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.

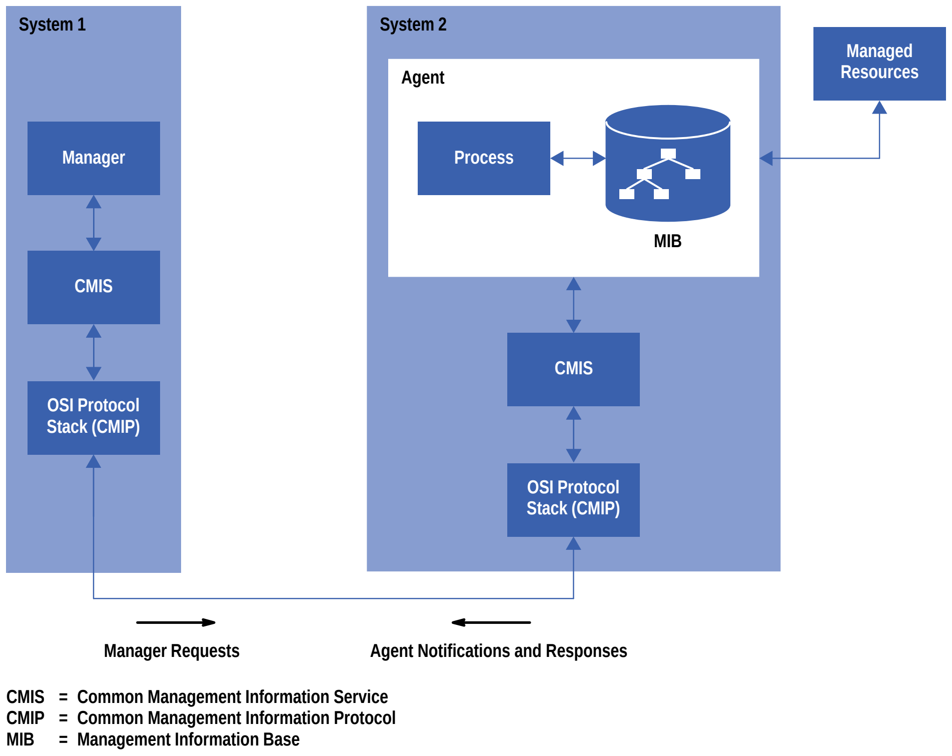
System 1
Manager
CMIS
OSI Protocol Stack (CMIP)
System 2
Agent
Process
MIB
CMIS
OSI Protocol Stack (CMIP)
Managed Resources
Manager Requests
Agent Notifications and Responses
CMIS = Common Management Information Service
CMIP = Common Management Information Protocol
MIB = Management Information Base

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.

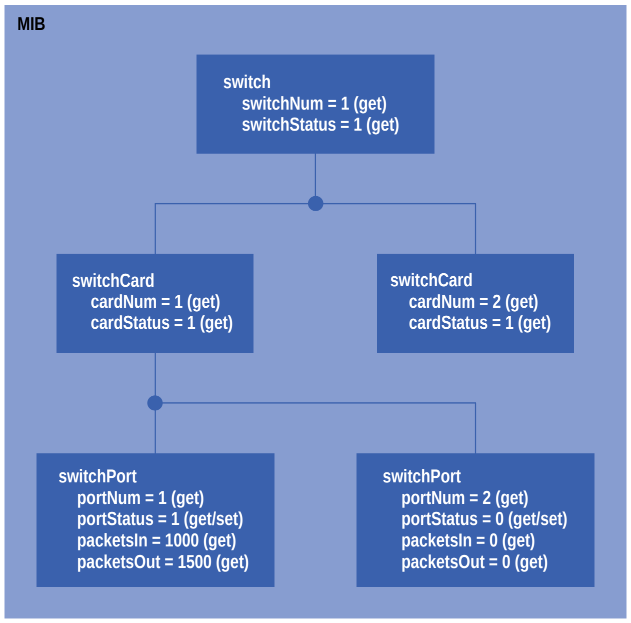

Fig. 2. *The contents of a subset of an agent's Management Information Base (MIB) showing a containment tree made up of object instances and their attributes.*

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).

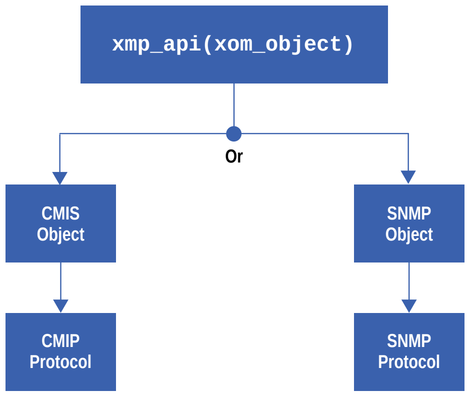

Fig. 3. *The relationship between XOM objects and XMP management functions. XMP functions use XOM objects as parameters to send the details of a manager's request via either a CMIP or an SNMP protocol stack.*

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):

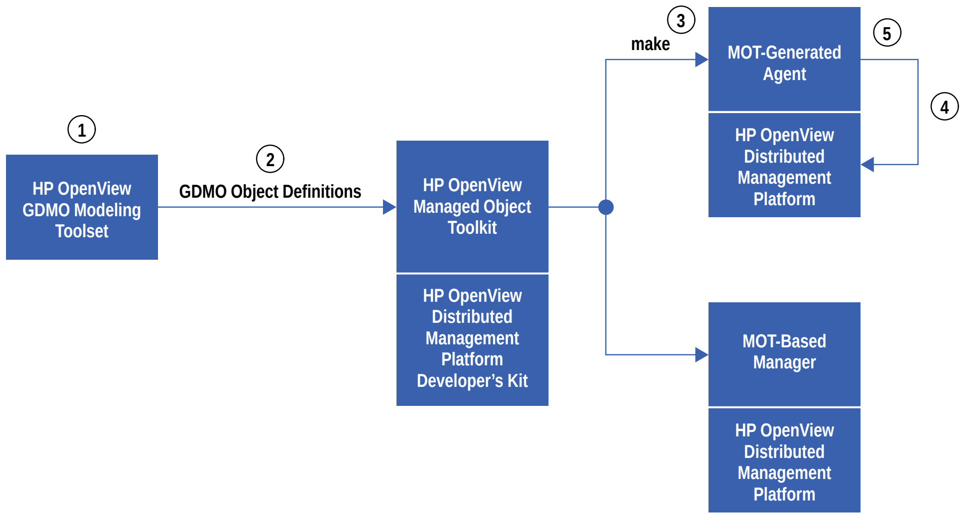
1
HP OpenView GDMO Modeling Toolset
2
GDMO Object Definitions
HP OpenView Managed Object Toolkit
HP OpenView Distributed Management Platform Developer's Kit
3
make
MOT-Generated Agent
HP OpenView Distributed Management Platform
5
4
MOT-Based Manager
HP OpenView Distributed Management Platform

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.

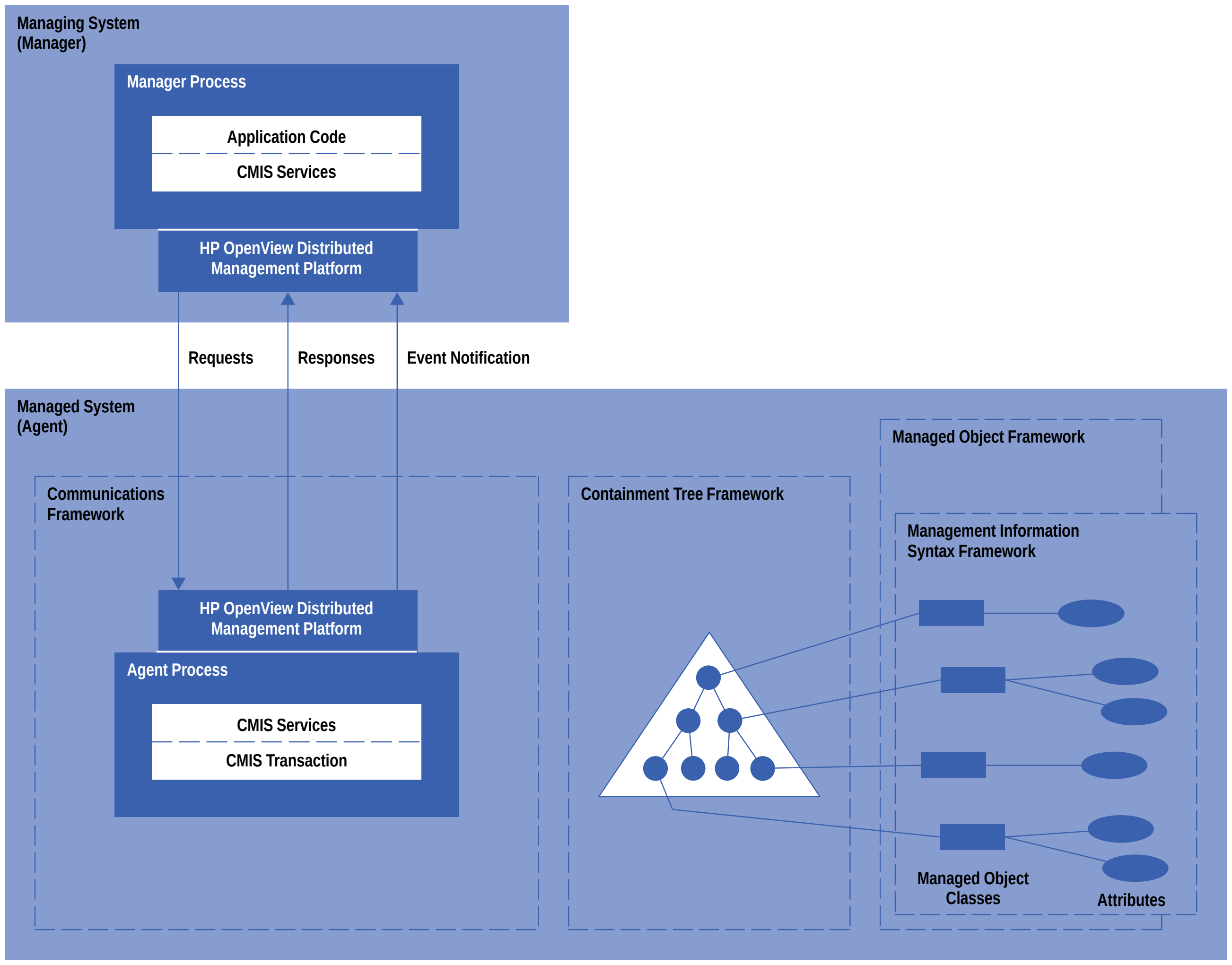

Fig. 6. *Managed Object Toolkit frameworks.*

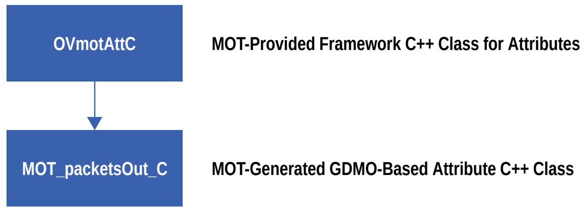
OVmotAttC
MOT-Provided Framework C++ Class for Attributes
MOT_packetsOut_C
MOT-Generated GDMO-Based Attribute C++ Class

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.

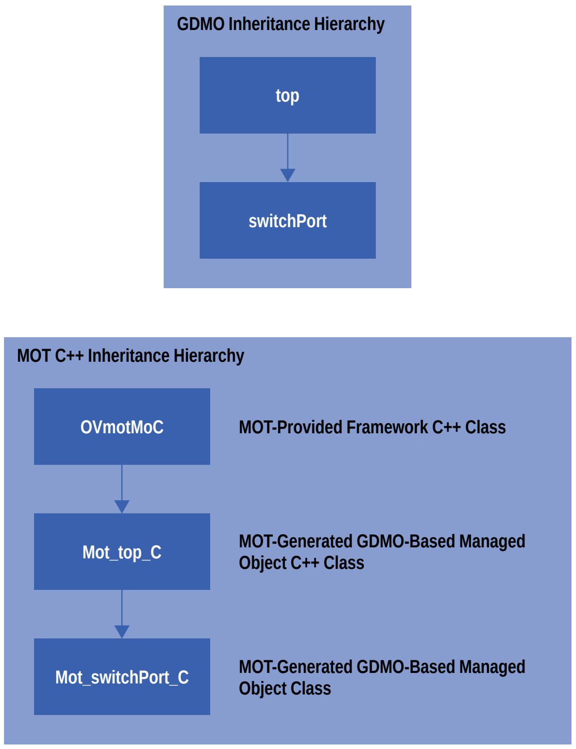
GDMO Inheritance Hierarchy
top
switchPort
MOT C++ Inheritance Hierarchy
OVmotMoC
MOT-Provided Framework C++ Class
Mot_top_C
MOT-Generated GDMO-Based Managed Object C++ Class
Mot_switchPort_C
MOT-Generated GDMO-Based Managed Object Class

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.

▶ Go to Article 7

▶ Go to Table of Contents

▶ Go to HP Journal Home Page