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.

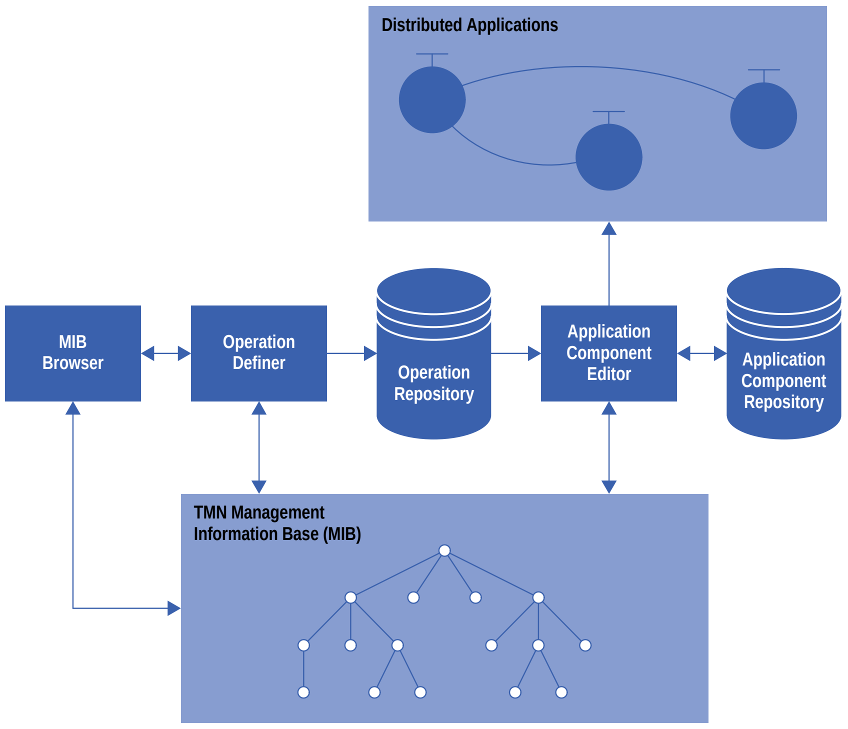

Distributed Applications
MIB Browser
Operation Definer
Operation Repository
Application Component Editor
Application Component Repository
TMN Management Information Base (MIB)

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:

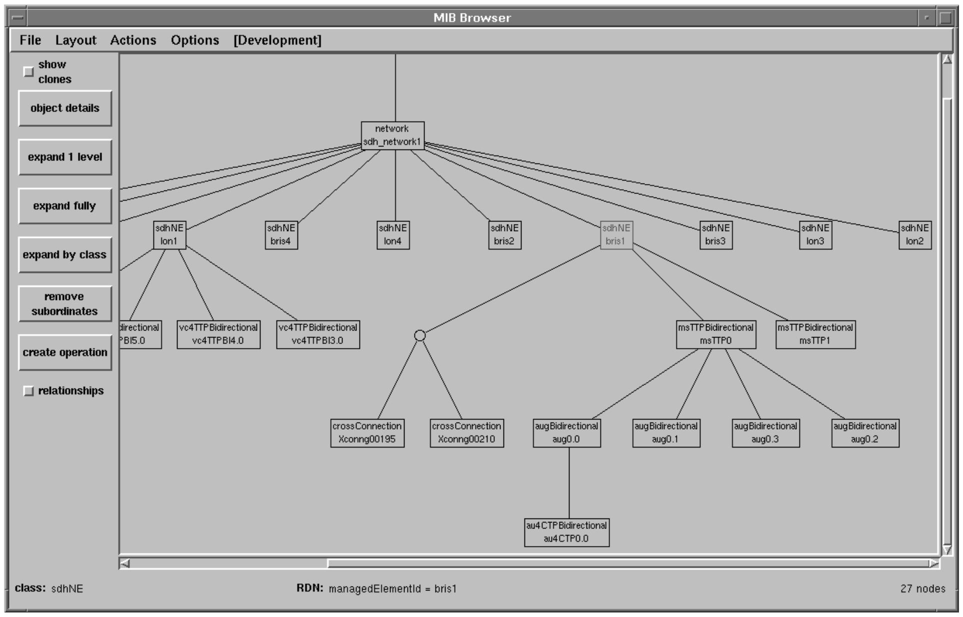
- Navigation through and exploration of the MIB using the *MIB browser* to build an understanding of the data and the way it is used
- Prototyping CMIS operations using the *operation definer*, which helps to expand or verify the developer's understanding (The results of executing these operations may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



class: sdhNE   **RDN:** managedElementId = bris1   27 nodes

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®.

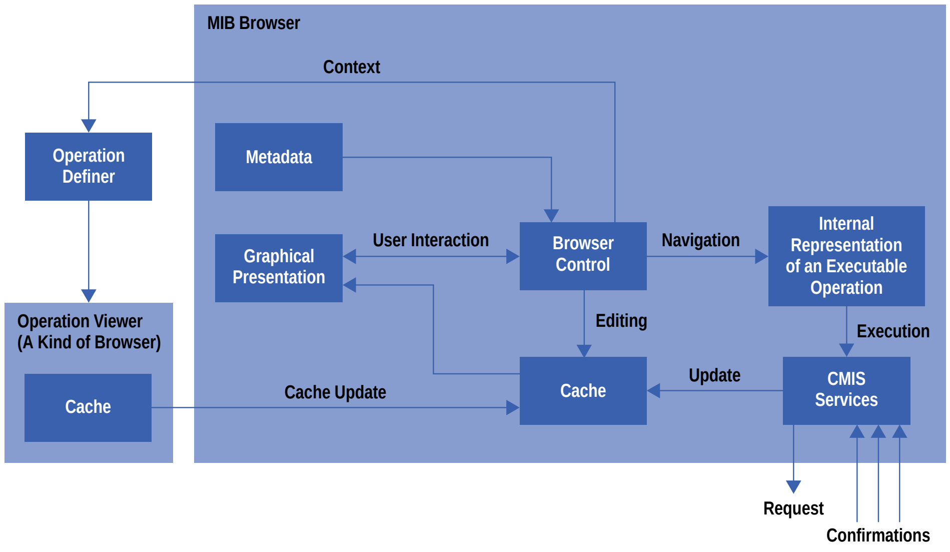


Fig. 3. *The design of the MIB browser.*

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.

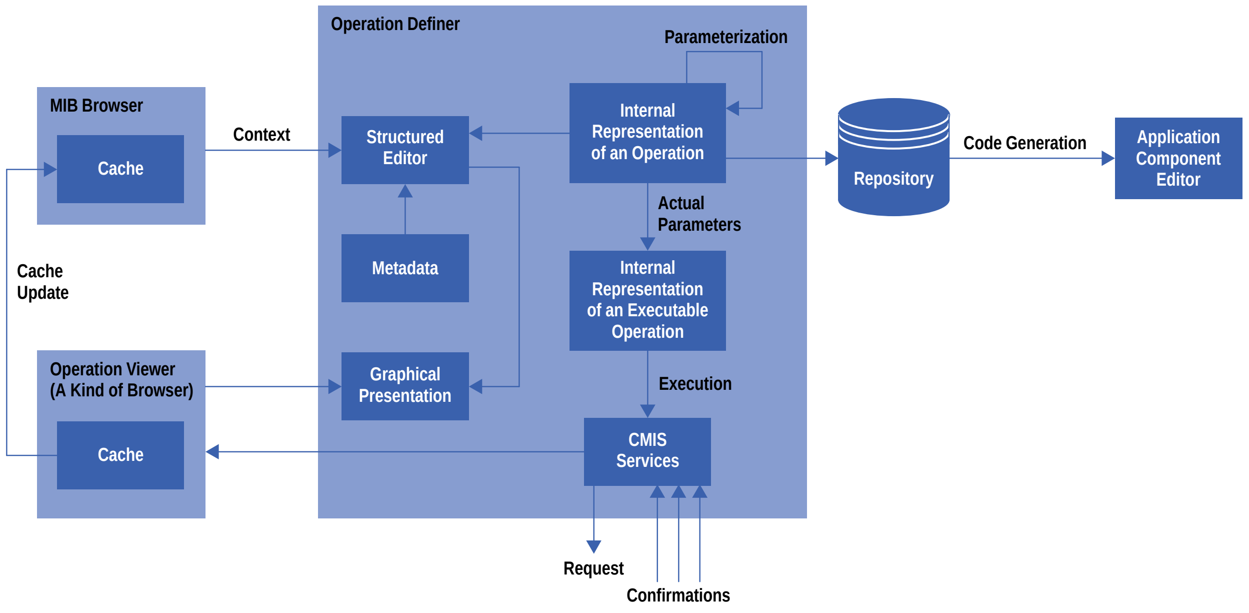


Fig. 5. *The design of the operation definer.*

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

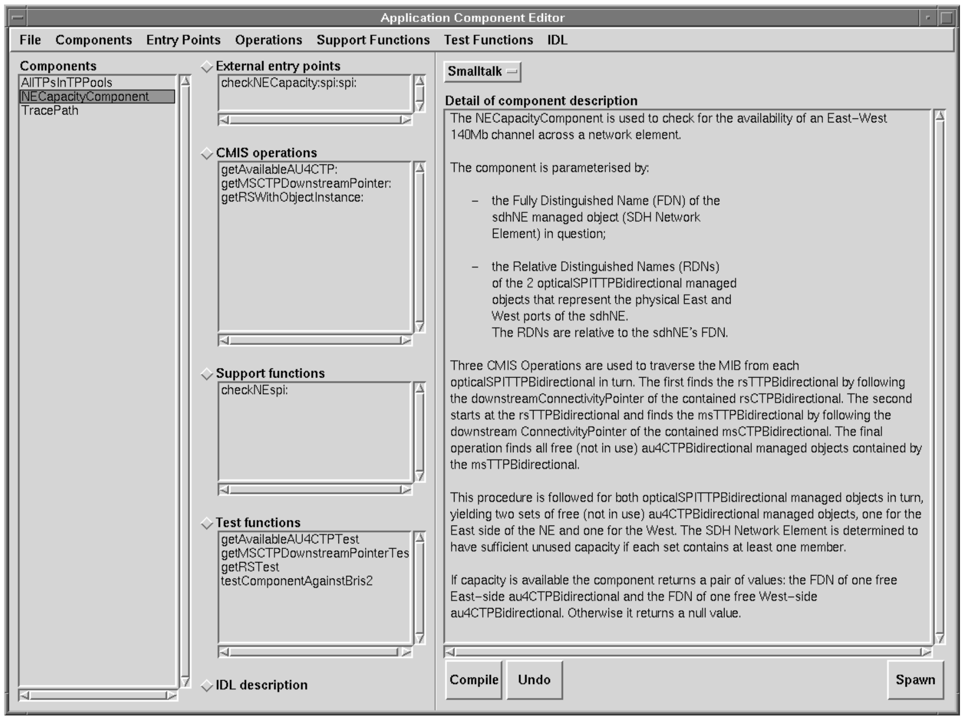
As stated earlier, we expect that future telecommunications applications will be made up of a number of large-grained, distributed objects. We call the objects application components. Some application components will communicate with data sources, including the MIB, to obtain the information that other 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.



AllTPsInTPPools
NECapacityComponent
TracePath

checkNECapacity:spi:spi:

getAvailableAU4CTP:
getMSCTPDownstreamPointer:
getRSWithObjectInstance:

checkNEspi:

getAvailableAU4CTPTest
getMSCTPDownstreamPointerTes
getRSTest
testComponentAgainstBris2

Detail of component description

The NECapacityComponent is used to check for the availability of an East–West 140Mb channel across a network element.

The component is parameterised by:

– the Fully Distinguished Name (FDN) of the sdhNE managed object (SDH Network Element) in question;

– the Relative Distinguished Names (RDNs) of the 2 opticalSPITTPBidirectional managed objects that represent the physical East and West ports of the sdhNE. The RDNs are relative to the sdhNE's FDN.

Three CMIS Operations are used to traverse the MIB from each opticalSPITTPBidirectional in turn. The first finds the rsTTPBidirectional by following the downstreamConnectivityPointer of the contained rsCTPBidirectional. The second starts at the rsTTPBidirectional and finds the msTTPBidirectional by following the downstream ConnectivityPointer of the contained msCTPBidirectional. The final operation finds all free (not in use) au4CTPBidirectional managed objects contained by the msTTPBidirectional.

This procedure is followed for both opticalSPITTPBidirectional managed objects in turn, yielding two sets of free (not in use) au4CTPBidirectional managed objects, one for the East side of the NE and one for the West. The SDH Network Element is determined to have sufficient unused capacity if each set contains at least one member.

If capacity is available the component returns a pair of values: the FDN of one free East–side au4CTPBidirectional and the FDN of one free West–side au4CTPBidirectional. Otherwise it returns a null value.

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

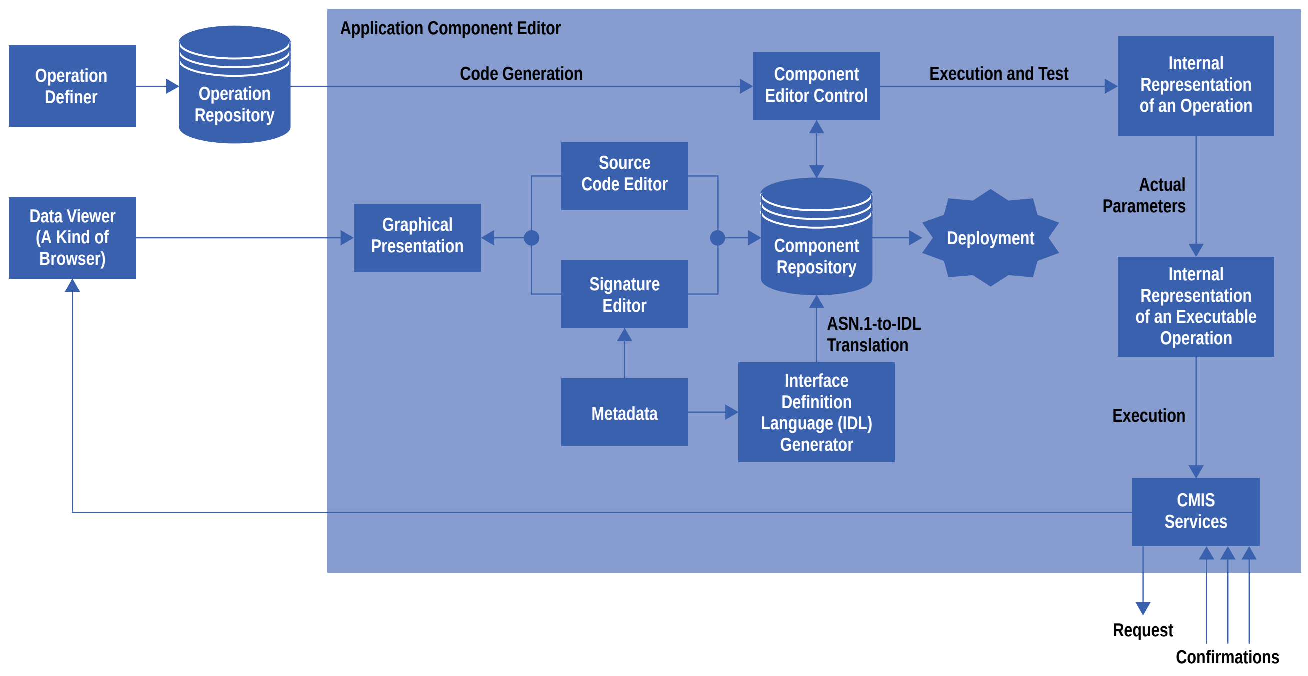


Fig. 7. *The design of the application component editor.*

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.

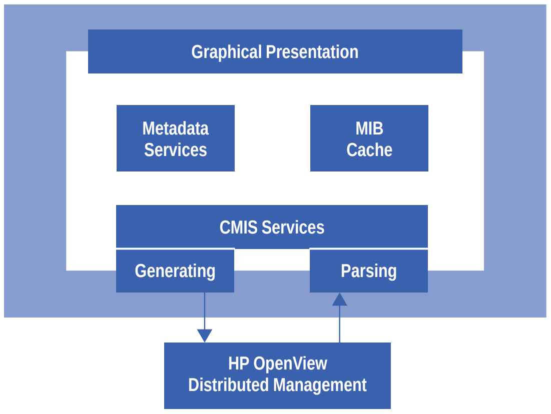


Fig. 8. *The architectural framework for the three prototype tools.*

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

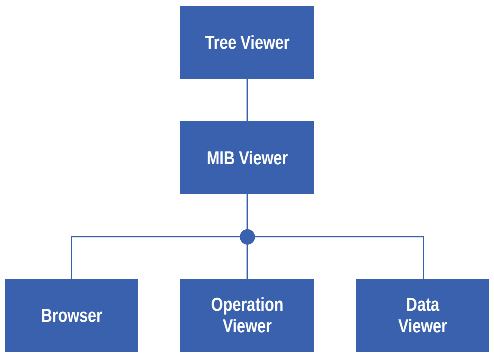


Fig. 9. *Inheritance hierarchy of the graphical presentation classes.*

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.

▶ Go to Article 8

▶ Go to Table of Contents

▶ Go to HP Journal Home Page