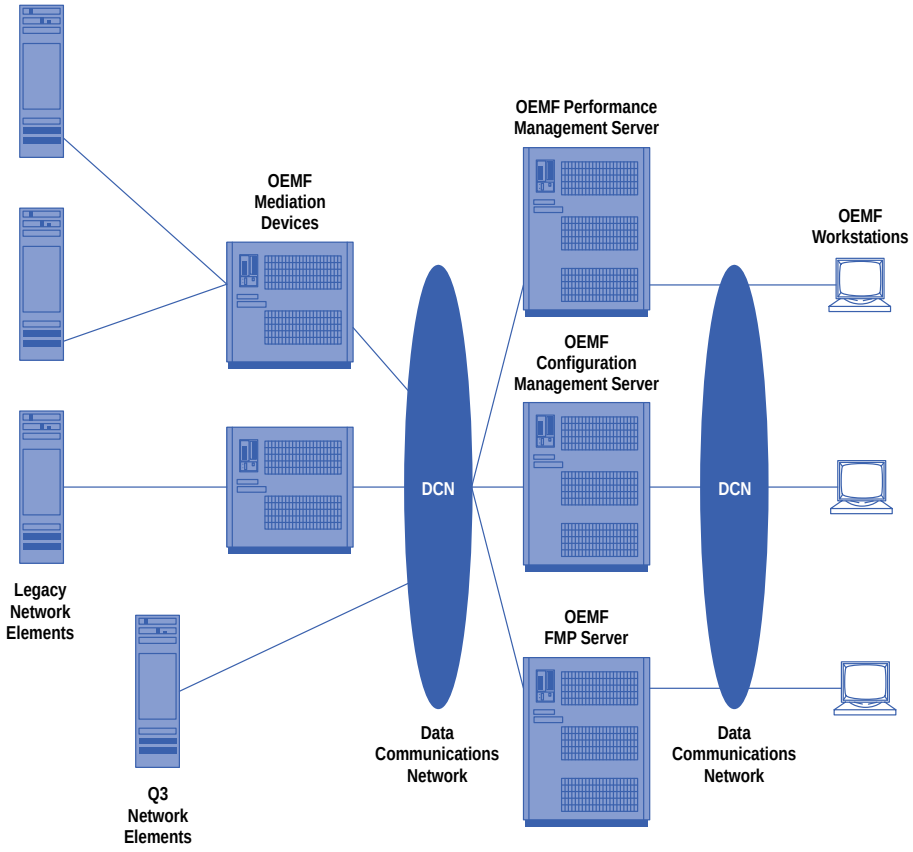


Fig. 2. *Physical architecture of the HP OpenView Element Management Framework (OEMF). FMP is the HP OpenView Fault Management Platform.*

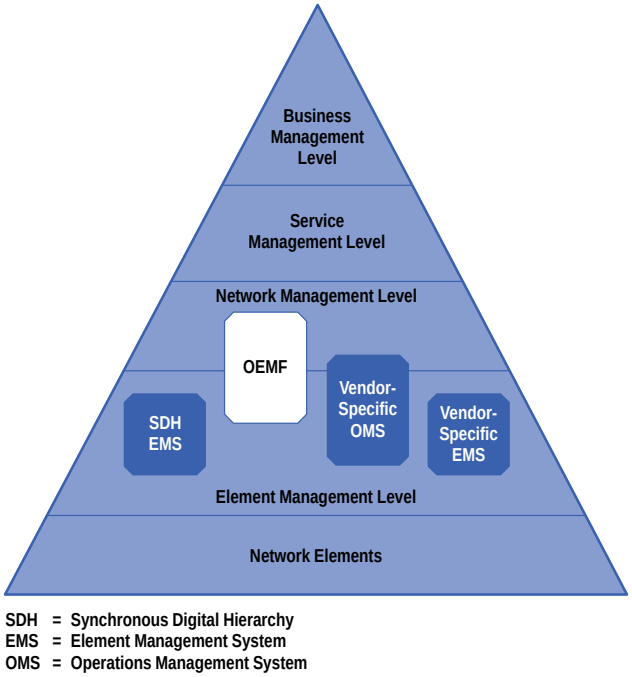


Fig. 3. *In the TMN hierarchy, the HP OEMF resides between the network management level and the element management level.*

Fault Management Platform

The FMP is a fault management platform that provides utility tools for managing alarms from multivendor devices and network element managers. It is based on the HP OpenView Distributed Management Platform. It has an extremely open architecture, which facilitates a seamless integration of third-party applications, as manifested by the OpenView Element Management Framework described earlier. Fig. 4 illustrates the FMP functional blocks. The main components of the FMP are the mediation device block, the FMP server block, and the graphical operator interface.

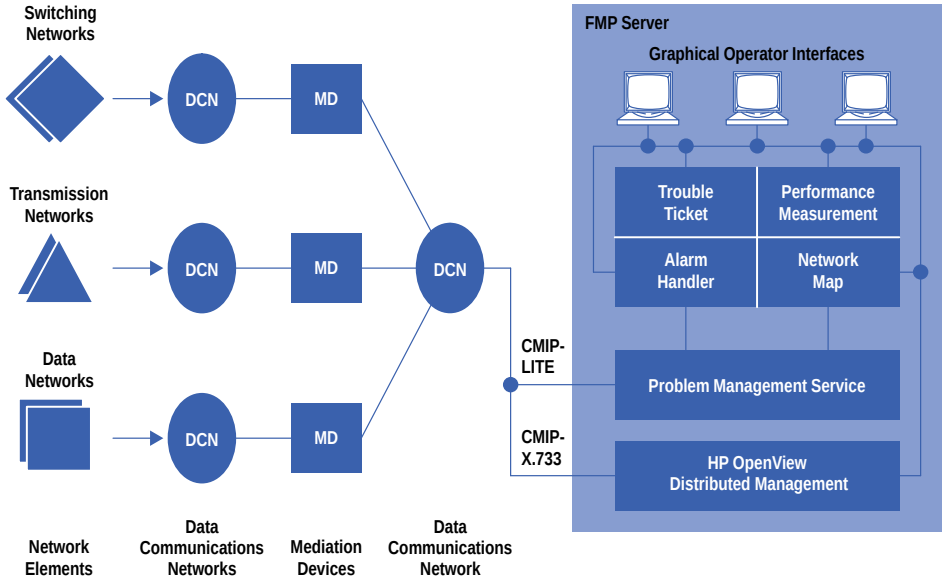


Fig. 4. *Functional blocks of the fault management platform (FMP) of the OEMF.*

The mediation device block provides the mediation and Q adapter functions and the FMP server provides the operations systems function. The mediation device logs, formats, filters, maps, and finally correlates all alarms it receives from network elements into ITU-T X.733 alarm reporting format and sends these alarms to the FMP server for alarm management. The mediation device can send the alarms to the FMP server using the CMISE protocol over the CMIP stack provided by the HP OpenView Distributed Management Platform, or optionally, using the CMIP-LITE protocol (an FMP representation of the X.733 alarm report) over TCP/IP.

The FMP server performs the problem condition management services. It provides graphical operator interfaces to aid in the management of the alarms being received from the network elements (which are performing the network element function). These graphical interfaces provide the means to interpret TMN information for the management information user. They perform the workstation function.

The FMP provides the fault management activities in a telecommunications network. However, to manage a telecommunications network, other management activities such as trouble management, performance management, and configuration management are also required. This requirement contributed to the OEMF concept, which allows a broad spectrum of best-in-class applications, regardless of manufacturer, to be integrated with FMP to provide an integrated network management solution. This integration is made possible by a range of open APIs provided in the FMP. The HP OpenView Distributed Management Platform APIs further enhance the integration capabilities.

Mediation Device Block

The mediation device logs raw alarms, formats and filters alarms from events, correlates these alarms, and then forwards them to the FMP server in the X.733 alarm reporting format. The mediation function is extremely important as the FMP server receives and manages alarms in a heterogeneous, multivendor, multinetwork environment in which the network elements send events in varying formats. Fig. 5 illustrates the functional blocks within the mediation device.

The mediation device provides a set of data collectors, which collect data over RS-232, TCP/IP, and SNMP. Reports in X.733 format can be sent directly to the FMP server using CMIP protocol. For data collection over X.25 and other types of networks, customer-specific data collectors can be written using mediation device connection APIs. These data collectors forward the events to the event logging module, which logs them into raw log files. The event logging module forwards the valid events to the event formatting module, which parses the incoming events and then classifies and formats them into message classes based on the parsing rules defined in the configuration. These formatted events are then logged into message class files corresponding to the message classes. The event formatting module forwards the events to the event mapping module, which filters the alarms from events, converts the alarms into the X.733 alarm report format and forwards them to the event filtering and correlation module. The correlation module correlates repeated alarms, transient alarms, and related alarms for a network element. The FMP supports a two-stage correlation approach in which the correlation

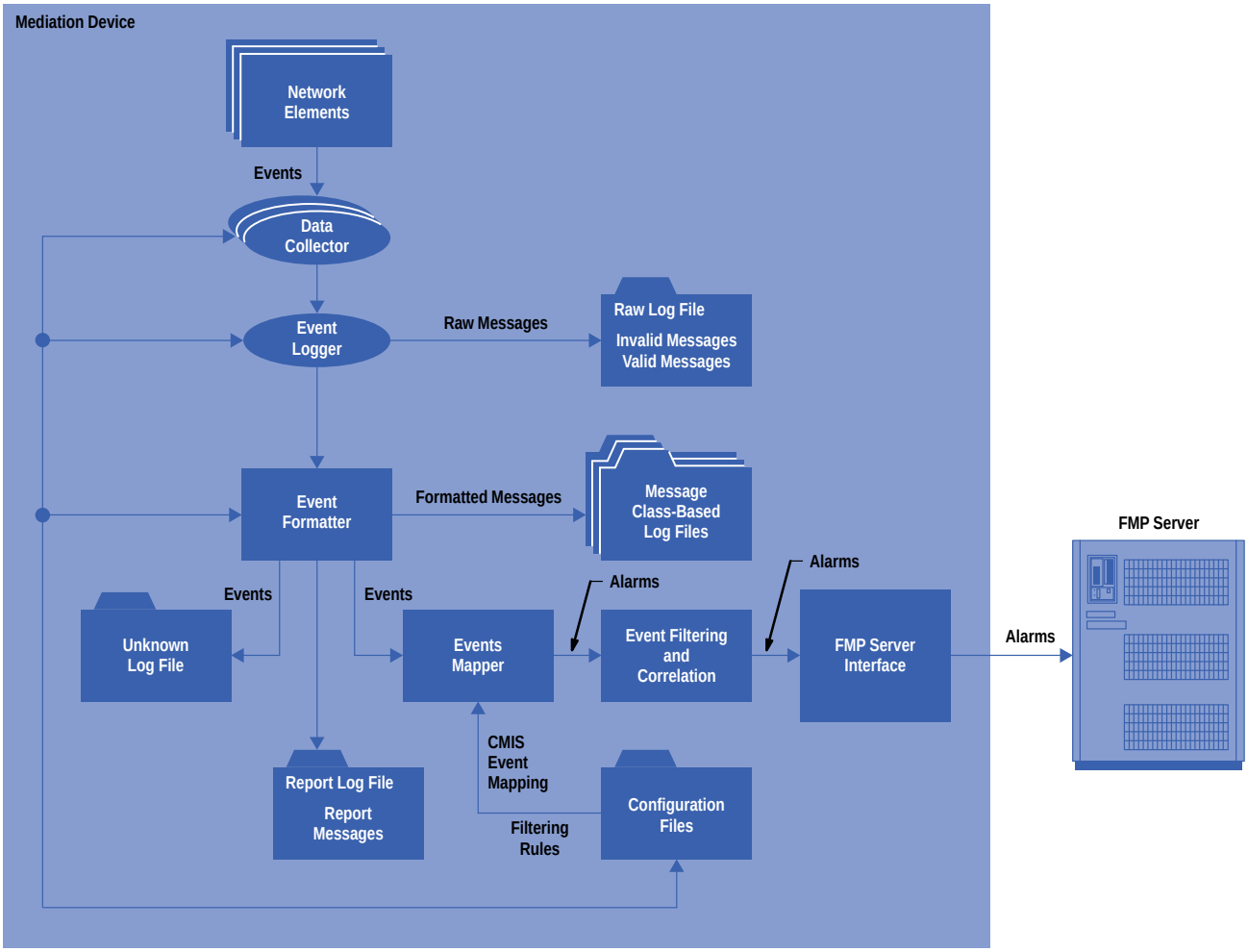


Fig. 5. *Mediation device functional block diagram.*

functionality is provided at the mediation device and at the FMP server. The correlation module forwards the alarm to the module interfacing to the FMP server. This module encodes the alarm into CMIP and sends it over the CMIP stack provided by the HP OpenView Distributed Management Platform or optionally (depending upon the configuration) encodes the alarm into CMIP-LITE and sends it over TCP/IP to the FMP server.

Correlation in the FMP

The FMP allows two stages of event correlation: one at the mediation device level and the other at the FMP server level. The correlation at the FMP server is done across the network being managed because the server has access to the topology database. The correlation at the mediation device is restricted to the network elements to which the mediation device is connected. The correlation at the mediation device level is done primarily to prevent not-so-important data from being forwarded from the mediation device to the FMP server.

The FMP has two types of correlation: repeated/transient correlation and root-cause/related correlation. Repeated/ transient correlation correlates alarms that are identical (they may have different severities) and are being emitted continuously by a network element. Root-cause/related correlation correlates alarms that have occurred because of a root-cause alarm and are not as important to the operator.

Let's take an example of root-cause/related correlation in a GSM network. Assumptions:

- MSC-1 is connected to BSC-1 which is connected to BTS-1 through BTS-4.
- An alarm A:MSC-1 (that is, an alarm of type A:MSC at MSC-1) causes an alarm B:BSC-1 (an alarm of type B:BSC at BSC-1).
- The alarm B:BSC-1 causes an alarm C:BSC-1.
- The alarm C:BSC-1 causes alarms D:BTS-1, D:BTS-2, D:BTS-3, and D:BTS-4.
- Of all these alarms, only A:MSC-1 is significant.

Fig. 6 illustrates the scenario. Based on the above assumptions, if an alarm A:MSC-1 occurs, the operator will also receive the alarm B:BSC-1 which will further cause alarms C:BSC-1 and D:BTS-1 through D:BTS-4. Even though the real problem is with MSC-1, the operator receives numerous alarms, many of which are of no significance.

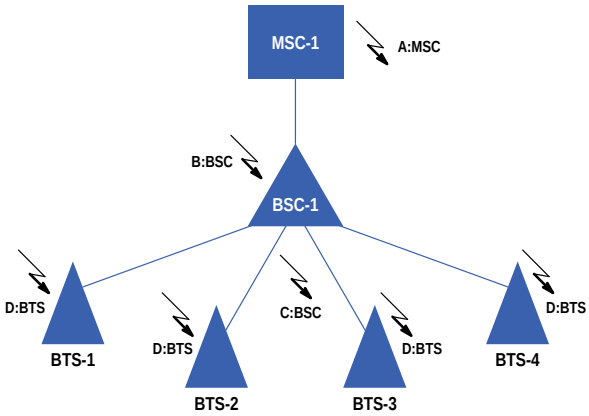


Fig. 6. *Scenario of alarm generation in an example GSM network without correlation. Many extraneous alarms can be generated in addition to the root-cause alarm.*

Let's specify the correlation rules to be as follows (the format of the event correlation rules specification has been simplified to explain the concept):

```
Rule 1: ROOTCAUSE : A:MSC RELATED : B:BSC
Rule 2: ROOTCAUSE : B:BSC RELATED : C:BSC
Rule 3: ROOTCAUSE : C:BSC RELATED : D:BTS

Correlation window: 20 seconds
```

With these correlation rules in effect, the operator will receive only the alarm A:MSC-1, which is the significant alarm. This behavior is illustrated below:

```
Correlation window (total elapsed time) = 20 seconds
|<------------------------------------------------------------>|
                                                               |
Arrival of A:MSC-1 -->|-->A:MSC-1 sent                         |
Arrival of B:BSC-1 --------->|                                 |
  (correlated by rule 1)                                       |
Arrival of C:BSC-1 ------------>|                              |
  (correlated by rule 2)                                       |
Arrival of D:BTS-1---------------------->|                     |
  (all BTS correlated by rule 3)                               |
Arrival of D:BTS-4 -------------------------->|                |
                                                               |
 End of Correlation Window---------------------->|
```

All alarms matching the correlation rules and occurring within the correlation window are subject to correlation. The correlation window can be a fixed or a sliding window. The arrival order of alarms is not important—in the above example, the alarms could have arrived in any order. As long as they arrive within the correlation time window, they get correlated. If a related alarm arrives before a root-cause alarm, it is held in the correlation module until the end of the correlation window. If the root-cause alarm does not arrive within the time window, the related alarm is sent out as uncorrelated. If the root-cause alarm arrives before the related alarms, it is sent out immediately. In the scenario above, the A:MSC-1 is sent out by the correlation module immediately and is not held back until the end of the correlation window.

Event correlation services are available in HP OpenView Distributed Management Platform 4.21 as an option (see ***Article 4***). Event correlation services further complement the event correlation provided by the FMP and, when integrated with the FMP, greatly enhance the correlation functionality of the FMP.

FMP Server Block

The FMP server provides problem management services. It logs, correlates, and distributes the alarms to the graphical operator interfaces. Fig. 7 shows the functional blocks within the FMP server.

An alarm is received from the mediation device (there may be one or more mediation devices) either in CMIP or CMIP-LITE by an interfacing module, which decodes and forwards it to the alarm logging module. The alarm logging module logs the alarm in the alarm database. This alarm is then passed to the alarm correlator module for correlation. This is the second

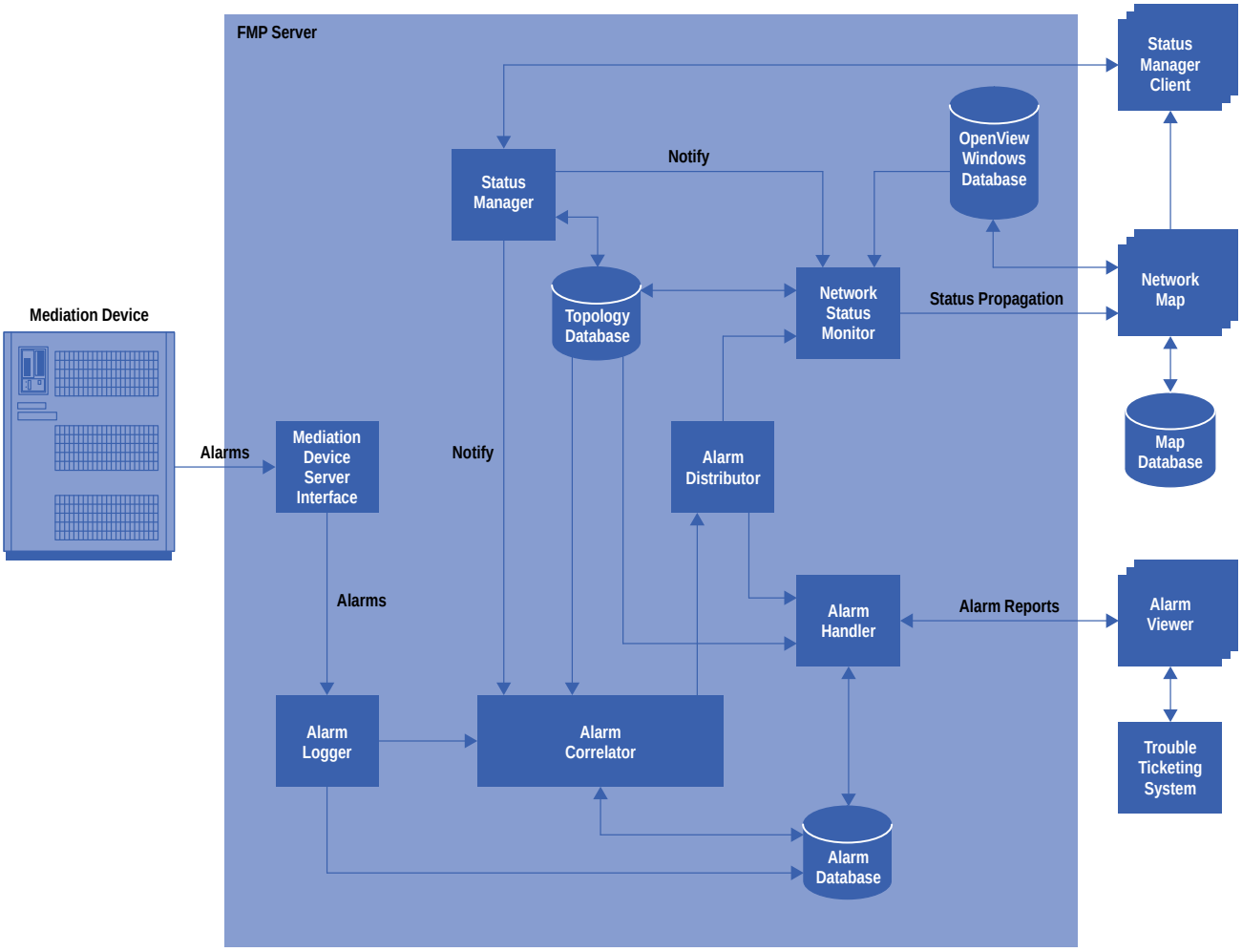


Fig. 7. *FMP server functional block diagram.*

stage of correlation, the first being at the mediation device. Correlation at the FMP server is performed across the network because the server has access to the topology information stored in the topology database, which resides at the server. After correlation, the alarm is distributed to the alarm handling module and the network status monitor module. The alarm handling module manages problems. Every alarm need not be a new problem—many alarms may be sent for the same problem. The alarm handling module identifies the alarm as belonging to a problem if it originates from the same network element and has the same probable cause and specific problem fields as the problem (these fields are specified by ITU-T X.733). The alarm handling module checks whether a problem condition already exists for the alarm received. If it does, then an update to the problem is sent to all the connected alarm viewers. Otherwise, a new problem condition is created and sent to the alarm viewers depending upon the span of control defined for each alarm viewer.

The network status monitor is responsible for status propagation according to the configuration rules. Depending upon the severity of the alarm, the network status monitor calculates and sets the status of the alarming network element on the network map. The network status monitor calculates the status of objects based upon the severity of objects both on the map and not represented on the map (*nonmap objects*). The need to support the concept of nonmap objects arises because in a telecommunications environment, the number of objects being managed can be very large. Also, the operator may prefer not to see all of the objects being managed on the map, but would want the status of the nonmap objects to be considered while calculating the status of the higher-level objects (in the containment hierarchy) that are represented on the map.

The status manager server (or simply status manager) facilitates the submission of outage schedules and maintains information regarding the outage of the network elements. Operators submit the outage information through the status manager clients.

Graphical Operator Interfaces

The FMP includes a set of utility tools that provide a graphical interface for the operator to manage the telecommunications network. The tools provided are the alarm viewer, the network map application, and the status manager clients for submitting outage schedules.

Alarm Viewer. The alarm viewer is an X11/Motif-based application that allows the operator to view the faults occurring in the network being managed and provides facilities to take corrective actions to handle these faults. Fig. 8 shows the alarm viewer window.

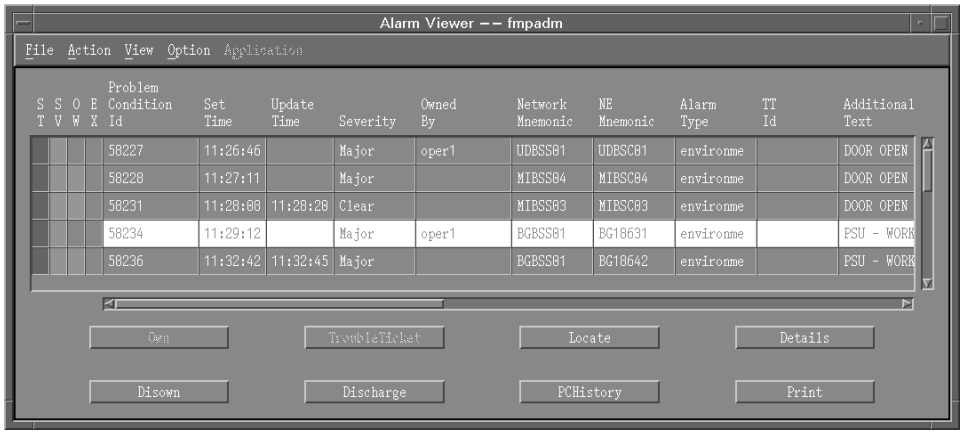


Fig. 8. *Alarm viewer window.*

The alarm viewer receives the problem condition from the alarm handling module. It allows the operator to select and perform the following actions on the selected problem condition:

- Own
- Disown
- Discharge
- Locate
- View Problem Condition History
- Create Trouble Ticket
- View Details
- Print.

By owning a problem condition, the operator acknowledges the presence of a fault. The operator can then locate the alarming network element on the network map and create a trouble ticket for the problem. Once the problem has been rectified, it can be discharged. On being discharged, the problem disappears from the alarm viewer. An audit trail is maintained in the alarm database regarding the actions performed on the problem. A list of all the alarms corresponding to the selected problem condition is displayed by clicking the Problem Condition History button. The alarm viewer can be configured to display only the fields in which the operator is interested. The Details button can be clicked to view the details of the problem condition. A hard copy of the selected problem condition can be obtained by selecting the print option.

The alarm viewer provides visual aids for quick identification of the severity of the problem. The columns in the problem condition row are color-coded and signify the outage, the severity, and the ownership status of the problem.

An operator can invoke an alarm viewer and perform actions on the selected problem conditions. However, operators need to be registered with the FMP server, and their spans of control, or *management domains*, need to be defined. Their control can be defined on the basis of the network instance, the network element class, and the network element instance. The alarm handler ensures that all the alarm viewers are consistent in case there is overlap in the operators' spans of control (it is common to have multiple operators responsible for the same network elements). For example, if a problem condition is owned at one alarm viewer, this owned status is propagated to other alarm viewers displaying this problem.

Alarm viewer menu options allow an operator to customize the alarm viewer. Operators can set their own sorting criteria for problem condition display. They can also set their own view preferences, for example to view problems from a certain network, view problems of a certain severity, view problems they own, and so on. These menu options offer flexibility and convenience to help the operators efficiently manage the problems occurring in the network.

The alarm viewer allows external and customer-specific applications to be registered with it. A problem condition can be selected and any of the registered applications can be invoked for the selected problem condition. This is an extremely useful feature which allows integration of the FMP with best-in-class applications from any vendor to complement the FMP functionality.

Network Map. The network map is an OpenView Windows application that displays the network being managed and the status of the network elements within it. The network elements are represented by icons and the colors of the icons represent their severity. The network map gets status updates from the network status monitor module. It allows the operators to navigate through the managed network and isolate the network elements generating the problem conditions.

It also allows updating of the topology either interactively through the menu options provided or programatically through the map loader APIs. The network map can be customized by creating logical views of the network. Thus, an operator can navigate through the whole network hierarchy while a manager can choose to look only at the status of the network at a higher level without going into the details of the network elements within the network.

The network map has the FMP registered as an application. The FMP menu option allows the operators to invoke various applications like the alarm viewer and the status manager clients.

Fig. 9 shows a network submap for an example GSM network.

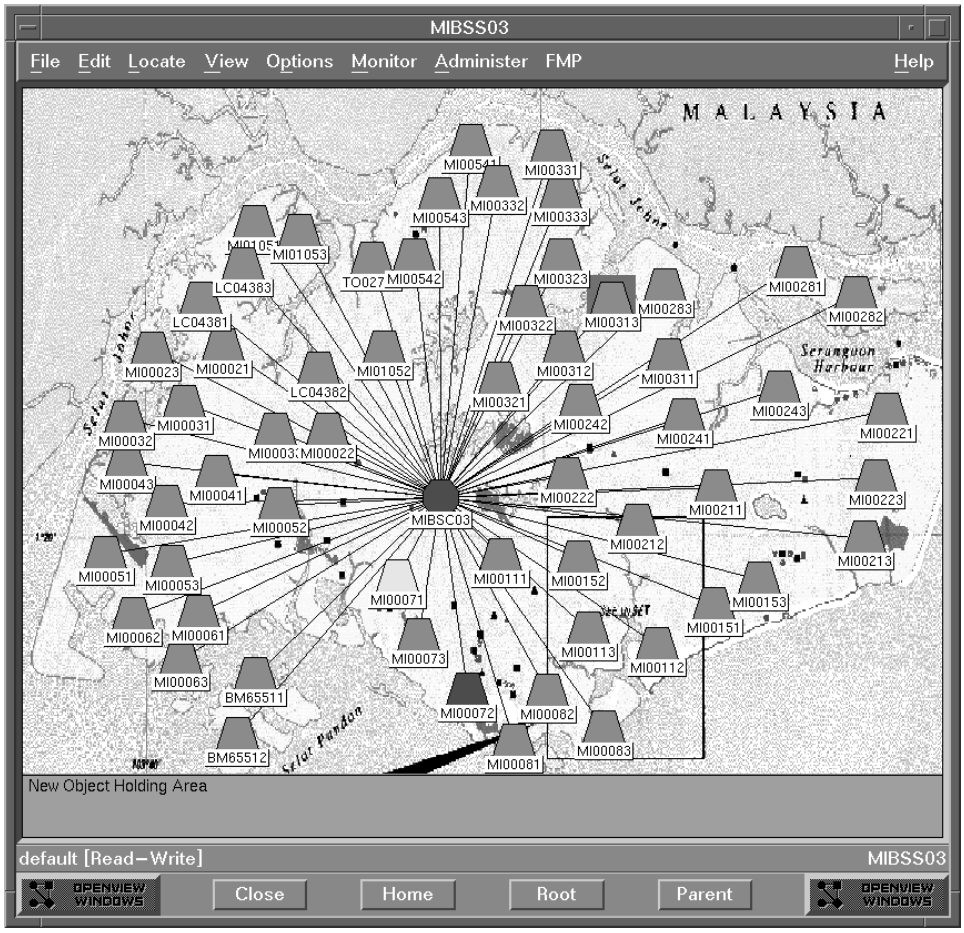


Fig. 9. *Network submap for a region in an example GSM network.*

Status Manager Client. The status manager client is an X11/ Motif application that allows an operator to submit outage schedules, that is, an operator can submit a proposal to put a network element out of service or restore a network element back into service. The operator needs to be configured for the status management capability to submit the outage schedules. The operator can specify the start and end times of the outages and whether the network element will be restored manually or automatically to in-service status. If an alarm is generated by a network element that is in the outage state, it is flagged by a different color in the first column of the alarm viewer. The FMP server can also be configured such that alarms from network elements in outage status are not sent to the alarm viewers at all. Information regarding the outage status of the network elements is maintained by the status manager server.

FMP Configuration

The mediation device has to be configured to be able to receive, format, and map events received from multivendor network elements in the X.733 alarm format. The object model of the network being managed, the different types of network elements, event correlation rules, operators' spans of control, and status propagation rules all have to be configured. Information regarding the mediation devices connected to the FMP server (the FMP server can be connected to more than one mediation device) and any customer-specific data collectors connected to the mediation devices also needs to be supplied.

FMP provides a screen-based GUI utility—the configurator—to aid in configuring the FMP and customizing it to manage a heterogeneous telecommunications network.

FMP Application Programming Interfaces

Throughout the design of the FMP, an open architecture and ease of integration were always given maximum importance. The FMP allows seamless integration with other applications as a result of its rich set of C and C++ APIs. The various APIs are described in the following paragraphs.

Data-Collector-to-Mediation-Device Connection APIs. These APIs can be used to write customer-specific data collectors to send events received from the network elements to the mediation device. Apart from data collection, these data collectors can also use these APIs to inform the mediation device regarding their operational status and to get information regarding the operational status of the mediation device.

High-Level Parsing APIs. High-level parsing APIs facilitate the validation of events received from the network elements. The data collectors can use these APIs to find the validity of an incoming event and flag the event as valid or invalid. This information is used by the event logging module in the mediation device to tag the event as valid or invalid in the raw log.

Pseudocode for a sample data collector using the data collector and high-level parsing APIs is as follows:

```
main()
{
   Open the network element port
   for(;;)
     {
       If (Data received from network element
           port)
         {
            //High-level parse the input data
            //received
            FaultBuf = HLPParse(Data Received)
            //Send the structure returned by the
            //HLPParse to the mediation device
            DCSendFaultToMD(FaultBuf)
         }
       //Inform the mediation device if the
       //network element port is not OK
       If (Error in network element port)
         DCSendPortStatusToMD(Port is not OK)
       //If the mediation device is shutting
       //down, shut down this data collector
       If (((msg = DCReceiveFromMD(control
             message from MD )) == MM_SHUTDOWN)
         ShutdownThisDC();
     }
}
```

Log APIs. The mediation device logs raw data received from the network elements. It then classifies these events into message classes and logs them in the corresponding message class file. The log APIs can be used to access these files. A number of applications can use the mediation services of the FMP and have the data collected at the mediation device in a format desired by them. These applications can then access this formatted information using the log APIs. Many interesting and useful applications can be written using the mediation and logging services provided by the FMP. An example is a raw log browser, which allows the operator to select an X.733 alarm from the alarm viewer and then extract and browse the raw alarm data corresponding to the selected alarm.

Application Registration APIs. These APIs allow the registration of external applications with the alarm viewer and facilitate passing information about the selected problem conditions to these applications. Application registration APIs can be used to integrate customer-specific applications. Once registered, the applications can be invoked from the Applications menu option of the alarm viewer. Taking the example of the raw log browser, the browser would first be registered with the alarm viewer. Then a problem condition can be selected and the raw log browser application can be invoked. Assuming that a raw log index is passed as some field in the X.733 alarm (e.g., rawlogindex as a part of additional information), this application can use the APIs to extract this index, which can be used to access the raw log. The trouble management system provided under the OEMF also uses these APIs. When a problem condition is selected and a trouble ticket is created for it, the trouble ticketing application uses these APIs to get information regarding the problem condition.

Problem Condition Management APIs. These APIs allow an application to interface to the problem condition management services. The alarm viewer uses these APIs. Customized alarm viewers, an automatic trouble ticketing application, an automatic problem condition discharge application, and other applications can be written using these APIs. Take, for example, an automatic problem condition discharge application for managing problems for which the switch normally does

not send a clear event. Based on certain criteria like the age of the problem condition, its severity, and so on, this application can be designed to discharge the problem condition automatically without any manual intervention from the operator.

Map Loader APIs. These APIs allow the customer's topology to be loaded into the FMP without having to add the objects manually into the FMP topology. This is extremely useful because the number of objects being managed can range anywhere from 4000 to more than 40,000. Map loader applications can be written to access the topology database of the customer and then populate the FMP topology using the map loader APIs.

Conclusion

The FMP APIs have led to the expansion of the number of products integrated under the OEMF umbrella. The FMP integrated with other best-in-class applications provides an enhanced telecom network management solution for efficient management of the multivendor, multi-equipment telecommunications networks of today.

Acknowledgments

I would like to acknowledge the members of the FMP team who have made it all happen, especially Chew Chye-Guan, Tok Wu-Chuan, Kuan Siew-Weng, Lin Chee-Kheong, Ho Yong-Boon, Vasu Sankhavaram, Prem N. Devadason, and not the least, project manager David Chua. Special thanks to the TMN, Integration Solutions Center, and Application Integration Center teams whose valuable suggestions have contributed to the growth of this product.

HP OpenView Event Correlation Services

When a fault occurs in a telecommunications system, it can cause an event storm of several hundred events per second for tens of seconds. HP OpenView Event Correlation Services (ECS) helps operators interpret such storms. It consists of an ECS Designer for the interactive development of correlation rules and an ECS engine for execution of these rules.

by Kenneth R. Sheers

Modern telecommunication technologies such as SDH/SONET (Synchronous Digital Hierarchy/Synchronous Optical Network) and ATM (Asynchronous Transfer Mode) can generate large numbers of events when a fault occurs. Every logical and physical element involved in delivering a service that uses the failed or faulty element can generate multiple events. This can result in an event storm of several hundred events per second for tens of seconds. The task of telecommunications operations staff is to determine the underlying cause from the thousands of events presented to them.

HP OpenView Event Correlation Services (ECS) is designed to deal with the problems associated with event storms in the telecommunications environment. The theory from which HP OpenView ECS technology has evolved was developed by HP Laboratories in Bristol, England.[1] ECS is delivered as two distinct components: the ECS engine and the ECS Designer.

The ECS engine is a run-time correlation engine. It executes a set of downloaded correlation rules that control the processing of event streams. At first release, the ECS correlation engine is integrated into the HP OpenView Distributed Management (DM) postmaster.

The ECS Designer is a graphical user interface (GUI) development environment that allows the correlation rules to be developed interactively by selecting, connecting, and configuring logical processing blocks. Once the rules have been created, their operation can be simulated and visualized using a log of events input to a correlation engine attached to the ECS Designer, with the engine's concept of time controlled by the ECS Designer.

The correlation rules are created using a visual paradigm of nodes connected together to form a correlation circuit. Events logically enter nodes via input ports and leave via output ports. An output port of one node is connected to an input port of another node in the circuit, so that events flow through the circuit from one node to the next. The functional node types provided with ECS are a superset of the basic types considered necessary and sufficient to perform real-time event correlation.

An event correlation circuit, Fig. 1, constitutes a set of correlation rules that can be compiled and downloaded to an event correlation engine. It consists of a series of interconnected and appropriately configured nodes, together with any associated data and relationship information. Fig. 2 shows the ECS architecture and the relationship of the correlation circuit to the ECS Designer and the ECS engine.

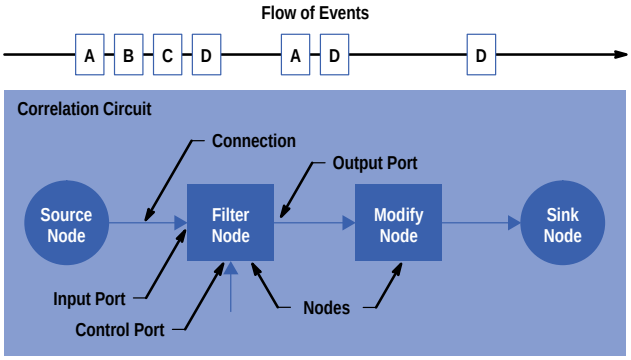


Fig. 1. *Generic correlation circuit. Correlation rules are specified by such circuits.*

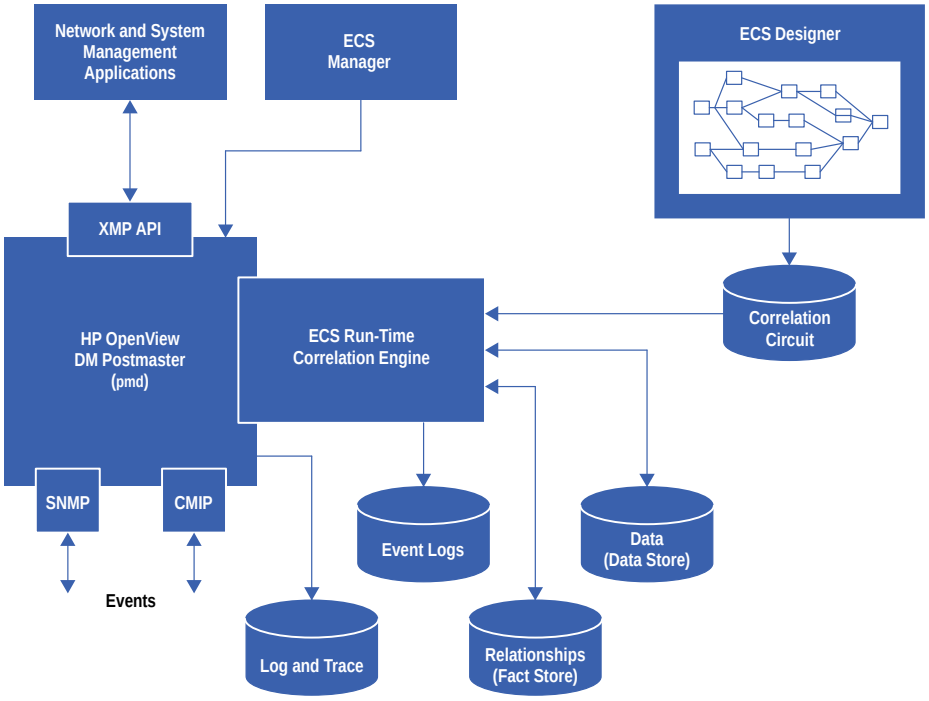


Fig. 2. *Architecture of HP OpenView Event Correlation Services.*

Real-Time Engine

A key differentiator of ECS compared with other correlation systems is that it operates in real time while taking into account the real-world problem that events will often be delayed in the management network and delivered to the correlation system out of order.

If events always arrived in bursts, it would be possible to buffer them on receipt and perform the correlation between bursts. However, event storms may be continuous, and the correlation engine should be capable of receiving the events, decoding them, and correlating them at the speed at which they arrive, continuously, without needing a mainframe to do the work. In ECS, any buffering required for correlation of time-separated events is dynamic and completely integrated into the normal engine operation.

In an ideal environment, a correlation engine is embedded into each piece of equipment that generates events. Correlated events from each piece of equipment are forwarded to another correlation engine where correlation across multiple systems is performed. This strategy reduces event volumes as early as possible, reducing the network traffic significantly. Thousands of events can be reduced to only a few events at each level of correlation.

Delayed Events

If all events were delayed by an equal amount within the management network, the correlation would simply be some delta time offset from real time. In a management network stressed by an event storm, some events will be delayed more than others. Consider a scenario in which an event B is to be suppressed if a prior event A has been detected. An A event should always occur two seconds before a B event.

Normally, it would be sufficient to remember that A had arrived, and when B arrives, discard it. But what if there has been no previous A event? Was no corresponding A event generated, so that the B event should be forwarded, or has the A event been delayed? If no A event has been received, it is necessary to hold B until there is no possibility of an A event arriving. If an A event doesn't arrive within some configured time, the B event should be transmitted. The permissible delay after which the B event will be forwarded is dependent upon the probable worst-case delay for any particular management network.

For any event, the delay imposed by the management network is known as the *transit delay* for that event. All events are considered to have a transit delay. ECS processes delayed and incorrectly ordered events in real time. Events can be reordered within the engine and held by individual nodes until related events are delivered even if subject to significant transit delays. Since events from multiple sources can be correlated, it is necessary that these sources have their clocks synchronized within known limits.

Information Concentration

ECS allows all pertinent information to be concentrated during correlation, with redundant and superfluous information suppressed. The useful information can be forwarded to management systems using either new events created by ECS or original events modified by ECS.

In the above scenario, a B event was to be suppressed if a corresponding, previously generated A event had been received within a specified time window. It is possible that the B event contains some useful information not contained in the A event. ECS allows the correlation engine to create a C event that contains the useful information from both the A event and the B event, along with any public information. In this situation, both the A and the B events would be suppressed, while the C event is transmitted. The event volume is reduced without losing any management information. Alternatively, the information content of event A could be modified to include the useful information from event B.

Correlation Circuit

A correlation circuit is a set of interconnected and appropriately configured nodes. Fig. 3 shows a typical correlation circuit. Events enter a circuit (visually) on the left through source nodes, flow through a user-specified interconnection of nodes, and depart the circuit through sink nodes.

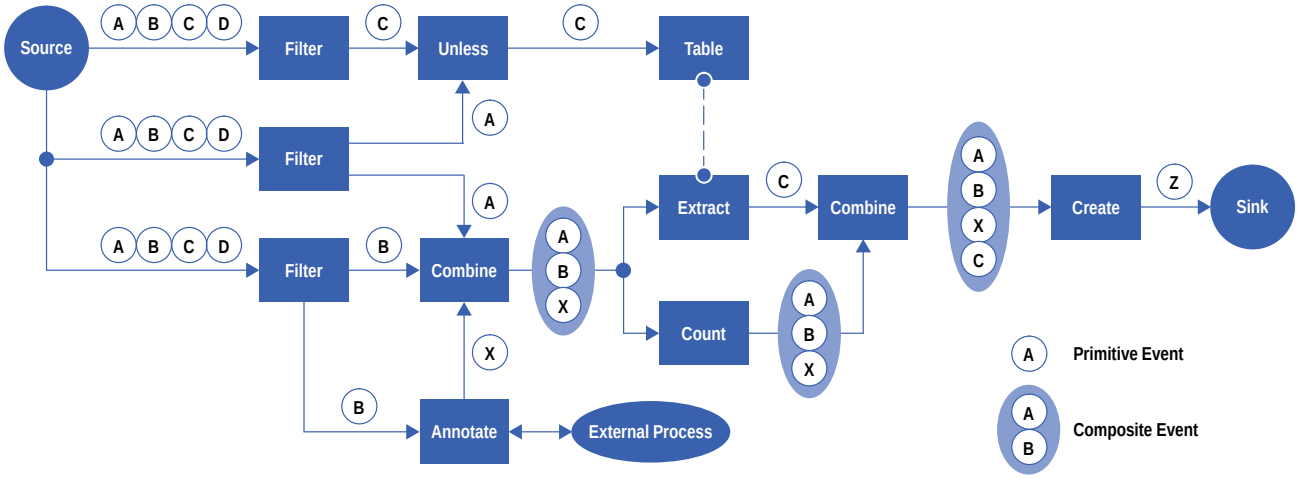


Fig. 3. *Typical correlation circuit.*

Events can be distributed down multiple paths within a circuit, and separate paths can be collected together. The event manipulation is controlled by the nodes in any given path. Events can be discarded, combined, created, and so on based upon the node types, the configuration of the nodes, and the dynamic conditions within the correlation circuit.

The correlation circuit paradigm results in a very intuitive implementation using a visual design and debugging interface that allows the circuit designer to see what operations depend upon each other.

Correlation Nodes

There are fifteen primitive node types, each type having a unique logical function (see sidebar: ***Correlation Node Types***). For example, a filter node will either forward or suppress an event depending upon its configuration. The primitive nodes can be combined to create powerful and efficient correlation circuits, in a manner similar to RISC processors, in which simple instructions are used to deliver powerful solutions.

Circuits begin and end with *source* and *sink* nodes, respectively. Events can be filtered conditionally or explicitly with *unless* nodes and *filter* nodes. Event ordering can be modified with a *delay* node, and events can be stored for future reference with the *table* node. Events stored in a table node can be extracted later with the *extract* node. Where information external to the correlation engine is required, the *annotate* node is used. Since ECS is used to assemble and consolidate information from multiple events, the *combine*, *rearrange*, *modify*, and *create* nodes are essential. Decisions are often based on the values maintained by *count* and *rate* nodes. Testing for the absence of expected events is facilitated with the *clock* node.

Event filtering is a degenerate case of event correlation requiring only filter nodes (and source and sink nodes) to complete the circuit. Filtering only considers discrete events.

Compound Nodes

A compound node encapsulates a subset of a correlation circuit and defines a new node type with user-defined functionality (see Fig. 4). In the ECS Designer, a compound node can be exploded to show the contained circuit, which may contain additional compound nodes. Compound nodes can be nested to any number of levels.

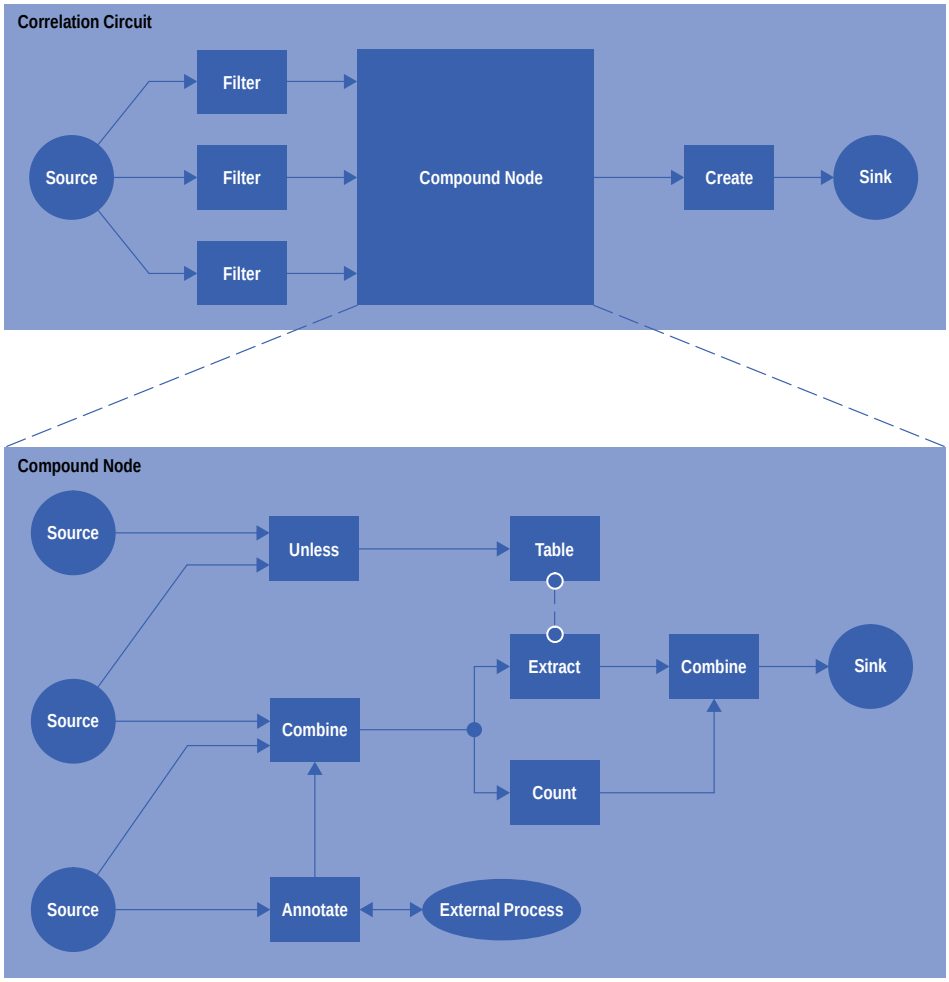


Fig. 4. *A compound node contains a correlation circuit that defines a new node type with user-defined functionality.*

An empty compound node can be placed on the ECS Designer canvas and interconnected with other nodes. The compound node can be exploded to add the required internal circuit. Alternatively, an interconnected set of existing nodes can be selected and converted to a compound node. This supports both top-down and bottom-up design methodologies, information hiding, abstraction, modular design and construction, improved circuit readability, functional reusability, and component testing.

Architecturally there is no difference between a top-level correlation circuit and a compound node. A circuit can be considered to be a compound node and vice versa. The contents of the compound node consist of an interconnected set of nodes including source and sink nodes. The source and sink nodes are connected to ports on the outside of the compound node. The ports are used to connect the compound node into a higher-level circuit. At the top level, the ports are connected to the external environment (see Fig. 5).

A compound node can be added to a library of compound nodes. A library can be selected and added to the tool bar of the ECS Designer so that the member nodes can be selected and used just like any of the primitive nodes. A library compound node can be copied, allowing local customization, or used by reference to the library copy. Reference use allows single-point maintenance of the library copy.

Node Configuration

Each instance of a node must be configured to the user's requirements. Node parameters are configured by the user to customize the functionality of the instance. Each node has a set of ports, which can be connected to ports of other nodes, with connections to some ports being required before the circuit will compile. The components of the count, unless, and table nodes are described in the three sidebars: ***Count Node***, ***Unless Node***, and ***Table Node***.

Parameters

All nodes are customized to specific requirements by setting the values of node parameters via the node's configuration dialog. Parameters are either evaluated by the engine at run time, or repeatedly whenever an event arrives at the input port of the node. Thus, parameters are either *static* or *dynamic*.

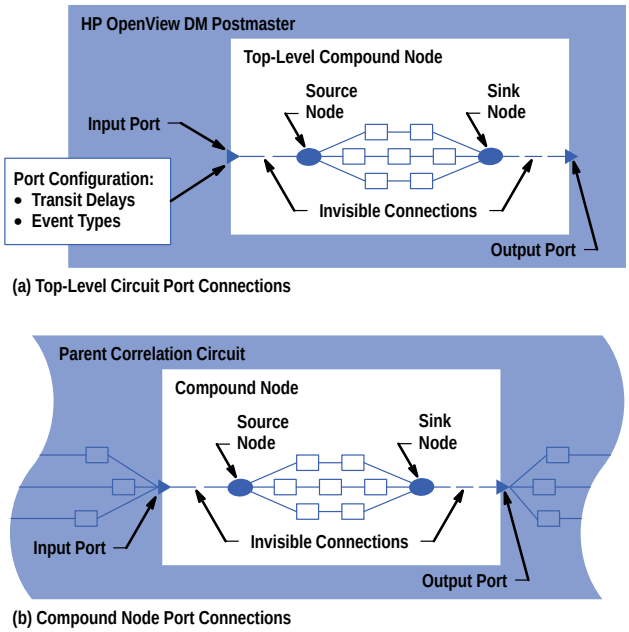


Fig. 5. *Compound node port connections. (a) Top-level. (b) Lower-level.*

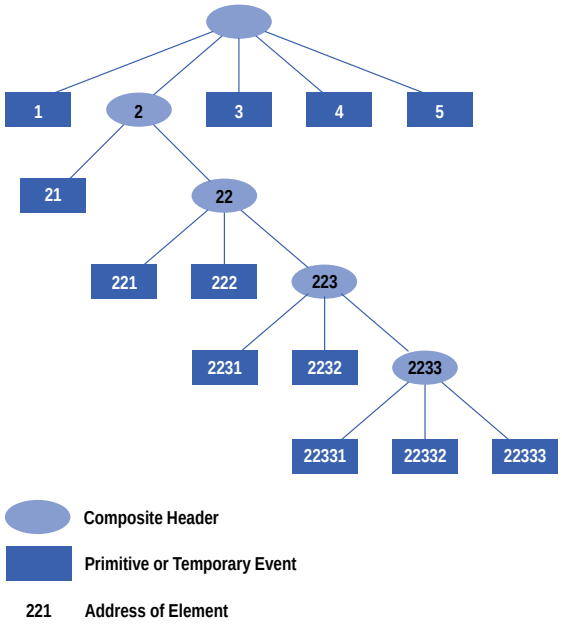


Fig. 6. *Structure of a composite event.*

Static parameters typically set resource and operational limits, such as the size of a table node, the frequency with which a clock node generates an event, how long an unless node should wait for an inhibiting event, and so on. Dynamic parameters define the operational behavior of a node. For example, a dynamic parameter of a filter node is a Boolean condition that is evaluated when an event enters the input port. The expression can take the incoming event as an argument, along with any other data from throughout the circuit. The result determines whether the event is forwarded to the true output or the false output.

All parameter values are actually expressions that can use references to values stored in the *data store* and to relationships stored in the *fact store* (see sidebar: ***Fact Store and Data Store***). For example, where a value should be supplied for a parameter, the name of a variable defined in the data store can be used. The data store entries can be changed without changing the configuration of the node. Dynamic node parameters are evaluated whenever a new event arrives at the input port of the node. If these parameter expressions reference data store or fact store entries, updates to these stores will affect subsequent parameter evaluations. This allows the behavior of a correlation circuit to be modified dynamically.

Like any primitive node, a compound node can have parameters that must be set or configured when the node is instantiated. The parameters are defined and documented by the designer of the compound node.

Ports

All nodes have one or more ports, which are visible in the ECS Designer. Nodes are interconnected via ports appropriate to the required functionality. Depending upon type, nodes can have input ports, output ports, error ports, reset ports, and other types of ports. Normal operations occur through the normal input and output ports. For a filter node, the configured expression should evaluate true or false, causing the incoming event to be routed to the true output or the false output port as appropriate.

Where run-time errors occur in evaluating the expressions configured for particular node instances (not everything can be known before run time), an event will be output through an error output port.

Except for the combine and compound nodes, primitive nodes have a fixed number of ports. The combine node can have up to 50 input ports (combining event streams). The compound node can have up to 50 input and output ports to support the encapsulated functionality.

Reset Ports

Delay, unless, combine, and annotate nodes can hold events in memory associated with an input port pending some condition becoming true. Table nodes can hold events in long-term memory.

When the engine is to be stopped or the correlation circuit is to be modified, the future conditions can now never be true, and the applicability of stored events is indeterminate in the context of the modified circuit. Even if the circuit were to be stopped and restarted without change, the potential for pertinent events to have been missed invalidates the state of the correlation. Critical events that should have been output may now be discarded.

It is necessary to be able to output the stored events before engine reset or reload so that potentially critical events are not lost. All nodes that store events have reset input and reset output ports. If an event is forwarded to a node's reset input port, any stored events will be output or discarded in a defined manner. The reset event will be output via the node's reset output port, possibly to a downstream node's reset input port, allowing a reset circuit to be added to an operational circuit.

Public Data

The nodes of a correlation circuit typically make decisions and perform actions based upon the arriving events, the information within the events, and the current time of the engine. Other data is also available to the dynamic node expression parameters to control node processing, including node attributes, the data and fact stores, and annotation data.

Node Attributes. A node can export one or more data values for use by expression and condition parameters configured for other nodes throughout the circuit. The count node exports a count attribute which increments or decrements for each arriving event. The table node exports two attributes: a count attribute whose value is the number of events currently stored and a contents attribute whose value is a list of all events stored in the table.

Compound nodes can export the attributes of any contained nodes as attributes of the compound node. If they are not exported, the attributes of internal nodes are private to the compound node and will not be visible outside the compound node.

Data and Fact Store. The data store contains entries of name-value pairs and the fact store contains entries of name-relation-name triples (see sidebar: ***Fact Store and Data Store***). These stores operate as in-memory databases and are accessible globally throughout the correlation circuit.

The data store entries allow values to be referenced by name, so that particular values need not be known when the circuit is designed. Using indirection through the data store also allows correlation circuits to be reused at multiple sites, with specialization via appropriate data store values.

The fact store allows the relationships between objects to be tested by node parameter conditions and expressions. Typically the entries will be used to reflect network topology information, and will allow event relationships to be determined dynamically without building the network model information into the actual correlation circuit.

Annotation. An annotate node (see sidebar: ***Annotation***) can be used to obtain data from outside the correlation engine for use within the engine. This data will be used to make correlation decisions, or to add to created events or modified events.

Event Types

Several event types can logically exist within a circuit. These include primitive events, composite events, and temporary events.

Primitive Events. A primitive event is a single event as it entered the correlation engine, possibly with data values modified within the circuit, or an event created within the engine, suitable to be returned to the external environment. ECS currently supports CMIP (Common Management Information Protocol, ISO/IEC 9596-1) and SNMPv1 (Simple Network Management Protocol, version 1) event types. The engine isolates the external event type from the internal functionality using specific decode and encode modules for each event type. This modular design allows additional event types to be supported in the

future. Since the internal processing is not dependent upon the format of primitive events, it is possible to correlate events of any supported type with events of any other supported type.

Composite Events. A composite event is an ECS internal mechanism that allows multiple events to be collected into a single addressable structure in which all members are accessible (Fig. 6). The event aggregation capability is fundamentally important for ECS. Collecting and processing multiple events as a single event allows all important information to be collected together. Members of a composite event can be primitive, composite, or temporary events. A composite event is only defined within a correlation engine, and cannot be output from a top-level circuit back to the environment. Composite events can be passed into and out of compound nodes.

Temporary Events. Where an event is required as an internal container for data, or where a trigger event is required, the engine will create a temporary event. For example, the clock node will emit a temporary event at each clock period. There is no relevant data in this empty temporary event. It can be used to trigger correlation activity elsewhere in the circuit. Where results are returned in response to a request by an annotate node, a temporary event is created to hold the data and returned to the circuit as a component of a composite event. A temporary event can enter or leave a compound node, but it cannot be output from a top-level circuit back to the environment.

Enhancing Event Information

A fundamental value proposition of ECS is that event information can be enhanced. What this means in reality is that all available information—for example, all information relevant to some network fault condition—is consolidated from multiple time-separated events, the data store and fact store, and data external to the engine via annotation.

When all pertinent information has been assembled (and all superfluous data discarded), it must be forwarded to interested operations systems. This can be done by:

- Creating a new primitive event containing the consolidated information, using the create node to create the event and copy the data into the event.
- Modifying the data values in an existing primitive event, using the modify node to change the values of the event's attributes before the event is output.

The input primitive events, which each contain only a fraction of the total relevant information, can be suppressed. Only the new or modified events are forwarded to interested management entities. The result is that the events actually delivered to management systems contain enhanced information content.

To aggregate all pertinent data into the delivered events, it is necessary to be able to collect the information together and process it through the engine as a single data unit. This allows correlation decisions to be applied to the logical block as a single unit, providing major efficiencies in circuit design and processing loads. Composite events are used to aggregate events into a single unit.

Event Processing

When an event is received by the correlation engine it must be decoded from the specific format (BER encoded*) sufficiently to determine the event creation time and the event identification. These values are fundamental to the operation of ECS. The creation time must be known to allow the time relationships between events to be known. The event identification can be used explicitly to control which branch of the circuit the event will logically enter.

HP OpenView ECS has been designed for high performance. Event encoding and decoding, and event copying within the engine, are implemented using a just-in-time encode and decode mechanism and sophisticated systems of header structure lists, event lists, pointers, and reference counts. An event is not fully decoded if not required by the correlation rule parameter expressions. References to events pass from node to node, rather than active events or copies of events.

When an event is forwarded down multiple paths in a circuit, reference counts are incremented. Only when one of these logical copies is modified (say with the modify node), is the event duplicated before modification. When the reference count is decremented to zero, possibly when the event is output from the circuit, the event is removed from the event list.

Retained Events

The transit delay of an event is defined as the number of seconds between the creation time of an event and the time that the correlation engine receives the event, assuming that both clocks are synchronized.

The example previously used considered the case in which an event A has arrived and a consequential event B is suppressed when it subsequently arrives. The correlation must consider the permissible transit delay range for event A to cover the situation in which event A arrives after event B. This requires that either event A or event B be retained in the circuit at the point where the condition is being tested. In a real-time engine, in which memory resources must be conserved, the event

* BER stands for Basic Encoding Rules (ISO/IEC 8825, ITU-T X.209). The BER define how ASN.1 (Abstract Syntax Notation 1, ITU-T X.208) data types are encoded to be transported on the network. Both of the primitive event types supported by HP OpenView ECS, that is, SNMP traps and CMIP notifications, are encoded using BER.

should be retained only while there is a possibility that it may be required in an active correlation, and automatically destroyed when no longer required.

A circuit must be configured with a transit delay window, which acts as an initial filter to eliminate any events with creation times outside this window relative to the correlation engine time. The circuit transit delay window is propagated into the circuit to calculate the transit delay limits on all nodes whose operation depends on event time differences. If a node imposes an additional time window for event comparisons (e.g., an unless node allows an inhibiting event to occur at some time offset from the exciting event), the allowable transit delays for the subsequent circuit are automatically adjusted to include the additional possible transit delay.

Events can be retained in port or node memory pending some condition becoming true. Each such event will be examined at each engine clock cycle to ensure that the creation time relative to the engine time is within the computed transit delay window at that point. Events failing to meet this requirement are released from memory automatically.

Data Access and GDMO MIBs

The tests and comparisons performed by the nodes must allow events to be tested for content. ECS provides high-level access to any element of an event, and has language data types that map onto the ASN.1 data types in an event. In ECS, each addressable component of an event is referenced as a named attribute. For example, an event may be significant if its severity attribute has a value of critical. ECS provides a sophisticated mechanism that allows the designer to specify this test (for a filter node) as:

```
input_event("severity") = "critical"
```

This Boolean expression extracts the severity attribute of the input event, tests the value of the attribute, and evaluates as either true or false.

The concept that an event has a series of attributes that have values that can be examined or modified is fundamental to ECS. The attributes of an event are all the components or elements of an event that are specified by the MIB (Management Information Base) that defines the event. The MIB is added to the underlying HP OpenView DM platform so that it can be accessed by the correlation engine. MIBs must conform to the GDMO model (Guidelines for the Definition of Managed Objects, ISO/IEC 10165-4, ITU-T X.722) or the HP OpenView DM platform will not accept them. Once part of the platform, ECS accesses the components of the events using the textual names from the MIB registration tree. The MIB is described in *Article 6*.

Language

Underlying the ECS Designer GUI is a complex and sophisticated language called ECDL (Event Correlation Description Language), which supports the complete specification of the correlation circuit including all the dynamic node expressions and conditions. ECDL includes data types, operations, and functions that allow read access to all component data in the events as they traverse the circuit, and to all public data within the circuit. (Event attributes can be altered with the modify node.) The ECS Designer ensures that the circuit designer does not need to understand this language in great detail. The circuit is specified by the visual interconnection of selected nodes. Node parameters are specified wherever possible using simple ECDL constructs and supplied library functions written using ECDL or actually built into ECDL. Advanced users are able to create specialized reusable functions. The ECDL code produced by the ECS Designer is encrypted in source form and compiled for downloading to the correlation engine. Direct coding using ECDL is not supported and cannot be compiled.

Building and Testing Correlation Circuits

The ECS Designer (which includes circuit design and simulate modes) is a GUI that allows the circuit designer to use a highly productive intuitive paradigm to build a correlation circuit by interconnecting primitive and compound nodes (see Fig. 7).

In ECS Designer build mode, nodes are selected from the tool palette, placed on the canvas, and interconnected to form the correlation circuit. Subsets of the circuit can be encapsulated as compound nodes to improve readability, implement top-down design rules, or promote reuse. Each node must be configured with appropriate values or expressions for its parameters. The general flow of events is determined by the circuit layout, subject to the conditions imposed by individual node parameters.

When the circuit is complete, the ECS Designer can be switched to simulate mode. In this mode, events can be input to the circuit to allow visualization of the flow of events through the circuit.

Various visual techniques are used to provide circuit operation feedback to the circuit designer. Events can be input in various modes: stepped by event or by time, free-run at selectable speed until a breakpoint, and others. The status of each node can be examined at any time. For instance, the contents of table nodes can be examined, the number of events through various ports can be checked, and so on.

In the simulate mode, events are input to the circuit from an input event log. The circuit designer can view both the input events and the correlated output events by means of associated event browsers.

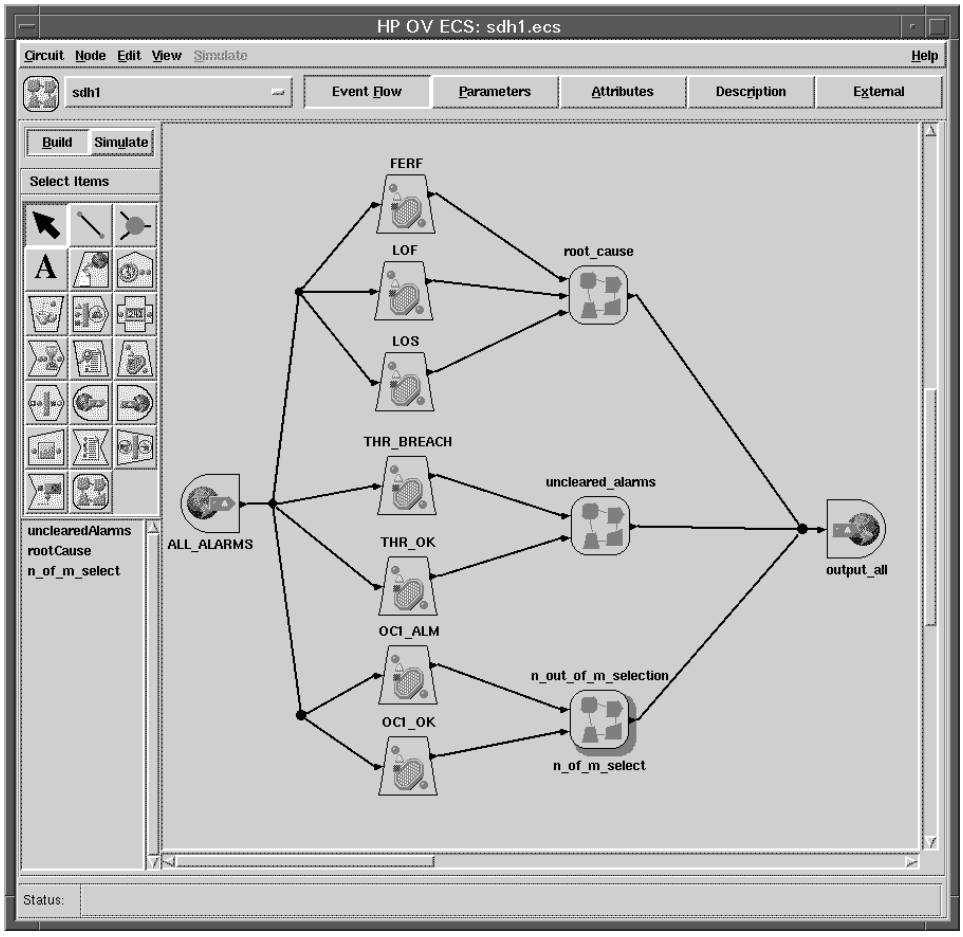


Fig. 7. *Constructing a correlation circuit using the ECS Designer GUI.*

The simulation is performed using a fully functional correlation engine, except that the engine does not free-run. The engine's notion of time is under the control of the simulator.

When a circuit has been developed and tested, the ECS Designer is used to compile the circuit so that it can be down-loaded into correlation engines, possibly in remote locations.

The event log that is input to the ECS simulator is in a structured ASCII format, allowing the events to be manually created or edited. It is desirable to use a log of real events, and to collect them automatically. Support is provided to collect real events in the required format. It may be necessary to make some simple edits to this event log to simulate the possible worst-case transit delays.

HP OpenView DM Interfacing

The correlation engine is integrated with the HP OpenView Distributed Management (DM) postmaster, ensuring that correlation is applied at a common point so that all events can be subjected to correlation (see Figs. 2 and 8). A correlation engine can be installed wherever an HP OpenView DM platform is installed. The distributed nature of HP OpenView DM event management services allows a distributed hierarchy of correlation engines to be readily implemented.

Adding correlation engines to the HP OpenView DM postmasters is transparent to existing agent and manager entities communicating via the postmasters, except that events can now be correlated. If the loaded correlation circuit were to pass all events, there would be no observable difference in the operation of these entities, or in the events being generated and received.

The HP OpenView DM platform is described in ***Article 1***.

Events entering the postmaster are routed to the correlation engine where they may be accepted into the engine depending upon the configuration of the circuit input ports (see Fig. 8). Confirmed CMIP events are immediately returned to the postmaster by the correlation engine. It is normally expected that a management entity will receive a confirmed event, and that the confirmation is returned as a consequence of some action having been taken, frequently by an operator. Typically, if the confirmation is not returned, the agent entity that generated the event will issue another (possibly different) event. The operation of the agent entity may be effected by the absence of the confirmation. If confirmed events were accepted into the correlation engine where they were subsequently suppressed as part of the correlation, the confirmation would not be

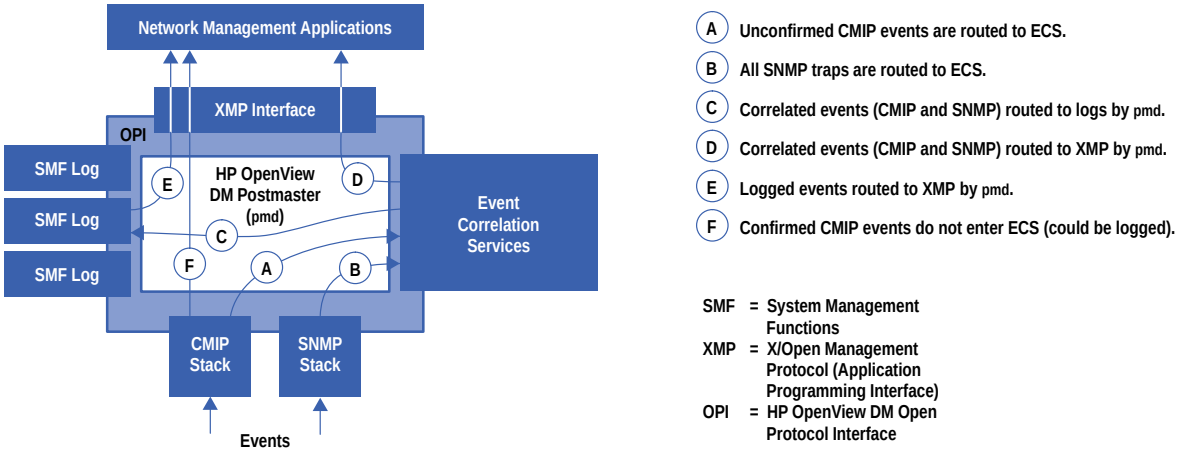


Fig. 8. *Event routing to and from HP OpenView Event Correlation Services.*

returned since the normal receiving entities would never see the event. Conversely, if the correlation engine generates the confirmation, the agent entity can modify its function based upon the assumption that an operator has seen the event and taken appropriate action.

The input ports of the correlation engine must be configured to accept all events received or they will be filtered out and discarded by the engine. Where significant events are expected for which correlation has not been designed into the circuit, a branch of the circuit can be arranged as a pass-through to transmit all events not explicitly handled by the other circuit branches.

Acknowledgments

I would like to thank Keith Harrison and Michele Campriani of HP Laboratories, Bristol for developing the theory upon which this technology is based, and for their encouragement and assistance in bringing the technology to market.

Reference

1. K.A.Harrison, *A Novel Approach to Event Correlation*, HPL-94-68, HP Laboratories, Bristol, England, July 1994.

Correlation Node Types

Fifteen primitive node types are supplied with HP OpenView Event Correlation Services (ECS). Every correlation circuit must have one or more source nodes and one or more sink nodes.

Source Nodes. This is where events enter a correlation circuit. There can be more than one source node in a circuit. For a top-level circuit, source nodes are connected to the engine's input ports, where events are delivered by the HP OpenView DM postmaster.

Sink Nodes. This is where events leave a correlation circuit. For a top-level circuit, the events are returned to the HP OpenView DM postmaster for delivery to management entities that have registered to receive them.

It is necessary to be able to suppress unwanted events. In the circuit paradigm, events are filtered by preventing them from flowing through different paths in the circuit. This is done by filter nodes and unless nodes.

Filter Nodes. These suppress events based upon a configured expression which typically uses the incoming event as an argument.

Unless Nodes. These forward an event unless another event was created within a configured (positive or negative) time period relative to the creation time of the first event, and the configured (filtering) expression evaluates false.

Events can be delayed in the management network, especially when the network is stressed. This may result in delayed events and events arriving in a different order than the order in which they were created. Delay nodes can be used when correlation decisions depend upon events being processed in the strict event creation time order.

Delay Nodes. These hold an event until the creation time is a configured number of seconds before the current time. This has the effect of guaranteeing that the events are output from this node in creation time order.

Events may need to be stored for extended periods so that future correlation decisions can be made using the event history. Subsequently, it may be necessary to extract complete copies of events from the storage.

Table Nodes. These hold a logical copy of all events sent to the table, subject to configured retention parameters and conditions. Other nodes can examine the event list, stored in creation time order, and make processing decisions based upon the contents.

Extract Nodes. These search a table node and extract a copy of one or more events from the stored list, subject to a configured condition. The search is triggered by an event arriving at the input port of the extract node, and the extracted events are output as a composite event (see ***Event Types***).

While most of the useful information will come from the event stream, data may need to be obtained from outside the correlation engine.

Annotate Nodes. These obtain data external to the engine and add it to an output composite event. The external data is now available in the event for subsequent use in the downstream circuit. The annotate request provides time for the request to be serviced, after which it will time out. Other events continue to be processed during this period.

One of the fundamental features of ECS is the ability to collect and consolidate discrete pieces of data from the event stream and from outside the engine to produce value-added information. Events need to be manipulated, including combining events into a single data unit, changing the structure of this unit, changing event data values, and creating new events.

Combine Nodes. At these nodes, two or more input event streams are combined into a single output stream, with each output event being a composite event containing an event from each input stream. Events on one stream can be held until events on other streams arrive.

Rearrange Nodes. These change the structure of a composite event, including pulling a single normal event out of a composite.

Modify Nodes. These change attribute values of incoming events to any values required. The values can be copied or calculated from any publicly available data. A copy of the original event is made, and the event copy is modified and output. The original event is not modified.

Create Nodes. These create a new event with a format controlled by a configured specification and attribute values set according to a configured specification. Event creation is triggered by an event arriving at the input port. The event's attribute values can be set from any publicly available data throughout the engine, including from the incoming event.

Some basic utility functions are provided by the following nodes.

Count Nodes. These count the events passing through the node.

Clock Nodes. These generate an empty event every configured time interval. This allows circuit logic to be triggered in the absence of any incoming events, enabling the absence of required events to be detected.

Rate Nodes. These calculate the rate at which events are passing through the node.

Count Node

The count node (Fig. 1) is an example of a simple node. It has a single parameter, initial count, which sets the initial value of the count attribute. An event entering the increment input port increments the count attribute and is immediately output via the increment output port. An event entering the decrement input port decrements the count attribute and is immediately output via the decrement output port. An event entering the reset input port resets the count attribute to the initial count value and immediately exits via the reset output port. At least an increment input port or a decrement input port must be configured or the compiler will generate an error. All output port connections are optional, and events routed to unconnected ports are discarded. The default value for initial count is zero. The count attribute is exported as a read-only attribute, which can be referenced in node parameters and expressions as <countnodename>.count. If both the increment input and decrement input ports are connected, the count attribute will indicate the difference in the number of events entering these ports.

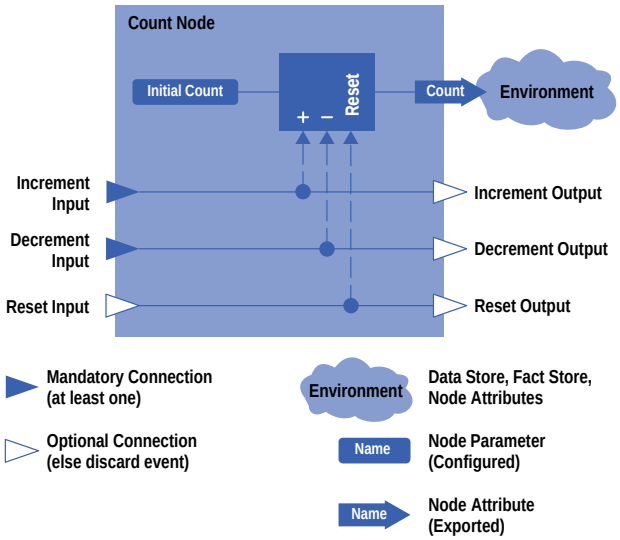


Fig. 1. *Count node.*

Unless Node

The unless node (Fig. 1) is an example of a complex node. The unless node will transmit an event arriving at the input port (the exciting event) to the output port, provided that no event arrives at the inhibitor input port (the inhibiting event) which satisfies the criteria specified by the window parameter and the condition parameter. These events can arrive at different times; either order is supported. The transit delays of arriving events must be within the window parameter limits to be accepted by the relevant port. If an accepted inhibiting event arrives and there is an accepted exciting event in memory, the condition parameter is evaluated. If an accepted inhibiting event arrives and there is no exciting event in memory, the inhibiting event will be held pending the arrival of an accepted exciting event, at which time the condition parameter will be evaluated. The condition parameter is a Boolean expression that can take both the exciting event and the inhibiting event as arguments. For example,

```
condition: input_event("device") = inhibitor_event("device")
```

will evaluate true if both events were generated by the same device (assuming this information is contained in the events). Any environment data (data store, fact store, and node attributes) can also be used in the condition expression. If the condition evaluates false, the exciting event is output via the output port. If the condition evaluates true, the exciting event is inhibited and is output via the inhibited output port if one is connected, or discarded if not. If the evaluation of the condition causes an error (e.g., if a referenced event attribute does not exist), the exciting event will be combined with the inhibiting event and output via the error output port as a composite event. If the error output port is not connected, the composite event will be logged. If the transit delay of the exciting event does not satisfy the window parameter transit delay window, the event is output via the fail output port. If the inhibiting event fails this test, it is silently discarded. The inhibiting event is never output except as a composite event as described above.

An event arriving at the reset input port causes any events held in the memory of the input (exciting) port to be output immediately via the fail output port, and any events held in the inhibitor input port memory to be silently discarded. The reset event is immediately output via the reset output port.

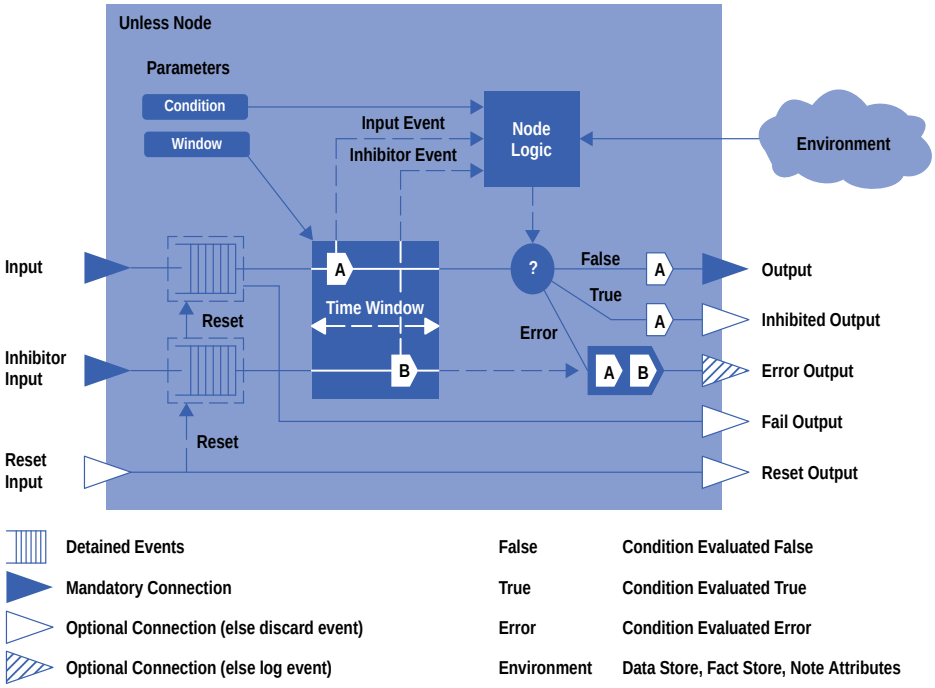


Fig. 1. *Unless node.*

Table Node

The table node (Fig. 1) is the only example of a special (and complex) node. The table node stores events for extended periods. The number of stored events is exported as the count attribute. All events stored in the table node are exported as a list of events in the contents attribute. The events in the contents attribute can be examined by expression and condition parameters of other nodes in the circuit, and extracted by the extract node. The table node stores a single list of events (the contents) in two logical areas. The *current area* is controlled by the save until and max events parameters, and the *retained area* is controlled by the retain condition and delete condition parameters. The two areas have different mechanisms for storage. The current area is based on physical and event age limits, while the retained area is based on evaluated conditions, which typically test data values within the events.

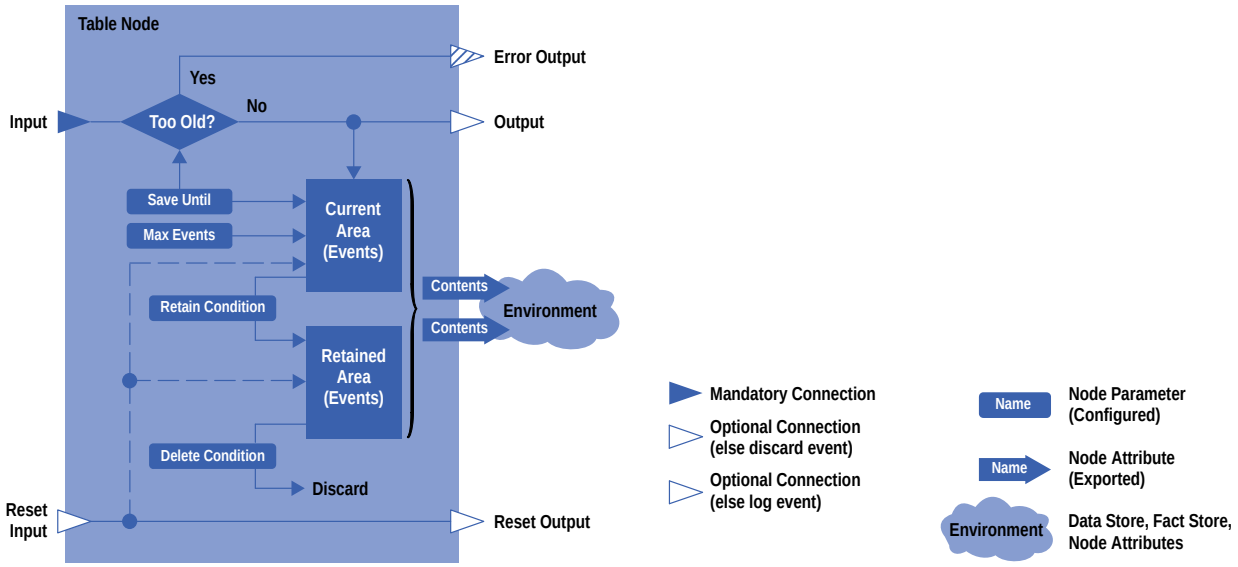


Fig. 1. *Table node.*

The current area stores each event until the creation time of the event is more than save until seconds before the current time of the correlation engine, or until the number of events in the region exceeds max events. Each event arriving at the input port is tested to see if the creation time of the event is less than save until seconds before the correlation engine's current time. If it is, the event is added to the current area (subject to the max events limit), and immediately output via the output port if connected, or discarded if the port is not connected. If the creation time of the event is more than save until seconds before the correlation engine's current time, the event is not added to the table, and is either output via the error output port if connected, or logged if the port is not connected.

When an event is to be retired from the current area (based on age or volume), the condition specified in the retain condition parameter is evaluated. This is a Boolean expression that can take the retiring event as an argument and can access any environment data from throughout the circuit. If the expression evaluates true, the retiring event is logically moved to the retained area. If the condition evaluates false, the event is discarded. Events are retained in the retained area until they meet the condition specified in the delete condition parameter. The condition is tested for all events in the retained area whenever an event arrives, or at each correlation engine clock cycle. Any event for which the condition evaluates true is silently discarded.

An event entering the reset input port causes all stored events to be silently discarded and the count attribute to be set to zero. The reset event is immediately output via the reset output port.

Fact Store and Data Store

The data store and fact store permit business, topology, and operating factors specific to a device or a set of circumstances in a managed network to be separated from the general correlation rules defined by the correlation circuit. The behavior of the correlation circuit can be changed by updating the data and fact stores as local conditions change without affecting the integrity of the correlation circuit. This makes it possible to develop correlation circuits that are data-driven, promoting circuit reusability and reliability and design generality.

The data store contains a set of name-value pairs. Any user-defined names can be used to identify the assigned values. In a circuit, the value can be referenced using the configured name. If the reference is by a static node parameter, the reference will be resolved at circuit load time. For dynamic parameters, references are resolved every time an event triggers activity at the node.

The fact store contains triples: thing1-relation-thing2. A relationship can be any user-defined concept, such as is_contained_in, is_the_parent_of, is_equal_to, is_gzumped_by, and so on. The related things can also be anything the user requires in the circuit, such as switch1, rack17, cabinet10, circuitABC, and so on. This means that a fact such as equipment10 is_contained_in rack27 can be defined. In a circuit node, a condition parameter can test whether this relationship is true and take appropriate action if it is.

The fact and data stores are loaded from files into memory. This model conforms with the notion that the run-time engine is designed for very high speed—faster than normal file I/O speeds. The stores can be updated while the correlation engine is running, resulting in dynamic changes to the correlation rules.

Annotation

The annotate node (Fig. 1) is special because it is the only node other than the source and sink nodes that interacts with the external environment. It is necessary to create an external *annotate server* to service the annotation requests. When an event arrives at an annotate node, it causes the engine to generate a CMIP event (an EcsAnnotateRequest notification) which is transmitted to the annotate server via the HP OpenView DM postmaster (pmd) and the XMP (X/Open® Management Protocol) application programming interface. The server must have registered with HP OpenView DM to receive the annotate request event. Any data from the incoming event, or from elsewhere in the circuit, can be output with the request. This data is used to parameterize the request. The annotate server must perform some user-implemented action or inquiry to obtain the information required by the request. The server will return the obtained data to the requesting annotate node with another CMIP event (an EcsAnnotateResponse notification) issued by the server (acting as an agent entity). The response must be returned within a configured time limit or the request will time out. Any required data can be returned (subject to the limits of the protocol). All ECS data types are supported (string, integer, real, time, duration, Boolean, list, tuple, etc.), including any combination of these types. The data in both the request and the response is specified as a list. Since all requests use the same CMIP event, it is necessary for the designer of the annotate server to specify some mechanism to differentiate between requests. The response event will include the requesting node name and request ID provided in the request event. These are used to route the response to the requesting annotate node.

The end user must create the annotate server. Examples of the server's agent and manager functions are provided as source code along with guidelines.

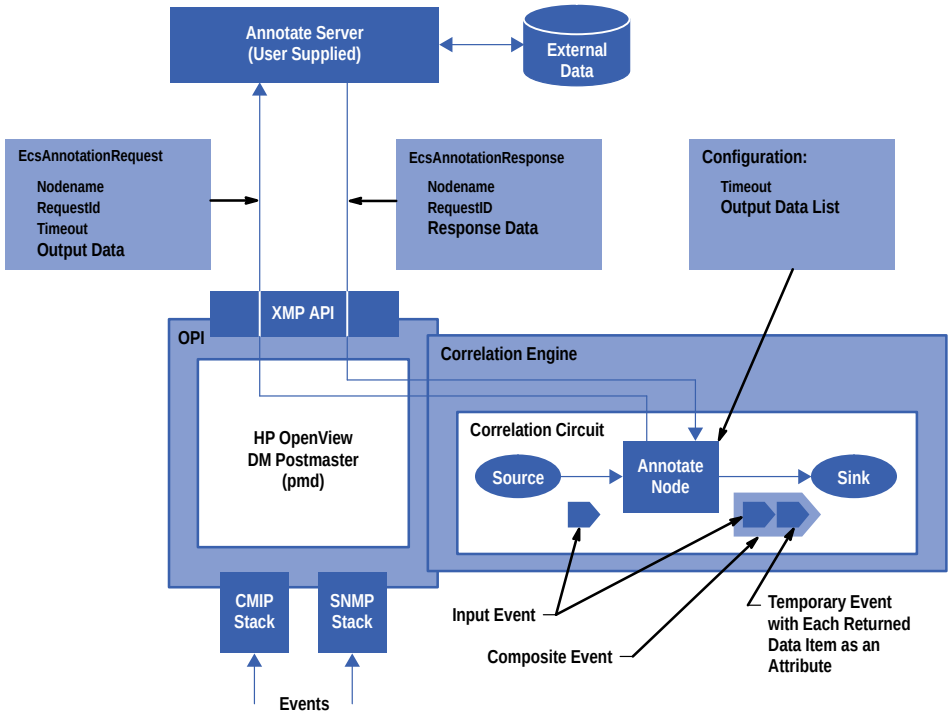


Fig. 1. *The annotate mechanism.*

A Modeling Toolset for the Analysis and Design of OSI Network Management Objects

To deal with the complexity of network management standards and the increasing demand to deploy network management applications quickly, analysts and designers need a set of tools to help them quickly and easily model, define, and develop new network management objects.

by Jacqueline A. Bray

The HP GDMO (Guidelines for the Definition of Managed Objects) Modeling Toolset (GMT) is the first tool in the HP OpenView TMN (Telecommunications Management Network) developer tool chain (Fig. 1). This toolset consists of a set of integrated tools designed to aid developers in the analysis and design of OSI network object models. The key components of the toolset are a graphical modeling tool, import and export facilities, and a conformance report generator. The tools operate independently of other HP OpenView products so that network specifiers can work independently of implementers. This article provides an overview of the network modeling process, GDMO, and the modeling tools.

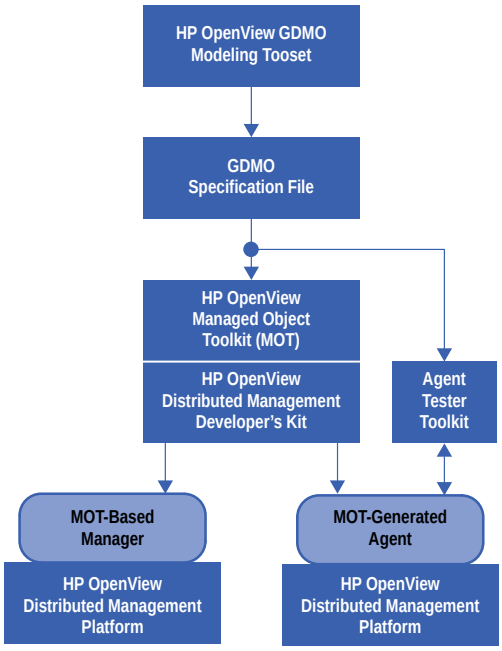


Fig. 1. *HP OpenView TMN developer tool chain.*

The Modeling Process

The first step in developing a network object model is the analysis of the environment to be managed. The network and system resources to be managed are identified and their characteristics and the operations that can be performed upon them are defined. The managed resources might be physical (e.g., a router or workstation) or logical (e.g., a software process). A managed resource might also represent a collection of different resources. Other elements might be managed that are not actually resources but are required to support management functions, such as an event log. These requirements are translated into a GDMO object model, with the managed resources represented as managed objects. The managed objects define the interface to a managed resource.

GDMO

The Guidelines for the Definition of Managed Objects is an ISO standard (ISO/IEC 10165-4 (ITU X.722))[1] that defines how network objects and their behavior are to be specified, including the syntax and semantics. This specification language allows network object designers and manager/agent implementers to communicate designs and build upon existing designs. GDMO is an object-oriented environment, using the concepts of inheritance, containment, and encapsulation. It is used to define:

- Managed object classes for managed resources
- Attributes and behaviors of a managed object
- Operations that can be performed on an attribute or object
- Notifications (events) an object might issue
- Relationships with other managed objects
- The names of object instances.

GDMO is organized into templates, which are standard formats used in the definition of a particular aspect of the object, with rules for how these templates refer to each other. A complete object definition is a combination of interrelated templates. There are nine of these templates.

- *Managed object class templates* define a model for managed object instances that share the same characteristics. The inheritance relationships with other managed object classes are specified, along with the packages that define the class characteristics.
- *Package templates* are groups of logically related sets of behaviors, attributes, attribute groups, actions, notifications, and parameters. With each attribute is a property list of valid operations (Get, Replace, Add, and Remove), initial values, and other value characteristics.
- *Behavior templates* describe, in textual form, the behavior of a component.
- *Attribute templates* define an actual data element of an object, including its syntax and behavior.
- *Attribute group templates* define a set of attributes to allow operations to be performed on the group as a whole.
- *Action templates* define additional operations for a managed object that cannot be modeled using the standard operations defined in the package template.
- *Notification templates* define unsolicited events that may be sent by the agent.
- *Parameter templates* define error conditions specific to the object and extend the definition of information used by actions and notifications. The context within which this parameter can be used is specified.
- *Name binding templates* define where an object may be located in the global containment tree, along with the attribute used to distinguish object instances. These templates also specify rules for the creation and deletion of the object instances.

An example of each of these templates can be found in ***Appendix A***, which shows a portion of a GDMO definition for a UNIX® password file (i.e., /etc/passwd). The details of the information to be exchanged between the manager and agent are defined using ASN.1 (Abstract Syntax Notation One).[2] ASN.1 is a formal description language used to define data types to be exchanged between systems. It includes primitive data types, such as integer and Boolean, and allows new data types to be constructed from these types. The data types are grouped into one or more ASN.1 modules within a GDMO definition. In the example in Appendix A, there is one ASN.1 module named PasswordFileInfo. The GDMO templates reference ASN.1 data types by prefixing the data type with the ASN.1 module name (e.g., PasswordFileInfo.LoginNameSyntax in the loginName attribute template).

Other ISO standards also relate to the definition of management object models. For example, The Management Information Model[3] is a companion document that defines modeling concepts, principles of naming and relationships, and scoping and filtering. Another example is the standard for the Definition of Management Information.[4] This standard defines, in GDMO and ASN.1, a set of managed object classes to be used as superclasses. It includes an object class named *top*, from which every other managed object class ultimately derives. The other classes form an inheritance hierarchy, with top as the root. The object class top includes attributes for object instance naming, which the other classes inherit. The set of rules for defining managed objects is referred to as the *Structure of Management Information*.

The Toolset

The GDMO modeling toolset stores the GDMO and ASN.1 definitions in an object dictionary, which acts as a central repository for all the tools (see Fig. 2). The toolset allows concurrent access to the tools and object dictionary and can be configured as a client/server architecture. GDMO and ASN.1 definitions are organized within documents. An import facility allows external standard and user-defined GDMO document files, such as ITU-T X.721, to be loaded into the object dictionary. (X.721 and other GDMO standards are included with the toolset.) New object classes can inherit from any object class in the object dictionary and reference other templates and data types for consistency and reuse of specifications. Within the toolset, each document has a short alias name to simplify references to documents. Object definitions can be added to new or existing documents.

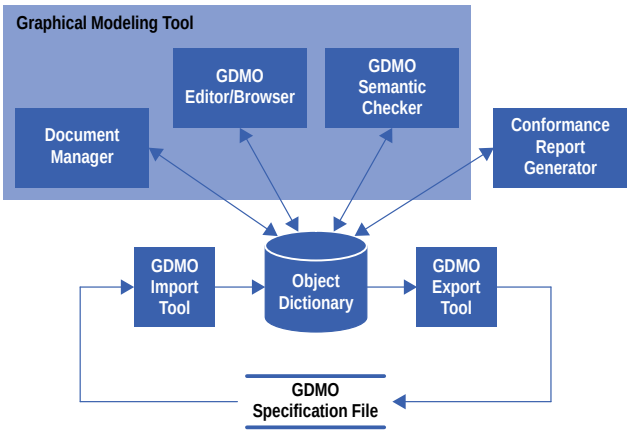


Fig. 2. *The GDMO modeling toolset.*

The graphical modeling tool can be used to learn the syntax of the GDMO language, to explore existing GDMO documents, and to create new ones. A template window exists for each of the nine GDMO template types. These windows show the details of an existing template or guide users in creating new templates. For example, Fig. 3 shows the template for a managed object class, with the GDMO keywords along the left (requiring no entry) and the specific entries for this managed object class entered in the table. Clicking the name of one of the referenced templates, such as the Characterized By Package template, and clicking on the Details button, displays the window for that particular template. The Package template window displays entries for several template types, including Attributes. Clicking on Details for an attribute in that window would display its Attribute template. In this way, successively more detailed information can be viewed until the lowest level, the ASN.1 definition, is reached.

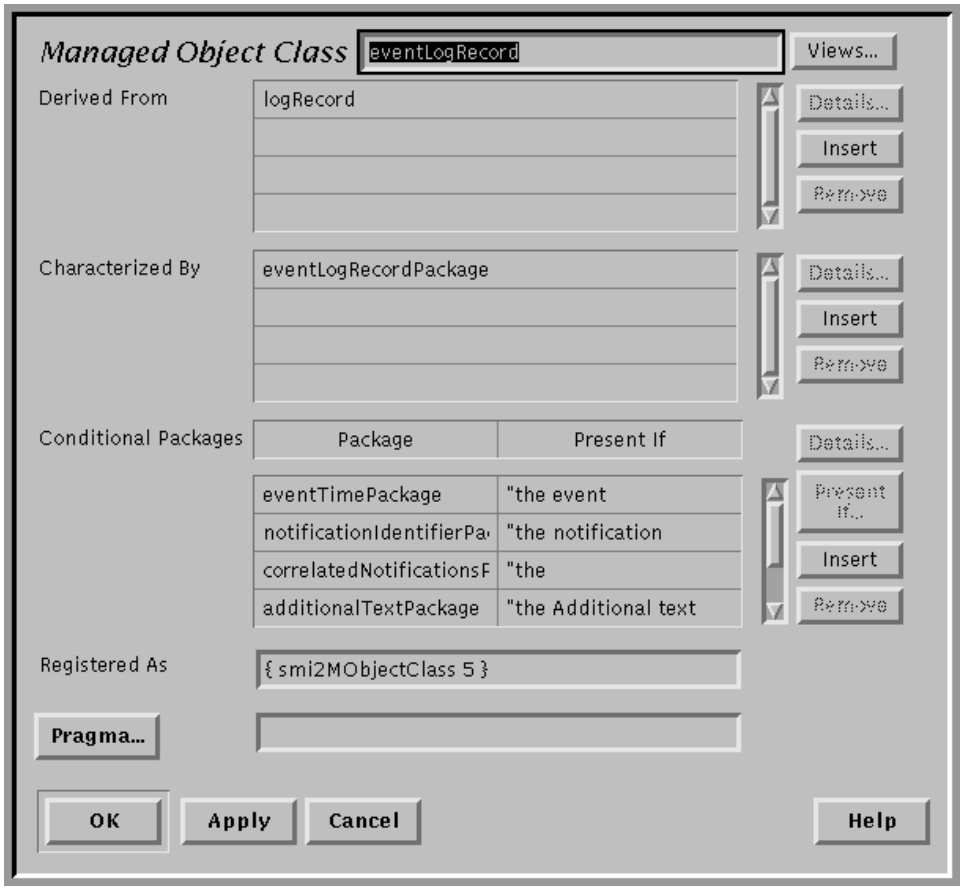


Fig. 3. *The window for a Managed Object Class template.*

Browsers for each of the template types can be invoked to display a list of all available entries of that type. A filter function is available in the browsers to display subsets of the complete list. Template entry names can be copied into another template window without retyping the name by selecting an entry with the mouse and then clicking the Insert button in the

appropriate template. The Details button described above also functions in the same way while in the browser windows. This detailed drill-down capability is available throughout the tool.

Two useful features for cross-reference checking the object model are the Viewpoint and Inherited Characteristics options. Clicking the Viewpoint button, available on all template and Viewpoint windows, displays the viewpoint for a selected item. In the Viewpoint of Managed Object Class window in Fig. 4, the selected object is represented by the vertical bar, the templates on the left reference the selected object, and the templates on the right are referenced by the selected object. The boxes on either side of the vertical bar contain the appropriate GDMO keywords. Above each template name is the template type (e.g., MOC (managed object class), PKG (package), etc.). The Viewpoint window is helpful when making changes to an object to verify that those changes will not adversely affect other objects that are dependent upon it.

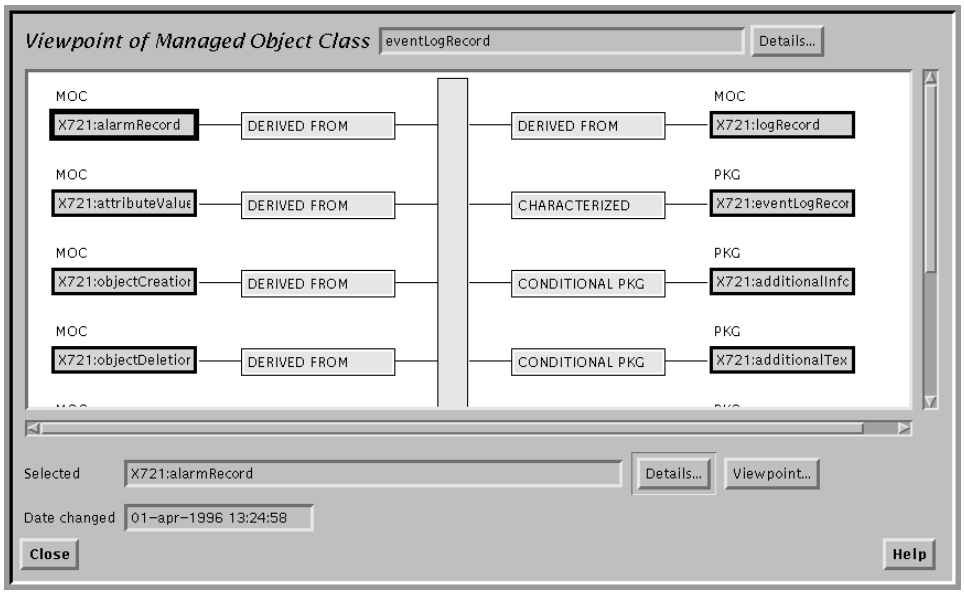


Fig. 4. *A GDMO tool for cross-reference checking: the Viewpoint of Managed Object Class window.*

Clicking Views and then the Inherited Characteristics button, available only on managed object class template windows, displays the window shown in Fig. 5. The characteristics available to a managed object class, whether specialized in that object class definition or inherited from an ancestor, can be displayed by selecting the characteristics of interest (Attributes, Notifications, etc.) from the row of buttons under Characteristics to Display and then clicking Compute. The scrolled window lists all of the inherited characteristics available and where they were referenced. Clicking on a characteristic and then the Details button displays its template.

The graphs available in the GDMO toolset represent three distinct and independent tree structures used in OSI system management. These graphs are the inheritance graph, the registration graph, and the name binding graph. The inheritance graph (Fig. 6) shows the inheritance hierarchy of all the managed object classes in a selected GDMO document, along with any superclasses derived from other documents. (All of the tool windows handle referencing across documents. When a template is referenced from another document, the template is prefaced with the document alias.) Object class nodes can also be added or deleted on this graph. GDMO and the GDMO toolset both support multiple inheritance, which allows classes to inherit properties from more than one superclass.

The registration graph (Fig. 7) shows part of the registration tree of object identifiers defined in ITU-T Recommendation X.721.[4] An object identifier is a unique ASN.1 data type that is a sequence of nonnegative integers representing a particular object. GDMO describes the registration tree structure adopted in the OSI system management standards for allocating globally unique identifiers to components of managed object definitions.[5] Objects can be registered via the registration browser or the registration tree. Registration is typically done in the last phase of GDMO modeling, when document definitions are stable.

The name binding graph (Fig. 8) displays the containment relationships defined via the name binding template. The name binding template specifies a subordinate (contained) object and a superior (containing) object, along with an attribute of the subordinate object that will be used to name instances of that class. The name binding template also specifies whether object instances can be created and deleted via remote management, along with any limitations on those actions. For example, it may specify that an object instance can be deleted via remote management, but only if that object instance does not contain other objects. This containment hierarchy represents the structure of the Management Information Base (MIB). It shows the objects an agent contains and the hierarchy and containment of those objects, which are used not only to define the MIB structure but also as a means of unambiguously referencing object instances.[6]

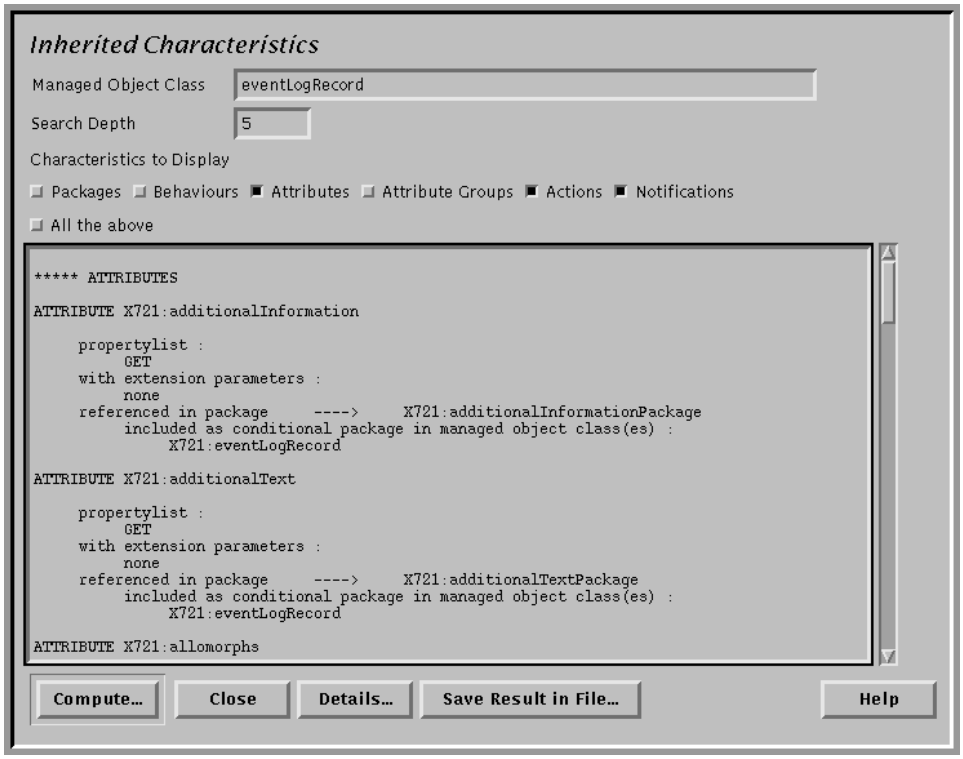


Fig. 5. *A GDMO tool for cross-reference checking: the Inherited Characteristics window.*

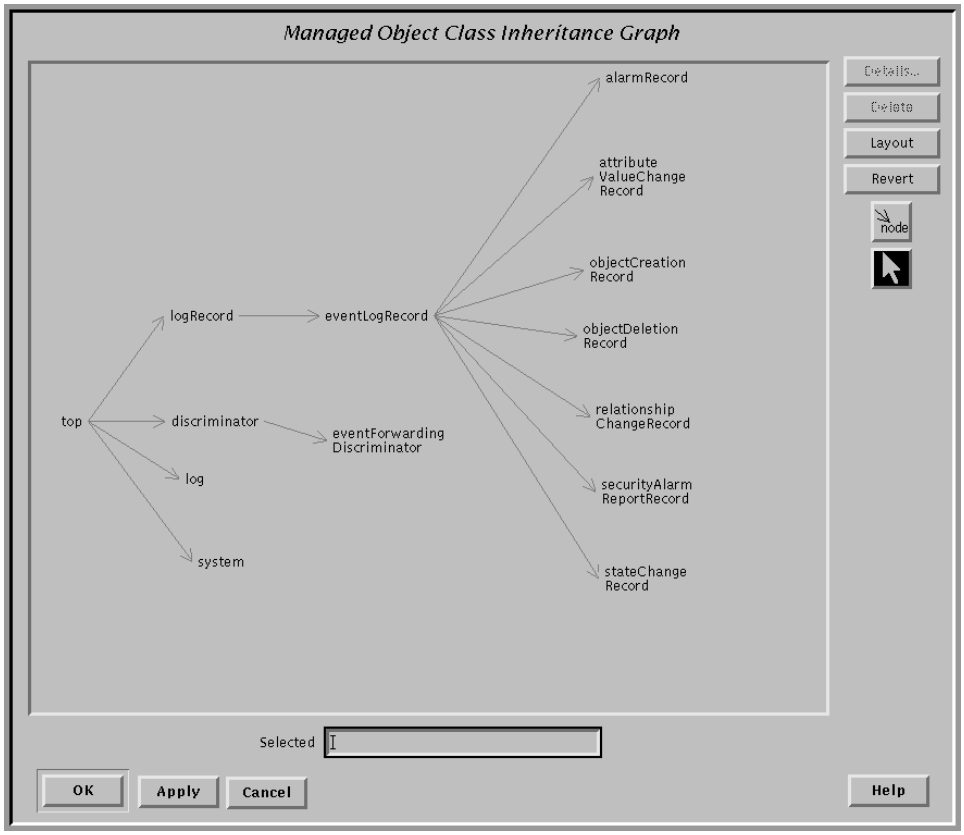


Fig. 6. *A managed object tree structure in a Managed Object Class Inheritance Graph window.*

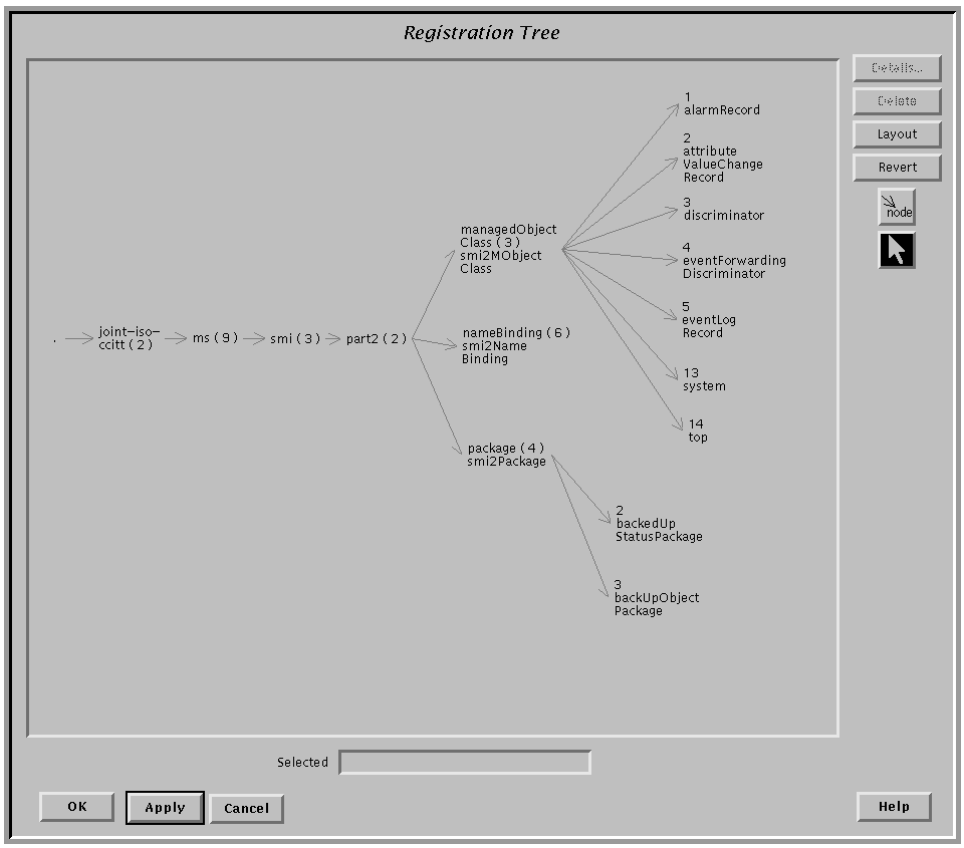


Fig. 7. *A registration tree graph.*

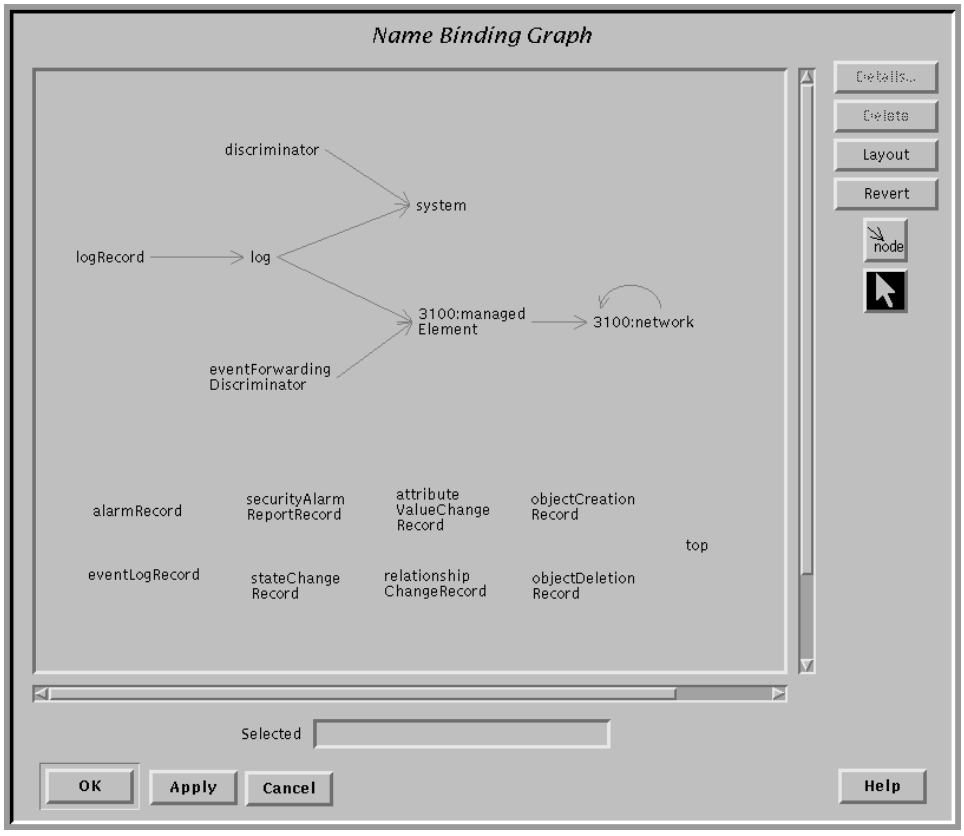


Fig. 8. *A name binding graph.*

Once the GDMO document is complete, the semantic checker verifies that the specifications are complete and correct. It checks references throughout the object dictionary, including the detection of templates that are defined but unreferenced. The document can then be exported to an ASCII file for subsequent use by code generators, such as the HP OpenView Managed Object Toolkit and other tools. The ASCII file contains both the GDMO definitions and ASN.1 modules. The Managed Object Toolkit is described in ***Article 6***.

The remaining tool, the conformance report generator, generates printed reports that conform to the ISO standard: *Requirements and Guidelines for Implementation Conformance Statement Proformas Associated with Management Information* (ISO/IEC 10165-6 (ITU–T X.724). Fig. 9 is an example of a proforma. The designer annotates the components in the report with any additional information that will be needed by the agent developer implementing the components. The agent writer will indicate in the proforma report the level of conformance to the definitions.

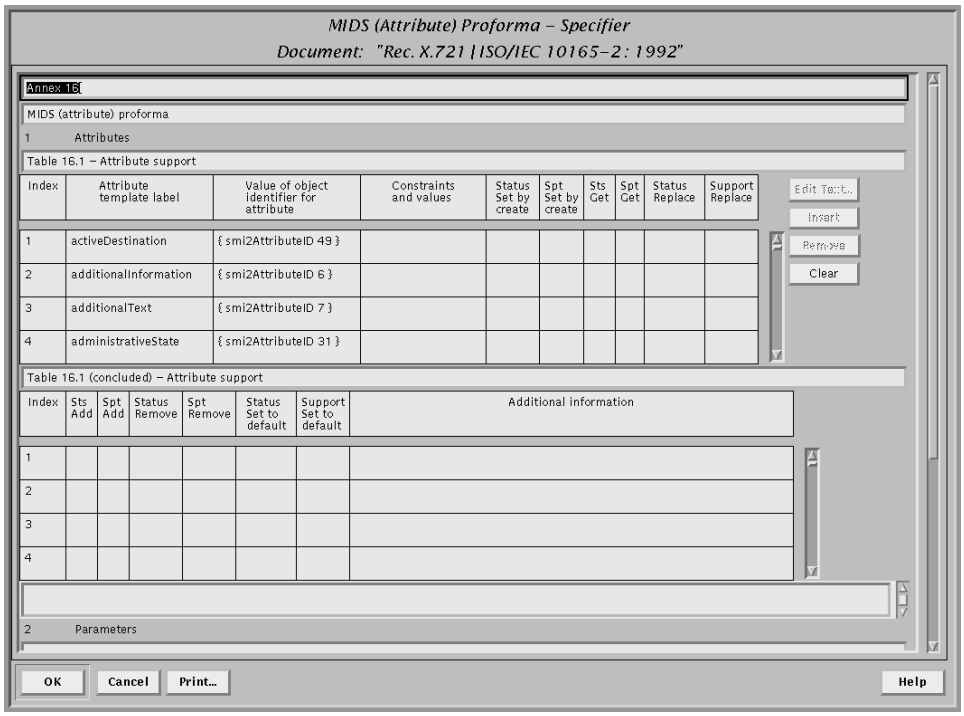


Fig. 9. *An example of a proforma, which is the output from the GDMO developer toolkit's conformance report generator.*

Conclusion

Developing a network object model and the associated GDMO specifications is a complex process. The GDMO Modeling Toolset aims to reduce the complexity and time involved in this definition by providing intuitive graphical tools and object model verification.

Acknowledgments

The author would like to acknowledge the contributions of many individuals who participated in the development and deployment of the GDMO toolset, including Paul Stoecker and Mark Smith for their technical contributions.

References

1. *Guidelines for the Definition of Managed Objects,* ITU-T Recommendation X.722, 1992.
2. *Specifications of Abstract Syntax Notation One (ASN.1),* ITU-T Recommendation X.208, 1993.
3. *Management Information Model,* ITU-T Recommendation X.720, 1993.
4. *Definition of Management Information,* ITU-T Recommendation X.721, 1992.
5. J. Westgate, *Technical Guide for OSI Management,* NCC Blackwell Limited, 1992.
6. W. Stallings, *SNMP, SNMPv2, and CMIP, The Practical Guide to Network-Management Standards,* Addison-Wesley Publishing Co, 1993.

Appendix A: A Portion of a GDMO Definition for a UNIX Password File

```
passwordEntryManagedObjectClass MANAGED OBJECT CLASS
  DERIVED FROM
    "Rec. X.721 | ISO/IEC 10165-2 : 1992":top;
  CHARACTERIZED BY
    passwordEntryPackage;
REGISTERED AS { passwordMOCObjID 1 } ;

passwordEntryPackage PACKAGE
  BEHAVIOUR                      passwordEntryPackageBehav;
  ATTRIBUTES
      loginName
          GET,
      password
          INITIAL VALUE          PasswordFileInfo.passwordInitVal
          GET-REPLACE,
      ...
  ATTRIBUTE GROUPS
      passwordEntry;
  NOTIFICATIONS
      passwordEntryWasCreated,
      passwordEntryWasDeleted;
REGISTERED AS { passwordPkgObjID 1 } ;

passwordEntryPackageBehav BEHAVIOUR
  DEFINED AS "This is a simple agent/manager designed to
  manipulate entries in a UNIX password file, /etc/passwd. ...";

loginName ATTRIBUTE
  WITH ATTRIBUTE SYNTAX          PasswordFileInfo.LoginNameSyntax;
  MATCHES FOR                    EQUALITY;
  BEHAVIOUR                      loginNameBehav;
  PARAMETERS                     loginNameErrorParam;
REGISTERED AS { passwordAttrObjID 1 } ;

passwordEntry ATTRIBUTE GROUP
  GROUP ELEMENTS
      loginName,
      password,
      ...
  DESCRIPTION "This is the mechanism whereby an entire password
               entry will be referenced in a single call.";
REGISTERED AS { passwordAttrGroupObjID 1 } ;

readObjectsFromDisk ACTION
  BEHAVIOUR                      readObjectsBehav;
  PARAMETERS                     readObjectsErrorParam;
  WITH INFORMATION SYNTAX        PasswordFileInfo.FileNameSyntax;
  WITH REPLY SYNTAX              PasswordFileInfo.SuccessSyntax;
REGISTERED AS { passwordActionObjID 1 } ;

passwordEntryWasCreated NOTIFICATION
  BEHAVIOUR                      passwordWasCreatedBehav;
  WITH INFORMATION SYNTAX        PasswordFileInfo.LoginNameSyntax;
REGISTERED AS { passwordNotifyObjID 1 } ;

loginNameErrorParam PARAMETER
  CONTEXT                        SPECIFIC-ERROR;
  WITH SYNTAX                    PasswordFileInfo.LoginNameError;
  BEHAVIOUR                      loginNameErrorBehav;
REGISTERED AS { passwordParamObjID 1 } ;

passwordEntryNameBinding NAME BINDING
  SUBORDINATE OBJECT CLASS       passwordEntryManagedObjectClass;
  NAMED BY SUPERIOR OBJECT CLASS passwordRootManagedObjectClass;
  WITH ATTRIBUTE                 loginName;
```

```
  BEHAVIOUR                     passwordEntryNameBindingBehav;
  CREATE                        WITH-REFERENCE-OBJECT;
  DELETE                        DELETES-CONTAINED-OBJECTS;

REGISTERED AS { passwordNameBindObjID 1 } ;
...
-- ASN.1 Definitions

PasswordFileInfo            { 1 3 6 1 4 1 11 1001 1 1 }
DEFINITIONS ::= BEGIN

maxNameLength            INTEGER ::= 8
LoginNameSyntax          ::= GeneralString (SIZE(maxNameLength))
...
passwordBaseObjID  OBJECT IDENTIFIER ::= { 1 3 6 1 4 1 11 1001 }
passwordAttrObjID  OBJECT IDENTIFIER ::= { passwordBaseObjID 2 }
...
passwordMOCObjID   OBJECT IDENTIFIER ::= { passwordBaseObjID 8 }
...
END

-- The passwd example is provided with various HP OpenView TMN products.
```

A Toolkit for Developing TMN Manager/Agent Applications

Developing manager and agent applications for telecommunications network management that conform to standards can be a time-consuming task because of the number of APIs and data types involved in dealing with network data and protocols. The HP OpenView Managed Object Toolkit aids and accelerates the development of these TMN applications.

by Lisa A. Speaker

Telecommunications Management Network (TMN) application developers have to implement large, complex solutions to manage today's heterogeneous and distributed telecommunication networks. Telecommunication service providers (carriers) rely on interoperability standards to integrate and deploy these solutions in a heterogeneous environment. Developing these solutions to conform to standards is a time-consuming task.

This article will provide an overview of the challenges involved in developing OSI-based TMN applications and then will describe the HP OpenView Managed Object Toolkit, which can be used to accelerate the development of TMN applications.

Background

Network equipment providers and network service providers historically have developed their own proprietary solutions for managing their telephone network equipment. Today, however, network equipment may come from different providers, and telecommunication network operators manage large, heterogeneous, and distributed networks. Thus, the main objective for standards being developed for managing telecommunications networks is to provide a framework for telecommunications management that promotes interoperability. By introducing the concept of generic network models for management, it is possible to perform general management of diverse equipment using generic information models and standard interfaces.

In 1977, the International Organization for Standardization (ISO) recognized the necessity for standards to enable the widespread use of communications networks and, as a result, established a subcommittee to initiate the standardization process. Because of the complexity of the environment, they concluded that no single standard would be sufficient. Rather, they decided that the communication functions should be partitioned into more manageable components and organized as a communications architecture. This architecture would then form the framework for standardization.[1]

The essential elements of the model were developed quickly, but the final ISO standard (ISO 7498) was not published until 1984. The International Telegraph and Telephone Consultative Committee (CCITT) issued a technically compatible version as X.200. The result is a massive set of standards referred to as OSI (Open Systems Interconnect) systems management. The ISO standards and the CCITT* recommendations continue to be developed with close collaboration.

The term OSI systems management actually refers to a collection of standards for network management that include a management service and protocol and the definition of a database and associated concepts. The first standard related to network management issued by the ISO was ISO 7498-4, which specifies the management framework for the OSI seven-layer model. Subsequently, the ISO issued a set of standards and draft standards for network management. A subset of these standards provides the foundation for TMN applications.

The TMN recommendations strive to leverage the OSI systems management standards and extend them into the telecommunications network management domain. As in OSI, the basic concept behind TMN is to provide an organized architecture and standardized interfaces, including protocols and messages, to achieve interconnection between various types of operations systems (OSs) and telecommunications equipment for the purpose of exchanging management information.

The OSI systems management standards fall into five categories:

- An OSI management framework and overview, which provides a general introduction to management concepts, including the OSI seven-layer model
- The Common Management Information Service (CMIS), which provides OSI management services to management applications, and the Common Management Information Protocol (CMIP), which provides the information exchange capability to support CMIS

* The CCITT recommendations are now called ITU-T (International Telecommunications Union-Telecommunications) recommendations.

- Systems management functions, which define the specific functions performed by OSI systems management, including fault, configuration, accounting, performance, and security management
- A management information model, which defines the Management Information Base (MIB), a database containing information about the resources and elements within the OSI environment that need to be managed
- Layer management, which defines management information, services, and functions related to specific OSI layers.

The fundamental function within OSI systems management is the exchange of management information between two entities: the managing system (the manager or requestor) and the managed system (the agent or responder) by means of a protocol (see Fig. 1). CMIS provides the services, invokable by the management process to initiate management requests, and CMIP specifies the protocol data unit (PDU) and associated procedures for transmitting management requests and responses.

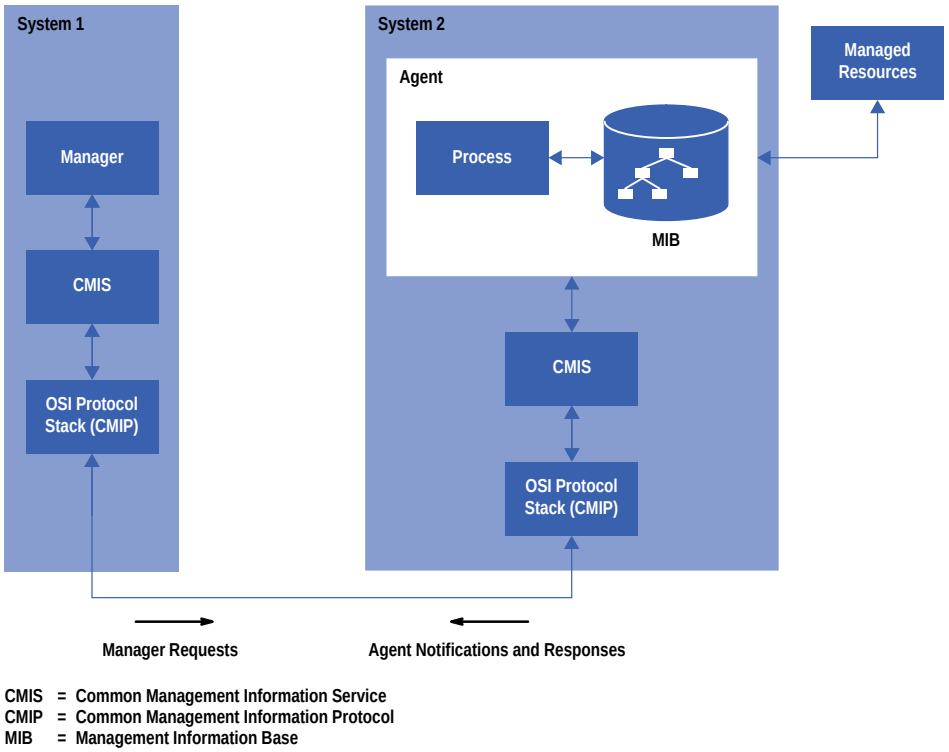


Fig. 1. *The OSI systems management architecture: the manager system and the managed system (agent).*

OSI systems management relies heavily on the concepts of object-oriented design. A *managed object class* is a model or template for managed object instances that share similar characteristics. An OSI systems management managed object class is defined in terms of its attributes, operations that can be performed upon it, notifications that it may emit, and its relationships with other managed objects. Attributes hold the data values associated with a specific managed object instance and may have a simple or complex structure. The data type for an attribute is defined using Abstract Syntax Notation One (ASN.1). The operations affiliated with a managed object class are closely associated with the CMIS services CREATE, DELETE, GET, SET, and ACTION.

A managed object class can be defined for any resource that an organization wishes to monitor or control. A single managed object class may represent a single network resource or a logical representation of many resources. Examples of hardware resources include switches, workstations, PBXs, LAN cards, and multiplexers. Examples of software resources are queuing programs, routing algorithms, and buffer management. Examples of logical resources include a network, a route, or a virtual private circuit.

An agent application provides a view of its associated managed object instances. Manager applications are able to access the managed object instance attribute values and manipulate managed object instances through the management interfaces published by each managed object class.

The foundation of any network management system is a database containing information about the resources and elements being managed. In OSI systems management, this data base is called the *Management Information Base* (MIB). The MIB is a structured collection of managed object instances, together with their attributes. The MIB is typically a multilevel hierarchy based on the managed object class containment relationships defined in the object model.

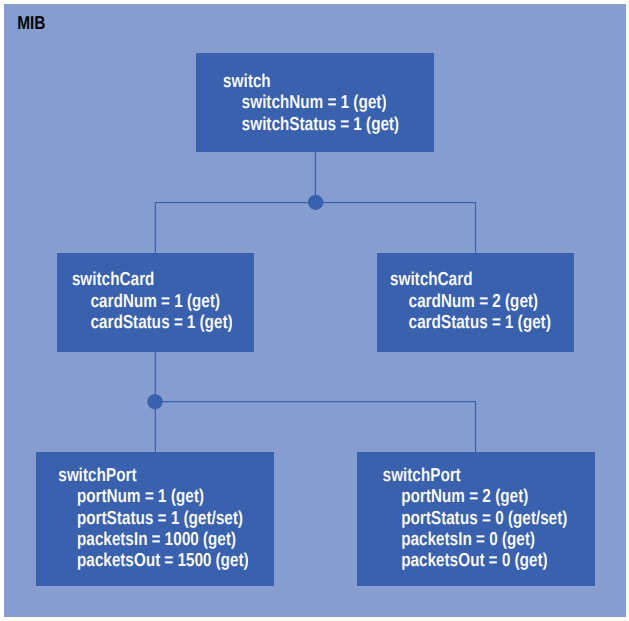


Fig. 2. *The contents of a subset of an agent's Management Information Base (MIB) showing a containment tree made up of object instances and their attributes.*

The MIB hierarchy is constructed and traversed by using the object instances' distinguishing attributes. For example, in Fig. 2 a switchPort object instance is identified by its portNum attribute, its associated switchCard cardNum attribute, and its switch switchNum attribute (switchNum = 1, cardNum = 1, portNum = 2).

The general framework within which a MIB can be defined and constructed is referred to as the *Structure of Management Information* (SMI). SMI identifies the data types that can be used in the MIB and how resources within the MIB are represented and named.

To encourage consistency between managed object definitions and to ensure the development of object definitions in a manner compatible with the OSI system management standards, the Guidelines for the Definition of Managed Objects (GDMO) (ISO/IEC 10165-4, ITU-T Recommendation X.722) has been developed. This standard provides a formal specification language for defining the interface for an OSI managed object class and the semantics for documenting the attributes and operations (behaviors) associated with a managed object class. The specification also defines the relationships among managed object classes in the management domain. See ***Article 5*** for more about GDMO.

OSI Application Development

OSI application development falls into two major categories: manager development and agent development. This section describes the development of the agent and manager applications without the assistance of a toolkit. Development with a toolkit is discussed in the next section.

Manager development involves the development of applications that issue management requests and process agent responses and notifications. Notifications are messages transmitted by agent applications when some trigger, such as a threshold or an error condition, has been tripped. Manager application developers must complete the tasks of:

- Developing the user interface through which management requests can be initiated and the status of managed objects can be viewed
- Developing the underlying communications for issuing requests and processing responses and notifications.

Agent development involves the development of the application that manages the managed object instance data, maintains its portion of the MIB, processes inbound management requests, and emits notifications as necessary. Agent application developers must complete the tasks of:

- Defining (usually in GDMO) the managed object classes associated with the resources managed by the agent application
- Developing the underlying communications for processing management requests
- Monitoring managed resources and emitting notifications as appropriate.

X/Open®, an independent, worldwide, open systems organization, provides application programming interfaces (APIs) to facilitate the development of OSI applications. A primary objective of X/Open is to promote the portability and

interoperability of OSI applications at the source-code level. For OSI systems management application developers, X/Open provides the X/Open OSI Abstract Data Manipulation (XOM) APIs and the X/Open Management Protocol (XMP) APIs.[2,3]

XOM is a C-language interface specifically designed for use with application-specific APIs that provide OSI services, such as X.400 and CMIS. The XOM API is a set of structured information objects and functions for accessing objects and shielding programmers from much of the complexities of manipulating the underlying ASN.1 data types.

XMP provides the TMN application developer with a C-language interface to the underlying management services consistent with the CMIS/CMIP and the Simple Network Management Protocol (SNMP). The XMP API is designed to be used and implemented with the XOM API. XOM objects serve as the parameters for the XMP management service functions (see Fig. 3).

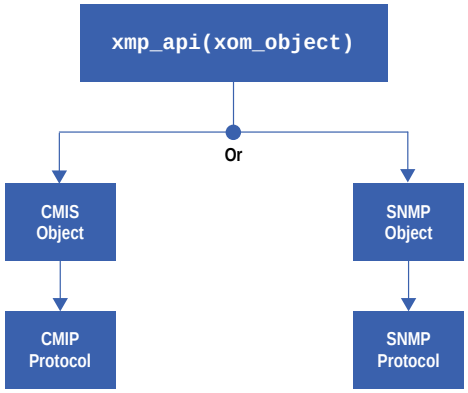


Fig. 3. *The relationship between XOM objects and XMP management functions. XMP functions use XOM objects as parameters to send the details of a manager's request via either a CMIP or an SNMP protocol stack.*

Manager applications initiate management requests and process responses returned from agent applications. XMP functions that support issuing management requests include:

- mp_create_req( ): Initiates a management request to create a managed object instance
- mp_get_req( ): Initiates a management request to get data from a managed object instance
- mp_set_req( ): Initiates a management request to set attributes in a managed object instance
- mp_receive( ): Receives the agent's response to support asynchronous requests or a notification emitted by an agent.

XOM objects, defined as C structures, must be created and passed to these functions to transfer the details of the management request. For example:

- mp_create_req( ) requires a CMIS-Create-Argument XOM object and is returned a CMIS-Create-Result XOM object
- mp_get_req( ) requires a CMIS-Get-Argument XOM object and is returned a CMIS-Get-Result XOM object
- mp_set_req( ) requires a CMIS-Set-Argument XOM object and is returned a CMIS-Set-Result XOM object.
- mp_receive( ) requires a CMIS-XXX-Result XOM object, with type depending on the response received, and is used for receiving asynchronous requests.

Fig. 4 shows the specification for the CMIS-Get-Argument XOM object class. The specification shows that the GET request XOM object class specification also contains references to other XOM object classes. To construct a CMIS-Get-Argument XOM object, all of its contained XOM objects must also be constructed. The complexities of developing applications using the XOM API can be seen. The developer is challenged with the tedious task of traversing several levels of nested XOM objects either to prepare a request or to extract data from a response.

When developing agent applications using the XMP/XOM APIs, the agent developer is responsible for creating functions to receive the request, determine the type of request (CREATE, GET, SET, ACTION, DELETE, etc.), validate the request, and process the request. In validating the request, the agent developer must determine, for example, if the request is for a valid object instance in the agent's management domain, or confirm that a SET request has been received on a modifiable attribute. A significant portion of the agent development task is in the implementation of request validation.

The agent developer must also manage the application's representation of the containment tree, which is the in-process structure holding the representation of the managed objects and their associated attribute values (see Figs. 1 and 2). As management requests are received, the agent application must operate on the associated managed object representation in the agent's containment tree to perform operations such as:

- Create new entries in the containment tree as CREATE requests are received
- Delete entries in the containment tree as DELETE requests are received

CMIS-Get-Argument XOM Object Class:

XOM Attribute	Value Syntax	
base-Managed-Object-class	Object (Object-Class)	Class of object designated as starting point of request
base-Managed-Object-Instance	Object (Object-Instance)	Identifier of object designated as starting point of request
access-Control	Object (Access-Control)	Permission and security information
synchronization	Enum (CMIS-Sync)	How to synchronize selected object instances
scope	Object (Scope)	Subtree to be searched
filter	Object (CMIS-Filter)	Characteristics to test attributes
attribute-Id-List	Object (Attribute-Id-List)	Attribute values to be returned

Attribute-Id-List XOM Object Class:

XOM Attribute	Value Syntax	
attribute-Id	Object (Attribute-Id)	Identifier for a managed object attribute

An instance of an Attribute-Id-List object will contain zero or more attribute identifiers, designating which attribute values to retrieve in the managed object instance. Each attribute identifier must be constructed and has the following form:

Attribute-ID XOM Object Class:

XOM Attribute	Value Syntax	
global-Form	String (Object-Identifier)	A registered attribute type identifier
local-Form	Integer	Integer identifier defined as part of the application context

An instance of an Attribute-Id object will designate the attribute to be retrieved either through its global-Form or its local-Form.

Fig. 4. *A specification for the CMIS-Get-Argument XOM object class.*

- Update entries in the containment tree as SET requests are received
- Traverse the containment tree when *scoped* and *filtered* requests are received.

A scoped request operates recursively on an entire branch of the containment tree, starting at a designated base managed object. A filtered request designates criteria that managed objects must have to have a management operation performed. Filters are an optional facility that the agent can provide. Scoping and filtering allow multiple managed objects to be selected and operated upon in servicing a single management request.

After processing the request, the agent developer must prepare the response by constructing the associated XOM objects and then use the XMP API to issue the management response.

The XMP API provides a collection of functions to support agent development, including:

- mp_receive( ): Receives the indication of a management request
- mp_create_rsp( ): Transmits a response to the manager's CREATE request
- mp_get_rsp( ): Transmits a response to the manager's GET request
- mp_set_rsp( ): Transmits a response to a manager's SET request

As in the manager scenario, the data received by the agent will be in the form of an XOM object, and the agent application developer must prepare an XOM object to return to the manager application after servicing the manager's request.

HP OpenView Managed Object Toolkit

As noted above, OSI systems management standards use object-oriented techniques to model applications in terms of managed objects, which represent the resources in a network. This makes object-oriented programming techniques, including the C++ programming language, a natural implementation choice to use for development, starting from object analysis to application coding.

Historically, developers implementing standards-based OSI applications have been required to implement the entire management application, agent, and manager from scratch, using the complex XOM/XMP APIs and C bindings, as described above.

However, the OSI systems management standards precisely define a generic structure and behavior that apply to all agent applications. The agent developer's task can be greatly simplified by implementing the generic pieces with reusable software components, which can be assembled to build agent applications. In addition, GDMO is provided for formally defining the specific managed object class attributes and interfaces. Given a GDMO specification, it is possible to define a C++ class mapping representation of the GDMO managed object classes.

These two fundamental philosophies form the foundation for the HP OpenView Managed Object Toolkit (MOT). The Managed Object Toolkit supplies:

- An object-oriented agent application framework that provides the general-purpose, reusable software components that make up the generic aspects of an OSI agent application (An application framework is a generic application that can be tailored to meet specific requirements. It handles all generic operations that are common to all applications in a specific domain. The agent framework is provided as a library with the Managed Object Toolkit.)
- A GDMO-to-C++ and an ASN.1-to-C++ code generator that provides the OSI application developer with a C++ interface for the implementation of the specific behaviors of a managed object in an agent application and C++ classes for preparing management requests in a manager application
- An agent application generation capability that merges the generic framework and the specific GDMO-based generated C++ classes into an operational agent application.

Agent application development, which previously had taken programmers weeks, or even months, to code using the XMP/XOM APIs can now be generated in a matter of minutes or hours. The Managed Object Toolkit allows agent developers to focus on the implementation-specific details of their applications, since they no longer need to be concerned with the tedious task of using the XMP/XOM APIs to implement the CMIS communications model.

The Agent Development Process

Using the Managed Object Toolkit, agent development is accelerated with a simple five-step process (see Fig. 5):

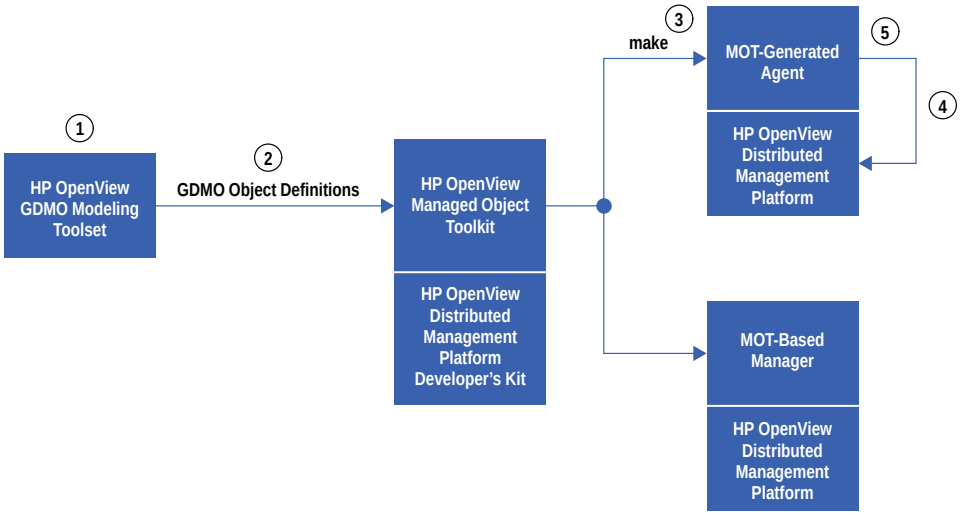


Fig. 5. *The process flow for agent and manager development using the Managed Object Toolkit (MOT).*

1. Define the GDMO managed object class specifications. The HP OpenView GDMO Modeling Toolset (GMT) can greatly simplify the design of managed object classes by providing a graphical, forms-based interface for defining and browsing GDMO managed object class definitions. It also provides a graphical representation of the object class inheritance and managed object naming hierarchies. The HP OpenView GDMO Modeling Toolset is described in ***Article 5***.

2. Submit the GDMO object model to the Managed Object Toolkit code generator. This will generate C++ class representations of the GDMO managed object classes, an agent main( ) function, makefiles to compile the agent executable, and an object registration file for the HP OpenView Distributed Management Platform's managed object directory service. The HP OpenView Distributed Management Platform is described in ***Article 1***.
3. Run the provided makefile, which compiles an operational "default" agent, so-called because it implements default behaviors for the managed object interfaces.
4. Register the agent's objects with the HP OpenView Distributed Management Platform's object directory service.
5. Run the agent application.

The executing agent is now ready to accept management requests, including CREATE, DELETE, GET, SET, and ACTION. Since the agent application is also tightly integrated with the HP OpenView Distributed Management Platform, it is able to leverage platform services immediately, such as the object registration service for object location transparency and event routing through the event management services. See ***Article 4*** for more about event management.

Managed Object Toolkit Agent Capabilities

The Managed Object Toolkit agent framework handles all aspects of the underlying management request and response communications between agent and manager, relieving the programmer of significant coding effort, including:

- Receiving CMIS requests (CREATE, DELETE, GET, SET, ACTION, etc.), and routing them to the appropriate CMIS service handler
- Validating requests and transmitting standard CMIS errors when invalid requests are received (For example, receiving a GET request on an attribute in a nonexistent managed object instance will generate a standard NO-SUCH-OBJECT-INSTANCE error message.)
- Creating and managing the agent's management information tree, which holds the managed object instances and their attributes representing the resources managed by the agent
- Supporting scoped and filtered requests in which a single request can be routed to several managed object instances
- Preparing and transmitting responses, including packaging multiple replies in response to a scoped request
- Emitting standard CMIS notifications when managed object instances are created, or deleted, or when an attribute value is modified.

By providing the CMIS communications handling infrastructure, the Managed Object Toolkit frees the programmer from the tedious task of implementing code for processing management requests and responses and allows the developer to focus on the value-added specific agent functionality.

Customizing the Managed Object Toolkit Agent

Because specific managed object behaviors cannot be completely specified using GDMO, the Managed Object Toolkit agent framework and the generated classes cannot fully implement managed objects on their own. The framework provides a default object behavior, and the Managed Object Toolkit C++ code generator provides a code skeleton for the implementation. The complete agent application is built by generating the agent skeleton code from the GDMO specification and then customizing the generated code stubs (C++ methods). Developers can also integrate Managed Object Toolkit-provided classes to implement communications with external devices (supported through standard file descriptors) and implement a cooperative multitasking agent.

For example, if a managed object class called switchPort includes an attribute called packetsOut which was defined in the GDMO specification to be "gettable" (see Fig. 2), the Managed Object Toolkit generates a file called MOC_switchPort.cxx and includes an empty method called get_packetsOut( ):

```
MOC_switchPort.cxx (filename)
virtual void Mot_switchPort_C::get_packetsOut(OVmotMoGetResultC & result_r)
{
}
```

If the developer wishes to override default behavior of the CMIS GET request for the packetsOut attribute to query a register in the associated physical device, the get_packetsOut( ) method is easily customized. This method includes a response parameter, to which the programmer assigns a response value, and the Managed Object Toolkit agent framework will appropriately package the response and return it to the requesting manager application.

The Managed Object Toolkit provides significant value for processing requests, especially if a GET packetsOut request is part of a scoped request, which is a single management request that will operate, potentially, on several managed object instances in the agent. The agent application must route the request to all of the relevant managed object instances, process the request, gather each of the individual responses, package them, and return them to the requestor. With the XOM/XMP APIs, the agent developer is required to implement the entire process, gather all of the responses, package them, and transmit the multiple

responses to the requestor. With the Managed Object Toolkit, the agent developer is only required to implement the handlers for each individual request. Since the agent framework manages all aspects of the request and response communications, the framework will track the scoped request, route it to all appropriate object instances in the Managed Object Toolkit-managed containment tree structure, collect all of the individual responses appropriately, and transmit the composite response to the requestor. The Managed Object Toolkit-based agent developer is required to implement only the actual details of each attribute's request handler.

Implementing an ACTION operation provides another good scenario. The CMIS ACTION service provides a general purpose object interface for implementing any operation in a managed object instance. For example, an action may be defined to reset a port on a switch. As in the GET scenario, the Managed Object Toolkit generates an empty C++ method for the programmer to fill in the specific details for servicing the ACTION request. But unlike the GET scenario, the standards cannot specify the appropriate response to an ACTION operation, and GDMO does not provide syntax for the specific implementation of an ACTION operation. Therefore, it is entirely the agent developer's responsibility to provide the implementation details associated with an ACTION request.

The following generated method provides the programmer with the ACTION information the management application passed along, and as in the GET scenario, an empty result object that the programmer fills in to transmit the agent's response. The MOT generates a file called MOC_switchPort.cxx, which includes the empty method resetPort_action( ).

```
Moc_switchPort.cxx (filename)
virtual void Mot_switchPort_C::resetPort_action(const Mot_resetPort_InfoC * actinfo_p,
                                        OVmotMoActResultC & result_r)
{
}
```

Again, the Managed Object Toolkit provides significant value, requiring the programmer to implement only the details of the action handler and assign the action response, allowing the programmer to disregard implementing any of the CMIS communications processing.

Managed Object Toolkit Agent Framework

In typical object-oriented design philosophy, the agent framework can be decomposed into several supporting frameworks (see Fig. 6). Each subframework, implemented as a class library, provides a particular category of functionality that contributes to the overall agent request processing task.

The agent framework is provided as a library with the Managed Object Toolkit and is made up of the following components:

CMIS Service. This class library provides classes that enable convenient access to the CMIS services. It contains base classes that define the CMIS services and subclasses that implement the CMIS services using the XMP API.

CMIS Transactions. This class library provides classes that implement the incoming CMIS requests. It provides an agent application with the functionality to process the receipt of CMIS CREATE, DELETE, GET, SET, and ACTION indications.

Containment Tree Framework. This class library provides an infrastructure for developing a containment tree representation. It also provides concrete classes that implement an in-memory representation of the containment tree.

Management Information Syntax Framework. This class library provides the infrastructure for developing C++ classes that represent syntaxes specified in ASN.1 (e.g., attribute values, action information, and action reply syntaxes). It provides base classes for representing ASN.1 syntaxes.

The Managed Object Toolkit C++ class generator generates C++ classes representing ASN.1 types defined in the GDMO specification. These C++ classes are derived from the base classes provided in the management information syntax framework (see Figs. 2 and 7). Fig. 7 illustrates how the Managed Object Toolkit generates C++ classes for GDMO-defined attributes. All toolkit-generated attributes will be derived from the OVmotAttC C++ class provided by the Managed Object Toolkit.

Managed Object Framework. This class library provides an infrastructure for developing managed object implementations. It provides C++ base classes for representing managed object instances and managed object metadata.

The Managed Object Toolkit C++ class generator will generate C++ classes representing the GDMO object classes defined in the GDMO specification. These C++ classes will be derived from the base classes provided in the managed object framework. The C++ inheritance hierarchy reflects the GDMO specified inheritance hierarchy. For example, Fig. 8 shows the C++ inheritance hierarchy for the GDMO-defined switchPort managed object class. In the GDMO specification, the switchPort managed object class is derived from the top managed class. Notice the similarity in the GDMO-defined inheritance hierarchy and the C++ inheritance hierarchy generated by the Managed Object Toolkit.

The managed object framework uses C++ classes from the management information syntax framework to represent attribute, action, and notification syntaxes.

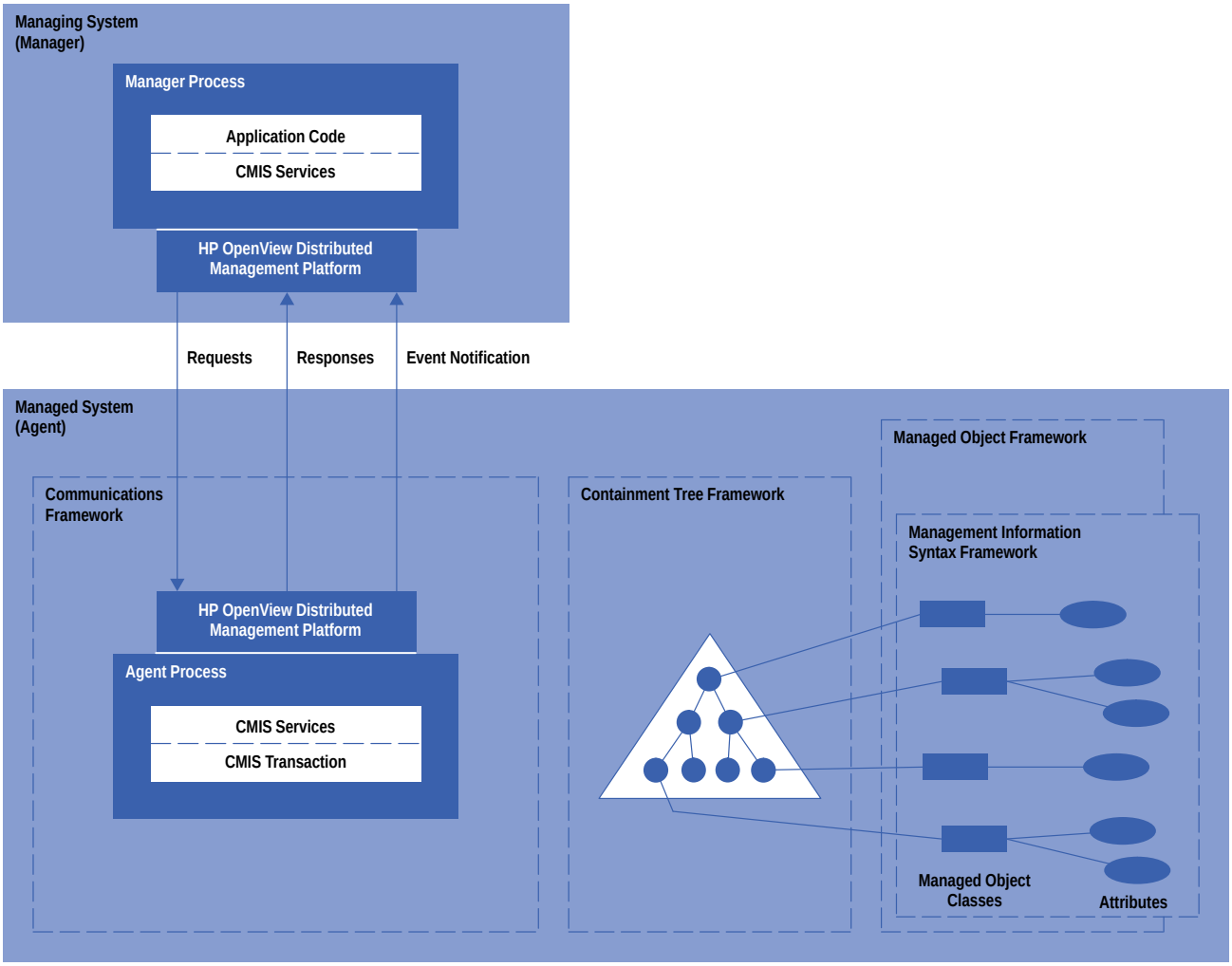


Fig. 6. *Managed Object Toolkit frameworks.*

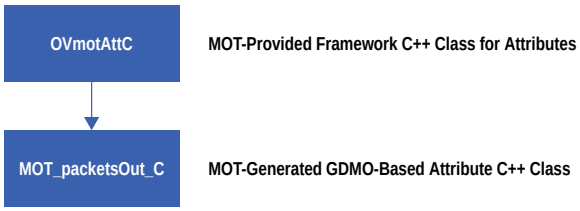


Fig. 7. *The inheritance hierarchy of C++ classes from the base classes provided in the management information syntax framework.*

CMIS ASN.1 Types. This class library provides specializations of the ASN.1 framework classes for representing CMIS arguments (e.g., CMIS-Create-Argument and CMIS-Get-Argument).

Communications Framework. This class library provides an infrastructure for developing C++ classes that are responsible for controlling communication with external devices. It coordinates between objects responsible for different communication endpoints (file descriptors) using an event-driven environment, which encapsulates the handling of the UNIX® select( ) function. The library also includes concrete classes for handling communication with the HP OpenView Distributed Management Platform process management functions.

The communications framework also provides an abstract tasking class, based on USL's (UNIX System Laboratories) task library, which can be leveraged to implement a cooperative multitasking application. Each task is cooperative, in that it owns control of a process until it exits or explicitly gives up control. The Managed Object Toolkit CMIS transactions are a collection of concrete task classes developed for processing CMIS requests. The communications framework classes allow the developer to create a pseudo multitasking, event-driven interface for communicating with external devices.

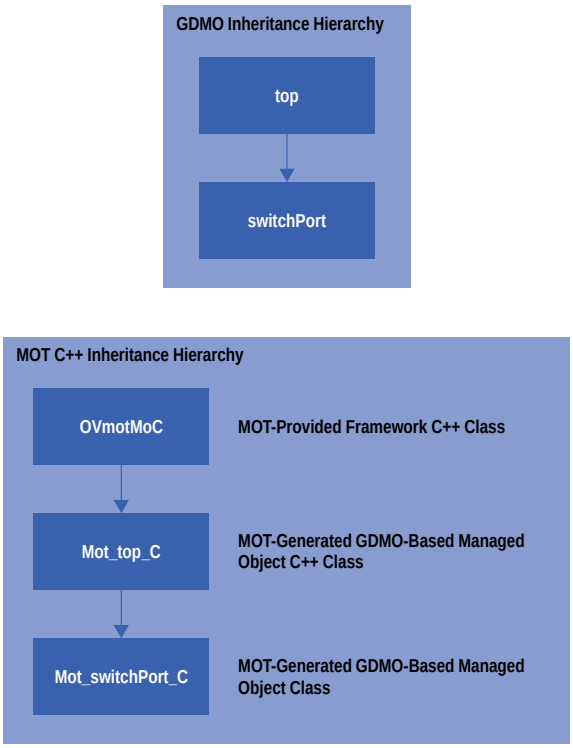


Fig. 8. *The C++ GDMO inheritance hierarchies of the GDMO-defined switchPort managed object class.*

Common Application Environment. This class library provides facilities for multilevel tracing and logging.

Foundation Types. This class library provides classes for representing common data structures such as lists and strings and classes for memory management and reference counted objects. It is based on the OSE class libraries for C++.

Using The Frameworks

The class libraries are leveraged by the Managed Object Toolkit agent library for processing manager requests. Many of these classes are also externally visible, allowing application developers to leverage them for their own agent development needs.

For example, when a CMIS GET request is received, the agent's communications framework will receive and identify the get-indicate, and then construct and initiate a Managed Object Toolkit-defined CMIS get-transaction object instance. This get-transaction object will manage the overall processing of the GET request, interacting with the containment tree framework, the managed object framework, and the management information syntax framework to complete the processing of the GET request. (The C++ object class representation of the managed object instance and the GET handler for this request are contained in the managed object framework, while the syntax associated with the attribute value is held in the management information syntax framework.)

An agent developer desiring to implement a multitasking interface to external entities, such as devices or databases, can derive a user-defined task class from the Managed Object Toolkit's abstract tasking base class. The application developer then constructs and initiates tasks in the application code, as in the following code fragment.

```
myTask.hxx (filename)
   myTaskC:public OVmotTxnC
   {
    public:
          myTask( );
          execute( );
         ...
    private:
         ...
    };
 myTask.cxx (filename)
   myTaskC :: execute( )
   {
       // provide code for task implementation
   }
```

The following code shows the construction and invocation of the above task in one of the CMIS service handling routines.

```
MOC_switchPort.cxx (filename)
 virtual void Mot_switchPort_C::get_packetsOut(OVmotMoGetResultC & result_r)
 {
        myTask   atask;  // construct a task
        atask.execute( );// run task
 }
```

The communications framework provides a communication coordinator that encapsulates the UNIX select( ) interface. The communication coordinator is used by the Managed Object Toolkit agent framework to receive and process incoming CMIS requests.

The application developer can also use the communications coordinator to process nonCMIS-oriented communications within the agent application. An example of this would be communicating to an external device through a serial interface. The agent developer need only register the opened file descriptors with the Managed Object Toolkit-provided communications coordinator and implement the associated communications handler (read, write, and exception). Then through the registration and callback mechanism, the communication endpoint processing code will be executed when the file descriptor triggers select( ) as in the following code.

```
somecc.hxx (filename)
  SomeCC : OVmotCC
  {
  public:
        SomeCC( );
        ~SomeCC( );
        void doRead( int fd );
  private:
        int fd; //File descriptor associated
                //with this communication
                //interface
  };
```

```
  somecc.cxx (filename)
  SomeCC::SomeCC ( )
  {
       fd = open (...); //open a file descriptor
       OVmotCoordC::registerCC (fd,OVmCoordC::OVMOT_KE_READ,this)
  // register file descriptor with MO commnications Coordinator
  // register for read operations, callback to
  // this->doRead( ) when data is on fd
     }
```

```
  SomeCC :: ~SomeCC ( )
  {
  OVmotCoordC::deRegisterCC ( fd,OVmotCoordC::OVMOT_KE_READ, this);
  close (fd);
  // deregister file descriptor with MOT Communications
  // Coordinator and close file descriptor
  }
```

```
  SomeCC :: doRead ( int fd )
  {
  // Receive and process the data buffered on
  // the file descriptor
  }
```

When data is sensed on this open file descriptor, the agent's communication coordinator will call the doRead( ) method, processing the data on the communications interface.

Developing Manager Applications

Unlike the agent development process, the Managed Object Toolkit does not generate an executable manager application (see Fig. 5). For manager developers, the Managed Object Toolkit provides an intuitive C++ interface which encapsulates the complexities of XOM object manipulations and assists the manager developer in the management communications implementation aspect of manager applications.

Manager developers use the XMP API to issue requests and leverage the Managed Object Toolkit to build the XOM object parameters required by the XMP API (see Fig. 3). The manager developer has access to:

- Managed Object Toolkit-provided CMIS service classes to build request objects and parse response objects
- Managed Object Toolkit-provided convenience classes, which represent the underlying components of the CMIS request objects, including C++ class representations for:
  - Fully distinguished names
  - Attribute identifier lists
  - Attribute lists
  - Base managed objects for scoped requests
  - Filters
- Managed Object Toolkit-provided stream-based classes for transforming C++ request objects to XOM request objects and XOM response objects to C++ response objects
- Managed Object Toolkit-generated GDMO-based C++ classes representing the managed object classes
- Managed Object Toolkit-generated ASN.1-based C++ classes representing the syntaxes associated with the managed object attributes, actions, and notifications.

The following scenario is an example of a Managed Object Toolkit-based manager GET request. Note that classes that begin with OVmot are Managed Object Toolkit-provided classes and classes that begin with Mot_ are Managed Object Toolkit-generated classes originating from the GDMO specification.

Scenario: Issue a scoped GET request for all of the "UP" ports on a specific card in a switch and return the in and out packet counts across ports that have traffic. Note the following containment relationship (see Fig. 2):

- A switch contains cards.
- A card contains ports.

1. Construct each of the attributes that make up the fully distinguished name for a card in a switch.

```
Mot_switchNum_C   switchNum(100);
Mot_cardNum_C   cardNum(10);
```

   This code fragment assigns switchnum to 100 and cardnum to 10.

2. Construct the fully distinguished name for the port base managed object of the request.

```
OVmotDnC        dn;
dn << switchNum << cardNum;
```

3. Construct the list of attribute identifiers associated with the attribute values to be retrieved.

```
OVmotAttIdListC   attr_ids;
attr_ids << Mot_portStatus_id <<
Mot_packOut_id << Mot_packetsIn_id;
```

4. Construct the base managed object identifier of the port associated with the request. Request processing to begin at the switch card with cardNum = 10 associated with the switchNum = 100.

```
OVmotBaseMoIdC    base_mo_id (Mot_switchCard_id, dn);
```

5. Construct the filter. This code fragment looks for instances where the values of packetsOut and packetsIn are greater than zero and the portStatus is "UP."

```
OVmotFilterC      filter(Mot_portStatus_id== 1
              // 1 denotes UP
              &&( Mot_packetsOut_id >  0
              ||Mot_packetsIn_id  >  0 ));
```

6. Construct the Managed Object Toolkit GET argument.

```
OVmotGetArgC      get_arg(base_mo_id,
              OVMOT_NIL//Omit Access Control
              OVMOT_CMISSYNC_BEST_EFFORT,
              OVMOT_SCOPE_WHOLE_SUBTREE,
              & filter,
              attr_ids );
// print to standard output
cout << "C++ Constructed Get Argument" << get_arg << endl ;
```

7. Build the XOM CMIS-Get-Argument from the Managed Object Toolkit C++ GET argument.

```
OM_object      xom_object;
OVmotOXomStrC get_strm (XomWorkspaceP -> qWorkspace());
get_strm << get_arg;
xom_object = get_strm.qAdoptXomPrivObj();
```

8. Issue a standard XMP get request.

```
mp_status =mp_get_req(Session,
         MP_DEFAULT_CONTEXT, xom_object,
         &result, &invoke_id);
```

Without the Managed Object Toolkit-provided and Managed Object Toolkit-generated classes the manager developer would be faced with the challenge of constructing the XOM CMIS-Get-Request object passed in the mp_get_req( ) function, a task that could require at least six times as many lines of code.

Summary

Developers who build telecommunications network management applications are implementing large, complex solutions and telecommunication service providers rely on interoperability standards to integrate and deploy these solutions in a heterogeneous networked environment. Developing applications that communicate via the standard CMIS and CMIP communication interfaces has historically been an extremely complex and time-consuming task using the XMP/XOM APIs. The HP OpenView Managed Object Toolkit offers the developer significant assistance in this task by helping to transform a GDMO specification into an executable, extensible agent application and providing an intuitive C++ interface for implementing agent behaviors and manager applications.

Acknowledgments

The author would like to acknowledge the contributions of the many individuals who participated in the development and deployment of the HP OpenView Managed Object Toolkit, including Tom Burris, Mark Smith, and Dan Rice who assisted in the design, build, and test processes.

References

1. W. Stallings, *SNMP, SNMPv2, and CMIP, The Practical Guide to Network Management Standards*, Addison-Wesley Publishing Company, 1993
2. *X/Open CAE Specification, Systems Management: Management Protocols API (XMP)*, X/Open Company Limited. 1994.
3. *X/Open CAE Specification, OSI Abstract Data Manipulation API (XOM)*, X/Open Company Limited and X.400 API Association, 1991.

Bibliography

1. *Managed Object Toolkit Technical Evaluation Guide*, Hewlett-Packard, 1995.
2. *Principles for a Telecommunications Management Network*, ITU-T Recommendation M.3010, 1992.
3. *TMN Interface Specification Methodology*, ITU-T Recommendation M.3020, 1992.
4. *TMN Management Functions*, ITU-T Recommendation M.3400, 1992.
5. *Information Technology—Open Systems Interconnection—Structure of Management Information: Management Information Model*, ITU-T Recommendation X.720 (ISO/IEC 10165-1), 1992.
6. *Information Technology—Open Systems Interconnection—Structure of Management Information: Guidelines for the Definition of Managed Objects*, ITU-T Recommendation X.722 (ISO/IEC 10165-4), 1992.

A Software Toolkit for Developing Telecommunications Application Components

To be effective, application developers must understand the data available to their applications, the operations required to access the data, and the steps required to turn their understanding into an implementation. A prototype development environment has been built that helps the developer explore and understand the data in the Management Information Base (MIB) and construct and deploy pieces of TMN management applications.

by Alasdair D. Cox

Telecommunications network operators own the largest distributed computing systems in the world. Their networks carry enormous volumes of traffic, much of which is highly valuable. Maintaining service is essential. The penalties for failure are great, and not just financial—emergency services and some air traffic control transmissions use the same telephone network. Not surprisingly, network operators have considerable systems and network management needs.

The ability to provide new services is becoming vital in the telecommunications business. Speed and flexibility are key requirements, not only of the initial implementation and its deployment, but also of the management systems that ensure that service continues to operate efficiently. The rapid development of effective management systems is therefore a major concern.

The applications that telecommunications companies use to manage their equipment, networks, and services they provide are known as *operations support systems*, or OSS. An established network operator will have hundreds of existing applications and a continuing need to develop more as their systems and technologies change.

The development of new applications in the Telecommunications Management Network[1] (TMN) area is still carried out with the aid of a C or C++ compiler. The developer must understand the data that is available to the application, the operations that can be performed to reach it, and the application program interfaces (APIs) and tools available to support those operations.

HP OpenView products provide support in a number of these areas. The GDMO Modeling Toolset (***Article 5***) helps the application developer understand the kind of data that is stored in the TMN world. The OpenView Distributed Management platform (***Article 1***) provides standard APIs that the developer can use to send CMIP (Common Management Information Protocol) messages to the data. The Managed Object Toolkit (***Article 6***) provides further support to the C++ programmer.

In this paper we describe a prototype development environment that addresses some of the demands of application development in the telecommunications world. This prototype helps the user explore the available management data and make enough sense of it so that the user can construct and deploy pieces of management applications.

Background

The Telecommunications Management Network is an attempt to standardize the management of telecommunications networks. It consists of a set of existing and evolving recommendations from the International Telecommunications Union's Telecommunications Standardization Sector, known as the ITU-T.[2] These recommendations are based on a number of previous recommendations on Open Systems Interconnection (OSI) systems management, now adopted as international standards.[3]

The OSI systems management standard proposes a Management Information Base (MIB),[4] which is a collection of data necessary for managing a network. This data is organized hierarchically and related by containment. The data is in the form of objects, called managed objects, which are defined by the Guidelines for the Definition of Managed Objects.[5]

The Guidelines for the Definition of Managed Objects, or GDMO, is the language used to define the structure and some of the relationships between managed objects. The GDMO definition is in the form of templates used to define managed object classes (classes in standard object-oriented terminology), attributes (instance variables), actions (methods), notifications (events that can be emitted by objects), and name bindings, which specify the ways in which objects can be related by containment in the MIB. See ***Article 5*** for more about GDMO.

The Common Management Information Service (CMIS)[6] is used to interact with the MIB, and the Common Management Information Protocol (CMIP)[7] is the way service messages are encoded for transmission between TMN management applications and the MIB.

The available services include getting information from managed objects, changing their values, making method calls, and creating and deleting managed objects. In addition, managed objects can emit events.

The Development Environment

We believe that telecommunication management applications of the future will be composed of a number of large-grained, distributed objects. We call these objects *application components*. Application components differ from managed objects in that, at their simplest, managed objects represent logical and physical parts of a network. Application components, on the other hand, are pieces of the system that manages the network and the services running on the network, and they may use, manipulate, and create managed objects as they work.

Some components will be specialized for a particular management function while others will be of a more general nature and may provide services to more than one application. Applications will need access to data in a number of sources, including other applications, traditional databases, and the MIB.

We believe that the parts of the application that act as data bridges will be split into components, each capable of supporting transactions to a data source. Since our intention is to support the development of telecommunication management applications, we decided to focus on the construction of application components that interact with data, and thus we have concentrated on the TMN MIB.

We have built a prototype development environment that includes three tools that operate together to support the development life cycle of application components (see Fig. 1). In the initial stages of using the prototype, this means providing aid in understanding the problem and progressing through to enabling the implementation, testing, and deployment of the solution. By taking this approach we believe the development process for application components can be greatly improved.

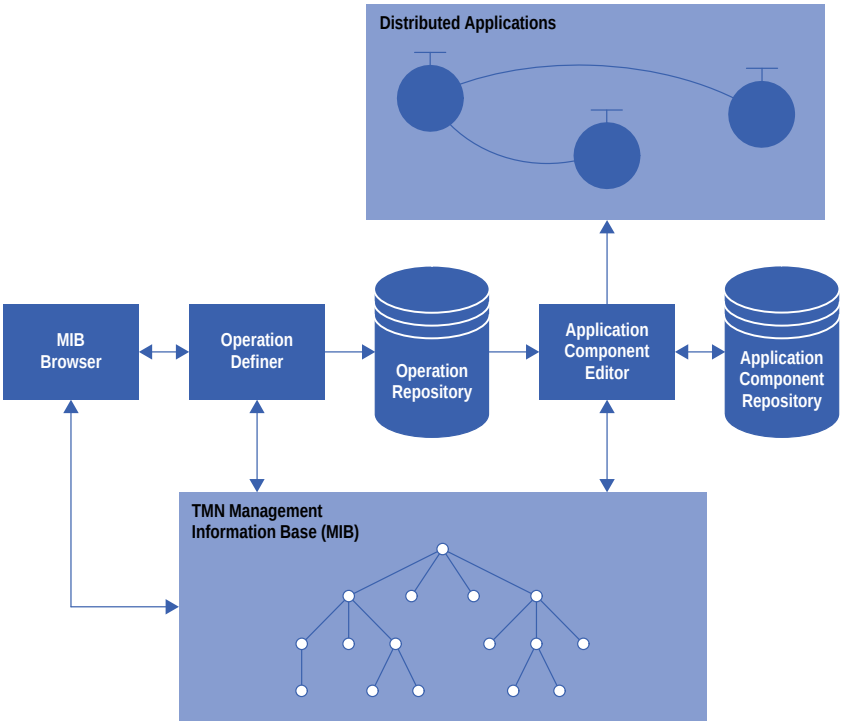


Fig. 1. *Prototype TMN development environment.*

Although the process is likely to be iterative, the basic steps in developing an application component using the TMN prototype environment shown in Fig. 1 are:

- Navigation through and exploration of the MIB using the *MIB browser* to build an understanding of the data and the way it is used
- Prototyping CMIS operations using the *operation definer*, which helps to expand or verify the developer's understanding (The results of executing these operations may be fed back into the browser to aid navigation.)

- Storing operations away for future reuse or as documentation aids
- Construction of fragments of application functionality from a number of operations using the *application component editor*
- Deployment of the completed components as distributed CORBA* (Common Object Request Broker Architecture) or OLE (Object Linking and Embedding) objects or as source code for inclusion in libraries or directly into applications.

The use of a common underlying architectural framework is a major reason why the prototype appears and behaves as an integrated development environment rather than as a set of standalone tools. This framework is discussed later in this article.

MIB Browser

The MIB Browser provides the user with a graphical view of the MIB (see Fig. 2). By interacting with and manipulating this view, the user is able to navigate through the MIB to explore the structure and content of the data stored in its managed objects.

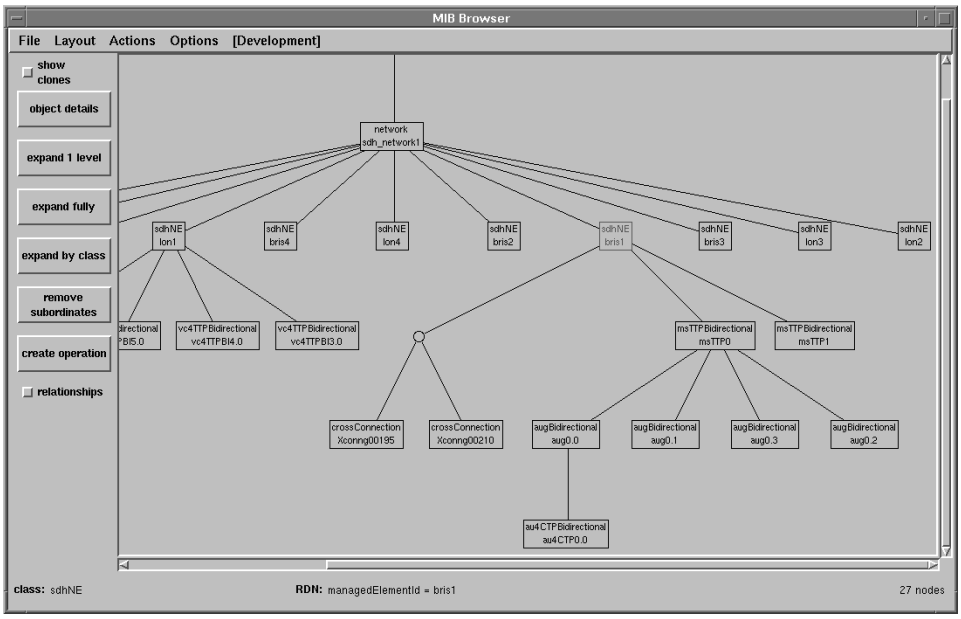


Fig. 2. *Output from the MIB browser showing a cached subset of the managed objects in the MIB.*

The view shown to the user is a cached subset of the managed objects that exist in the MIB. The contents of this cache are built up as the user navigates through the MIB. A number of simple operations are provided for this navigation, each causing a CMIS service request to be sent to the MIB. The replies to this request, which can vary from none to very many, are used to update the cache and hence the view presented to the user.

The browser uses metadata to add meaning to the presentation and to help the user navigate. For example, the names of managed object classes and attributes and the details of attribute values are presented to the user as words, rather than as the numbers that the underlying infrastructure uses. In addition, the browser is sometimes able to advise the user in advance when a navigational operation is guaranteed to find nothing new. This decision is arrived at by using metadata that describes the ways in which the MIB can be organized. The design of the MIB browser, including how the cache and metadata fit in, is shown in Fig. 3.

The MIB browser is useful to application developers and operations staff who understand the network and how it is managed. It is also useful as an educational aid for training the entire staff. Its main benefit is that it is not necessary to understand the technical details of a particular area to use the browser successfully. In fact, we find that it helps users to increase their understanding of the network.

Navigation by CMIS

The MIB browser provides five predefined CMIS operations which can be used to retrieve data from the MIB. The user is shielded from the execution details of CMIS operations by the user interface.

* CORBA is from the Object Management Group (OMG) and OLE is from Microsoft®.

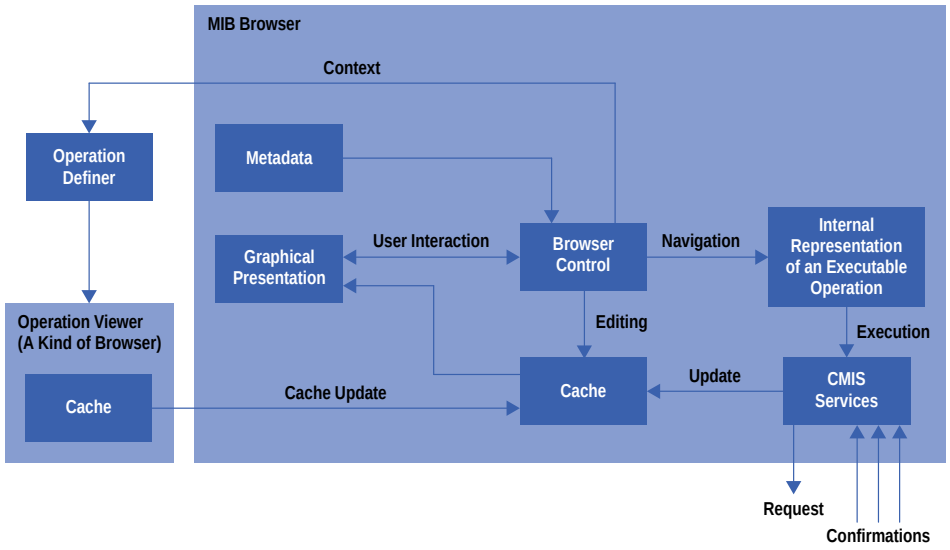


Fig. 3. *The design of the MIB browser.*

Three of the CMIS operations (expand fully, expand one level, and expand by class) are used to discover the structure of the MIB. The result of executing them is a set of managed object names. The MIB browser interface presents them as nodes on the tree displayed to the user.

A fourth operation (all attributes) is used to extract the details of a managed object. These details, or attribute values, of the managed object are stored in the browser's cache.

Finally, the fifth operation (find class) is used to find out about the managed object class of an object whose existence has been inferred from the result of an earlier operation but about which we know only the name.

We decided against making the structure-finding operations also discover the contents of the managed objects they encountered, even though this would have reduced the need for the fourth operation. There were two reasons for the decision: performance and size. It is not usually possible to predict how many responses will be returned from executing an operation. For example, a large area of the MIB may lie within the scope of an operation, possibly containing several thousand objects. Since the nature of the underlying CMIP protocol means that each object discovery results in the transmission of an asynchronous message to the browser, if an operation requested the contents of each managed object, the size of each message would increase greatly and performance would be severely affected. In addition, the user is probably not interested in the details of most discovered objects. Knowing how they are organized is often what matters. So the browser does not need to store the details of every managed object.

We realized that we could let the user decide which managed objects had interesting contents, so we provided a set of navigational operations and a drill-down* operation, for the user to execute appropriately.

The following sections describe the navigation operations in more detail, and Fig. 4 shows the values assigned to the fields in CMIS GET requests for each of the operations. ***Article 6*** provides more information about CMIS GET requests.

CMIS GET Request	Base Managed Object Class	Base Managed Object Instance	Access Control	Synchronization	Scope	Filter	Attribute ID List
Expand Fully	<From Browser Entry>	<From Browser Entry>	—	bestEffort	wholeSubtree	—	{ }
Expand One Level	<From Browser Entry>	<From Browser Entry>	—	bestEffort	firstLevelOnly	—	{ }
Expand by Class	<From Browser Entry>	<From Browser Entry>	—	bestEffort	wholeSubtree	*	{ }
All Attributes	<From Browser Entry>	<From Browser Entry>	—	bestEffort	baseObject	—	—
Find Class	actualClass [5]	<From Browser Entry>	—	bestEffort	baseObject	—	{ }

Values

```
* { or
      {
        {equality{objectClass,<user-supplied>}},
        . . .
      }
  }
```

Fig. 4. *The values submitted in CMIS GET requests to implement the MIB browser navigation operations.*

* A drill-down operation is one that enables the user to see greater detail about a managed object.

Expand Fully. This is the crudest navigational operation. It discovers all the managed objects below a specified position in the MIB. The user selects a managed object and then presses the expand fully button on the browser window. The class and name of the selected managed object provide the context for the operation. These values are used to fill in the base managed object class and instance fields of the request.

Expand One Level. This is a safer operation than expand fully in that it can be used when expand fully would be inappropriate. Rather than discover all the managed objects below the selected position in the tree, expand one level discovers only those objects immediately beneath the selected position. In MIB-speak, it finds all the managed objects contained by the base object. In computer-speak, it finds the children.

Expand by Class. In this operation the user is presented with a list of those managed object classes that could possibly have instances below the selected position in the MIB. This list is computed from the metadata (described below). The user can select the managed object classes of interest or scan the list and choose likely candidates. Prior knowledge is helpful but not essential. The choices are used to parameterize the request.

All Attributes. This drill-down operation targets a single managed object that was discovered by an earlier navigation. The values of all the object's attributes are obtained.

Find Class. This operation can be invoked on a managed object whose existence and name have been inferred from the results of an earlier operation, but whose class is unknown.

Operation Definer

The MIB browser provides the user with a small number of ways to construct CMIS service requests. While this is an advantage in terms of ease of use, it can appear limiting to users with a need for selective information. To those with a greater understanding of the area (i.e., the protocols and information models used) the operation definer gives full flexibility in the construction of CMIS requests. Operations defined this way can be used to extend the browser's repertoire. The design for the operation definer is shown in Fig. 5.

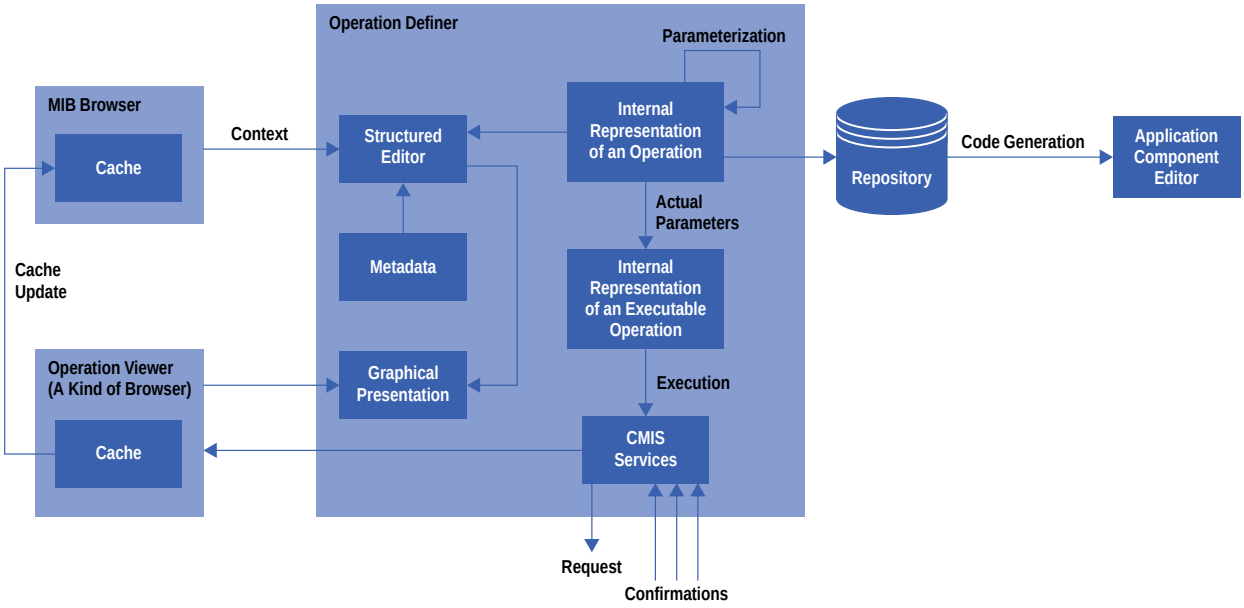


Fig. 5. *The design of the operation definer.*

As the name implies, the operation definer helps the user specify an operation to be performed on the MIB. Once its specification is completed, the operation can be sent as a service request and the corresponding results can be shown to the user using an operation viewer, which presents information in a way that is similar to the MIB browser. The user can examine the results, and if necessary, modify the operation and reissue it. This cycle can continue until the user is satisfied with the operation.

The results obtained by executing an operation can be added to the MIB browser to expand the view it presents of the MIB. In this case, the operation definer can be seen as a powerful navigational tool that augments the basic browser.

Alternatively, the real benefit of the operation definer might be the operation itself, rather than the results of executing it once. The results returned will be useful because they can help prove that an operation works correctly. The user might want to store such an operation so that it can be used again. The operation definer maintains a repository for this purpose. By adding to this repository, the user can build up a toolbox of useful operations, similar to the way a system administrator builds up a library of shell scripts.

For an operation to be executable, all aspects of its specification must be fixed. The operation definer picks up the starting point, which is the base managed object's name and class, from the browser context. All other aspects of an operation are defined by the user. Once this is done, the operation can be tested and its definition refined until the user is satisfied with it. If the finished operation is going to be used again, it is likely that the user will want it to be made more general-purpose. A stored operation can be made more general-purpose by specifying that some aspects of it become parameterized, meaning that each time it is used the values of those parameters must be supplied. This allows the effect of the operation to be tailored to the context in which it is applied. In this way, for example, it becomes possible for an operation defined on a particular named network element to be applied to any network element by supplying the appropriate name and type information.

Application Component Editor

As stated earlier, we expect that future telecommunications applications will be made up of a number of large-grained, distributed objects. We call the objects application components. Some application components will communicate with data sources, including the MIB, to obtain the information that other components will use to perform management functions. We decided to concentrate our efforts on supporting the development of application components that interact with the MIB. They play a more constrained role that is better suited to automation, and the data they interact with is described by a standard form of metadata. A window dump from the application component editor shown in Fig. 6.

An application component has four parts:

- Entry points that make up the visible interface of a component (Other parts of an application make calls to this interface to use a component.)
- Operations that interact with the MIB and issue CMIS service requests and receive results
- Support functions, or scripts, that tie together the operations to implement the required functionality
- Test functions, some of which test individual operations and some of which test the whole component.

As shown in Fig. 7, the application component editor uses the operation definer's repository. Operations stored in the repository can be selected for inclusion in components. They are translated into source code and, like a method, will perform the same operations when they are executed. The fields that are identified as parameters when the operation is stored can be treated as formal parameters to the method. In addition, simple test functions are automatically generated that test the operation with its original parameter values. The results obtained from running the test functions are presented to the developer in the same style as the MIB browser.

Although source code generation is not strictly necessary, the ability to generate source code helps make the tools acceptable to the traditional telecommunications OSS (operations support systems) developer market.

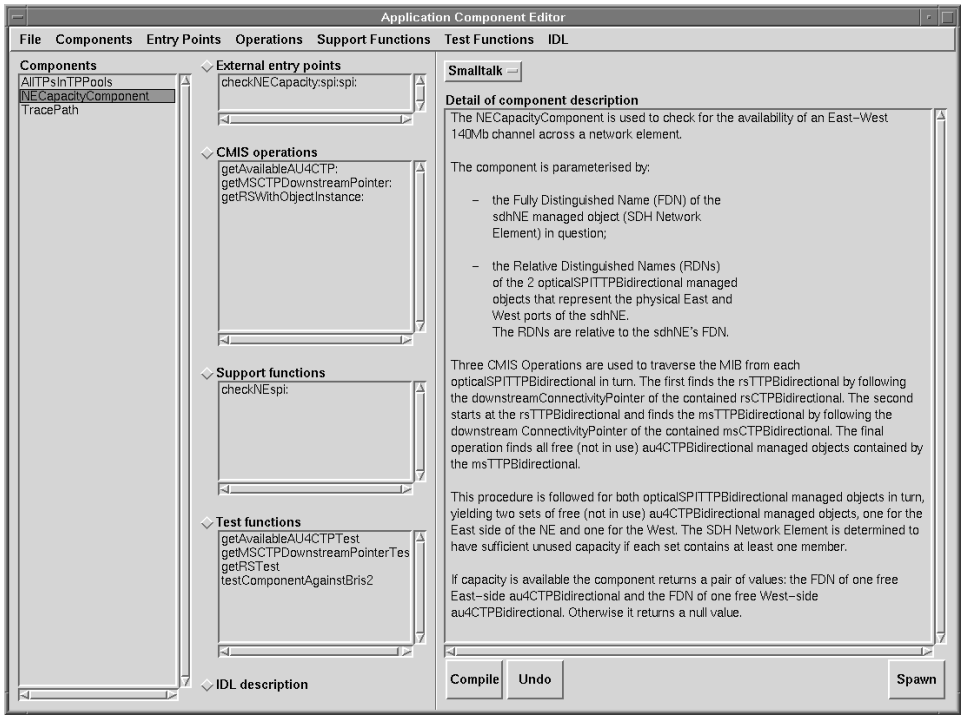


Fig. 6. *Output from the application component editor.*

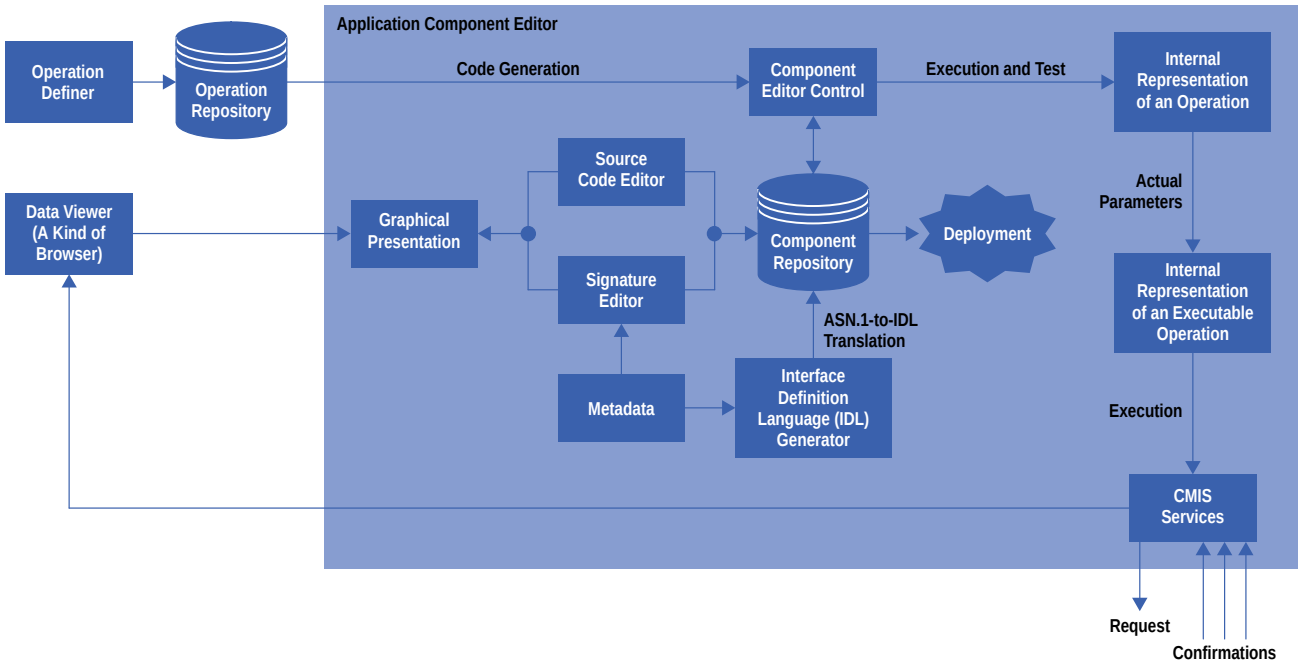


Fig. 7. *The design of the application component editor.*

The component editor is not restricted to editing new and existing components, but also provides help in deployment. There are several ways in which a completed component can be made available for inclusion in an application, such as:

- As a CORBA object
- As an OLE/COM object
- As a fragment of application source code for direct inclusion in larger application programs
- As a library routine that can be linked into a number of applications
- As part of the implementation of a managed object class's run-time behavior.

We have concentrated on the first option. The signature editor shown in Fig. 7 can be used to define formal signatures for entry points, using ASN.1 types for the parameters and results. This information is then available to the IDL (Interface Definition Language) generator which produces equivalent CORBA IDL interfaces using a standard translation algorithm.[8,9] These interfaces help the developer towards deployment of application components as CORBA objects. A similar process would enable their distribution as OLE objects.

The prototype application component editor generates Smalltalk source code. In fact, we used HP Distributed Smalltalk[10] to automate the entire process of deploying application components as CORBA objects. Although Smalltalk is increasingly being used for product development, it is usually restricted to research and prototyping work. A fully fledged tool would have to generate C or C++.

Architectural Framework

Fig. 8 shows the architectural framework upon which the three prototype tools (the MIB browser, the operation definer, and the application component editor) are built. By building on top of this framework we were able to increase commonality in implementation, appearance, and behavior among the tools.

Graphical Presentation

All the prototypes were implemented in VisualWorks Smalltalk, which meant we were able to use its interface construction tools to develop the dialog boxes, menus, lists, and buttons that make up most of the tools' interfaces. In addition, because Smalltalk is an object-oriented language, we could subclass interfaces and specialize them for particular tasks. For example, there are several browser-type interfaces used by all three tools in different ways. These were not implemented independently. Instead, we implemented the common features in a superclass, which was inherited from the supplied VisualWorks classes, and created subclasses that became the browser and viewers for displaying the results of operations and test functions. In this way the three tools share a common look and feel because much of the code is common to them all. Fig. 9 shows the inheritance hierarchy.

The managed objects in the MIB are organized into a tree called a *containment tree* because the tree's edges represent a containment relationship. This reflects the equipment-oriented origins of the OSI systems management standards. For example, networks contain equipment such as multiplexers, which contain circuit boards which in turn contain software.

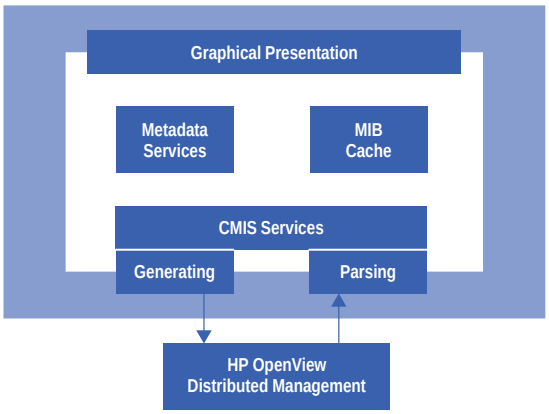


Fig. 8. *The architectural framework for the three prototype tools.*

The MIB browser enables users to navigate through the containment tree. It seemed natural to present a view of the discovered containment tree and allow the user to interact with it via buttons and menu selections. These operations cause the browser to execute CMIS operations to extend the browser in the way the user directs. The browser shows a view of the tree as it is discovered during a browsing session. We felt that users would not see the larger picture if they focused only on one managed object at a time or were presented with only a view of the path through the tree to that object. This larger picture often provides the context that helps the user understand the smaller picture.

One lesson we learned from this prototyping experience is that more flexibility is necessary when presenting information to the user. It is possible to discover quickly many hundreds or thousands of managed objects using the browser. This is generally more than the user wants to deal with. Currently we advise the user to retrace steps and try again, applying more constraints to the search. In the future we will help the user with ways to reduce the clutter on the screen rather than put the responsibility on the user to figure out how to reduce it.

Metadata Services

Many different types of metadata are used by the tools that make up the MIB browser, including:

- GDMO, which describes the kinds of data that can be stored in the MIB and how it can be structured
- ASN.1, which describes the basic data types that can be stored
- IDL, which the application component editor generates from the signature of the components' external entry points
- Descriptions of how an operation should be parameterized
- Descriptions of each application component.

The tools obtain the GDMO and ASN.1 metadata via the HP OpenView metadata services. The metadata is stored in a repository and can be queried by all of the tools. Some added services are implemented. For example, when the user wants to find objects of a particular class or classes that are below a selected object in the MIB, the allPossibleSubordinateClasses service provides a list of the classes that it makes sense to allow the user to select from. This list is often much shorter than the list of all known managed object classes, making it quicker and easier to perform the action.

The CMIS requests and confirmations that flow between the browser and the MIB use the CMIP protocol. The information passed in the messages is numeric. For example, while the user wants to refer to the network class, the downstreamConnectivityPointer attribute, and the internalTimingSource, speech, locked, and sunday data values, the CMIP protocol will

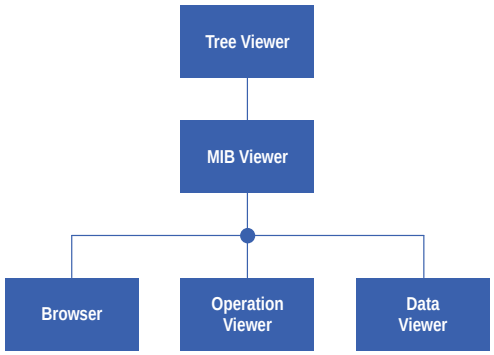


Fig. 9. *Inheritance hierarchy of the graphical presentation classes.*

expect to see the numerical values { 0 0 13 3100 0 3 1 }, { 0 0 13 3100 0 7 19 }, 0, 0, 0, and 0. Our metadata services perform these context-sensitive translations automatically in both directions.

The MIB Cache

The MIB cache contains a subset of the managed objects contained in the MIB. This subset is built up in the cache as the user navigates through the MIB while using the MIB browser. Operations used to handle this navigation result in responses being sent from the MIB. A response equates to a single managed object involved in the operation. Some responses indicate errors and others return data. Each data-bearing response contains the name and class of the responding managed object and possibly some additional data, such as the values of attributes.

The tools parse the results and store the values in the MIB cache. This reflects what has been learned about the MIB by the tools. The cache can be preloaded (seeded) on startup, which allows the browser to provide an initial context. When the MIB browser or similar viewers present a picture of the MIB, they are really presenting views of information in the MIB cache.

CMIS Services

The CMIS services enable the tools to issue multiple synchronous or asynchronous CMIS requests and receive multiple responses to each request. We built these services on Smalltalk's support for multiple thread execution and synchronization.

Smalltalk classes that represent CMIS requests and confirmations were defined. A request object can be submitted to the CMIS services component, causing the operation that it represents to be executed. A number of confirmations will later be received and a confirmation object will be created for each confirmation. These confirmations are then sent to the Smalltalk process that made the request.

HP OpenView Distributed Management Platform

The tools connect to an intermediary program called the CMIS interpreter, which in turn uses the HP OpenView Distribution Managment Platform as the distribution mechanism and communications provider. The CMIS interpreter uses OpenView's standard XOM/XMP APIs to generate, send, receive, and parse CMIS requests and confirmations. Communication between the tools and the CMIS interpreter is via an ASCII language, which is like a symbolic form of CMIS.

This arrangement provides us with the power of a symbolic, object-oriented language for rapid development while still enabling us to make use of the communication facilities of the HP OpenView DM platform, which is designed to work with C and C++ clients.

ASN.1 Representation

The representation of ASN.1 types and values is important to all the major architectural components. Values are passed to and from the CMIS services, stored in the MIB cache and displayed by the graphical presentation component. ASN.1 types are stored in the metadata's repository described above.

Conclusion

We have described the prototype of a software environment that aids the construction of telecommunications management applications. It is made up of three tools that together address many aspects of the development life cycle, from investigation of the problem to the deployment of the solution.

In choosing to address application components that interact with the TMN MIB, we deliberately focused on a well-defined subset of the overall application development area. We were then able to build tools that partially automate the task. We believe this automation could greatly increase developer productivity. The tools' usefulness is not restricted to the development of applications. The MIB browser, combined with the operations and components built using the other tools, is a powerful environment for exploring, understanding, and troubleshooting the MIB.

Acknowledgments

This work was carried out in collaboration with Simon Love and Paul Jeremaes at Hewlett-Packard Laboratories, Bristol, England. The author and his colleagues wish to acknowledge the contribution made by Ina Heider of the Technical University of Berlin.

References

1. *Principles for a Telecommunications Management Network*, ITU-T Recommendation M.3010, 1992.
2. *Principles for a Telecommunications Management Network: Overview of TMN Recommendations*, ITU-T Recommendation M.3000, 1994.
3. *Information Technology—Open Systems Interconnection—Systems Management Overview*, ITU-T Recommendation X.701 (ISO/IEC 10040), 1992.
4. *Information Technology—Open Systems Interconnection—Structure of Management Information: Management Information Model*, ITU-T Recommendation X.720 (ISO/IEC 10165-1), 1992.
5. *Information Technology—Open Systems Interconnection—Structure of Management Information: Guidelines for the Definition of Managed Objects*, ITU-T Recommendation X.722 (ISO/IEC 10165-4), 1992.
6. *Information Technology—Open Systems Interconnection—Common Management Information Service Definition*, ITU-T Recommendation X.710 (ISO/IEC 9595, 1991.
7. *Information Technology—Open Systems Interconnection—Common Management Information Protocol—Part 1: Specification*, ITU-T Recommendation X.711 (ISO/IEC 9596-1), 1991.
8. *Inter-Domain Management: Specification Translation, Preliminary Specification*, X/Open Company Ltd., 1995.
9. Ina Heider, *Encapsulation of TMN Application Components as CORBA Objects*, submitted to the Technical University of Berlin, Interdepartmental Research and Service Centre for High-Speed Networking and Multi Media (FSP-PV/TUBKOM), 1995.
10. E. Keremetsis and I. Fuller, "HP Distributed Smalltalk: A Tool for Developing Distributed Applications," *Hewlett-Packard Journal*, Vol. 46, no. 2, April 1995, pp. 85-92.

Business Process Flow Management and its Application in the Telecommunications Management Network

HP OpenPM is an open, enterprise-capable, object-oriented business process flow management system that manages business activities supporting complex enterprise processes in a distributed heterogeneous computing environment. It is a middleware service that represents a substantial evolution from traditional workflow technologies.

by Ming-Chien Shan, James W. Davis, Weimin Du, and Qiming Chen

Business process reengineering is emerging as one of the crucial business strategies of the 1990s. Business process reengineering is the fundamental rethinking and reimplementation of business processes to achieve never-before-possible levels of quality, cost, throughput, and service. This is especially significant in an era of workforce downsizing and greater demands for shortened time to market and faster customer response. The need for business process reengineering is pervasive. Organizations are currently engaging in business process reengineering in many domains, including financial services, telecom services, healthcare services, customer order fulfillment, manufacturing procedure automation, and electronic commerce.

While business process reengineering provides a business management concept, business process flow management (BPFM) software—or more accurately, *middleware*—provides the enabling technologies for business process reengineering to support flexible solutions for the management of enterprise-wide operations, including:

- Process flow control, automation, and monitoring
- Resource allocation, authorization, and authentication
- Task initialization and data exchange
- End-to-end communication and security.

BPFM is more than just a technology. It offers an overall environment and approach to unifying, automating, and measuring business processes. In addition, BPFM is not a technology supporting only business process reengineering. It can be used to manage existing nonautomated legacy processes—what is often called "paving the cow paths."

Business Process Flow Management System

At the enterprise level, the process to be managed can be very complex, spanning several organizations with multiple steps being performed in parallel. In such cases, a BPFM system can act as the superstructure that ties together disparate systems whose business purposes are interconnected.

A BPFM system provides procedural automation of a business process by managing the sequence of process activities and the invocation of appropriate human, instrument, or computer resources associated with various activity steps. It involves the high-level specification of flows, and provides the operational glue and environment support for managing and automating the flows, recovering from failures, and enforcing consistency. A BPFM system also enforces various administrative policies associated with resources and work.

The structure and flow of a business process managed by a BPFM system can be preplanned or ad hoc. In the case of a BPFM system managing the process of providing telecommunications service, the flow of the process is ad hoc and depends on the services required by a customer. However, certain aspects of the process will be preplanned and deliberately structured. For instance, regardless of the individual services required by a customer, the process always originates in the sales department and is always ends in the billing department.

Typically, a BPFM system:

- Provides a method for defining and managing the flow of a business process.
- Supports the definition of resources and their attributes.
- Assigns resources to work.
- Determines which next steps will be executed within a business process and when they will be executed.

- Ensures that the business process flow continues until proper termination.
- Notifies resources about pending work.
- Enforces administrative policies such as access control.
- Tracks execution and supports user inquiry of status.
- Provides history information in the form of an audit trail for completed business processes.
- Collects statistical data for process and resource bottleneck analysis, flow optimization, and automatic workload balancing.

HP OpenPM

HP OpenPM is an open, enterprise-capable, object-oriented BPFM system developed at HP Laboratories to manage business activities supporting complex enterprise processes in a distributed heterogeneous computing environment. It is a middleware service that represents a substantial evolution from traditional workflow technologies.

Given the trend towards open systems and standards, a BPFM system must coexist with and take advantage of standards-based commercial products for network communication, legacy application invocation, and system monitoring. In particular, the OMG's CORBA (the Object Management Group's Common Object Request Broker Architecture), the OSF's DCE (the Open Software Foundation's Distributed Computing Environment), HP OpenView, and ISO OSI (International Standards Organization Open Systems Interconnection) X.400 technologies are expected to play an important role in the development of BPFM systems. HP OpenPM provides a generic framework and a complete set of services for business process flow management using the above-mentioned standard technologies, with emphasis on performance, availability, scalability, and system robustness.

Basically, HP OpenPM provides:

- An open system adhering to the CORBA communications infrastructure and providing a WfMC (Workflow Management Coalition) standard interface.
- High performance as a result of optimized database access and commitment.
- Effective management with an HP OpenView-based system management environment.
- A comprehensive solution for business reengineering including an extensive set of products.

The overall architecture of an HP OpenPM system is depicted in Fig. 1. The core is the HP OpenPM engine, which supports five interfaces for business process definition, business process execution, business process monitoring, resource and policy management, and business object management.

A business process is specified via the process definition interface. An instance of the business process can be started, stopped, or controlled via the process execution interface. Status information of each process instance and load information of the entire system can be queried via the process monitoring interface. The resource and policy management interface is used to allocate, at run time, execution resources to a task, according to the policies defined by the organization (including authorization and authentication) and the availability of the resources. Interaction with the external world (e.g., the invocation of an application, the control of an instrument, or the delivery of a work order to a person's e-mail intray) is the task of the business object management interface.

HP OpenPM Process Model

A business process is a description of the sequencing, timing, dependency, data, physical agent allocation, business rule and organization policy enforcement requirements of business activities needed to enact work.

An HP OpenPM process is a directed graph consisting of a set of nodes connected by arcs. Fig. 2 shows an example of the user interface. There are two kinds of nodes—*work nodes* and *rule nodes*—and two kinds of arcs—*forward arcs* and *reset arcs*. A work node has at most one inward arc and one or more outward arcs. A rule node can have any number of inward and outward arcs.

Work nodes represent activities to be performed external to the HP OpenPM engine. These activities include authorization, resource allocation, the execution of business objects, and the provision of input data for the business objects and output data from them. Rule nodes represent processing internal to the HP OpenPM engine. This processing includes decisions of what nodes should execute next, the generation or reception of events, and simple data manipulation.

A work node is a place holder for a *process activity*, which is a logical representation of a piece of work contributing towards the accomplishment of a process. A process activity is mapped to the invocation of an operation on business objects during the execution of the process. Each process activity can represent a manual operation by a human or a computerizable task to execute legacy applications, access databases, control instrumentation, sense events in the external world, or even effect physical changes. A process activity definition includes a *forward activity* and optionally, a *compensation activity*, a *cancel activity*, a *resource management activity*, timeout and deadline information, and input and output data.

Rule nodes are used to specify process flows that are more complex than a simple sequence. A rule language is used to program the rule node decision. When executed, a rule node determines which outward arcs to fire, based on the status

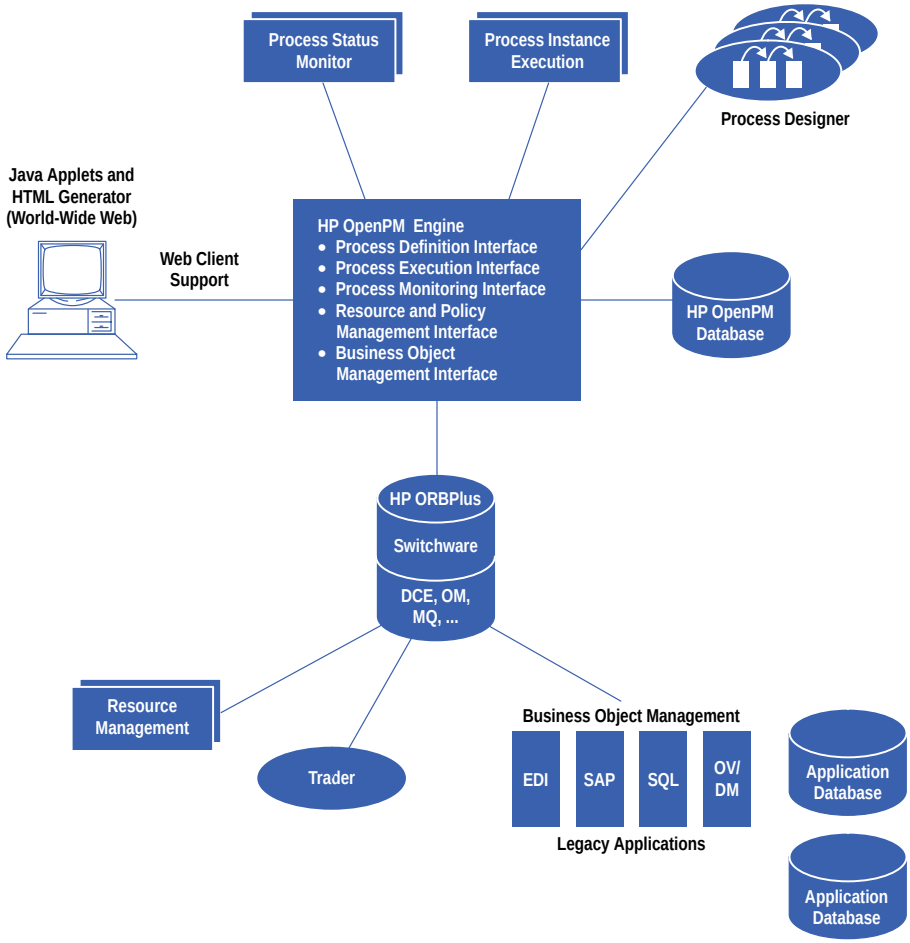


Fig. 1. *Architecture of the HP OpenPM business process flow management middleware.*

passed along the inward arcs, the time at which each inward arc is fired, and the process-relevant data associated with the process instance.

Rule nodes are also used to support events. A rule node can raise events when certain conditions are met as defined by the rules, and an event can activate rule nodes that have subscribed to receive the event.

Forward arcs represent the normal execution flow of process activities and form a directed acyclic graph. Successful completion of a node at the source end of a forward arc triggers the starting of the node at the destination end of the forward arc.

Reset arcs are used to support repetitions or explore alternatives in a business process. Reset arcs differ from forward arcs in that they reach backwards in the process graph.

Rule nodes are executed each time any inward arc fires. Work nodes have states of initial or fired. When the inward arc is fired on a work node in the initial state, the work node changes its state to fired and performs its associated activity. When the inward arc is fired on a work node in the fired state, nothing is done.

A reset arc, together with the forward arcs between its destination and source, forms a loop. When traversed, a reset arc causes all nodes within its loop to be reset. Resetting a fired work node changes its state to initial so that the node can be reexecuted. Resetting an active work node cancels the current execution of the corresponding process activity and change its state to initial.

Associated with each business process, there is a *process data template* defined by the business process designer. The process data template is used by users to provide initial data for the creation of process instances. At run time, based on the process data template and *read/write lists* of activities defined in a business process, HP OpenPM will generate a *case packet* for each process instance to facilitate data passing between activities and the HP OpenPM engine.

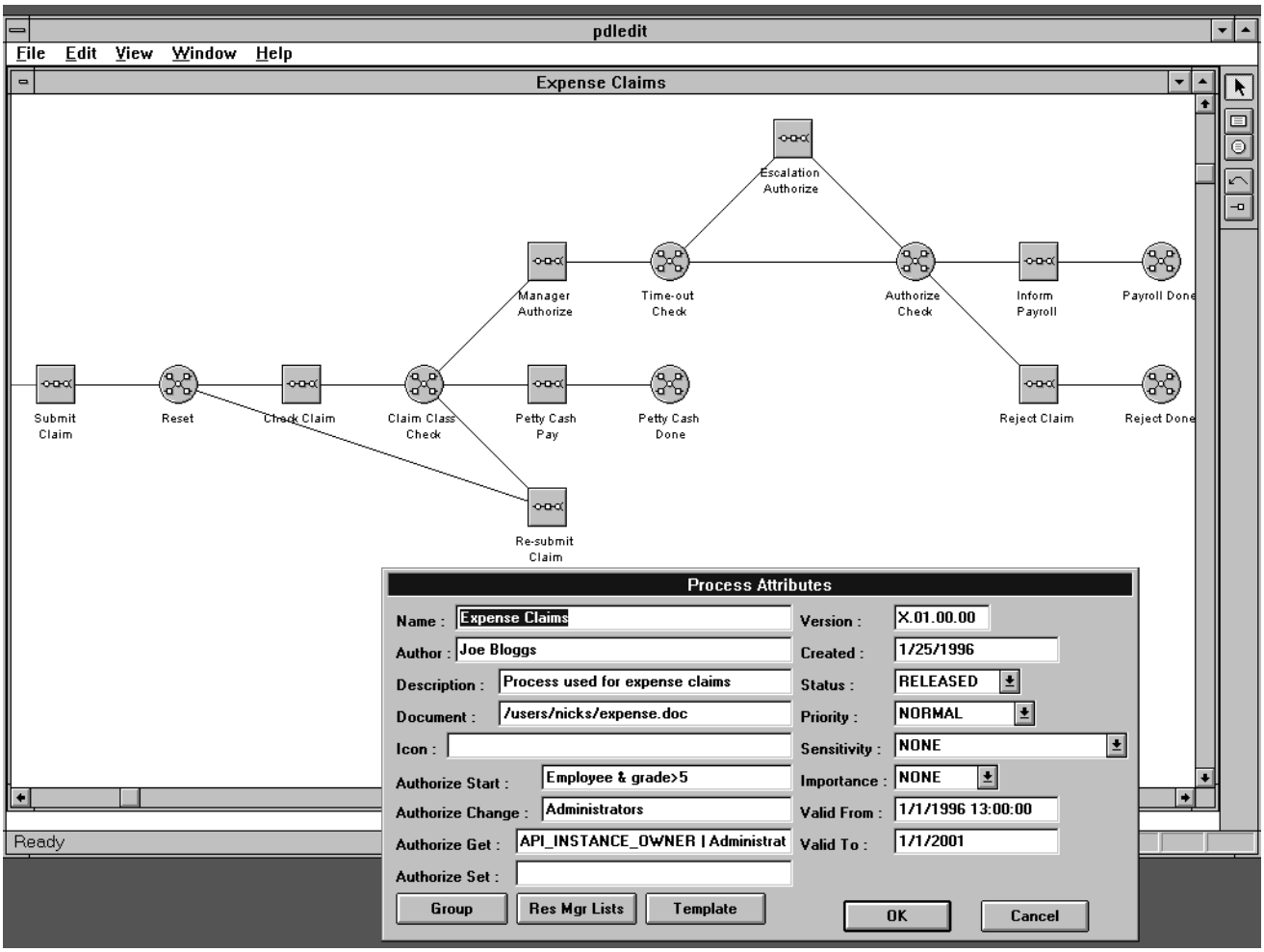


Fig. 2. *An example of the HP OpenPM user interface. An HP OpenPM process is a directed graph. There are two kinds of nodes: work nodes (square) and rule nodes (round).*

HP OpenPM Process Execution

Fig. 3 shows a simplified version of the component structure of the HP OpenPM engine, which coordinates the overall execution flow of business processes. It functions as a highly reliable, log-based state machine. The HP OpenPM engine interfaces with external environments through a uniform CORBA-based transport interface, independent of the actual physical dispatch of the requests.

The HP OpenPM engine launches business process instances in response to user requests. For each instance, the HP OpenPM engine steps through the nodes according to the order specified in its business process definition. For work nodes, the HP OpenPM engine will execute the associated process (forward) activity. For rule nodes, the HP OpenPM engine will evaluate the rules and perform the rule actions when the rule conditions are met.

Each node transition is durably logged to facilitate forward rolling of incompleted business processes at system restart time in the event of a system failure, or to facilitate a support activity compensation process in the case of a business activity failure. In addition, HP OpenPM allows flexible specification of compensation scopes and actions (e.g., compensation activity or cancel activity) to support various application needs.

In HP OpenPM, different versions of similar business processes are supported by the engine under the concept of a *process group*. The user can designate a particular version as the default to be used when no specific version is requested at the time a business process instance is created.

To monitor the progress of running business activities and support system management, the HP OpenPM engine maintains a comprehensive log of all events and provides a native interface as well as SNMP/CMIP gateways to facilitate integration with the HP OpenView environment. The formats and contents of the logged information can be customized to support specific application needs.

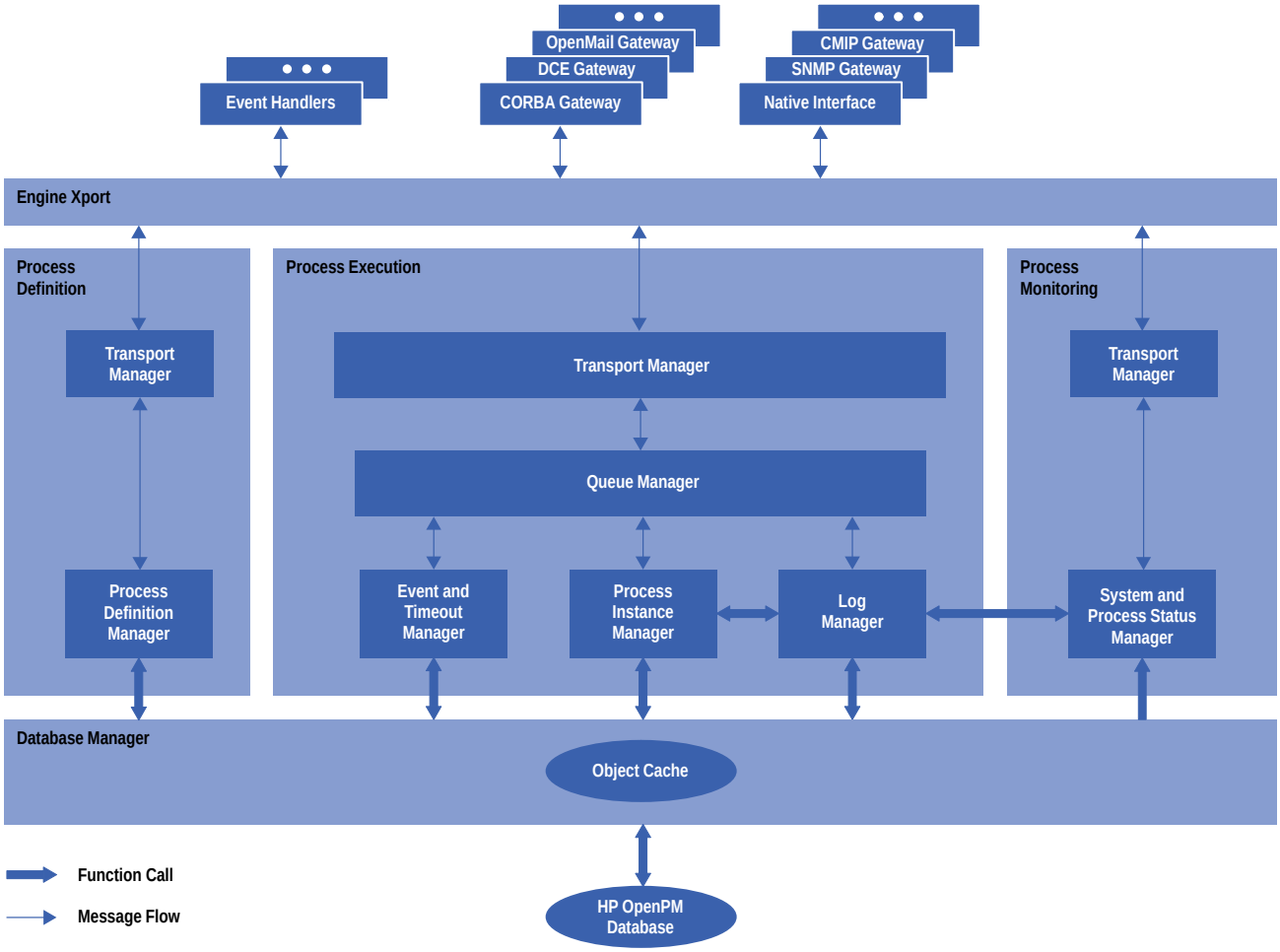


Fig. 3. *Block diagram of the HP OpenPM engine.*

HP OpenPM Business Objects

HP OpenPM has to interact with business activities supported by various implementations encountered in real life. These can range from manual handling by humans to automated processes executed by computers. An infrastructure is needed to enable the effective management and invocation of these business activities.

Distributed object technologies have become the primary infrastructure for enterprise-scale distributed computing. Among them, the OMG (Object Management Group) CORBA (Common Object Request Broker Architecture) technology has been developed to support interoperability for application integration.

Based on CORBA technology, in HP OpenPM an abstraction called a *business object* is built to encapsulate whatever piece of work each process activity has to accomplish. The wrapping code provides an IDL (Interface Definition Language) interface and the business objects are catalogued in the HP OpenPM business object library.

A business object, as defined by the OMG, is a representation of something active in the business domain, including its business name and definition, attributes, behavior, and constraints. It provides a uniform way to encapsulate legacy systems and applications, and a direct mapping, in understandable business terms, between the business model and the possibly sophisticated operational procedures of the business process system.

By representing these process activities in business objects, new business processes can be quickly created by assembling business objects to describe business processes. The business object library avoids repetitive coding to tailor the business activity implementation to each individual business process.

HP OpenPM Resource and Policy Management

A *resource* is a person, computer process, or machine that can be used to accomplish a task. A resource has a name and various attributes defining its characteristics, such as job code, skill set, organization unit, and availability.

A *policy* is a set of rules that determines how resources are related to tasks within a BPFM system. One common use is for task assignment. Policies can be used to specify which resource, under which role, is eligible or available to perform a task. Policies are also used to ensure proper authorization and authentication.

In HP OpenPM, the mapping between the business activity (task) specified in a business process and the business object (resource) to be invoked is performed by the *resource manager* during run time as part of the execution of the business activity. HP OpenPM allows multiple resource managers to be used to resolve a single resource assignment request; each resolves the request at a different level within an organization.

HP OpenPM Worklist and Application Data Handlers

Two optional components that can be added into the HP OpenPM environment to facilitate the execution of business processes are the *worklist handler* and the *application data handler* (see Fig. 4). Both components are designed to enhance the scalability of HP OpenPM systems.

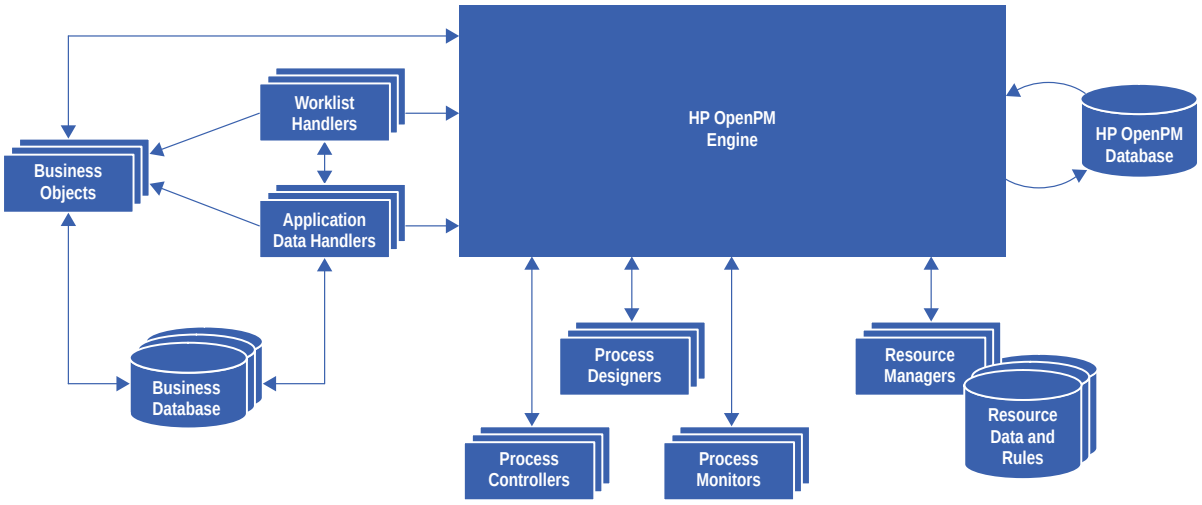


Fig. 4. *HP OpenPM system architecture including optional elements.*

The worklist handler supports both *engine-push* and *client-pull* modes to provide more freedom in task assignment. In addition, the worklist handler can be used to support the concept of *integration on demand*. Based on the task performer's profile, the worklist handler determines and launches a specific environment for an activity at run time, rather than hard-wiring it into the process definitions.

The application data handler supports the separation of application-specific data and process-relevant data to reduce the amount of data flow over the network. It also provides the preparation facility for application-specific data to remove the burden of database access from activity performers.

HP OpenPM Security

In today's business environments, security must be implemented enterprise-wide. The security service developed by the OMG provides authentication and encryption for HP OpenPM to prevent eavesdropping and forgery. The HP OpenPM infrastructure components can identify each other and vouch for the credentials of end-user components.

BPFM in the Telecommunications Management Network

The Telecommunications Management Network (TMN) defined by the International Telecommunications Union is changing the way operations support systems and business support systems solutions are being developed. The TMN architecture separates layers of functionality and provides access by elements in any one layer to any element in the layer immediately below. Before the introduction of the TMN model, operations support systems and business support systems solutions were isolated from each other and could not interoperate.

The HP OpenView Distributed Management platform supports the realization of TMN operations support systems and business support systems solutions for the TMN element management layer and network management layer (see ***Article 1*** for a description of the TMN layers). Still needed is a middleware service supporting the service management layer and even the business management layer of the TMN model. This need offers a great opportunity for BPFM added value. The next section presents an example of this support.

At the service management layer, the BPFM process enabling framework is required to be able to:

- Support reengineering and transformation processes for strategic operations support systems and business support systems.
- Integrate existing operational environments to form an enterprise hub for service management and provisioning.
- Deploy new management services as rapidly as possible.

- Monitor and measure processes.
- Tune processes to benefit from experience.
- Automate processes to reduce execution time.

The overall deployment of BPFM technology in the TMN environment is depicted in Fig. 5.

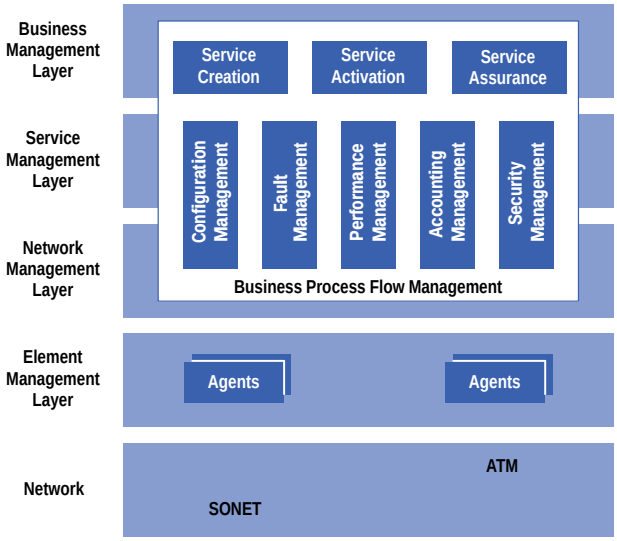


Fig. 5. *Telecommunications Management Network layers, showing management functions provided by business process flow management.*

SONET Configuration Management Prototype

Based on an HP OpenPM system, we built a prototype to demonstrate the application of BPFM technology in the specific domain of SONET (Synchronous Optical Network) configuration management. The prototype was a joint project between HP Laboratories in Bristol, England and Palo Alto, California to demonstrate the middleware technologies required to automate the processes supporting the configuration management of a SONET telecommunications network.

The scenario demonstrated by this prototype consists of the provision of a new VC4/VC12 path for customers. It goes through several different steps for this operation: search for a new route, negotiate the service level agreement (SLA) with the customer, configure the new path, and finally, update the SLA for this customer. The HP OpenPM process definition supporting the process of providing this new SONET data path is sketched in Fig. 6.

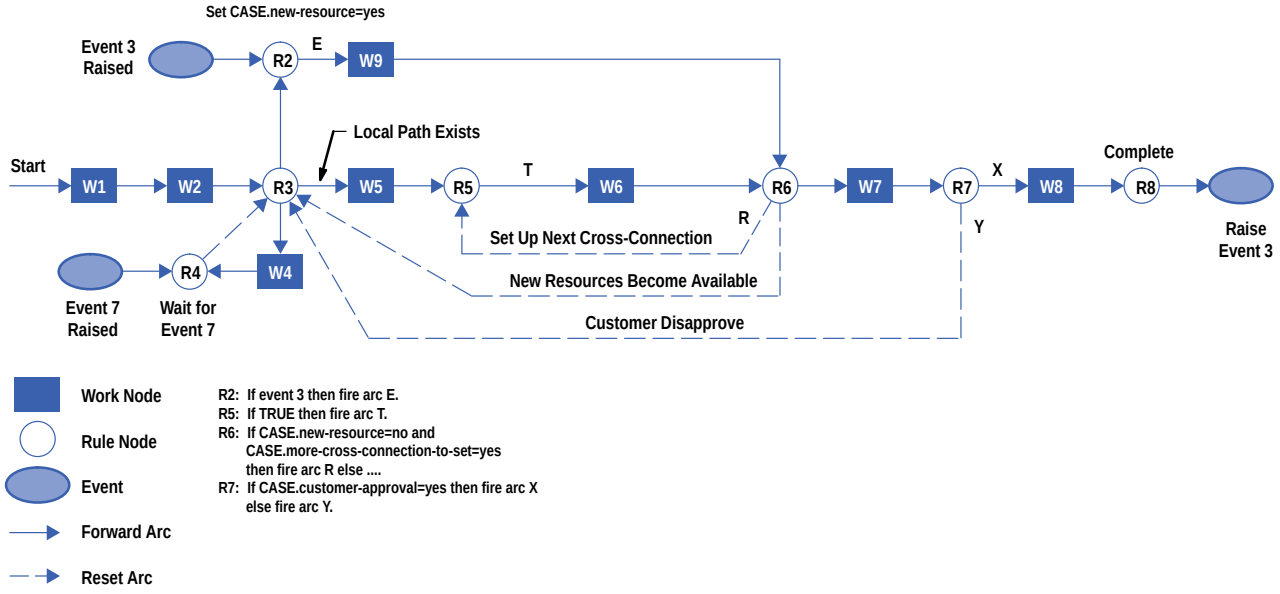


Fig. 6. *HP OpenView process definition for SONET configuration management.*

Searching for and configuring a new path in SONET are complex processes requiring a lot of interaction with the SONET MIB (Management Information Base) and network elements. This type of operation is a source of errors when it is performed manually by an operator as a set of individual, uncorrelated activities.

In the prototype, such complex operations as searching and configuring new paths are handled as business processes and automated by an HP OpenPM engine in an environment interacting with HP OpenView DM and Oracle DBMS applications.

Depending upon the changing business needs, a customer can request to add or drop communication paths between certain endpoints in a *private virtual network* (PVN). In HP OpenPM, these services can be modeled as business processes to be executed by the service provider. Adding a new path may consist of the following activities and decision points:

1. Retrieve the customer's profile from the customer database for customer-PVN-specific information.
2. Locate the closest add-drop multiplexers (ADMs) to the endpoints, based on the information stored in the SONET physical configuration database.
3. Check whether fiber connections exist between the endpoints and the two end-ADMs.
4. If not, issue a request for an engineer to go onsite and physically connect the endpoints to the end-ADMs. After the establishment of the connection, the process continues on to step 5 and an independent subprocess is initiated to watch for resource changes.
5. Find valid routes between end-ADMs. This requires access to the routing table in the SLA database to determine whether any valid routes exist between the two end-ADMs. Either a list of ADMs is returned signifying the ADMs that must be configured to realize the route, or "No Route Found" is returned. For a returned list of ADMs, this activity will then use the HP OpenView DM facility agent to collect port information stored in the MIB to determine the available ports between the ADMs that are fibered together and can be used to enable the path.
6. Check network element (NE) capabilities. For an ADM in the route, this activity uses the HP OpenView DM NE agent to access the MIB information to determine whether a VC4 cross-connection can be set up in the ADM between the selected ports of the ADM. This activity has to be executed for each ADM in the route. During steps 5 and 6, if any additional resources become available, HP OpenPM cancels any currently running activity and starts the process over from step 5 to consider these newly available resources.
7. Get customer's approval of the selected configuration. Once a suitable path is identified, the customer will review the offer, including available date, charges, quality of services (QoS), and so on. Depending upon the business factors (e.g., cheapest service wanted), the customer may request that a new search be initiated, that is, loop back to step 5 to find another valid route.
8. Configure the selected route. This activity is responsible for setting up the cross-connections in each ADM by invoking the HP OpenView DM NE agent and updating the SLA database.

Acknowledgments

The authors would like to acknowledge the contributions of several individuals. John Manley and Mike Robinson provided leadership and guidance in the development of an early version of the SONET configuration management prototype for demonstration at Telecom '95. Chris Whitney helped provide the SONET environment simulator. Clemens Pfeiffer spearheaded the initial creation of HP OpenPM and helped gather the momentum needed for the project to survive. Nick Sheard led the product development to commercialize the HP OpenPM research. Chip Vanek helped drive the design of HP OpenPM as a major internal customer.

Bibliography

1. S. Aidarous and T. Plevyak, *Telecommunications Network Management into the 21st Century*, IEEE Press, 1994.
2. W. Du, C. Whitney, and M. Shan, "SONET Configuration Management with HP OpenPM," *Proceedings of the 12th International Conference on Data Engineering*, New Orleans, Louisiana, February 1996.
3. J. Davis, W. Du, E. Kirshenbaum, K. Moore, M. Robinson, M. Shan, and F. Shen, "CORBA Management of Telecommunications Networks," *Proceedings of the Workshop on Distributed Object-Oriented Computing*, Object World Frankfurt '95, October 1995.
4. J. Davis, W. Du, and M. Shan, "HP OpenPM: An Enterprise Process Management System," *IEEE Computer Bulletin*, June 1995.
5, W. Du, S. Peterson, and M. Shan, "Enterprise Workflow Architecture," *Proceedings of the 11th International Conference on Data Engineering*, Taipei, Taiwan, March 1995.
6. U. Dayal and M. Shan, "Issues in Operation Flow Management for Long-Running Activities," *Data Engineering Bulletin*, Vol. 16, no. 2, June 1993.

7. M. Shan, “OpenPm: An Enterprise Business Process Flow Management System,” *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Montreal, Canada, June 1996.

HP OpenView Agent Tester Toolkit

In developing HP OpenView agents, a major challenge is to develop and test both the agent and the manager simultaneously. To fill this need, the HP OpenView Agent Tester Toolkit generates tests and allows the developer to execute these tests individually or as a set.

by Paul A. Stoecker

HP OpenView agents can be created by telecommunications network management developers either by using tools or by writing the code directly. The tools available include the *GDMO* Modeling Toolset* (see ***Article 4***), which helps in the design and specification of network management objects using the GDMO language, and the *HP Managed Object Toolkit* (see ***Article 6***), which accepts GDMO documents and produces C++ code to implement a default agent that meets those specifications. Whether the developer builds an agent using these tools or writes the code by hand, one of the major challenges is to develop and test both ends of the communications link simultaneously—the *agent* controlling the managed device and the *manager* that sends requests to the agent and receives the responses. To fill this need, the new HP OpenView Agent Tester Toolkit generates tests and allows the developer to execute these tests individually or as a set.

The Role of an Agent

An agent program enables other programs, called managers, to control physical and logical resources. Examples of resources that are controlled by agents are telephone switching equipment and phone service databases. From a centralized location, a telephone service provider can use automated processes to monitor the performance of the communications lines, reroute traffic as necessary, and maintain the business and accounting records. Because the communication protocol between managers and agents has been standardized, a wide area network of multivendor equipment can be efficiently controlled from a small number of central locations.

The resources being monitored and controlled are modeled as objects called *managed object classes*. Managed object classes are logical groupings of the attributes, events, and actions associated with a resource. A GDMO specification defines the various managed object classes that make up the interface to the resource. Instances of these classes are called into existence by sending a create request. The attribute values for an instance are accessed by issuing set and get requests to change or retrieve the attribute values, respectively. Other message types remove an object instance, allow the agent to notify interested parties of an asynchronous change, or cause the agent to perform some agreed-upon activity.

A collection of managed object instances and their relationships is called the *containment tree*. Subobjects are logically contained or grouped within other objects. Fig. 1 depicts a portion of a containment tree. Each of the boxes in Fig. 1 represents an object instance. The label in each box identifies the object class of that instance. For example, in Fig. 1, a fiber-optic network is composed of two network elements. In one of those network elements, the regenerator and multiplexer sections are shown.

One of the attributes within each of the contained object instances is designated as the *distinguishing attribute*, and the value of this attribute is used to uniquely distinguish that instance from all of its siblings. The containment tree is used to uniquely identify, or *name*, an object instance. An object instance anywhere in the containment tree is identified by specifying the distinguishing attribute and its value from the top of the tree down to the desired instance. The concatenation of all of the distinguishing attributes along this naming path is called the *fully distinguished name*. In Fig. 1, the fully distinguished name for the multiplexer section would consist of the sequence networkId = “net1”; elementId = 5; muxId = 56.

Agent Development

The Managed Object Toolkit saves an enormous amount of work by handling all of the overhead of decoding and validating incoming requests, locating the selected object instance within the containment tree, and invoking an appropriate C++ method on the selected object. However, the attribute values that are set or retrieved by the initial Managed Object Toolkit output are only internal representations. The developer is responsible for filling in empty C++ stubs to make the internal attribute values reflect the state of external physical devices. During this coding process, it is helpful to simulate the requests that will eventually be sent by a manager.

The Agent Tester Toolkit performs this task in two steps. First, it creates test requests from the GDMO specification. Second, it transmits these requests over the network to the agent and receives the responses. During the development phase, these test files can be sent individually and the responses viewed interactively. As each agent operation is implemented,

* GDMO is the ISO (International Standards Organization) *Guidelines for the Definition of Managed Objects*.

the associated test requests can be added to a test suite. The accumulated tests can then be run in a batch mode to check that previously implemented functionality still works properly. Fig. 2 depicts the sequence of steps needed to generate and send the test requests, and shows how the Agent Tester Toolkit relates to other development tools.

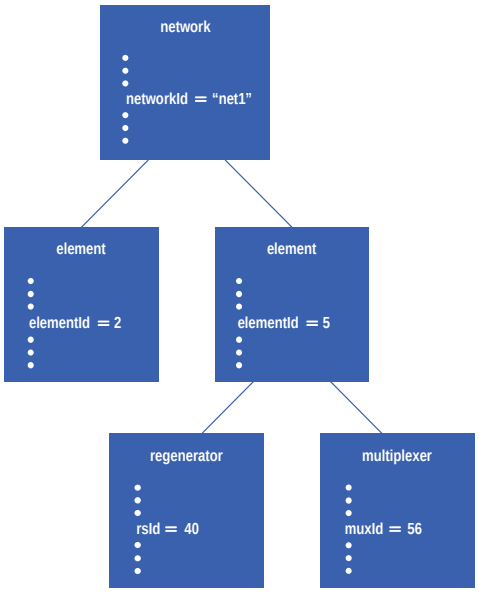


Fig. 1. *A containment tree.*

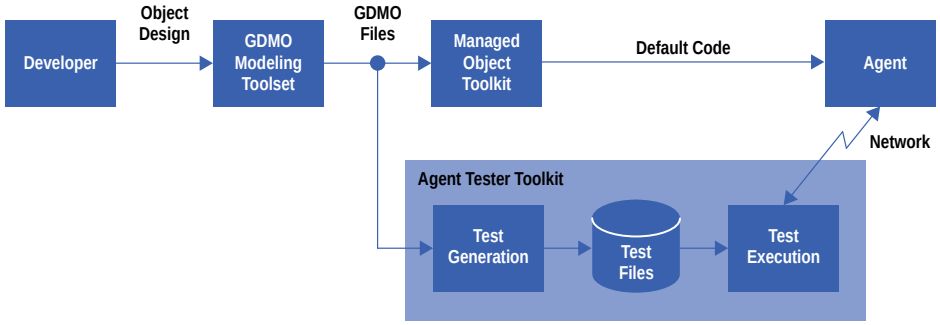


Fig. 2. *Agent development and testing tools.*

Running the Agent Tester

The components of the Agent Tester Toolkit are command-line tools that are invoked in a straightforward way. For example, ovatgen -t /tests gdmo.mib reads the GDMO description in the file gdmo.mib and generates a set of test requests stored in files under the directory /tests. Next, tests for a particular object instance can be sent to the agent as follows:

```
$ ovatrun -i
>create
...
>getall
...
>mytest
...
>delete
...
```

The -i option to ovatrun specifies the interactive mode, in which the user can type the name of a test file in response to the > prompt and the response from the agent is displayed immediately (shown by the dots above).

The test files represent CMIS (Common Management Information Service, ISO/IEC 9595) operations, such as create, set, get, and so on, and are stored in a directory layout that mirrors the organization of the agent's containment tree, with each directory named by its associated managed object class name. At each level in the containment tree, test files are generated that create an object instance, get all attributes, and delete the instance. If there are changeable attributes, tests are also generated that set those attributes to new values and retrieve the changed attributes. In addition, files are generated that test

attribute groups and actions. Documentation files describe the object identifiers used in the tests and optional features called *conditional packages*.

Each test request is written in a format called ASN.1 value notation, which is a standardized format described in ISO and ITU-T documents (8824 and X.208, respectively). ASN.1 (Abstract Syntax Notation One) is a notation for expressing the types of the attributes and operations. For example, a test file that contains a get request to retrieve the current values of several attributes might appear as:

```
GetArgument {
  -- passwordEntryManagedObjectClass
  baseManagedObjectClass {1 3 6 1 4 1 11 9 81},
  baseManagedObjectInstance distinguishedName : {
    {
      {
          -- passwordRootName
          attributeType {1 3 6 1 4 1 11 9 29},
          attributeValue Mod.RootSyntax 0
      }
    },
    {
      {
          -- loginName
          attributeType {1 3 6 1 4 1 11 9 21},
          attributeValue Mod.LoginSyntax "paul"
      }
    }
  },
  attributeIdList {
    -- password
    {1 3 6 1 4 1 11 9 22},
    -- userID
    {1 3 6 1 4 1 11 9 23}
  }
}
```

In this example, the first word, GetArgument, announces the ASN.1 type whose value follows. A GetArgument is a structured type, and in this example its fields are baseManagedObjectClass, baseManagedObjectInstance, and attribute-IdList. Lines beginning with -- are comments inserted by the Agent Tester Toolkit generator to help the reader identify the various *object identifiers* (OIDs), which are strings of digits (e.g., {1 3 6 1 4 1 11 9 23}) that uniquely identify attributes, classes, and other fields. Returning to the GetArgument request, when sent by the Agent Tester Toolkit it asks the agent to return the current value of the password and user ID attributes of an object of class passwordEntryManagedObjectClass. The particular instance is identified by an object instance passwordRootName = 0, which in turn contains the desired subobject loginName = paul. A typical response would be:

```
GetResult {
  managedObjectClass {1 3 6 1 4 1 11 9 81},
  managedObjectInstance distinguishedName : {
    {
      {
          attributeType {1 3 6 1 4 1 11 9 29},
          attributeValue Mod.RootSyntax 0
      }
    },
    {
      {
          attributeType {1 3 6 1 4 1 11 9 21},
          attributeValue Mod.LoginSyntax "paul"
      }
    }
  },
  currentTime "19960327145135",
  attributeList {
    {
      attributeId {1 3 6 1 4 1 11 9 22},
      attributeValue Mod.PasswordSyntax "secret"
    },
```

```
      {
        attributeId {1 3 6 1 4 1 11 9 23},
        attributeValue Mod.UserIDSyntax 4463
      }
    }
  }
```

This response returns the requested class and instance information, and reports that the values of the two requested attributes password and userID were secret and 4463, respectively.

It is also useful to gather as much information as possible when error conditions exist. For example, if we try to query an object that doesn't exist, an error is returned, letting us know what aspect of the request was rejected:

```
$ ovatrun –i
>getbad
–– Error: No such object instance
ObjectInstance distinguishedName : {
  {
    {
      attributeType {1 3 6 1 4 1 11 9 29},
      attributeValue Mod.RootSyntax 0
    }
  },
  {
    {
      attributeType {1 3 6 1 4 1 11 9 21},
      attributeValue Mod.LoginSyntax ”joe”
    }
  }
}
```

Test files are ordinary text files, and customized tests can be crafted using the generated tests as guides. Several supporting tools are included in the Agent Tester Toolkit.

Batch Testing

After portions of the agent have been developed and the tests are working individually, it is good practice to run the tests and check the results in an automated fashion. This is useful to monitor existing behavior of an agent as new code is added, or to be able to repeat the testing process as new versions of the agent are developed or the agent is ported to new hardware platforms. To this end, the Agent Tester Toolkit's run program can execute a sequence of tests in succession. The command is ovatrun without the -i option:

```
$ cd /tests
$ ovatrun
```

This causes the list of tests in a default test director file, batch_list, to be run and the responses stored. After all tests have been run, the responses are compared against a set of known-good results, and summary statistics are prepared in a log file, reporting the number of tests run, passed, and failed. The known-good result files are generally prepared by copying actual response files that have been manually verified. A utility tool is provided that copies result files into place as known-good comparison files. As part of the copying process, this utility removes lines that contain the current time, since this would needlessly cause comparison failures in future test suite runs.

The test director file in its simplest form contains the names of the test files and the order in which they are to be sent. Optional commands in this file allow for more complex situations. For example, ISO standard 10164-1 identifies situations (object creation, object deletion, and attribute value change) in which the agent should emit an event so that all interested managers can maintain a synchronized view of the agent's state. To alert the Agent Tester Toolkit to expect both a response to one of its own create, delete, or set requests and the resulting event emitted by the agent, the pair command can be used. For example, the command pair password/create sends the request command contained in the file password/create and then receives both the confirmation of the request and a notification that the creation has occurred. Similarly, if an isolated event is expected, the event command can precede the name of a file with which the arriving event will be compared. Other commands, such as a shell escape to execute any user command, allow customized testing. For example, a shell escape allows the test designer to send a signal to the agent process to trigger some behavior, such as the sending of an event. This simulates the behavior of the agent in actual operation where some asynchronous condition might cause the event, while still allowing the test process to receive a predictable stream of responses from the agent. Other commands allow finer control over the testing process. For example, a timeout value can be set that controls how long the tester will wait for a response before aborting any single test. An example of a test director file with some of these commands included is as follows:

```
# Comments begin with the '#' character
# The following files are regular tests to get
# the attributes in the already-created
# Root Managed Object Class
root/passFileMOC/getall
# Some of the next tests expect both a response
# and an event
pair root/passFileMOC/passEntryMOC/create
root/passFileMOC/passEntryMOC/getall
pair root/passFileMOC/passEntryMOC/set
root/passFileMOC/passEntryMOC/get
pair root/passFileMOC/passEntryMOC/delete

# Set the timeout to 30 seconds
timeout 30

# Send a UNIX signal that triggers an event
! kill SIGINT $(AGENT_PID)
# Receive the event
event root/passFileMOC/passEntryMOC/event1
```

Finer Control of the Generation Process

A powerful feature of the object-oriented design methodology is that the standards bodies have invested much energy into constructing managed object class building blocks. A side-effect, however, is that in most cases the standard documents from which specific agents inherit contain far more definitions than are needed for that agent. In the case of the Managed Object Toolkit, this causes needless code to be generated, producing a larger agent than is required. To counteract this effect, the Managed Object Toolkit allows developers to specify a subset of the managed object classes, so that code is generated only for that subset. The Agent Tester Toolkit accepts the same subset specifier, and tests are generated only for that subset.

In some cases, greater control over the nature of the generated tests is needed than simply selecting a subset of managed object classes. An example is the containment tree example given in Fig. 1 that began with a network object as the root node. The GDMO description of a network class might allow (as it does in ITU-T Recommendation M.3100) that network to be decomposed into subnetworks and subsubnetworks, and so on. To allow the test designers to specify how many levels of decomposition the agent is expecting, a containment tree specification file can be provided to the test generator. This specification file is formatted like an outline, with the level of indentation indicating how deeply under the root node each class is contained. For example, the containment tree in Fig. 1 would be depicted:

```
network
> element
> > regenerator
> > multiplexer
```

(Only one of the element nodes is shown. It will be explained later how to include both circuit branches.)

If the agent is expecting the network level to be expanded into network and subnetwork levels, this change can be incorporated in the specification file by introducing another network node and indenting its children by an additional level:

```
network
> network
> > element
> > > regenerator
> > > multiplexer
```

This change adds an element to the distinguished name (corresponding to the subnetwork distinguishing attribute value) in each test file, so such containment changes have far-reaching effects. Making such decisions early in the specification phase saves much work compared to adjusting already generated test files.

As mentioned earlier, the tests are placed in a UNIX directory structure that parallels the containment tree structure, with each level named by its associated managed object class name. In many cases, managed object class names can be lengthy, and a pathname to lower-level test cases composed of a sequence of those names can be unwieldy. For example, names such as trailTerminationPointBidirectional and connectionTerminationPointSource appear in the standards, and when several of these are joined (as is typically done when specifying containment relationships), the combination is hard to read. To populate the directory structure with shorter, meaningful names, a default heuristic is applied that selects a few letters from each segment of a managed object class name. For example, a file deep in the tree described so far might be named netw/netw/elem/mult/create. Alternatively, the test developer can override this heuristic by specifying shorter names in an optional field in the specification file:

```
network (net)
> network (subnet)
> > element (NE)
> > > regenerator (rs)
> > > multiplexer (mux)
```

Finally, the GDMO document doesn't specify actual attribute values, so the containment tree's distinguishing attributes have to be supplied by the test designer. Once again, much work is saved by specifying these early, rather than fixing tests after they are generated. These distinguishing attributes can be assigned in the last optional field of the specification file:

```
network (net) networkId="ftc"
> network (subnet) networkId="Bldg1"
> > element (NE2) elementId=2
> > element (NE5) elementId=5
> > > regenerator (rs) rsId=40
> > > multiplexer (mux) muxID=56
```

Note that by including the distinguishing attribute values, we can differentiate between the two element sibling branches.

A supporting tool called ovatct reads GDMO files and produces a skeleton specification file similar to the one above (using the same subset selection file as the Managed Object Toolkit, if provided). More meaningful abbreviations and attribute values can be noted in the specification file and then used as an input to the test generator to guide the production process:

```
ovatct gdmo.mib > spec_file
ovatgen -t /tests -f spec_file gdmo.mib
```

Summary

Key design goals of the Agent Tester Toolkit include supporting agent developers during the development and maintenance phases, and confirming compliance to the GDMO specifications the agent is to implement. Also important is the ability to generate tests iteratively for evolving designs, without time-consuming configuration changes of the test engine itself. The Agent Tester Toolkit complements other tools in the development life cycle.

Storage Management Solutions for Distributed Computing Environments

Strategies for dealing with the vast amounts of data generated by today's information technology environments involve more than just larger and larger disk drives. They include the right combination of different storage devices to deal with offline, nearline, and online data storage and scalable management software.

by Reiner Lomb, Kelly A. Emo, and Roy M. VanDoorn

Storage management is fast becoming one of the most important issues information technology (IT) managers face today. With data accumulating at enormous rates, and with end users demanding faster access to more information, storage management has moved from an operation that was done only at night to a mission-critical concern that requires full-time attention.

Storage management consists of all the activities related to the effective deployment, accessibility, and use of stored information across a computing infrastructure. Storage management involves several major disciplines, including backing up and restoring data, storing data online across multiple classes of storage devices such as disks and tape, archiving data for legal and historical purposes, and managing storage resources such as tape or optical media for optimal use. Managing these storage disciplines takes an effective combination of organization, processes, and technologies to meet end-user data-availability expectations.

In today's distributed computing environments, IT managers need consistent storage management strategies and processes across the enterprise. In addition, storage management processes cannot be separated from an integrated network and system strategy. Therefore, IT managers need complete solutions that integrate the various storage management components and technologies such as databases, file systems, storage peripherals, storage management applications, and network and system management strategies.

In the past, storage management solutions have been proprietary (mainframes) or piecemeal (early client/server point products), with specific peripherals working only with specific software and hardware. It was difficult to expand a solution to meet the demands of rapidly growing collections of data.

In this article we will describe trends driving storage management technology and the components that make up an ideal storage management solution. Finally, we'll introduce HP hardware and software products, services, and partners and describe how they work together providing storage solutions for our customers.

Storage Management Trends

Traditionally, the task of storage management was done after work hours when the system could be brought down for storage management functions such as backup and archiving. Today, much more data is generated, and storage management solutions need to provide much greater data availability and reliability. Complicating storage management are variables that determine data throughput and access. These variables include disk capacity, CPU, input/output channels, device speed, networks, and software (Fig. 1). New ways of transferring and storing large amounts of data without downtime have to be developed.

Probably the most important driving factor in storage management today is that customers demand continuous accessibility to huge amounts of data, very often in the terabyte range. You can see this demand occurring in the increased use of the Internet and online services such as CompuServe and America Online, and in the emergence of new applications such as imaging and multimedia. In the past, data accessibility was a fairly simple process when mainframes were the primary storage devices and the only limitation was disk size. Today, the answers to storage problems cannot be provided simply by installing a bigger disk on a central server.

As customers reengineer their businesses, many are choosing to migrate away from the mainframe via "mainframe downsizing." Mission-critical applications are moving to open systems, and the management of client/server workgroups is being consolidated across LANs and WANs. An enormous amount of company-sensitive data, which used to be under central control and located in the data center, is now distributed and available on the network (Fig. 2). Published market numbers show that the average amount of distributed data has surpassed the average amount of data in the data center. Companies must begin viewing storage management as integral to their network and system management solutions.

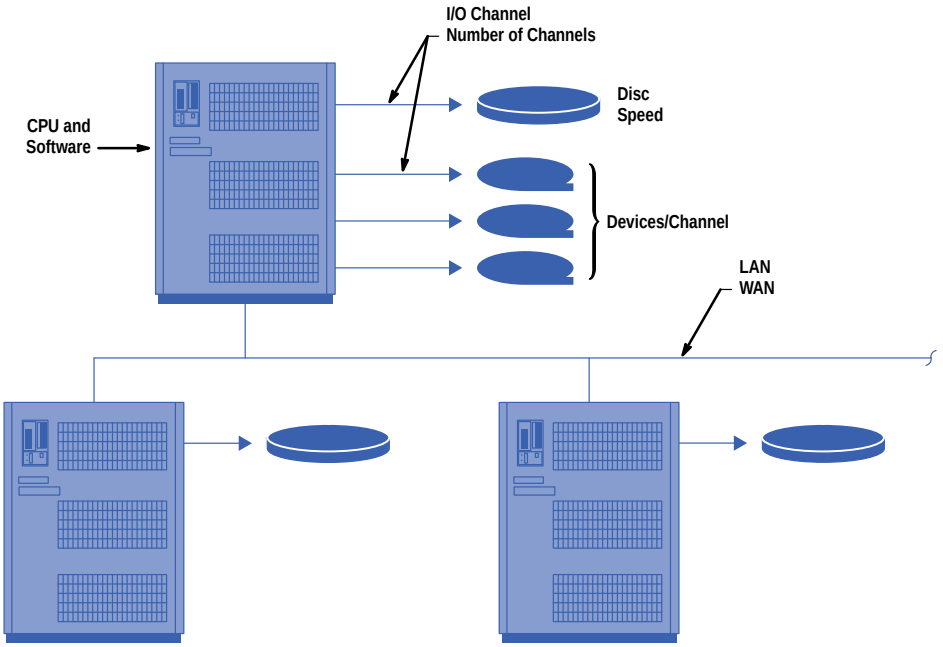


Fig. 1. *Components in the chain that have an impact on data throughput.*

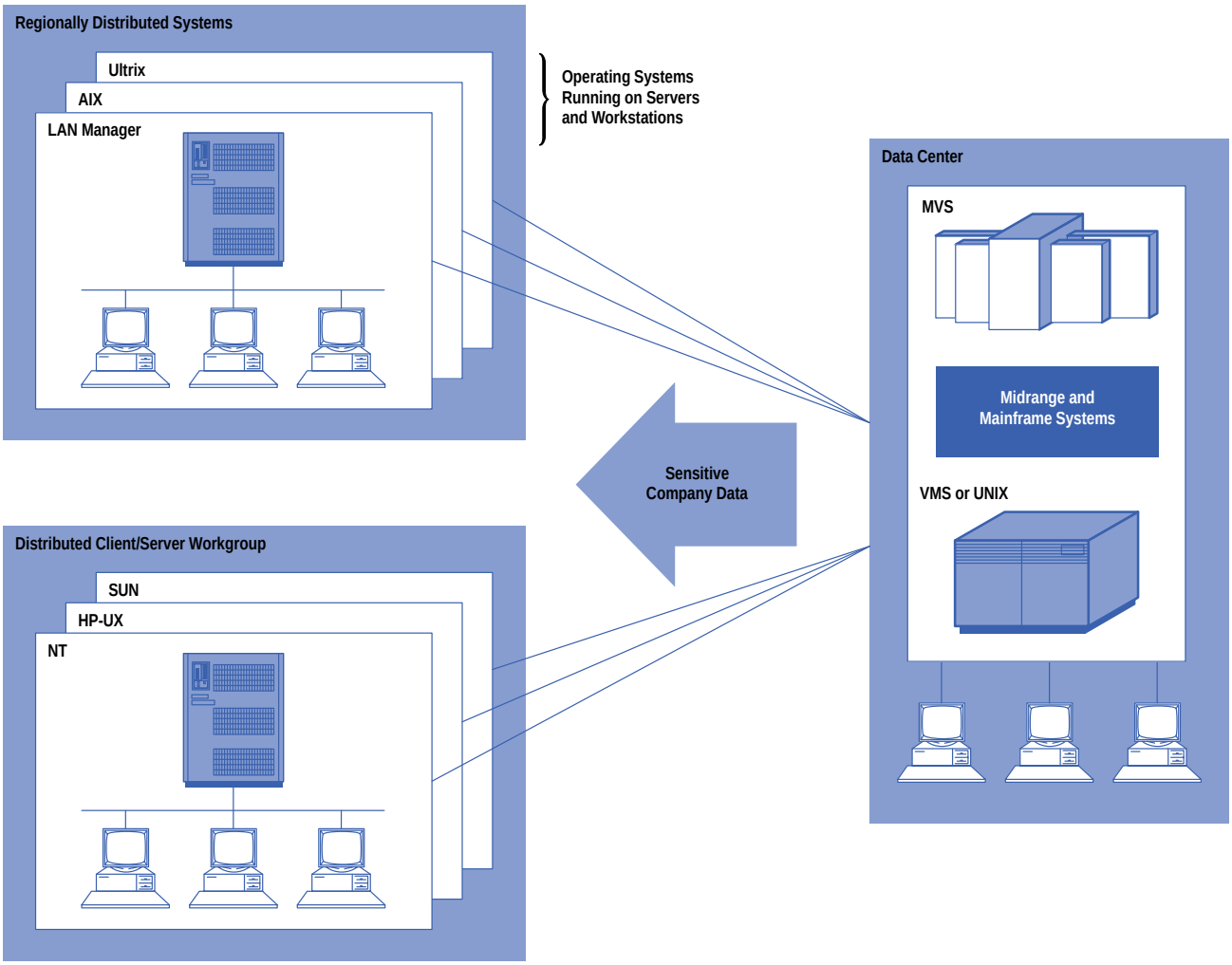


Fig. 2. *Decentralized storage management committed to the same service level as a mainframe environment.*

In addition to all the challenges raised by managing storage on distributed systems, IT managers must deal with the reality that the amount of data being stored is outstripping the network's capacity to handle it efficiently (Fig. 3). For example, a company might need to back up 100 Gbytes of data in an hour. As the storage staff looks for solutions, they see processor performance improving faster than disk performance. They also see the performance of both disks and processors outstripping the performance of the installed network infrastructure. At the rate network infrastructure is improving, it will be a huge challenge to catch up to processor performance.

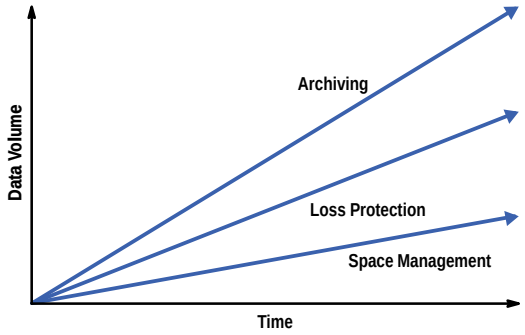


Fig. 3. *The volume of data being stored is outstripping the ability of network services to deal with it.*

If IT managers try to win this contest with only the traditional approach of centrally stored data, they will lose the storage management race. Instead, today's storage management solutions must allow distributed storage management to be performed centrally, decentrally, or in a hybrid fashion, depending on a company's policies and needs.

To solve the growing issue of storage management, they need to understand what constitutes a storage management solution.

Storage Management Requirements

The ideal storage management solution, which is made up of complementary software, systems, and peripherals, is integrated, scalable, and modular, and allows the solution to be implemented in phases and expanded over time. A flexible solution addresses both mission-critical enterprise-wide requirements and business-critical desktop needs. At the same time, this solution must be easy to use and robust, and must provide quick, reliable access to data.

The fundamental requirement of any storage management system is to provide data accessibility to all users, regardless of where and how the data is stored. To make data quickly accessible, yet store it efficiently, customers need a complete, integrated set of storage management functions, including backup and recovery, archiving and retrieving, hierarchical storage management, and media management.

In addition, an enterprise-wide storage solution must allow various storage management applications and peripherals to manipulate and share media in a consistent manner. It must also provide an easy and standardized way to access the various storage devices, library systems, and silos. Many companies are dedicating servers to specific tasks such as backup and restore servers or archival and retrieval servers. A storage solution must be optimized so data is stored and moved in the most efficient manner.

Other more generic services for storage management include a central policy definition and a single point of control.* Lights-out operation and unattended remote backup are also key to many storage management solutions.** Storage management solutions are most manageable when integrated into management systems such as HP OpenView, which provides integrated network and system management services dealing with monitoring, problem management, and configuration and change services.

Backup and Recovery

One of the most important needs in enterprise-wide storage is backup and recovery. Very early in a solution deployment, IT managers must establish a backup and recovery policy that provides the appropriate level of data integrity. This policy must ensure that critical data can be completely and quickly recovered from a backup even in the event of a disaster. Equally critical is minimizing planned downtime, or completely avoiding downtime, to create a backup while keeping user applications up and running.

* A central policy describes a set of features that allow an administrator to define policies about how distributed storage is to be managed from a central management console. For example, an administrator defines for a networked environment which data needs to be backed up, when it will be backed up, which device will be used for backup, and so on.

** In the IT community lights-out operation means that an IT environment can run without local operators or administrators.

Archiving and Retrieving

The main reason for implementing archival and retrieval solutions is the need to keep data long-term and guarantee retrieval when access is required. The data is typically copied onto a different medium such as tape or optical disk, while the original copy is deleted from magnetic disk. Archived data is not frequently accessed, but sophisticated retrieval mechanisms need to be available. In many cases, archiving data is required for legal or internal auditing purposes. The archiving procedure includes storing data that logically belongs together in long-term storage, such as a finished project, a finished design, or a client record.

Hierarchical Storage Management

Hierarchical storage management, or HSM, efficiently manages data stored on magnetic disks, optical disks, and tapes. Depending on cost versus performance requirements, data is kept on one or more of the different storage hierarchy levels and migrated transparently among the storage media according to customer-defined policies. An HSM system reduces ongoing storage configuration tasks, such as moving data manually between levels in the hierarchy and subsequent management costs. It also eliminates frequent storage maintenance, such as manually archiving files onto tape to free disk space, and it helps reduce the need to acquire more expensive media, such as magnetic disks, for infrequently accessed data. For example, files are migrated from disk to tape or optical storage if they are not accessed for a certain time period. Statistical data about access patterns can help to define the right migration policy. Also, keeping statistical data about migration patterns and creating appropriate reports will help to implement the right storage management policies for an organization.

Media Management

The storage services discussed above handle copying or moving data onto media or retrieving data from media. Media management, which keeps track of removable media such as tapes or optical devices, deals with the medium itself and not with the data on the medium. A media management system protects data on the media and makes the media pools available to storage management applications. Typical media management functions include mount and unmount media, rotate media, and provide statistical information about the media.

Most of today's backup, retrieval, archival, or HSM products have their own integrated media management functionality dedicated to a specific product. Enterprise storage management solutions require generic media management services delivered in an integrated way, so that media use can be managed and optimized across applications and systems.

Enterprise-Wide Storage Management

IT departments also require consistent and effective management capabilities for storage management across the enterprise environment. To provide these management services, a complete storage solution must provide:

- A single point of control, which is consolidated console management, including:
  - Central policy definition
  - Central monitoring and problem management
  - Central configuration of storage
- Multivendor availability and support
- Scalable, modular services
- Integration with an industry-standard network and system management framework
- High availability of key storage management components.

HP Storage Management Solutions

HP can offer many different solutions to an organization's storage needs because of the combined effort of major HP business organizations in the areas of network and system management, storage peripherals, and UNIX® servers.

However, each customer's needs for storage management solutions are different. No one vendor can provide a single solution for every environment. Rather than create a monolithic, proprietary solution, HP is working closely with third-party partners and many diverse HP divisions to create an open, standardized environment in which many vendors can participate in creating solutions.

HP Storage Management Software

Central to HP's storage management offering is the software that links all the other pieces of a storage management solution together. HP's two leading storage management products, HP OpenView OmniBack II and HP OpenView Omni-Storage, are client/server solutions that give IT managers the flexibility to manage distributed storage centrally and delegate management responsibility to distributed sites or departments. HP OmniBack II products address issues associated with data loss and protection, and HP OmniStorage products address issues associated with space management.

HP OpenView OmniBack II. HP offers two backup solutions: HP OpenView OmniBack II for Workgroups, which provides an entry-level backup solution ideally suited for the workgroup environment, and HP OpenView OmniBack II, which provides a comprehensive backup management solution to cover all sizes of environments, including the whole enterprise.

The HP OmniBack II architecture (Fig. 4) consists of three major pieces:

- Backup Manager. This module centrally administers and controls the backup environment.
- Backup Device Servers. These servers run on the system to which the backup device is connected. A backup environment can have many backup device servers.
- Clients. All systems being backed up need a client to invoke the backup utilities.

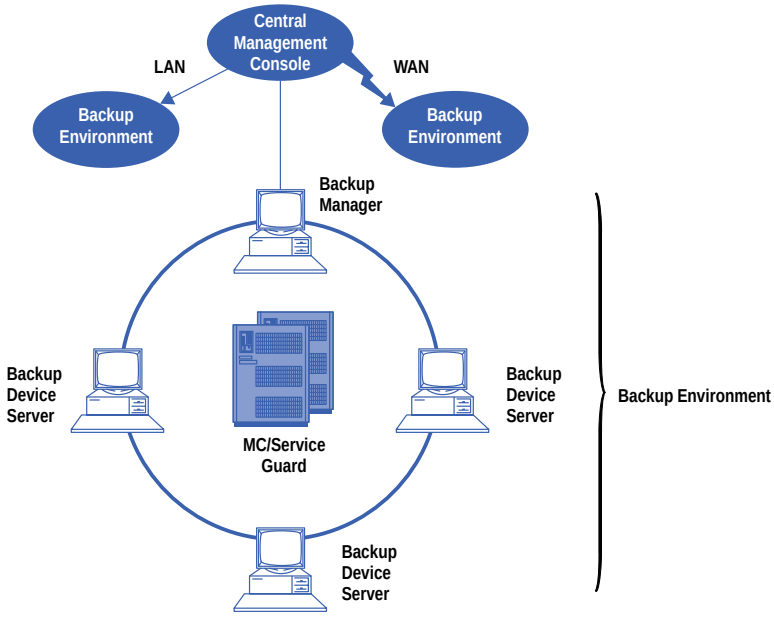


Fig. 4. *The backup environment and components for HP OpenView OmniBack II.*

All three components can run on the same system or can be distributed. OmniBack II has its own management interface and can be run inside or outside the HP Openview management interface.

OmniBack II is a scalable and flexible solution. Through its policy-driven, centrally managed, automated backup capabilities, OmniBack II reliably protects data distributed throughout the entire network. Easy-to-use backup and restore functionality provides management for desktop PCs to UNIX-based business servers. In combination with HP OpenView IT/Operations, administration and problem management for the entire enterprise can be centralized.

Sophisticated media and device management combined with support for mainframe-class library systems, including silos, make OmniBack II the ideal solution for data centers.*

Increased uptime for application and database servers can be achieved through high-performance offline backup, requiring only a minimum of application downtime. This can be extended to 100% application availability through online backup of business data.

The main features provided by OmniBack II include:

- Network backup and recovery
- Support for a broad range of devices and libraries
- Online backup of applications and databases such as SAP/R3, Oracle, and Sybase
- Sophisticated media management
- Support for major UNIX and PC platforms, including Windows NT
- High-performance backup and recovery from multiple drives in parallel, each running at its full native throughput
- Integration with HP OpenView IT/Operations
- Integration with HP OpenView OmniStorage, HP's hierarchical storage management solution.

HP OpenView OmniBack II for Workgroups. OmniBack II for Workgroups is a complete solution that offers everything needed for a low-administration, automatic, unattended, and reliable network backup and recovery solution. It is targeted toward

* Silos are very large tape libraries.

smaller multivendor computing environments without dedicated administrators. OmniBack II for Workgroups includes the following features:

- Automated and reliable network file system backup and recovery
- Sophisticated and automated media management, autoloader support
- Support for all major UNIX and PC platforms
- Easy-to-use intuitive graphical user interface with many built-in browsers and selection lists.

HP OpenView OmniStorage. OmniStorage is HP's hierarchical storage management solution. It offers benefits in environments where a significant amount of data needs to be online, but where not all of the data is frequently accessed.

OmniStorage provides high-capacity, cost-effective online storage by supporting HP's broad range of optical libraries and the newest tape libraries. According to policies defined by the customer, files are automatically and transparently migrated among the levels of storage hierarchy.

Fig. 5 shows a typical environment in which OmniStorage runs. The two main pieces of OmniStorage are the manager and the clients. The manager administers and controls the storage environment, and the clients invoke the OmniStorage functions on behalf of users.

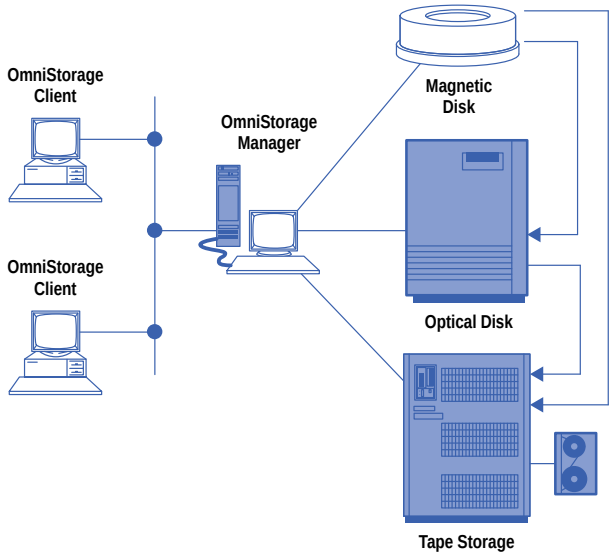


Fig. 5. *The components and environment for HP OpenView Omni-Storage.*

OmniStorage is tightly integrated with HP OpenView IT/Operations, providing easy administration and problem management of multiple OmniStorage installations from a central workstation console. OmniStorage also integrates with OmniBack II for automated backup and recovery of the HSM environment. However, OmniStorage can run as a standalone product, which allows customers to implement storage management in phases.

OmniStorage provides optimal performance if users frequently access only a subset of the data. Additionally, Omni-Storage can be used for databases if they are based on a file system and if major parts of the database, such as decision support systems, are not frequently accessed.

Finally, OmniStorage provides the following features:

- Policy-driven automatic and transparent file migration
- Network migration for HP-UX* and Solaris operating systems
- Additional multivendor support through NFS
- Exceptionally fast rebuild capabilities in case of data loss
- Configurable demigration strategy
- Archival to WORM (write once, read many) disks
- Integration with HP OpenView OmniBack II
- Integration with HP IT/Operations
- Support for data warehouse environments.

HP OpenView Solutions

HP OpenView's solutions are part of a strategy for managing multivendor networks, systems, applications, and databases from the mainframe to the desktop PC. The HP OpenView portfolio and companies that provide network and system

management solutions (solution partners) give IT managers the tools to control and manage all enterprise resources and devices centrally, while reducing the cost of systems operations and administration. Besides OmniBack and OmniStorage, more than 250 HP OpenView-based management solutions from HP and solution partners integrate with a complete set of common management services to help customers improve service and reduce operation costs.

Central to the HP OpenView products is a user interface that provides a focal point from which the IT staff can manage computer systems and network devices. Although control is centralized through the interface, management functions can be distributed across the enterprise. More important, flexible, distributed interfaces allow several operators and administrators to be involved in the process of IT management.

As an important part of the HP OpenView solutions framework, HP OpenView IT/Operations provides centralized operations and problem management with distributed intelligence across multivendor platforms. With intelligent agents (managed nodes) installed throughout the enterprise, IT/Operations collects up-to-date, accurate information to provide 24 × 7 (24 hours a day, seven days a week) uptime for mission-critical applications. IT/Operations-managed nodes gather information, messages, and monitoring values from a variety of sources. Filters and thresholds ensure that only relevant information is forwarded to the central management system and presented to the responsible IT/Operations operators.

HP and Third-Party Storage Peripherals

The HP 9000 supports mass storage products that provide online, nearline, and offline storage capabilities. The primary differentiator among these three categories of storage is access time. A storage device is considered online when the data access time is a fraction of a second. Nearline storage devices usually access data in the range of a few seconds to a few minutes. Offline storage devices typically require many minutes to hours to access data. Some offline storage strategies that require retrieval from a storage vault may take days before the data is available to the user.

HP and its partners can provide a wide variety of products to meet the individual needs of specific customer environments. These products can be mixed and combined with HP's storage management software to provide the needed end-user solutions.

Online Storage

HP offers two classes of online mass storage products: single-spindle disks and disk arrays.

Single-Spindle Disks. Single-spindle disks* offered by HP are either embedded in the host systems or provided externally within storage enclosures. These disks provide high-capacity, nonvolatile, fast-access mass storage. Single-spindle drives operate at 7200 rpm and are currently available in capacities of 1.05 Gbytes, 2.1 Gbytes, and 4.3 Gbytes as fast/wide differential drives.

These new drives are available as embedded devices in all of the HP 9000 servers, more than doubling the internal online storage capacity. With the addition of the 4.3-Gbyte drive, 21.5 Gbytes of external online storage capacity can now be housed in a single enclosure rack with up to 160 Gbytes in a 1.6-meter-high cabinet.

HP External Storage Enclosures. HP offers two families of storage enclosures for online storage: the HP 6000 SCSI mass storage family and the HP high-availability storage system. HP's high-availability storage system is based on a package design that delivers flexibility and ease of use while providing critical functionality to meet the needs of the enterprise. The system provides excellent availability, hot-pluggable power supplies, dual, power cords, cooling fans, and hot-pluggable storage modules. The subsystem connects to the server via dual SCSI buses, increasing reliability and enabling disk mirroring in the same enclosure.

HP Disk Arrays. A disk array is a storage system consisting of multiple disk drive mechanisms under the command of an array controller that communicates with the host (Fig. 6).[1] The key benefit of disk arrays is high data protection. Arrays also provide high storage capacities, connectivity, and configuration flexibility.

HP currently offers three primary disk array families associated with the HP 9000 business server product line. The first family is the HP high-availability disk arrays Model 10 and Model 20, which have a raw capacity of 6 to 80 Gbytes, support RAID levels 1 and 5, and have dual and hot-swappable controllers and redundant cooling and power.**

The second disk array offering is EMC's Symmetrix 3000. The Symmetrix 3000 is a high-performance integrated-cache disk array designed for online storage. As such, the Symmetrix 3000 provides a high level of online performance, an online capacity of up to 1.1 terabyte, and manageability and high availability to HP 9000 business servers. The result is a mainframe-class data storage solution that is simple to manage and is delivered in a high-performing, scalable, protected, and open architecture.

The final disk array is the fault-tolerant, self-configuring, high-performance HP disk array product with AutoRAID technology. The HP AutoRAID disk array eliminates the need for system administrators to understand RAID levels.

* A collection of disk platters on a single spindle.

** RAID = Redundant Array of Independent Disks.

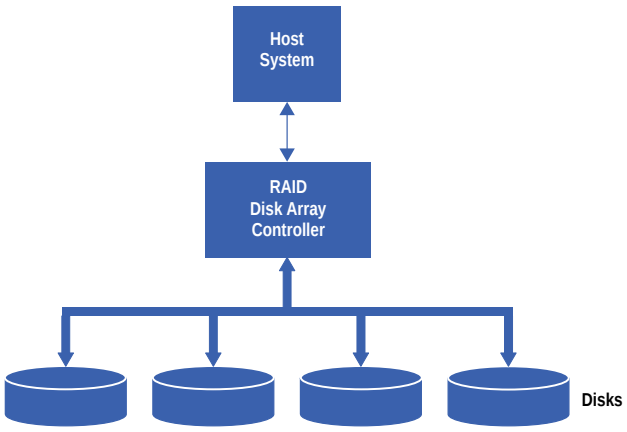


Fig. 6. *A typical RAID architecture.*

It dynamically adapts to the system's workload, thus optimizing for performance and cost. Finally, it offers a raw capacity of up to 24 gigabytes.

Nearline Storage

Most files and applications stored on hard disks are never used. Thus, the major benefit of hard disks—high-performance access—is squandered on dormant data. Cost-sensitive environments would be better served by a hierarchical storage management solution in which active data is stored on hard disks, while dormant or infrequently used data is cost-effectively stored offline or nearline in media such as optical disks.

HP SureStore Optical Storage Products. HP offers a broad family of optical disk drives, ranging in capacity from 40 Gbytes to 618 Gbytes. HP offers multifunction magnetooptical drives with rewritable and WORM disks. A rewritable optical disk can be written up to 10 million times. A WORM disk can be written once but cannot be erased or overwritten, adding a higher degree of security.

The main features of these nearline storage devices include:

- Fast, near-hard-disk transfer and seek times
- High capacity
- Low risk—no disk crashes with optical disks
- Online data availability on a random-access device
- Online drive replacement—provides assurance that the optical system is persistently available
- Removable media
- Long life—provides more than 100 years of media life without maintenance.

Offline Storage

HP's range of offline storage products provide high speeds and large capacities to meet the increasing demands of HP's high-performance workstations, network servers, and multi-user systems. HP has combined its reliable DAT products with an industry-leading autoloader design and networking software to give customers the flexibility they need for complete automated network backup.[2]

HP DAT Products. HP offers the latest DDS-2 tape drives in addition to DDS and DDS DC drives. The new DDS-2 format, combined with 120-meter tapes, has a native mode capacity of 4 Gbytes. With data compression, customers can typically store 8 Gbytes on a single tape. For unattended or lights-out operation, a six-cartridge autochanger is available to rotate media for full and incremental backup and restore operations. HP also supports 8 mm and QIC tape drives. The main features of these offline storage products include:

- Unattended backup
- High capacity with high reliability
- Easy storage in a fireproof safe according to industry standards
- DDS format can be interchanged with different manufacturers' tape drives.

Digital Linear Tape. HP 9000 servers support the HP DLT (digital linear tape) Library 4/48. This library consists of four Quantum DLT/4000 drives, accommodating 48 20-Gbyte tape cartridges and providing greater cartridge capacity than the DDS format. The DLT/4000 is a 0.5-in cartridge streaming tape with a capacity of 40 Gbytes per cartridge (with 2:1 compression), and a sustained transfer rate of 3 Mbytes/s. The HP DLT Library 4/48 enables fast, unattended backup of over

100 Gbytes of data within the brief windows of time available for backup in high-end OLTP (online transaction processing) and decision support system environments (Fig. 7).

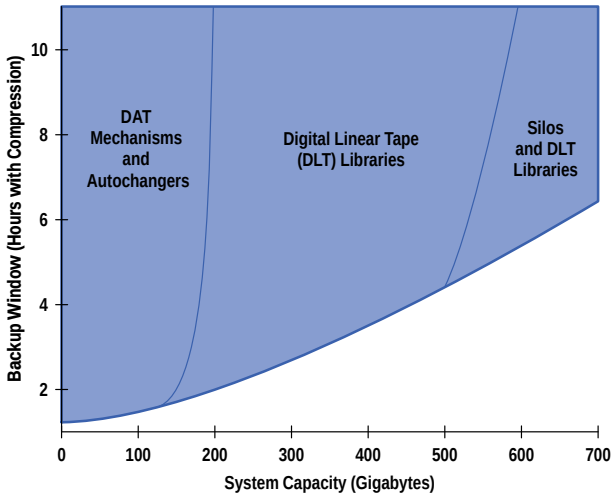


Fig. 7. *System capacities of different offline backup devices based on the backup window (i.e., the amount of time available for backup).*

The main features of digital linear tape include:

- A native-mode data transfer rate three times faster than competing technologies
- Greater media and drive head longevity
- Sophisticated tape indexing for fast-streaming file searches and restoration
- Higher compression ratio for most data types.
- Driver support provided by HP for the 18-track StorageTek tape drives (model 4781) and the 36-track single-ended tape drives (model 4791).

3480/3490 Compatible Tape Subsystems, Libraries, and Silos. HP provides driver support for 18-track StorageTek tape drives (Model 4781) and 36-track single-ended tape drives (Model 4791). Additionally, StorageTek offers Timberline 9490, a fast wide implementation of the Model 4791 with a 6-Mbyte/s drive. These drives are compatible with all StorageTek silos. HP supports StorageTek silos, including one with a 500-cartridge capacity and 90 cartridges/hour (upgradable to 1000 cartridges and 350 cartridges/hour) and another with a 6000-cartridge capacity and 350 cartridges/hour. Both connect to other devices and silos for easy growth.

HP 9000 Business Servers

Today's open systems for critical business computing environments require three essential elements. First, they must provide the storage management, data integrity, security, and manageability that information technology managers have come to expect in running business-critical applications on centralized processing systems. Second, they must provide connectivity and compatibility with the growing base of PC desktop users. Finally, they must offer flexibility, performance scalability, and technical innovation to keep up with emerging application demands.

As the leading open systems platform, HP 9000 business servers offer the benefits of all three elements in a single, unified, UNIX-based platform. The HP 9000 server platform is able to support environments of all sizes, ranging from workgroups and replicated sites to the departments and data centers of large enterprises. For storage management, the HP 9000 business servers offer the following features:

- Highly available and reliable systems environment
- Excellent data-movement management
- A dedicated storage server architecture that is designed for optimal database and file management
- Scalable from desktop to data center
- Hundreds of partners that ensure customizable solutions.

Storage Solutions for the Enterprise

To help understand how to plan a complete storage management solution, we have looked at scenarios common to many companies struggling with storage management demands. In each of the following examples, we'll examine the needs specific to each environment and the needs of centralized storage management, starting with small workgroups, and building up to the enterprise level. Then we'll show how HP and its partners can provide a unified solution.

Independent Workgroups

Many workgroups implement their own backup and recovery solutions. These solutions are typically managed by a part-time administrator. Major requirements for backup and recovery solutions include ease of use and automation. OmniBack II for Workgroups is the best backup and recovery product for independent workgroups. Combined with HP's low-cost, high-performance business servers and HP's DDS II device libraries, backup procedures can be automated and centrally controlled within the workgroup. OmniBack II for Workgroups provides easy and fast restoration of files and the potential to expand the workgroup or even consolidate multiple workgroups.

Solution Elements. The storage management solution from HP for independent workgroups includes:

- HP OmniBack II for Workgroups, with easy-to-use backup and recovery software for small environments
- HP 9000 Class D and E business servers for backup
- HP's DDS II autoloader as a cost-effective tape library.

Distributed Client/Server Workgroups

As information technology departments consolidate workgroup management, they need more centralized storage management for distributed, heterogeneous workgroups. Two main objectives of this scenario are to increase end user productivity by providing homogeneous and powerful storage services, and to increase operator productivity through central control and administration of those storage services over the LAN. Significant savings can be achieved through intelligent resource sharing and reduction of operational overhead.

OmniBack II centrally manages the complete backup and recovery process of large numbers of distributed workgroups by dividing large numbers of backup nodes into multiple manageable backup domains. Central control can be maintained at the enterprise console while delegating backup and recovery tasks to the individual end-user departments. OmniBack II can automate the complete backup process of distributed client/server workgroups.

In addition, data on shared file servers and on client disks grows dramatically. Migrating infrequently accessed data onto different storage media such as optical disks or tapes becomes an administrative nightmare. OmniStorage helps to increase the online storage capacity of clients and servers while keeping storage administration costs under control.

Solution Elements. The storage management solution from HP for distributed client/server workgroups includes:

- Centrally controlled backup and recovery for heterogeneous workgroups with OmniBack II
- Automated backup based on HP's DDS II Autoloader or DLT libraries
- HP 9000 business servers used as reliable and high-performing backup, restore, and HSM servers.
- Unlimited online storage based on OmniStorage combined with HP's optical or tape libraries
- Sophisticated problem management with HP IT/Operations.

Regional Distributed Systems and WAN Connections

Companies with branch offices in the retail or financial industries, for example, often have regional distributed systems connected via WANs. IT departments for these companies need to run the IT infrastructure of the branch offices without operators or administrators. Remote control and administration of storage services over the WAN are essential.

OmniBack II defines each remote branch office as a backup domain with one or more local backup servers. The complete backup and recovery administration process and control can be performed from a central console via WAN. By choosing the appropriate devices, with sufficient capacity for each of the remote offices, the backup and recovery process for the branch offices can be performed remotely.

Solution Elements. For regionally distributed systems the storage management solution from HP would include:

- Central backup and recovery for remote sites with OmniBack II
- HP 9000 Class D and E business servers for reliable backup and recovery at branches
- HP's DDS II autoloader for automated tape handling
- HP IT/Operations integration of OmniBack II for sophisticated central problem management.

Data Center and Mainframe Downsizing

When customers migrate from the mainframe to open systems, they expect the same functionality and scalability in storage services as they had with the mainframe. The IT department is expected to continue to provide the same level of assurance that computer services are available, reliable, and secure.

For example, a major multinational company using the HP-UX operating system with large SAP/R3 projects based on Oracle requires efficient, reliable, unattended automated backups and restores. These backups are in the multiple terabyte-per-week range and will move into 24 × 7 (24 hours a day, 7 days a week) operation. OmniBack II is a key part of the solution because it gives the customer online backup by integrating with either the SAP/R3 online utility or the Oracle database utility. OmniBack II also integrates with IT/Operations. This customer has complete centralized management of all backup sessions and devices with the option of managing a partially or fully distributed environment.

Because of OmniBack II's modularity and integration with other online backup services, such as those provided by Oracle and Sybase, customers can configure OmniBack II to expand to match their growing infrastructures. Many customers also may choose to add HP's MC/ServiceGuard to ensure high availability of stored data.

Solutions Elements

Data centers needing high-end backup and restore are presented with the following storage management solutions from HP:

- High degree of automation with OmniBack II and StorageTek's Tape Silo
- High backup and restore performance using StorageTek's Timberline tape drives
- High reliability through HP 9000 systems running the HP-UX operating system
- SAP/R3 online backup with OmniBack II integration
- Enterprise monitoring and problem management through HP IT/Operations integrated with OmniBack II
- Full consulting, from investigation through implementation, by HP's professional consulting services
- HP MC/ServiceGuard.

Data Warehousing: Scalable Storage Infrastructure

Much of a company's data remains valuable but does not need to be online and available all the time. Typical storage capacity requirements of data warehouses range from tens of gigabytes to several terabytes. Storage managers need the ability to move this data cleanly from primary to secondary storage and back to primary temporarily, as needed.

The combination of HP's business servers, magnetic disks, optical disks, and OmniStorage offer an ideal storage infrastructure for data warehouse environments. This solution offers efficient storage hardware costs and high scalability for large storage capacities.

The ideal ratio between magnetic and optical capacity depends on environment. Ratios in the range of 1:5 to 1:10 (i.e., 1 Gbyte of magnetic disk capacity assigned to 5 or 10 Gbytes of optical capacity) have been implemented successfully.

Solution Elements. HP provides the following storage management solutions for data warehouses:

- Automated data migration with OmniStorage
- Online access to secondary and tertiary storage through HP's optical and tape libraries
- High-performance data warehouse applications based on HP 9000 Class K and T business servers.

Conclusion

The ideal storage system would provide complete and integrated storage management functionality, smooth integration with multiple file systems and databases across a broad set of operating systems, and support for a large variety of peripherals to satisfy the needs of different storage management applications. Preventing this ideal solution from occurring are a multitude of nonintegrated storage components from different vendors. This situation forces administrators to use single-unit solutions for storage management, which leads to redundant and inconsistent management environments.

The HP storage architecture offers a streamlined and unified interaction among diverse storage components. This architecture allows for customized solutions using plug-and-play components and enables different storage components to interact consistently. For example, HP's backup solutions are being integrated via APIs with databases such as Oracle and Sybase.

References

1. T. Skeie and M. Rusnack, "HP Disk Array: Mass Storage Fault Tolerance for PC Servers," *Hewlett-Packard Journal*, Vol. 46, no. 3, June 1995, pp. 71-81.
2. S. Dimond, "DDS-2 Tape Autoloader: High-Capacity Data Storage in a 5 1/2-Inch Form Factor," *Hewlett-Packard Journal*, Vol. 45, no. 6, June 1994, pp. 12-20.

An Introduction to Fibre Channel

Fibre Channel is a flexible, scalable, high-speed data transfer interface that can operate over a variety of both copper wire and optical fiber at data rates up to 250 times faster than existing communications interfaces. Networking and I/O protocols, such as SCSI commands, are mapped to Fibre Channel constructs, encapsulated, and transported within Fibre Channel frames.

by Meryem Primmer

Fibre Channel is a standard, efficient, generic transport mechanism whose primary task is to transport data at the fastest speeds currently achievable with the least possible delay. It is a flexible, scalable method for achieving high-speed interconnection, communication, and data transfer among heterogeneous systems and peripherals, including workstations, mainframes, supercomputers, desktop computers, and storage devices. It handles both networking and peripheral I/O communication over a single channel using the same drivers, ports, and adapters for both types of communication.

Fibre Channel began in the late 1980s as part of the IPI (Intelligent Peripheral Interface) Enhanced Physical Project to increase the capabilities of the IPI protocol. That effort widened to investigate other interface protocols as candidates for augmentation. The first year of the project was spent looking for existing implementations to adopt, but none were found to be sufficient. The focus then changed to develop a new implementation. That implementation became Fibre Channel. Fibre Channel was approved as a project in 1988 by ANSI X3T9.

During the first year of investigation the ANSI working group decided to adopt a serial rather than a parallel bus interface. IBM's 8B/10B encode/decode scheme was adopted, and a decision was made to support both copper cable and optical fiber. Copper can be used for low cost while optical fiber can be used for distance. *Fibre* is a generic term used by the Fibre Channel standard to refer to all the supported physical media types.

The first draft of the Fibre Channel standard was developed in 1989. The standard addresses the need for very fast transfers of large volumes of data, while at the same time relieving systems of the need to support the multitude of channels and networks currently in use. The Fiber Channel standard covers networking, storage, and data transfers. In October 1994 the Fibre Channel physical and signaling interface standard, FC-PH, was approved as ANSI standard X3.230-1994.

Fibre Channel is structured as a set of hierarchical functions that support a number of existing protocols, such as SCSI (Small Computer System Interface) and IP (Internet Protocol), but it does not have a native I/O command set. It is not a high-level protocol like SCSI, but does contain a low-level protocol for managing link operations. Fibre Channel is not aware of, nor is it concerned with the content of the user data being transported. Networking and I/O protocols, such as SCSI commands, are mapped to Fibre Channel constructs and encapsulated and transported within Fibre Channel frames. The main purpose of Fibre Channel is to have any number of existing protocols operate over a variety of physical media and existing cable plants.

Fibre Channel is a high-speed data transfer interface that can operate from 2.5 to 250 times faster than existing communications interfaces. Its performance is both scalable and extendable and it supports multiple cost/performance levels, from small configurations such as disk arrays and low-cost, low-performance I/O devices and small systems to high-performance supercomputers and large distributed systems.

Fibre Channel runs at four speeds (actual data throughput): 100 megabytes per second (Mbytes/s), which translates to 1062.5 megabaud, 50 Mbytes/s or 531.25 megabaud, 25 Mbytes/s or 265.625 megabaud, and 12.5 Mbytes/s or 132.812 megabaud. A single 100-Mbyte/s Fibre Channel port can replace five 20-Mbyte/s SCSI ports, in terms of raw throughput. Fibre Channel provides a total network bandwidth of about one gigabit per second.

Fibre Channel operates over a variety of both copper wire and optical fiber at scalable distances, as shown in Table I. Distances are easily extendible using repeaters or switches.

Fibre Channel provides full duplex operation with separate transmit and receive fibers.

Another advantage of Fibre Channel is that it uses small connectors. The serial connectors used for Fibre Channel are a fraction of the size of SCSI parallel connectors and have fewer pins, thereby reducing the likelihood of physical damage. Also, depending on the topology, many more devices can be interconnected on Fibre Channel than on existing channels.

Topologies

Fibre Channel can be implemented in three topologies to interconnect varying numbers of devices, called *nodes* in Fibre Channel terminology. The topologies are point-to-point, arbitrated loop, and crosspoint switched, or *fabric* (a Fibre Channel term for a network of one or more switches connecting multiple nodes). Nodes contain one or more ports, such as an I/O adapter, through which they communicate over Fibre Channel. A generic node port is called an *N_Port*. The connections between ports are called *links*.

Table I
Fibre Channel Media, Data Rates, Distances, and Transmitters

Media Type	Data Rate (Mbytes/s)	Maximum Distance	Transmitter Type
150-ohm Twinax or STP	100	30 m	ECL
	50	60 m	ECL
	25	100 m	ECL
75-ohm Video Coax	100	25 m	ECL
	50	50 m	ECL
	25	75 m	ECL
	12.5	100 m	ECL
75-ohm Miniature Coax	100	10 m	ECL
	50	20 m	ECL
	25	30 m	ECL
	12.5	40 m	ECL
105-ohm Type-1 Shielded Twisted-Pair Electrical	25	50 m	ECL
	12.5	100 m	ECL
62.5-μm Multimode Optical Fiber	100	300 m	SW Laser
	50	600 m	SW Laser
	25	1 km	LW LED
	12.5	2 km	LW LED
50-μm Multimode Optical Fiber	100	500 m	SW Laser
	50	1 km	SW Laser
	25	2 km	SW Laser
	12.5	10 km	LW LED
9-μm Single-Mode Optical Fiber	100	10 km	LW Laser
	50	10 km	LW Laser
	25	10 km	

ECL = Emitter-Coupled Logic, LW = Longwave, SW = Shortwave,
LED = Light-Emitting Diode, STP = Shielded Twisted-Pair

Point-to-point (Fig. 1) is a direct channel connection between two N_Ports, typically between a processor and a peripheral device controller. In this topology exactly two devices are connected together. No fabric elements exist and no fabric services, such as name mapping, are necessary. Point-to-point is the default topology.

Fibre Channel arbitrated loop, or FC-AL, is a method for interconnecting from two to 126 devices through attachment points called *L_Ports* in a loop configuration. L_Ports can consist of I/O devices and systems of various performance levels. FC-AL is a low-cost solution because it does not require hubs and switches. FC-AL is a good choice for small to medium-sized configurations and provides an upward growth path by interconnecting the loop with a fabric through an *FL_Port*. Arbitrated loop is the most common Fibre Channel topology.

Fig. 2 shows the Fibre Channel arbitrated loop topology. A *private loop* (Fig. 2a) consists only of nodes, called *NL_Ports*, and does not connect with a fabric. A *public loop* (Fig. 2b) connects with a fabric via an FL_Port. A *disk loop* uses the loop topology to interconnect a number of high-performance disks, for example, a RAID (Redundant Array of Inexpensive Disks) device. Fig. 3 shows an office configured in a public arbitrated loop topology, and Fig. 4 shows a private disk loop.

All devices on the arbitrated loop share the bandwidth of the loop and the management of the loop. No dedicated loop master exists, and any node is capable of being the loop master. Which node performs the loop master functions is negotiated when the loop is initialized.

Each node has equal opportunity to communicate with another node by arbitrating for temporary ownership of the loop. An arbitration scheme using a fairness algorithm is used to establish a circuit between two NL_Ports on the loop before they

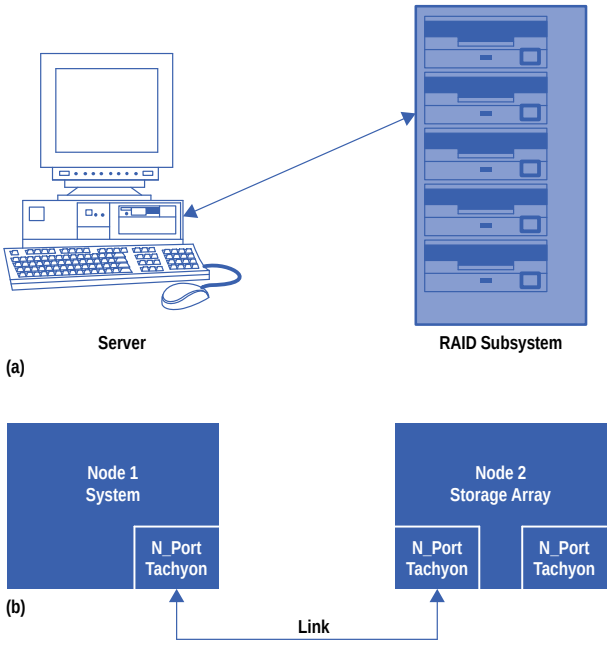


Fig. 1. *(a) Two devices connected point-to-point. (b) Fibre Channel point-to-point topology. Tachyon is HP's gigabit Fibre Channel controller chip.*

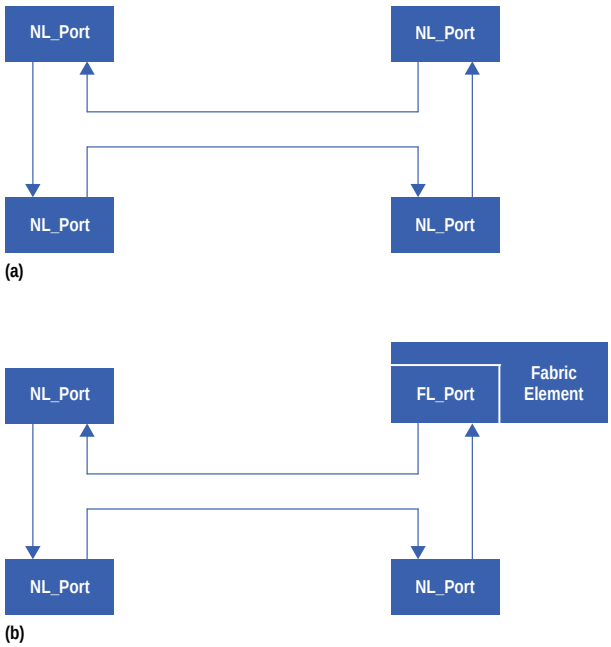


Fig. 2. *Fibre Channel arbitrated loop topology. (a) Private Loop. (b) Public loop.*

can communicate. Only one communication, or loop circuit, can be active at a time. After relinquishing the loop, an NL_Port cannot win arbitration again until all other arbitrating ports have had their turn.

The third Fibre Channel topology is crosspoint switched, or fabric. Fig. 5 shows a generic fabric topology, and Fig. 6 shows the Fibre Channel fabric topology with a single switching or fabric element.

A fabric topology is implemented as one or more switching elements. A fabric appears as a single entity to attached nodes, called *F_Ports*, even though the fabric can consist of multiple switches. Typically, a switch has from four to 16 F_Ports attached to it. In theory, there is no size limit to the number of nodes that can interconnect in a fabric, but addressing space limits the number to a maximum of 2^{24}. The fabric topology is good for interconnecting large numbers of devices and complex configurations.

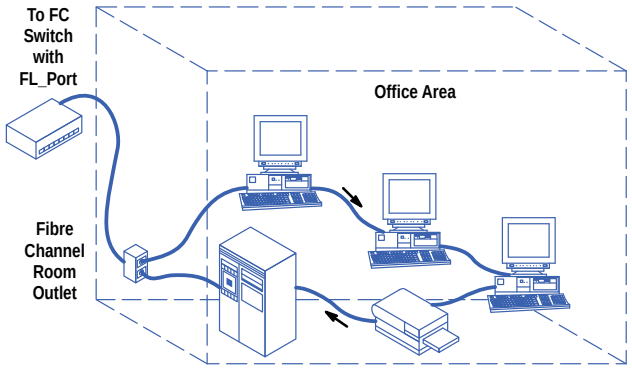


Fig. 3. *An office configured in a public arbitrated loop topology.*

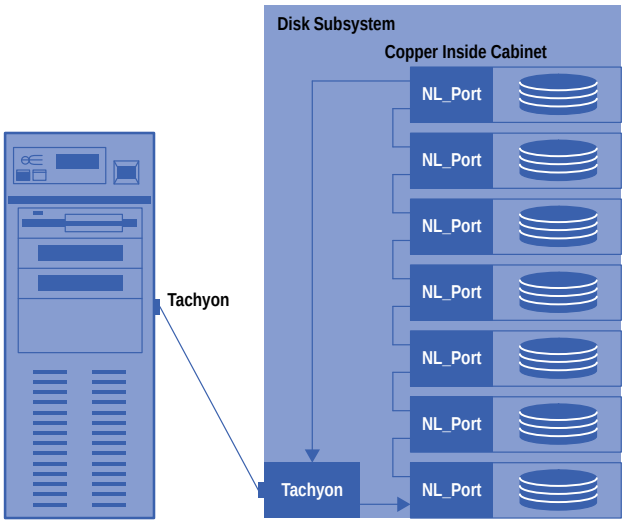


Fig. 4. *A private disk loop.*

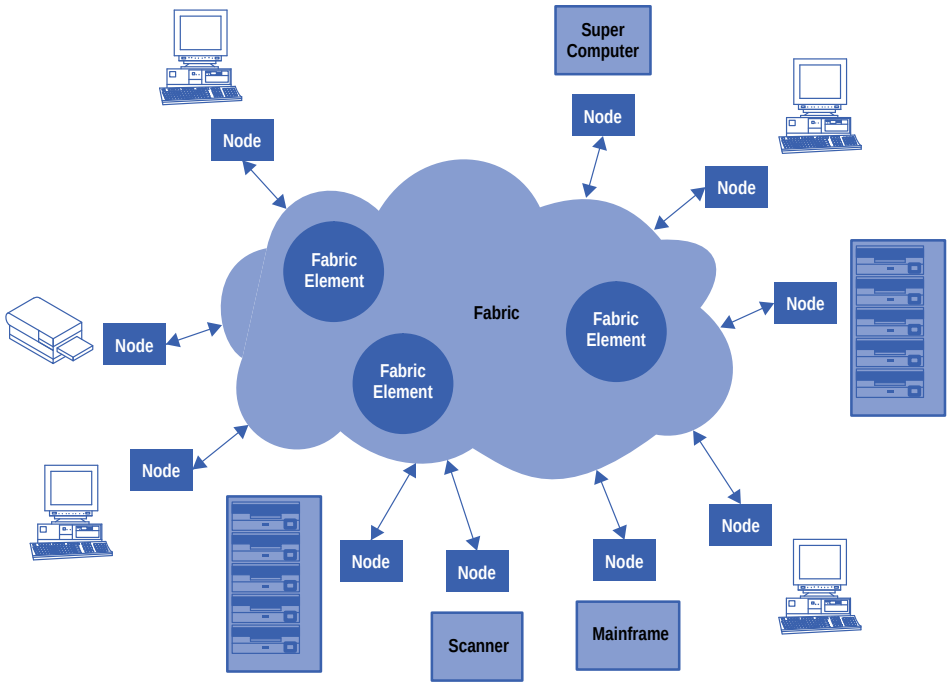


Fig. 5. *A generic fabric topology.*

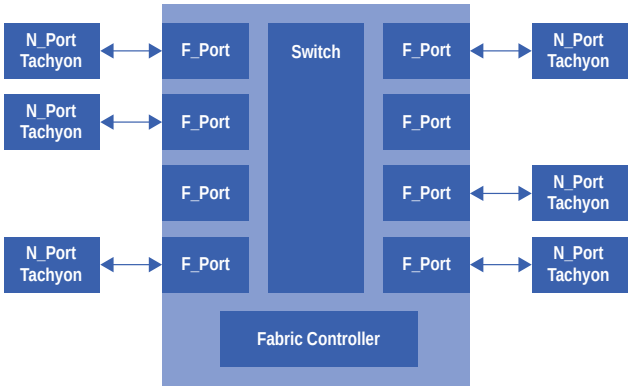


Fig. 6. *Fibre Channel fabric topology with a single switching element.*

The structure and operations of the fabric are transparent to the F_Ports attached to it. The fabric topology is self-managed, with the fabric performing station management functions and the routing of frames. Each port only needs to manage a point-to-point connection between itself and the fabric.

A Layered Approach

Fibre Channel is structured as a set of five hierarchical functional levels (see Fig. 7). The user protocol being transported over the Fibre Channel—SCSI or IPI (Intelligent Peripheral Interface), for example—is known as the *upper level protocol* (ULP) and is outside the scope of the Fibre Channel layers. The Tachyon Fibre Channel protocol chip described in ***Article 12*** implements the FC-1 and FC-2 layers, which are shaded in Fig. 7. Tachyon also implements SCSI assists and IP checksumming, shown as shaded boxes at the FC-4 level.

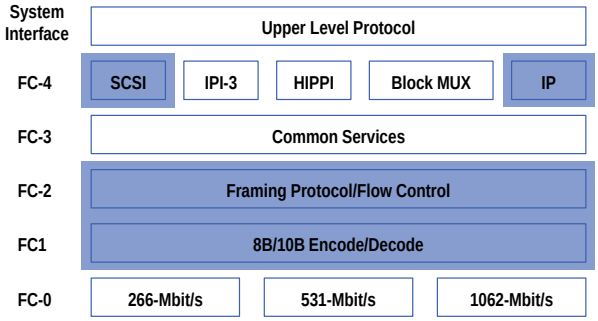


Fig. 7. *Fibre Channel's five layers.*

FC-4: The Protocol Mappings Layer. This topmost Fibre Channel level defines the mapping of the ULP interfaces to the lower Fibre Channel levels. Fibre Channel supports multiple existing protocols, including SCSI, IP, and IPI. Each ULP supported by Fibre Channel requires a separate FC-4 mapping and is specified in a separate FC-4 document. For example, the Fibre Channel protocol for SCSI, which is known as FCP, defines a Fibre Channel mapping layer that uses the services of the lowest three Fibre Channel layers to transmit SCSI command, data, and status information between a SCSI initiator and a SCSI target. ULPs are not tied to a particular physical medium or interface. For example, SCSI is supported without requiring a SCSI bus.

FC-3: The Common Services Layer. Nodes can be computer systems or peripheral devices. The FC-3 level defines a set of services that are common across multiple ports of a node. The FC-3 layer is still being formulated in the ANSI committee and no functions have been formally defined.

FC-2: The Framing Protocol Layer. This level defines the signaling protocol, including the frame and byte structure, which is the data transport mechanism used by Fibre Channel. Included in this level is the framing protocol used to break sequences into individual frames for transmission, flow control, 32-bit CRC generation, and various classes of service.

The FC-2 layer also handles hardware disassembly and reassembly of sequences of data. Defined in this layer are a few built-in command primitives, called *ordered sets*, for handling such functions as configuration management, error recovery, frame demarcation, and signaling between two ends of a link.

A *frame* (Fig. 8) is the smallest indivisible unit of user data that is sent on the Fibre Channel link. Frames can be variable in length, up to a maximum of 2148 bytes long. Frame size depends on implementation, not hardware or software. Each frame contains a four-byte Start of Frame delimiter, a 24-byte header, up to 2112 bytes of FC-4 payload consisting of zero to 64 bytes of optional headers and zero to 2048 bytes of ULP data, a four-byte CRC, and a four-byte End of Frame delimiter.

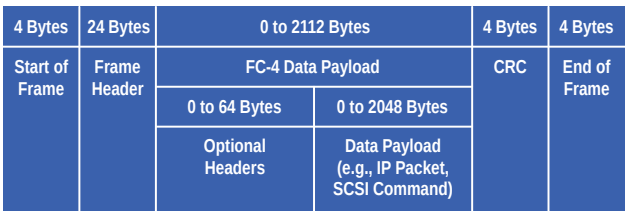


Fig. 8. *A Fibre Channel frame.*

A *sequence* is a set of one or more related frames. For example, a large file transfer would be accomplished in a sequence consisting of multiple frames.

An *exchange* contains one or more sequences. It is comparable to a SCSI I/O process, and is the mechanism for coordinating the exchange of information between two communicating N_Ports in a single operation.

In general, the sequence is the Fibre Channel error recovery boundary. That is, selective retransmission of frames for error recovery is not supported in the Fibre Channel physical and signaling interface, FC-PH. If an error is detected in a transmitted frame and the error policy requires error recovery, the sequence to which the frame belongs may be retransmitted.

Fibre Channel provides three classes of service, which are managed by the FC-2 layer. Class 1 dedicated connection service provides a dedicated or circuit-switched connection between two N_Ports. The connection must be established before communication can begin and must be torn down when communication is completed. Class 1 guarantees delivery of frames in the order in which they were transmitted. Confirmation of delivery also is provided. Class 2 multiplex service provides a connectionless, frame-switched link. Delivery is guaranteed, but not necessarily in order if multiple routes exist through the fabric. Class 2 also provides acknowledgement of receipt. Class 3 datagram service is a connectionless service similar to class 2, but without confirmation of receipt. Neither delivery nor receipt order is guaranteed in class 3.

FC-1: The Encode/Decode Layer. This layer defines the transmission protocol, including the 8B/10B encode/decode scheme, byte synchronization, and character-level error control. 8B/10B is a dc-balanced encode/decode scheme that provides good transition density for easier clock recovery and character-level error detection. In this scheme, 8-bit internal bytes are encoded and transmitted on the Fibre Channel link as 10-bit transmission characters. The transmission characters are converted back into 8-bit bytes at the receiver. Using 10 bits for each character provides 1024 possible encoded values rather than only the 256 values that are possible for 8-bit characters. Not all of the 1024 possible values are used. To maintain a dc balance on the link, only those that contain four zeros and six ones, six zeros and four ones, or five zeros and five ones are used. Some of the extra 10-bit characters are used for low-level link control. One special character called a *comma* is used for byte synchronization.

FC-0: The Physical Layer. FC-0, the lowest of the five levels, defines the physical characteristics of the media, including cables, connectors, drivers (ECL, LEDs, shortwave lasers, longwave lasers, etc.), transmitters, transmission rates, receivers, and optical and electrical parameters for a variety of data rates and physical media. Reference **1** describes HP products that implement the FC-0 layer.

Collectively, the three lowest layers constitute the Fibre Channel physical and signaling interface, FC-PH. FC-PH is a channel/network hybrid. It supports channel interfaces for peripherals—for example, SCSI, IPI, and HIPPI (High-Performance Parallel Interface)—as well as network protocols such as TCP/IP. FC-PH is similar enough to a network to gain connectivity, distance, and serial interfaces, while being enough like an I/O channel to retain simplicity, reliability, and hardware functionality.

Reference

1. J.S. Chang, et al, "A 1.0625-Gbit/s Fibre Channel Chipset with Laser Driver," *Hewlett-Packard Journal*, Vol. 47, no. 1, February 1996, pp. 60-67.

Bibliography

1. *Fibre Channel—Physical and Signaling Interface (FC-PH)*, X3.230-1994, Rev. 4.3, American National Standards Institute.
2. *Fibre Channel—Arbitrated Loop (FC-AL)*, X3.272-199x, Rev. 4.5, American National Standards Institute, June 1995.
3. *Fibre Channel Protocol for SCSI (FCP)*, X3.269-199x, Rev. 012, American National Standards Institute, May 30, 1995.
4. *Fibre Channel: Connection to the Future*, The Fibre Channel Association, 1994.
5. The Fibre Channel Association server URL: **http://www.amdahl.com/ext/CARP/FCA/FCA.html**

Tachyon: A Gigabit Fibre Channel Protocol Chip

The Tachyon chip implements the FC-1 and FC-2 layers of the five-layer Fibre Channel standard. The chip enables a seamless interface to the physical FC-0 layer and low-cost Fibre Channel attachments for hosts, systems, and peripherals on both industry-standard and proprietary buses through the Tachyon system interface. It allows sustained gigabit data throughput at distance options from ten meters on copper to ten kilometers over single-mode optical fiber.

by Judith A. Smith and Meryem Primmer

Relentlessly increasing demands of computer systems continue to stress existing communication architectures to their limits. Even as processor speeds continue to improve dramatically, they are barely keeping up with increasing numbers of concurrently running applications and CPU-intensive applications, such as multimedia, with higher data throughput requirements. Additionally, as the number of interconnects between systems and I/O devices continues to increase, I/O channels become bottlenecks to system performance. A channel such as SCSI (Small Computer Systems Interface), which operates at a maximum throughput of 20 megabytes per second in fast and wide mode, simply cannot keep pace with ever-increasing processor speeds and data rate requirements.

Another challenge of contemporary computer systems is the trend to more widely distributed systems, which require greater interface distances. Current parallel bus interconnects between systems and their I/O devices cannot operate over the distances needed for true distributed systems, such as LANs spanning campus areas and high-availability applications requiring remote mirrored disks for disaster recovery. SCSI, for example, is limited to a distance of six meters single-ended (single wire per signal) and 25 meters differential (two wires per signal).

Current peripheral interconnect protocols are limited in the number of devices they can interconnect. For example, parallel SCSI can connect eight devices and 16-bit wide SCSI can connect 16 devices. In addition, peripheral connectors are becoming too large to fit into the shrinking footprints of systems and peripherals. Other SCSI limitations include half-duplex operation only, lack of a switching capability, inability to interconnect individual buses, and the need for customized drivers and adapters for various types of attached devices.

Computer room real estate also is becoming scarce and expensive, fueled by increasing numbers of racked computers, insufficient room to connect desired numbers of peripheral devices, and more complex cabling. At the same time, data storage requirements are skyrocketing as backups of terabytes of data are becoming commonplace. An additional problem is that ever-increasing amounts of data must be backed up over too-slow LANs, making timely, low-cost backups ever more difficult to accomplish.

For all these reasons, today's parallel bus architectures are reaching their limits. Fibre Channel provides solutions to many of these limitations. Fibre Channel is a forward-thinking solution to future mass storage and networking requirements. ***Article 11*** presents a technical description of Fibre Channel.

HP and Fibre Channel

Searching for a higher-performance serial interface, HP investigated a number of technologies. HP chose Fibre Channel over other serial technologies because it supports sustained gigabit data transfer rates (the fastest throughput of any existing interface), it allows networking and mass storage on the same link, and it is an open industry standard.

Although Fibre Channel faces the challenges of lack of market awareness and industry coordination and a perception that it can be expensive, it is a stronger contender than alternative serial technologies for a number of important reasons. It is an open industry standard and an approved ANSI standard, it has vendor support from switch, hub, and disk drive suppliers, it is extensible, offering three topologies and four data transfer rates, and it supports both networking and mass storage.

Fibre Channel's increased bandwidth provides distance flexibility, increased addressability, and simplified cabling. Fibre Channel has versatility, room for growth, and qualified vendor support. Mass storage suppliers are using Fibre Channel to interconnect subsystems and systems and to control embedded disk drives. Some midrange system (server) suppliers are using Fibre Channel as a clustering interconnect and for specialized networking. Fibre Channel supporters and developers include HP, Sun Microsystems, SGI, and IBM for workstations, HP, Sun, Unisys, and Compaq in the server market, HP,

Seagate, Quantum, and Western Digital for disk drives, and Data General's Clariion Business Unit, DEC, Symbios, Fujitsu, and EMC for disk arrays, in addition to over 100 other vendors (source: The Fibre Channel Association).

Fibre Channel's main virtue is that it works as a networking interface as well as a channel interface. Fibre Channel is one of three complementary networking technologies that HP sees as the next step upwards in network performance (see Fig. 1). The other two technologies are ATM (Asynchronous Transfer Mode), and IEEE 802.12, which is also known as 100VG-AnyLAN or 100BT.[1] Each technology has a set of strengths and is best suited to a particular networking niche. Combined, these technologies support all aspects of networking.

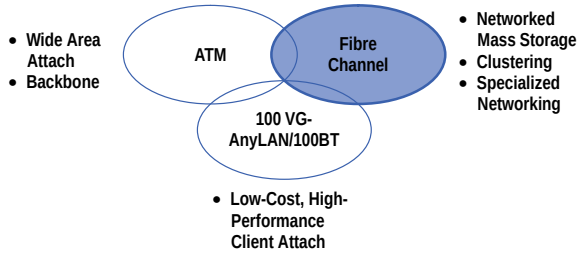


Fig. 1. *HP networking technologies.*

Both Fibre Channel and ATM are switched systems. They can share the same cable plant and encoding scheme and can work together in a network (Fig. 2). However, Fibre Channel and ATM standards are evolving independently to resolve different customer needs and objectives. ATM, which is telecommunications-based, is intended for applications that are characterized by "bursty" types of communications, thus lending itself to WAN applications. 100VG-AnyLAN or 100BT provides low-cost, high-performance client attachments. Fibre Channel is data communications-based and particularly well-adapted for networked and embedded mass storage, clustering, and specialized networking applications requiring sustained data flow rates.

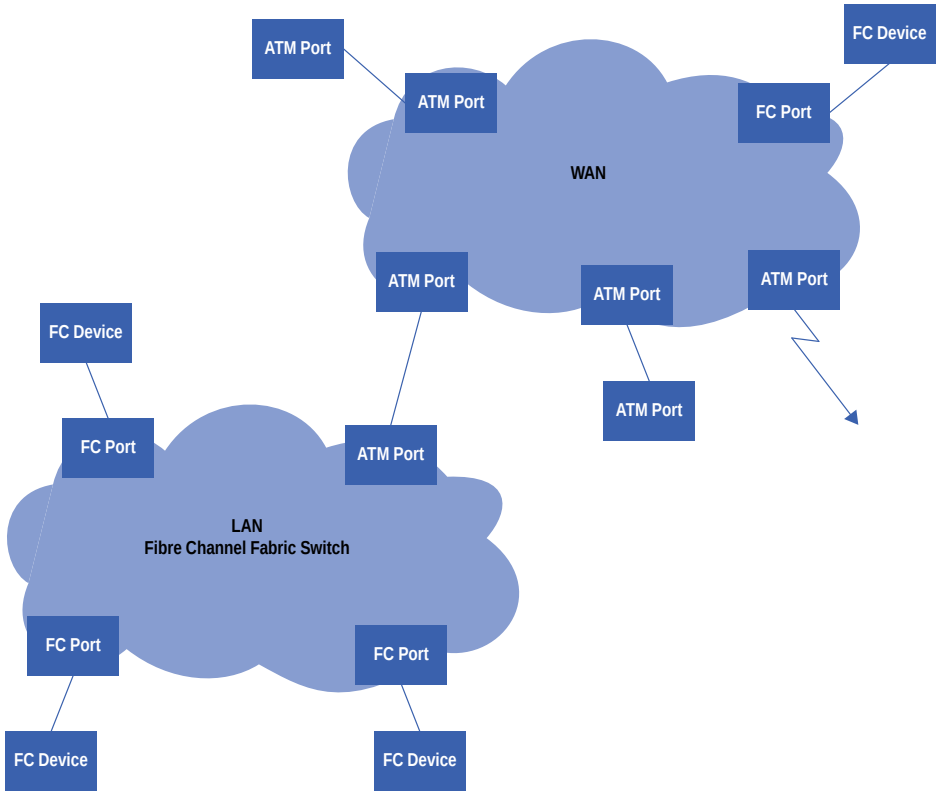


Fig. 2. *A network containing both Fibre Channel and ATM elements.*

In addition, Fibre Channel resolves the "slots and watts" problem that current symmetric multiprocessing systems have. For example, in 1995, three FDDI ports and six fast and wide SCSI ports were required to use fully the I/O capabilities of a symmetric multiprocessing HP server. Fibre Channel could support these I/O services with just three slots.

HP's vision of Fibre Channel is that it is at the core of the virtual data center containing diverse elements including:

- Fibre Channel switches connecting mainframes and supercomputers

- Network-attached disk arrays and storage archives
- ATM, FDDI, and Ethernet routers
- Imaging workstations
- Fibre Channel arbitrated loop disks and disk arrays
- High-performance mass storage peripherals
- Low-cost clients
- Clustered systems
- Video, technical, and commercial servers.

Interoperability and the establishment of a critical mass of Fibre Channel products are the keys to the success of Fibre Channel. HP is committed to Fibre Channel and is working with partners and standards bodies to ensure interoperability. HP is an active participant in the ANSI Fibre Channel Working Group, the Fibre Channel Association (FCA), and the Fibre Channel Systems Initiative, which has been integrated into the FCA. In 1994 HP purchased Canstar, a Fibre Channel switch company, which is now HP's Canadian Networks Operation. HP has developed Fibre Channel disk drives, gigabit link modules and transceivers,[2] system interfaces, and the Tachyon protocol controller chip, which is the subject of this article. HP is using Fibre Channel's versatility and speed for high-availability mass storage solutions and clustered system topologies.

Tachyon Chip

The system interconnect laboratory of the HP Networked Computing Division became interested in Fibre Channel in 1993 as a method of entering the high-speed serial interconnect market because Fibre Channel was the first technology that could be used for both networking and mass storage. HP decided to develop the Tachyon chip in mid-1993 after investigating which Fibre Channel controller chip to use in a Fibre Channel host adapter card under development for the HP 9000 Series 800 K-class server.[3] The investigation determined that no available chipset would meet the functional or performance requirements, so the decision was made to develop a controller internally.

The Tachyon chip (Fig. 3) implements the FC-1 and FC-2 layers of the five-layer Fibre Channel standard (see ***Article 11***). Tachyon's host attach enables low-cost gigabit host adapters on industry-standard buses including PCI, PMC, S-Bus, VME, EISA, Turbo Channel, and MCA. It is easily adaptable both to industry-standard and proprietary buses through the Tachyon system interface (a generic interface) and provides a seamless interface to GLM-compliant modules and components. GLM (gigabaud link module) is a profile defined by the FCSI (Fibre Channel Systems Initiative) and adopted by the FCA (Fibre Channel Association). It is a subset of the Fibre Channel FC-0 layer.[4]

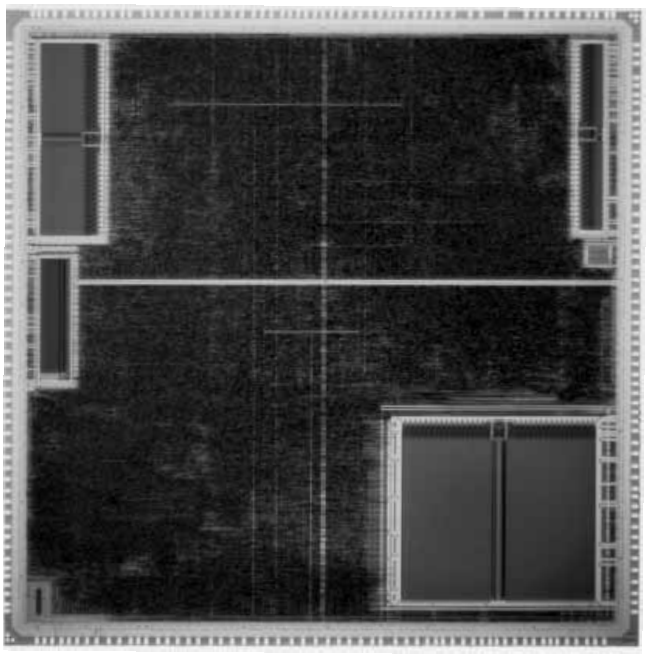


Fig. 3. *HP Tachyon Fibre Channel controller chip.*

Tachyon provides gigabit data throughput at distance options from 10 meters on copper to 10 kilometers over single-mode optical fiber. Tachyon host adapters save system slots, minimizing cost and cabling infrastructure.

Tachyon achieves high performance and efficiency because many of its lower-level functions are implemented in hardware, eliminating the need for a separate microprocessor chip. Functions such as disassembly of outbound user data from

sequences into frames, reassembly of inbound data, flow control, data encoding and decoding, and simple low-level error detection at the transmission character level are all built into hardware. One set of hardware supports all upper-level protocols. Errors and exceptions are offloaded to host-based upper-level software to manage.

Tachyon High-Level Design Goals

The Tachyon designers made several high-level design decisions early in the project. The primary design goal was to deliver sustained, full-speed gigabit performance while imposing the minimum impact on host software overhead. To accomplish this, Tachyon supports all Fibre Channel classes of service (see ***Article 11***), automatically acknowledges inbound frames for class 1 and class 2, handles NL_Port and N_Port initialization entirely in hardware, manages concurrent inbound and outbound sequences, and uses a messaging queue to notify the host of all completion information. To offload networking tasks from hosts, Tachyon is designed to assist networking protocols by supporting IP checksums and two different modes for splitting network headers and data.

The second major design goal was that Tachyon should support SCSI encapsulation over Fibre Channel (known as *FCP*). From the beginning of the project, Tachyon designers created SCSI hardware assists to support SCSI initiator transactions. These hardware assists included special queuing and caching. Early in the design, Tachyon only supported SCSI initiator functionality with its SCSI hardware assists. It became evident from customer feedback, however, that Tachyon must support SCSI target functionality as well, so SCSI target functionality was added to Tachyon SCSI hardware assists.

Tachyon Feature Set

To take advantage of Fibre Channel's high performance, Tachyon:[5]

- Provides a single-chip Fibre Channel solution.
- Manages sequence segmentation and reassembly in hardware.
- Automatically generates acknowledgement (ACK) frames for inbound data frames.
- Automatically intercepts and processes ACK frames of outbound data frames.
- Processes inbound and outbound data simultaneously with a full-duplex architecture.
- Allows chip transaction accesses to be kept at a minimum by means of host-shared data structures.

To provide the most flexibility for customer applications, Tachyon:

- Supports link speeds of 1063, 531, and 266 Mbaud.
- Supports Fibre Channel class 1, 2, and 3 services.
- Supports Fibre Channel arbitrated loop (FC-AL), point-to-point, and fabric (crosspoint switched) topologies.
- Provides a glueless connection to industry-standard physical link modules such as gigabaud link modules.
- Supports up to 2K-byte frame payload size for all Fibre Channel classes of service.
- Supports broadcast transmission and reception of FC-AL frames.
- Allows time-critical messages to bypass the normal traffic waiting for various resources via a low-latency, high-priority outbound path through the chip.
- Provides a generic 32-bit midplane interface—the Tachyon system interface.

To provide support for customer networking applications, Tachyon:

- Manages the protocol for sending and receiving network sequences over Fibre Channel.
- Provides complete support of networking connections.
- Computes exact checksums for outbound IP packets and inserts them in the data stream, thereby offloading the host of a very compute-intensive task.
- Computes an approximate checksum for inbound IP packets that partially offloads the checksum task from the host.
- Contains hardware header/data splitting for inbound SNAP/IP sequences.

To provide support for customer mass storage applications, Tachyon:

- Supports up to 16,384 concurrent SCSI I/O transactions.
- Can be programmed to function as either an initiator or a target.
- Assists the protocol for peripheral I/O transactions via SCSI encapsulation over Fibre Channel (FCP).

To reduce host software support overhead, Tachyon:

- Allows chip transaction accesses to be kept at a minimum by means of host-shared memory data structures.
- Manages interrupts to one or zero per sequence.
- Performs FC-AL initialization with minimal host intervention.

To provide standards compliance, Tachyon:

- Complies with Fibre Channel System Initiative (FCSI) profiles.

- Complies with industry-standard MIB-II network management.

To ensure reliability, Tachyon:

- Supports parity protection on its internal data path.
- Has an estimated MTBF of 1.3 million hours.

Fabrication

Tachyon is fabricated by LSI Logic Corporation using a 0.5-µm 3.3V CMOS process, LCB500K. The chip dissipates just under 4 watts and is contained in a 208-pin MQUAD package with no heat sink.

Tachyon Functional Overview

The host interface of the Tachyon chip is a set of registers used for initialization, configuration, and control and a set of data structures used for sending and receiving data and for event notification. This interface is very flexible and allows the customer to design an interface to Tachyon that best meets the capability, performance, and other requirements of a specific application.

Transmitting a Fibre Channel Sequence

To transmit an outbound sequence (see Fig. 4), the host builds several data structures and sets up the data to be transmitted. A data structure called the *outbound descriptor block* is built first. The outbound descriptor block provides much of the information Tachyon needs to send a sequence. The outbound descriptor block points to a data structure called the *extended descriptor block*, which points to data buffers containing the data for the sequence. The host then creates the Tachyon header structure, which contains important Fibre Channel-specific information such as which Fibre Channel class of service to use during sequence transmission. The host sets up the outbound descriptor block to point to the Tachyon header structure. The host then adds the outbound descriptor block to the outbound command queue.

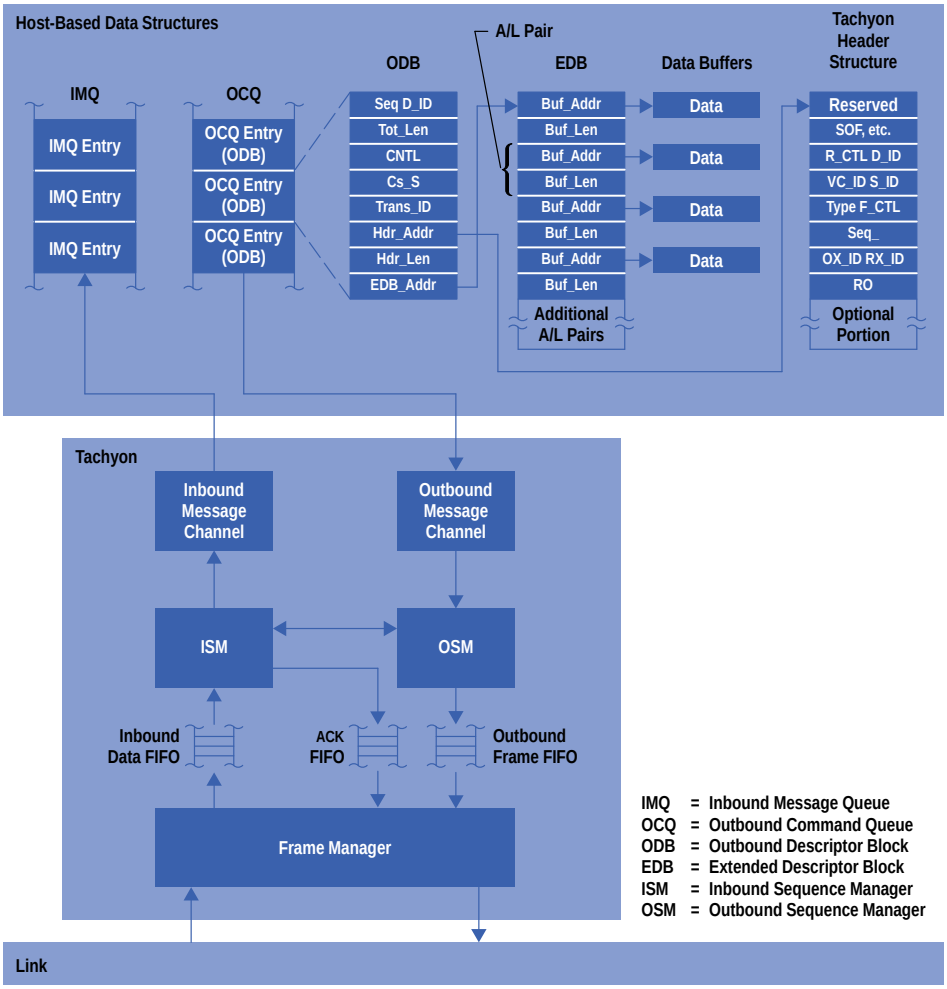


Fig. 4. *Transmit process overview.*

When Tachyon sees the new entry in the outbound command queue, it gets the outbound descriptor block from host memory via DMA. As Tachyon reads the Tachyon header structure to send the first frame of the sequence, it copies the header structure to internal registers for use in generating Fibre Channel headers for subsequent frames.

If this is class 1 service, after sending the first frame, Tachyon waits until it receives the ACK for the first frame of the sequence before continuing. Tachyon then inserts an identifier value, called the *responder exchange identifier* (RX_ID), which is returned in the ACK, into the Fibre Channel header on all subsequent frames of this sequence. Tachyon continues to transfer data from the host via DMA in frame-sized blocks and sends the frames with headers automatically generated from the previously stored header.

Tachyon keeps track of the frame count for the sequence. The Fibre Channel header for each frame contains an incremental count of the number of frames transmitted for the sequence along with the relative position of that frame within the sequence. As Tachyon sends the frames for the sequence, it also tracks flow control for the sequence using a Fibre Channel flow control method called end-to-end credit (EE_Credit). EE_Credit determines the number of frames that Tachyon can send to the remote destination without receiving an ACK. Each time Tachyon sends a frame, EE_Credit decrements. Each time Tachyon receives an ACK from the destination, EE_Credit increments. If EE_Credit goes to zero, Tachyon stops transmitting frames and starts a watchdog timer called the ED_TOV timer (error detect timeout value). The ED_TOV timer counts down until ACKs arrive. If ACKs arrive, Tachyon resumes transmission. If the ED_TOV timer expires, Tachyon sends a completion message to the host indicating that the sequence has timed out.

Once Tachyon has transmitted the last frame of a sequence and received all of the ACKs for the sequence, it sends a completion message to the host via the inbound message queue. This tells the host that it can deallocate all memory associated with this outbound descriptor block and inform any processes waiting on its completion. If a sequence terminates abnormally, Tachyon will notify the host of the error in the completion message.

Receiving a Fibre Channel Sequence

Fig. 5 shows an overview of the receive process. To enable Tachyon to receive data, the host first supplies Tachyon with two queues of inbound buffers. Two inbound queues are required because single-frame sequence reception and multiframe sequence reception are handled independently. Tachyon needs minimal resources to receive a single-frame sequence, but for multiframe sequences the chip needs to use additional resources to manage the reassembly of frames. Because of this resource demand, Tachyon can reassemble only one incoming multiframe networking sequence at a time. Tachyon supports the reception of single-frame sequences while reassembling a multiframe sequence. This process allows a host to receive short sequences while Tachyon is reassembling a longer incoming sequence.

Single-Frame Sequence Reception. The hosts uses the single-frame sequence buffer queue to inform Tachyon of the location of host-based buffers that Tachyon should use to receive sequences contained within a single frame. As Tachyon receives a single-frame sequence, it places the entire sequence, which includes the Tachyon header structure followed by the sequence data, in the buffer defined by the address from the single-frame sequence buffer queue. If the sequence is larger than one buffer size, the remaining data of the sequence is packed into the next buffers, as required, until all of the sequence is stored. Next, Tachyon sends an inbound_sfs_completion message to the host via the inbound message queue and generates an interrupt to the host.

Multiframe Sequence Reception. The host uses the multiframe sequence buffer queue to inform Tachyon of the location of host-based buffers that Tachyon should use to receive and reassemble incoming data that has been split into an arbitrarily large number of frames. When the first frame of a new sequence arrives, Tachyon copies the Tachyon header structure into the beginning of the next available multiframe sequence buffer. Tachyon packs the data payload of the frame into the next buffer following the buffer with the Tachyon header structure. As each new frame arrives, Tachyon discards the Fibre Channel header information and sends the data to the host. Tachyon packs this data into each of the buffers on the multiframe sequence buffer queue, obtaining a new buffer when the current buffer is full, until the entire sequence is stored. Once all the frames arrive and the sequence is reassembled in memory, Tachyon notifies the host that the entire sequence has been received by generating a completion message and placing it into the inbound message queue. Tachyon then generates a host interrupt to process the entire sequence.

Tachyon can also handle multiframe sequences that are received out of order. When Tachyon detects an out-of-order frame, Tachyon generates a completion message that indicates the in-order portion of the sequence and the last sequence buffer used. Tachyon passes the completion message to the inbound message queue, but does not generate an interrupt until all frames of the sequence are received. Next, Tachyon obtains the next available sequence buffer and copies the Tachyon header structure of this out-of-order frame into it. Then, into the next sequence buffer, it copies the data payload of this out-of-order frame. At this point, if the frames that follow the out-of-order frame are in order, Tachyon discards the Tachyon header structures and packs the data into the host buffers. Tachyon packs this data into each of the buffers on the multiframe sequence buffer queue, obtaining a new buffer when the current buffer is full, until the entire sequence is stored. If another frame arrives out of order from the previous out-of-order portion, Tachyon generates a new completion message and the process is repeated. When it receives the final frame of the sequence, Tachyon passes it to the host and generates a completion message. At this time, Tachyon generates a host interrupt to process the entire sequence. With the information in each of the Tachyon header structures that Tachyon passed to the host for each in-order portion and the information in the completion messages, the host has enough information to reorder the out-of-order multiframe sequence.

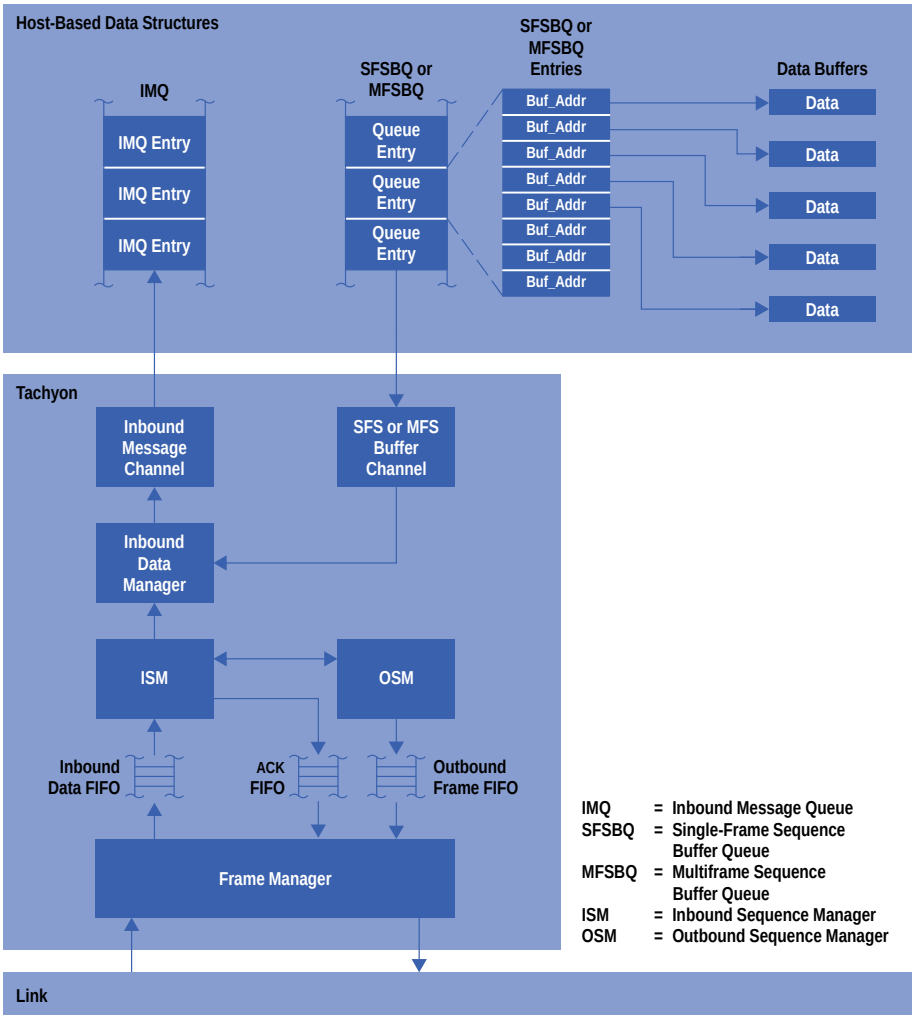


Fig. 5. *Receive process overview.*

Tachyon Internal Architecture

Tachyon's internal architecture is illustrated in Fig. 6. Each functional block in the architecture is described below.

Outbound Message Channel. The outbound message channel block manages the outbound command queue. It maintains the outbound command queue as a standard circular queue. The outbound message channel informs the outbound sequence manager block when an outbound command is waiting in host memory to be processed. When requested by the outbound sequence manager, the outbound message channel then reads one 32-byte entry from the outbound command queue and delivers it to the outbound sequence manager block for processing.

High-Priority Message Channel. The high-priority message channel block manages the high-priority command queue. The host can use the high-priority channel to send urgent single-frame sequences that need to bypass the dedicated outbound command queue. For example, the host could use the high-priority command queue to send special Fibre Channel error recovery frames that might not otherwise be transmitted because of a blocked outbound command queue. The high-priority message channel maintains the high-priority command queue as a standard circular queue. The high-priority message channel informs the outbound sequence manager block when a high-priority outbound command is waiting in host memory to be processed. When requested by the outbound sequence manager, the high-priority message channel reads one entry from the high-priority command queue and delivers it to the outbound sequence manager block for processing.

Outbound Block Mover. The outbound block mover block's function is to transfer outbound data from host memory to the outbound sequence manager via DMA. It takes as input an address/length pair from the outbound sequence manager block, initiates the Tachyon system interface bus ownerships, and performs the most efficient number and size of transactions on the Tachyon system interface bus to pull in the data requested.

Outbound Sequence Manager. The outbound sequence manager block is responsible for managing all outbound sequences. The outbound message channel, the high-priority message channel, and the SCSI exchange manager notify the outbound sequence manager block when they have data to send. The outbound sequence manager must determine which channel has priority. High-priority sequences have first priority, but the outbound sequence manager determines priority between

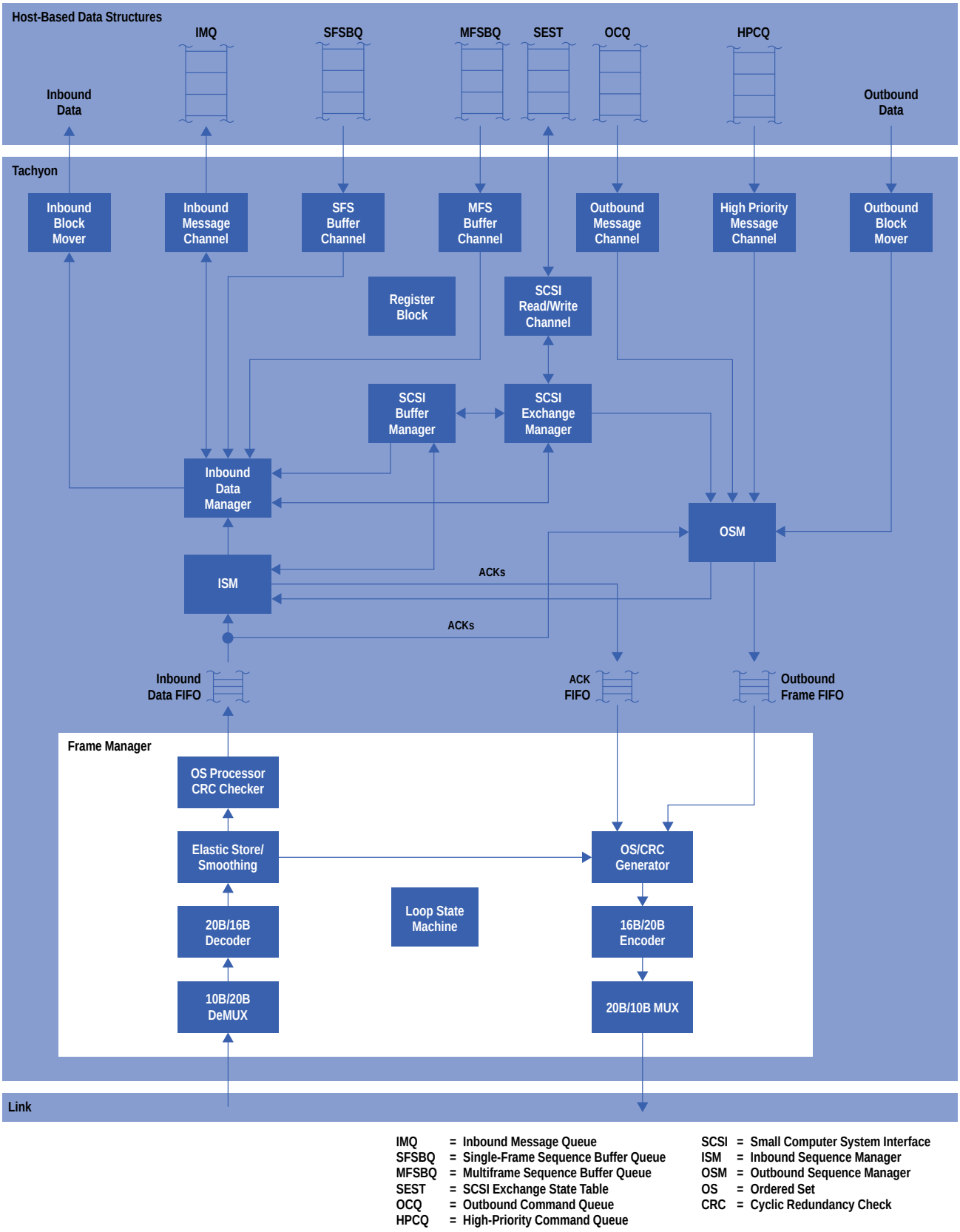


Fig. 6. *Tachyon internal architecture.*

networking and SCSI transactions using a fairness algorithm. Once priority is determined, the outbound sequence manager programs the outbound message channel to retrieve a data sequence from host memory and segment it into individual frames for transmission. The outbound sequence manager transmits the sequence, performs a TCP or UDP-type checksum on the sequence, verifies that each frame is acknowledged by the receiving node, handles errors if required, and sends a completion message to the host through the inbound message channel.

Outbound Frame FIFO. The outbound frame FIFO buffers data before transmission to prevent underrun. This FIFO is sized to hold one maximum-size frame. As Tachyon sends the current frame onto the link, the outbound frame FIFO is simultaneously filled with the next frame, maximizing outbound performance and reducing latency.

ACK FIFO. The ACK FIFO holds Fibre Channel class 1 and class 2 ACKs until they can be sent out by the frame manager.

Frame Manager. The frame manager is Tachyon's interface to the physical link module. The frame manager implements the N_Port state machine described in the FC-PH specification and the loop state machine described in the FC-AL specification. The frame manager can be configured to support link speeds of 1063, 531, and 266 megabits per second. It also implements initialization in hardware for both NL_Ports and N_Ports.

The frame manager implements portions of the FC-1 and FC-2 specifications. It is responsible for the FC-1 functions of transmitting and receiving Fibre Channel frames and primitives. It calculates and verifies the CRC for frame data, checks parity of the transmit data, and generates parity for the receive data. It generates primitives, encodes and receives Fibre Channel frames and primitives, decodes 10-bit or 20-bit physical link module data, and implements the running disparity algorithm in FC-1. The frame manager can be configured to generate interrupts to the host when certain link configuration changes occur to which the host must respond. The interrupt process occurs as part of N_Port initialization and loop initialization and any time the link has been disrupted.

The ordered set and CRC generator encapsulates data into FC-1 frames, generates a 32-bit cyclic redundancy check (CRC), writes it into the frame, and passes the encapsulated data to the 16B/20B encoder. The 16B/20B encoder converts outbound 16-bit-wide data into two 8-bit pieces, each of which is encoded into a 10-bit transmission character using the 8B/10B encoding algorithm.

The 20B/10B multiplexer is responsible for selecting the proper width, either 10 bits or 20 bits, of the data path for the specific type of physical link module being used. The data width and speed are specified by the parallel ID field of the physical link module interface.

The 10B/20B demultiplexer is responsible for receiving incoming encoded data, either 10 bits wide or 20 bits wide, from the physical link module and packing it into 20 bits for decoding. The data width and speed are specified by the parallel ID field of the physical link module interface.

The 20B/16B decoder is responsible for converting 20-bit-wide data received from the 10B/20B demultiplexer into two 8-bit data bytes.

The elastic store and smoothing block is responsible for retiming received data from the receive clock to the transmit clock. The elastic store provides a flexible data FIFO buffer between the receive clock and transmit clock domains. Receiver overrun and underrun can be avoided by deleting and inserting duplicate primitives from the elastic store as needed to compensate for differences in receive clock and transmit clock frequencies (within specifications).

The ordered set processor and CRC checker block is responsible for detecting incoming frame boundaries, verifying the cyclic redundancy check, passing the data to the inbound FIFO, and decoding and recognizing primitives.

Inbound Data FIFO. The inbound data FIFO buffers frame data while the frame manager verifies the CRC. It also serves as high-availability temporary storage to facilitate the Fibre Channel flow control mechanisms. This FIFO is sized to hold a maximum of four 2K-byte frames (including headers).

Inbound Sequence Manager. The inbound sequence manager is responsible for receiving and parsing all link control and link data frames. It interacts with the SCSI buffer manager block to process SCSI frames. The inbound sequence manager block is also responsible for coordinating the actions taken when a link reset is received from the frame manager block and for passing outbound completions to the inbound data manager block. The inbound sequence manager also manages class 1 Fibre Channel connections.

Inbound Data Manager. The inbound data manager routes incoming frames to their appropriate buffers in host memory, transferring SCSI FCP_XFER_RDY frames to the SCSI exchange manager, sending completion messages to the inbound message queue, and sending interrupts to the interrupt controller inside the inbound message channel.

Inbound Block Mover. The inbound block mover is responsible for DMA transfers of inbound data into buffers specified by the multiframe sequence buffer queue, the single-frame sequence buffer queue, the inbound message queue, or the SCSI buffer manager. The inbound block mover accepts an address from the inbound data manager, then accepts the subsequent data stream and places the data into the location specified by the address. The inbound block mover also accepts producer index updates and interrupt requests from the inbound message channel and works with the arbiter to put the interrupt onto the Tachyon system interface bus.

Inbound Message Channel. The inbound message channel maintains the inbound message queue. This includes supplying the inbound data manager with the address of the next available entry in the inbound message queue and generating a completion message when the number of available entries in the inbound message queue is down to two.

The inbound message channel also generates interrupts for completion messages, if necessary, and handles message and interrupt ordering. The completion message must be in the host memory before the interrupt. The inbound message channel also handles interrupt avoidance, which minimizes the number of interrupts (strobes on the INT pin on the Tachyon system interface bus) asserted to the host for each group of completions sent to the host.

Single-Frame Sequence Buffer Channel. The inbound single-frame sequence buffer channel manages the single-frame sequence buffer queue. It supplies addresses of empty single-frame sequence buffers to the inbound data manager and generates a completion message when the supply of single-frame sequence buffers runs low.

Multiframe Sequence Buffer Channel. The inbound multiframe sequence buffer channel manages the multiframe sequence buffer queue. It supplies addresses of empty multiframe sequence buffers to the inbound data manager and generates a completion message when the supply of multiframe sequence buffers runs low.

SCSI Buffer Manager. The SCSI buffer manager is responsible for supplying the inbound data manager with addresses of buffers to be used for inbound SCSI data frames. The SCSI buffer manager maintains a cache of 16 unique SCSI descriptor blocks. Each block contains eight SCSI buffer addresses. Depending upon the direction of the exchange, the originator exchange identifier (OX_ID) or the responder exchange identifier (RX_ID) of the current frame is provided by the inbound data manager block and is used to point to the correct entry in the SCSI exchange state table.

SCSI Exchange Manager. In conjunction with the SCSI exchange state table data structure, the SCSI exchange manager provides Tachyon with the hardware assists for SCSI I/O transactions. It converts SCSI exchange state table entries to outbound descriptor block format for the outbound sequence manager block's use. The SCSI exchange manager accepts SCSI outbound FCP_XFER_RDY frames from the inbound data manager block. It then uses the OX_ID or RX_ID given in the frame as an offset into the state table, reads the entry, and builds the outbound descriptor block.

Register Block. The register block consists of control, configuration, and status registers. These registers are used for initialization, configuration, control, error reporting, and maintenance of the queues used to transfer data between Tachyon and the host. Each register is 32 bits wide and may be readable or both readable and writable by the host depending upon its function.

SCSI Read/Write Channel. The SCSI read/write channel manages requests from the SCSI exchange manager and interfaces to the Tachyon system interface arbiter block.

Tachyon SCSI Hardware Assists

Tachyon supports SCSI I/O transactions (exchanges) by two methods. The first method uses host-based transaction management. In this method, the host transmits and receives the various sequences using the general transmit and receive processes. By using the host-based transaction management method, Tachyon reassembles only one SCSI unassisted multiframe sequence at a time. The second method uses Tachyon's SCSI hardware assists. With this method, Tachyon assists the host transaction management through the use of a shared host data structure called the SCSI exchange state table.

By using Tachyon's SCSI hardware assists, the host can concurrently reassemble up to 16,384 SCSI assisted sequences. Tachyon maintains an on-chip cache for up to 16 concurrent inbound transactions. Tachyon uses a *least recently used* caching algorithm to allow the most active exchanges to complete their transfers with the minimum latency.

The protocol for SCSI encapsulation by Fibre Channel is known as FCP. Fig. 7 shows an overview of the FCP read exchange and Fig. 8 shows an overview of the FCP write exchange. The exchanges proceed in three phases: command, data, and status.

SCSI Exchange State Table. The SCSI exchange state table is a memory-based data structure. Access is shared between Tachyon and the host. The SCSI exchange state table is an array of 32-byte entries. The starting address of the table is programmable and is defined by the SCSI exchange state table base register. The length of the table is also programmable and is defined by the SCSI exchange state table length register. Each used table entry corresponds to a current SCSI exchange, or I/O operation. Each entry contains two kinds of information: information supplied by the host driver for Tachyon to manage the exchange, and information stored by Tachyon to track the current state of the exchange. For initiators in SCSI write transactions, the outbound SCSI exchange state table entries contain information indicating where outbound data resides in memory and what parameters to use in the transmission of that data on the Fibre Channel link. For initiators in SCSI read transactions, the inbound SCSI exchange state table entries contain information indicating where inbound data is to be placed in memory. SCSI exchange state table entries are indexed by an exchange identifier (X_ID) —either the originator exchange identifier (OX_ID) or the responder exchange identifier (RX_ID). In an initiator application, the OX_ID provides the index into the SCSI exchange state table. In a target application, the RX_ID provides the index into the SCSI exchange state table.

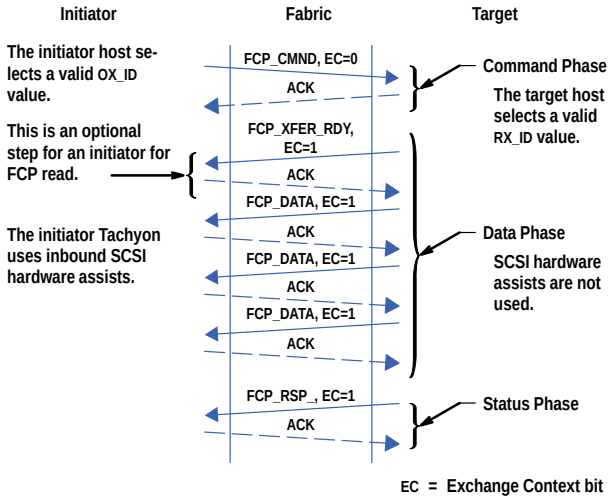


Fig. 7. *FCP read exchange overview.*

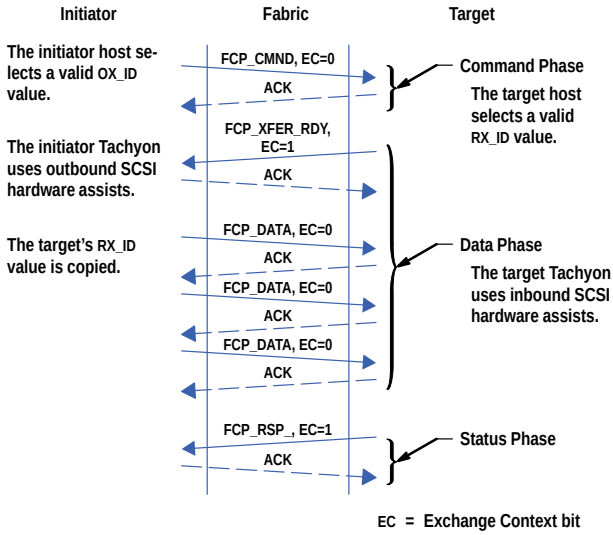
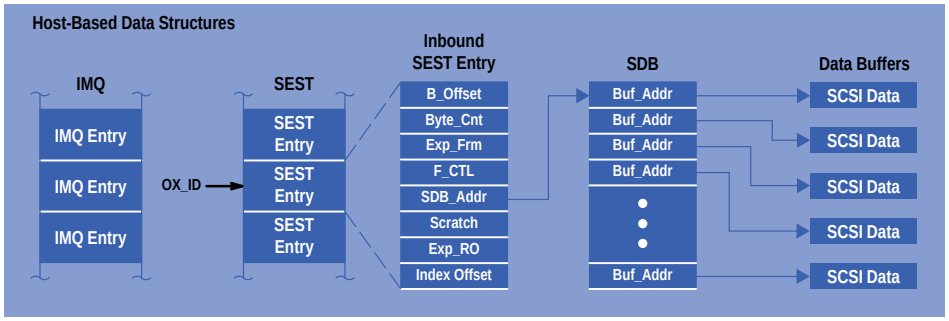


Fig. 8. *FCP write exchange overview.*

FCP Read Exchange—Tachyon as an Initiator. Fig. 9 shows the FCP read exchange host data structures for an initiator Tachyon. For the initiator host to receive inbound SCSI data, it selects a valid OX_ID value that points to an unused location in the SCSI exchange state table. The OX_ID value identifies this particular exchange. Using the OX_ID value, the initiator host builds a data structure called an inbound SCSI exchange state table entry. The inbound SCSI exchange state table entry includes the address of the SCSI descriptor block. The SCSI descriptor block defines the host buffers that Tachyon will use to store the received read data.



IMQ = Inbound Message Queue
SEST = SCSI Exchange State Table
SDB = SCSI Descriptor Block

Fig. 9. *FCP read exchange initiator host data structures.*

In the command phase, once the host creates the inbound SCSI exchange state table entry, it creates an FCP_CMND for an FCP read exchange. The initiator Tachyon sends the FCP_CMND to the target via the outbound command queue.

In the data phase, the initiator Tachyon may receive an FCP_XFER_RDY from the target. This is an optional step for an initiator in an FCP read because the data frames contain all the information needed to process them. When Tachyon receives the optional FCP_XFER_RDY from the target, it acknowledges the frame if appropriate and discards the FCP_XFER_RDY. As each data frame is received, the SCSI exchange manager uses the OX_ID to access the appropriate inbound SCSI exchange state table entry for the address of the SCSI data buffer. The SCSI descriptor block and the relative offset of the data frame determine where data is to be placed in host memory. Tachyon maintains an internal cache of 16 inbound SCSI exchange state table entries. If the SCSI exchange state table information associated with the received frame is not in cache, Tachyon writes the least recently used cache entry back to the host SCSI exchange state table. Tachyon then fetches into cache the inbound SCSI exchange state table entry associated with the received frame and transfers the read data to host memory via DMA. The initiator Tachyon automatically handles both single and multiple data phases for inbound data transfers.

In the status phase, when the data phase is complete, the initiator Tachyon receives an FCP_RSP from the target. The FCP_RSP is a Fibre Channel information unit that contains status information that indicates that the SCSI exchange has completed. The initiator Tachyon passes the FCP_RSP to the host via the single-frame sequence channel and sends a completion message to the initiator host. This informs the initiator host that the exchange is completed.

FCP Read Exchange—Tachyon as a Target. For FCP read exchanges for the target Tachyon, SCSI hardware assists are not used. Fig. 10 shows the read exchange target host data structures.

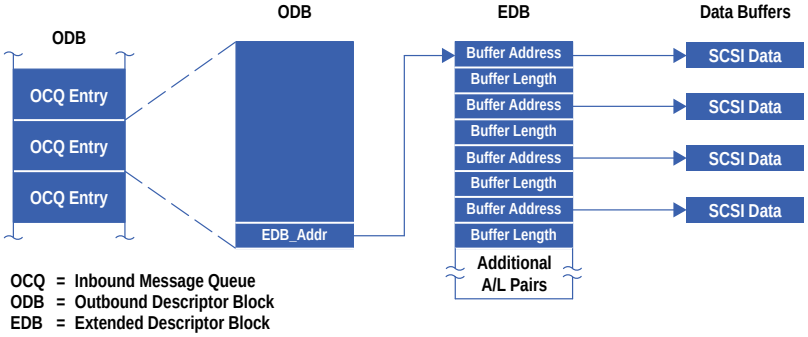


Fig. 10. *FCP read exchange target host data structures.*

In the command phase, the target Tachyon receives an FCP_CMND for an FCP read from the initiator and sends the FCP_CMND to the target host via the single-frame sequence channel. If configured to automatically acknowledge, the target Tachyon immediately returns an ACK (for class 1 and class 2) to the initiator. The target host selects a valid, unused RX_ID value. The RX_ID is placed into the header of the ACK and sent via the high-priority command queue.

In the data phase, the target host builds an outbound descriptor block that contains the extended descriptor block address. The target host builds an extended descriptor block that defines where the read data is located in the target host memory. The target host may send an FCP_XFER_RDY to the initiator host to indicate that it is ready to send the requested data. The target Tachyon sends the FCP_XFER_RDY(s) with the appropriate RX_ID value to the initiator Tachyon. Using the outbound command queue, the target Tachyon then sends the appropriate SCSI read data to the initiator via the outbound command queue.

In the status phase, the target Tachyon sends an FCP_RSP to the initiator to indicate that the exchange has completed.

FCP Write Exchange—Tachyon as an Initiator. Fig. 11 shows the FCP write exchange initiator host data structures. For the initiator host to perform an outbound data transfer, it selects an unused SCSI exchange state table entry whose index will be used as the OX_ID value. Using the OX_ID value, the initiator host builds an outbound SCSI exchange state table entry. The outbound SCSI exchange state table entry includes information about the frame size, the Fibre Channel class, and writable fields that the initiator Tachyon uses to manage the SCSI transfer. The outbound SCSI exchange state table entry also contains a pointer to the extended descriptor block that contains pointers to the data to be sent.

In the command phase, once the host creates the outbound SCSI exchange state table entry, the host creates the FCP_CMND for an FCP write exchange. The initiator Tachyon sends the FCP_CMND to the target via the outbound command queue.

In the data phase, when the initiator Tachyon receives the FCP_XFER_RDY from the target, it uses its SCSI hardware assists and manages the data phase for this FCP_XFER_RDY independent of the host. Tachyon uses the information in the SCSI exchange state table and the FCP_XFER_RDY to build an outbound descriptor block to be sent through the outbound state machine. If the outbound sequence manager is busy, the SCSI exchange state machine adds the request to a linked list of outbound transactions waiting for transmission. As the outbound sequence manager becomes available to process a SCSI transfer, the SCSI exchange state manager dequeues a waiting transaction and passes it to the outbound sequence manager. The outbound sequence manager transmits the write data to the target Tachyon.

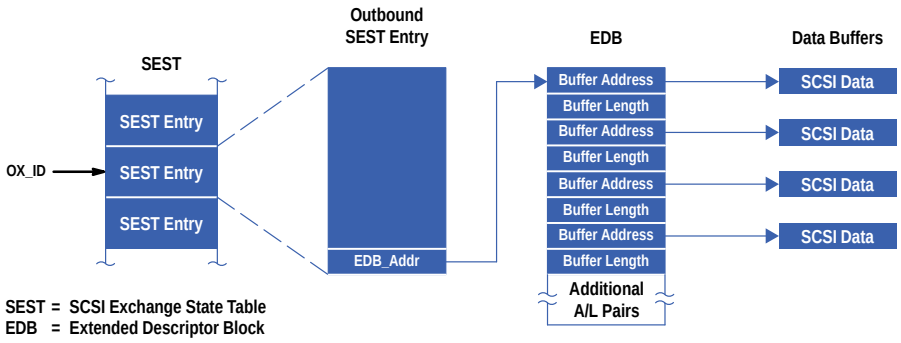


Fig. 11. *FCP write exchange initiator host data structures.*

If this FCP_XFER_RDY is part of a multiple-data-phase transfer, the initiator Tachyon passes this FCP_XFER_RDY as a single-frame sequence to the initiator host along with a completion message. The initiator host is responsible for managing the data phase for this multiple-data-phase transfer by using the general sequence moving services. The OX_ID field in the FCP_XFER_RDY is an index into the SCSI exchange state table and identifies the appropriate outbound SCSI exchange state table entry that points to the extended descriptor block in which the write data is located. Using the information in the outbound SCSI exchange state table entry, the initiator host builds an outbound descriptor block. The initiator Tachyon uses the outbound descriptor block to transmit the write data to the target.

In the status phase, after the data phase completes, the initiator Tachyon receives an FCP_RSP from the target. The initiator Tachyon passes the FCP_RSP to the host and sends a completion message to the initiator host. This informs the initiator host that the exchange has completed.

FCP Write Exchange—Tachyon as a Target. Fig. 12 shows the FCP write exchange target host data structures.

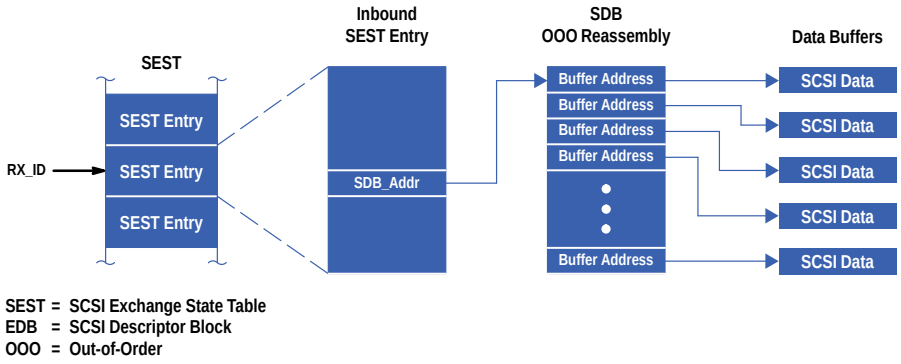


Fig. 12. *FCP write exchange target host data structures.*

In the command phase, the target Tachyon receives an FCP_CMND for an FCP write from the initiator. If configured to do so, the target Tachyon immediately returns an ACK to the initiator for class 1 and class 2. If Tachyon is not configured to return an ACK, the target host is responsible for sending the ACK. The target host selects a valid RX_ID value, places the RX_ID into the ACK header and sends the ACK via the high-priority command queue.

In the data phase, the target host selects an unused SCSI exchange state table entry whose index will be the RX_ID value. Using the RX_ID value, the target host builds the inbound SCSI exchange state table entry that points to the SCSI descriptor block. The SCSI descriptor block contains as many buffer addresses as necessary to receive the data. If the target host has enough buffers and is ready to receive all of the data from the initiator, the host programs the inbound SCSI exchange state table entry and the target Tachyon sends the FCP_XFER_RDY to the initiator via the outbound command queue. The initiator will send the data when it is ready.

The target Tachyon checks the RX_ID of the data frame to locate the appropriate inbound SCSI exchange state table entry. The inbound SCSI exchange state table entry helps Tachyon determine exactly where within the buffers the data is to be placed. When the last data frame is received, the target Tachyon sends a completion message to the target host to inform the target host that all frames for the SCSI sequence have been received.

In the status phase, the target host sends an FCP_RSP to the initiator via the outbound command queue.

Tachyon System Interface

The Tachyon system interface describes the backplane protocol for the Tachyon chip. It is a flexible backplane interface that allows customers to interface to Tachyon using many existing backplane protocols, including PCI, MCA, S-bus, EISA, VME, and Hewlett-Packard's High-Speed Connect (HSC) bus. The Tachyon system interface is capable of 100-Mbyte/s data transfers. Fig. 13 shows the Tachyon system interface signals.

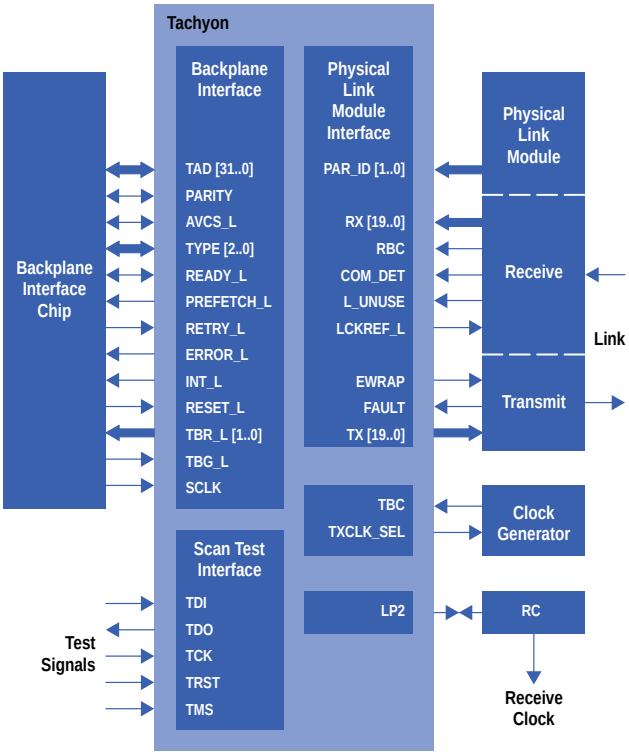


Fig. 13. *Tachyon system interface.*

The Tachyon system interface provides a basic transaction protocol that uses two major operations: write transactions and read transactions. Every transaction has a master and a responder. If the host is the master of a transaction, Tachyon is the responder in that transaction, and vice versa.

The master of a transaction drives an address and transaction type onto the TAD[] and TYPE[] buses, respectively, while asserting AVCS_L to indicate the start of the transaction. If Tachyon masters a transaction, the host, as the responder, uses READY_L as its acknowledgment signal. Similarly, if the host masters the transaction, Tachyon, as the responder, uses READY_L as its acknowledgment signal. A Tachyon system interface bus master has the choice of using one-word, two-word, four-word, or eight-word read and write transactions on the Tachyon system interface bus.

Tachyon System Interface Streaming

To maximize performance, the Tachyon system interface allows the host to configure the length of Tachyon's bus tenancy. When Tachyon obtains mastership of the Tachyon system interface and has more than one transaction to perform, Tachyon may extend its bus tenancy and perform several Tachyon system interface transactions (up to the maximum programmed limit) before releasing mastership. Table I shows how streaming increases performance over nonstreaming.

Table I
Tachyon Performance Savings with Streaming

	Tachyon Stream Size			
	1 Transaction (no streaming)	4 Transactions	16 Transactions	64 Transactions
Savings on Writes	0%	14.2%	18.5%	19.6%
Savings on Reads	0%	5.7%	7.2%	7.5%

Tachyon Host Adapter Requirements

A generic Fibre Channel host bus adapter board using the Tachyon chip contains the following:

- Backplane Connector. Connects the backplane interface chip to the system bus.
- Backplane Interface Chip. Enables the connection of the Tachyon system interface bus to PCI, EISA, HP-HSC or other bus.
- Tachyon Chip. HP's Fibre Channel interface controller.
- Physical Link Module. Tachyon interfaces to many GLM-compliant physical link modules currently in the marketplace. Types of modules include:
  - 1063-Mbit/s DB9 copper connectors for distances up to 30 meters.
  - 1063-Mbit/s shortwave laser physical link modules for distances up to 500 meters.
  - 1063-Mbit/s longwave laser physical link modules for distances up to 10 kilometers.
  - 266-Mbit/s shortwave laser physical link modules for distances up to 2 kilometers.

A block diagram of a typical host bus adapter is shown in Fig. 14. An HP-HSC Fibre Channel adapter is shown in Fig. 15. This adapter was developed by HP to provide Fibre Channel connection to the HP 9000 K-Class servers[3] for fast networking and mass storage.

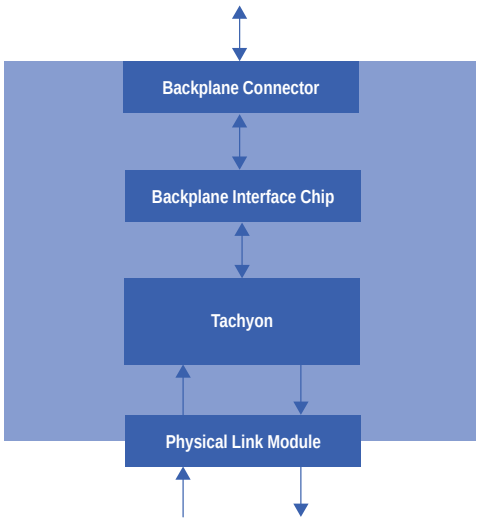


Fig. 14. *Block diagram of a typical host adapter board.*

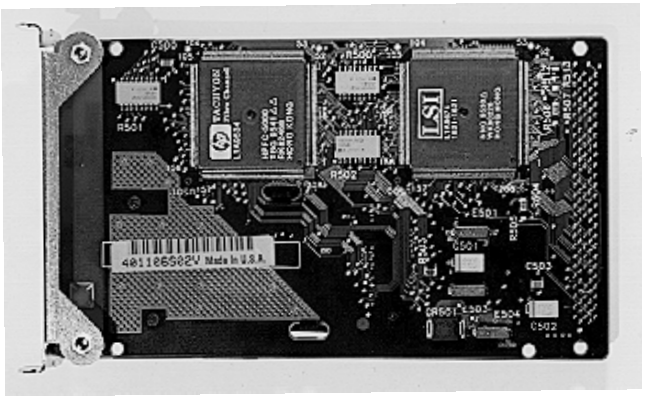
Development Tools

Effective tools were key to the success of the Tachyon chip.[6] The following tools were used in the development project:

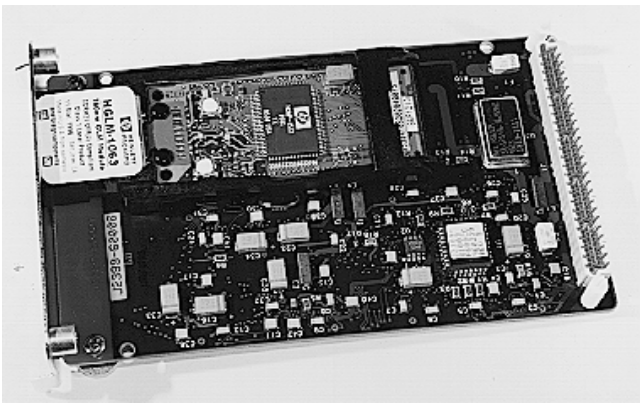
- Cadence Verilog-XL Turbo was used for interactive and batch Register Transfer Language (RTL) simulations, gate functional simulations, and dynamic simulation.
- Chronologic Verilog Compiled Simulator (VCS) was used for batch RTL simulations.
- Quad Motive was used for static timing analysis.
- Synopsys was used for chip synthesis.
- Veritools Undertow was used as a waveform viewer.
- Inter-HDL Verilint was used to check syntax and semantics for RTL code.
- SIL Synth was used to manage and launch synthesis jobs.
- History Management System (HMS),[5] written by Scott A. Kramer, was used as a revision control system.
- LSI Logic Corporation's Concurrent Modular Design Environment (CMDE) was used for floorplanning bonding, delay calculation, and layout.
- Hewlett Packard Task Broker[6] was used to distribute jobs to the various compute engines.

Verification Environment

The verification environment used for Tachyon was very sophisticated and automated. The Tachyon chip model (as either a Verilog RTL code or a gate-level netlist) was tested in simulation using Verilog's programming language interface capability. Test modules written in C or in Verilog provided stimulation to the Tachyon chip. Other test modules then verified the functionality of the chip and reported errors.



(a)



(b)

Fig. 15. *(a) HP HSC Fibre Channel adapter, side A. (b) HP HSC Fibre Channel adapter, side B.*

Conclusion

The Tachyon Fibre Channel interface controller provides a high-performance, single-chip solution for mass storage, clustering, and networking applications. Tachyon has been selected by many other companies as the cornerstone of their Fibre Channel product designs. As an understanding of the capabilities of Fibre Channel technology grows in the marketplace, Tachyon is expected to be present in a large number of new Fibre Channel products.

Acknowledgements

We wish to acknowledge the contributions of Randi Swisley, section manager for Tachyon and the HP HSC FC adapter projects, Bob Whitson, Bryan Cowger, and Mike Peterson, technical marketing, Tachyon chip architects Bill Martin, Eric Tausheck, and Mike Thompson, Fibre Channel standards committee members Kurt Chan and Steve Dean, project managers Margie Evashenk and Tom Parker, VLSI design team lead Joe Steinmetz, hardware and VLSI design team members Catherine Carbonaro, Dave Clark, Mun Johl, Ted Lopez, Bill Martin, George McDavid, Joseph Nuno, Pery Pearson, and Christi Wilde, Tachyon simulation team members Narayan Ayalasomayajula, Tony de la Serna, Murthy Kompella, Brandon Mathew, Mark Shillingberg, John Schimandle, Gordon Matheson, and Matt Wakeley, Tachyon and HSC FC adapter bringup team members Navjyot Birak, Bob Groza, and Donna Jollay, and customer support specialist Rick Barber, who is happy to answer questions from U.S. customers at 1-800-TACHYON. Special thanks to learning products engineer Mary Jo Domurat, who wrote the Tachyon user's manual, and disk support engineer Leland Wong, who provided photographs for this article.

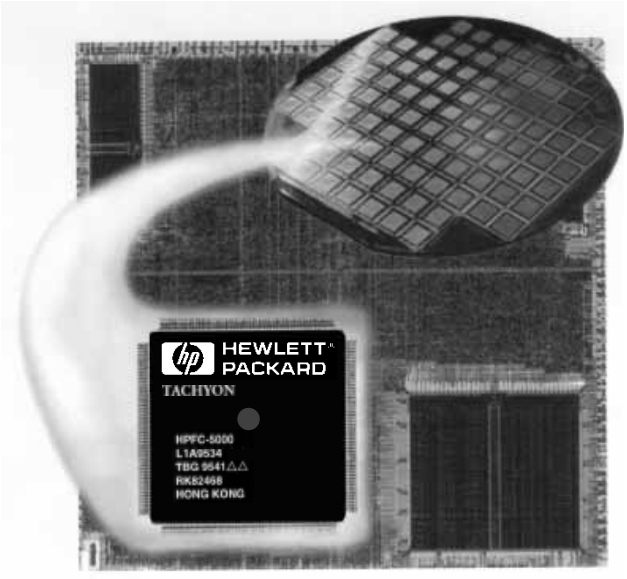
References

1. A.R. Albrecht and P.A. Thaler, "Introduction to 100VG-AnyLAN and the IEEE 802.12 Local Area Network Standard," *Hewlett-Packard Journal*, Vol. 46, no. 4, August 1995, pp. 6-12.
2. J.S. Chang, et al, "A 1.0625-Gbit/s Fibre Channel Chipset with Laser Driver," *Hewlett-Packard Journal*, Vol. 47, no. 1, February 1996, pp. 60-67.
3. M.J. Harline, et al, "Symmetric Multiprocessing Workstations and Servers System-Designed for High Performance and Low Cost," *Hewlett-Packard Journa*l, Vol. 47, no. 1, February 1996, pp. 8-17.
4. *Gigabaud Link Module*, FCSI-301, Rev. 1.0, Fiber Channel Association.

5. S.A. Kramer, "History Management System," *Proceedings of the Third International Workshop on Software Configuration Management (SCM3)*, June 1991, p. 140.
6. T.P. Graf, et al, "HP Task Broker: A Tool for Distributing Computational Tasks," *Hewlett-Packard Journal*, Vol. 44, no. 4, August 1993, pp. 15-22.

Bibliography

1. *Tachyon User's Manual*, Draft 4, Hewlett-Packard Company.
2. *Fibre Channel Arbitrated Loop Direct Disk Attach Profile (Private Loop)*, Version 2.0, ad hoc vendor group proposal.



When we started to write this article, the possibility existed that the HP Journal would be able to feature the Tachyon chip on the cover. We came up with a cover concept, and many people worked very hard on the cover design. Unfortunately, it turned out that Tachyon could not be featured on the cover. We would like to thank the many people who contributed to this graphic: Margie Evashenk for her concept drawing, LSI Logic Corporation who contributed the photomicrograph, Leland Wong who supplied photographs of Tachyon, and graphic artist Marianne deBlanc who put all the pieces together so beautifully.