1 Introduction

Telecoms 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 financially – the emergency services and some air traffic control transmissions use the telephone network. Not surprisingly, network operators have considerable systems and network management needs.

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

The applications that telecoms companies use to manage their equipment, their networks and the services they provide are known as Operational Support Systems (OSSs). 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] area is still carried out with the aid of a C or C++ compiler. The developer must understand what data that is available to the application, the operations that can be performed in order 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 helps the application developer to understand the kind of data that is stored in the TMN world. The OpenView Distributed Management platform itself provides standard APIs that the developer can use to send messages to the data. The Managed Object Toolkit provides further support to the C++ programmer.

At present, though, there is no OpenView product that lets the user see the data itself, only the metadata, the GDMO and ASN.1, which describes what sorts of data might exist. The standard APIs are complicated and quite frustrating to use. Application development therefore proceeds slowly.

In this paper we describe a prototype development environment built at Hewlett-Packard Laboratories, Bristol that addresses some of the demands of application development in the telecoms world. We believe that its use could greatly improve the process. Our work was shown publicly at the Telecom ’95 exhibition in Geneva and was used to construct part of a second demonstration that was also shown there.

In this paper we start with a little background information then in section 3 we present an overview of the development environment. The individual tools are described in sections 4, 5 and 6 and their common architectural framework is covered in section 7. Finally, the conclusion.

Internal Accession Date Only

2 Background

This section describes briefly some of the technical background to this paper.

2.1 Telecommunications Management Network

The Telecommunications Management Network (TMN) is an attempt to standardize the management of telecoms networks. It consists of a set of existing and evolving recommendations [2] from the International Telecommunications Union's Telecommunications Standardization Sector, known as the ITU-T. These build on a number of previous recommendations on Open Systems Interconnection (OSI) management, now adopted as international standards [3]. See [1] for a discussion of TMN.

2.2 Management Information Base

OSI management proposes a Management Information Base (MIB) [4] that is a collection of the data necessary for management of the network. This data is organised hierarchically, related by containment. The data itself is in the form of objects, called managed objects, that is described by the Guidelines for the Definition of Managed Objects (GDMO).

2.3 Guidelines for the Definition of Managed Objects

GDMO [5] is the language used to define the structure and some of the relationships between managed objects. The 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.

2.4 Common Management Information Services and Protocol

The Common Management Information Services (CMIS) [6] are the services which can be 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.

3 The development environment

We believe that telecoms management applications of the future will be composed of a number of large-grained, distributed objects. We call these application components. Some of them will be specialised to 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 out into components, each capable of supporting transactions against a data source. Since our intention is to support the development of telecoms management applications we decided to focus on the construction of application components that interact with data, and to concentrate on the TMN MIB.

We have built a prototype development environment that includes three tools that operate together to support the development lifecycle of such application components. See Figure 1. In the initial stages this means providing aid in understanding the problem and progresses through to enabling the implementation, testing and deployment of the solution. By taking this approach we believe the application development process can be greatly improved.

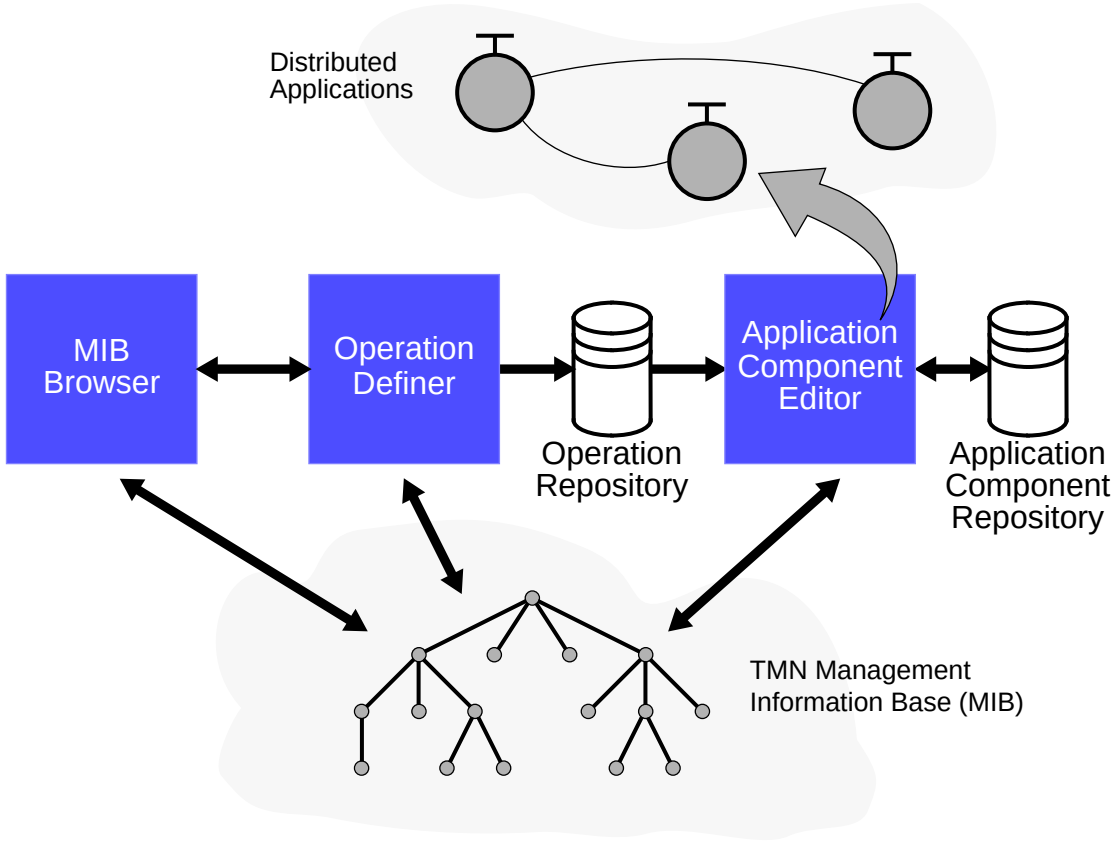

Figure 1: Prototype TMN development environment

Although the process is likely to be iterative the basic steps in developing an application component are:

- navigation through and exploration of the MIB using the MIB Browser, to build up an understanding of the data and the way it is used.
- prototyping of CMIS operations using the Operation Definer. These help to expand or verify the developer's understanding. The results of executing these operations may be fed back into the browser to aid the navigation.
- storage of operations for future re-use or as documentation aids.

- construction of fragments of application functionality out of a number of operations, using the Application Component Editor.
- deployment of the completed components, as distributed CORBA or OLE 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 tools appear and behave as an integrated development environment rather than standalone tools. This framework is discussed in a later section.

4 MIB Browser

The MIB Browser presents a graphical view of the MIB to the user, as shown in Figure 2. By interacting with and manipulating this view the user is able to navigate through the MIB, exploring the structure and content of the data stored in its managed objects.

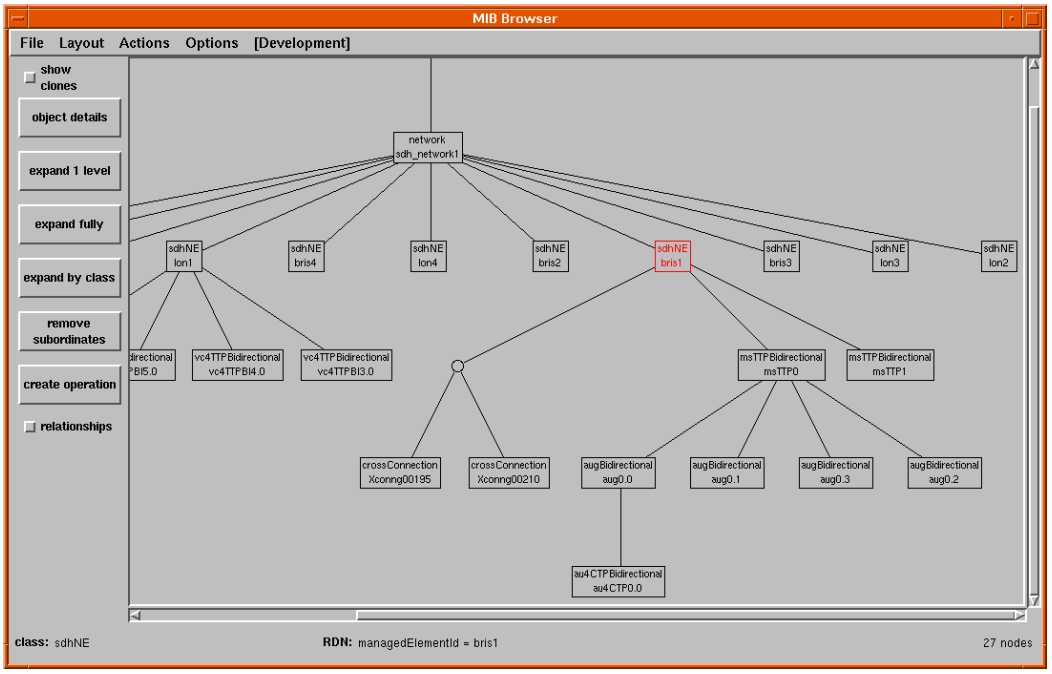

Figure 2: Window dump of the MIB Browser

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 in number from none to very many, are used to update the cache and hence the view presented to the user. This is described in more detail below.

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 details of attribute values are presented to the user as words, rather than as the numbers that the underlying infrastructure deals in. 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 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 Figure 3 and discussed in more detail in section 7.

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 in the training of these and other staff. Its main benefit is that it is not necessary to understand the technical details of the area in order to use it successfully. In fact, we find that it helps users to increase their understanding.

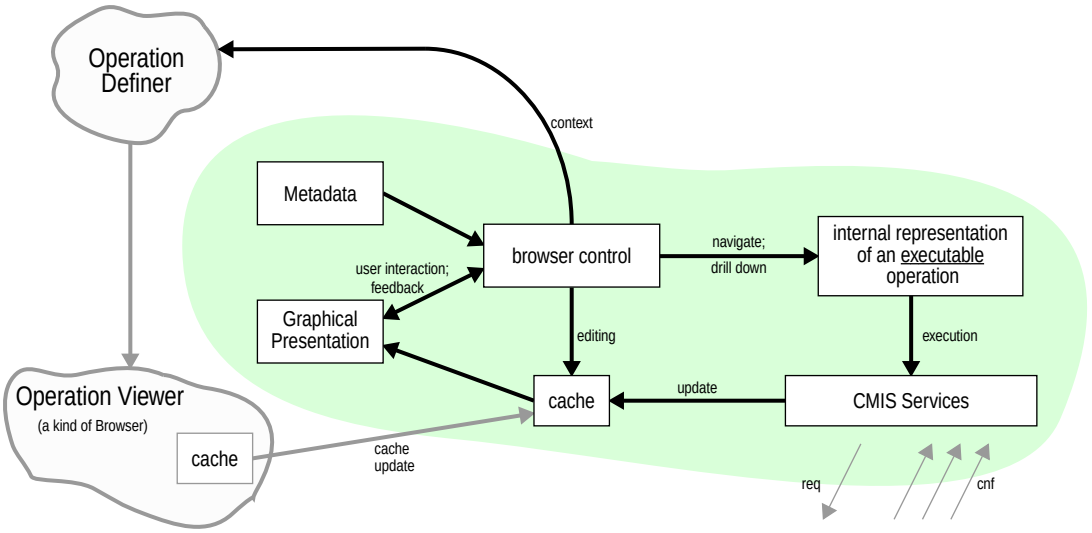

Figure 3: Design of the MIB Browser

4.1 Navigation by CMIS

The MIB Browser provides five pre-defined CMIS operations that can be used to retrieve data from the MIB. The execution of the operations is concealed by the user interface and the user is not aware of the precise nature of the interactions.

Three of the operations 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 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.

The fifth operation is used to find out 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, although 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. A large area of the MIB may lie within the scope of an operation, containing thousands or many more ob-

jects. 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 organised is often what matters. So the browser does not need to store the details of every managed object.

We realised that we could leave the user to 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.

4.1.1 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. The selected managed object is used to provide the context necessary to define the base managed object class and instance fields of the request. The details of the request, hidden from the user, are shown in Table 1.

Table 1: CMIS M-GET request for the “expand fully” operation

Field	Value
base managed object class	<taken from browser context>
base managed object instance	<taken from browser context>
access control	*omitted*
synchronization	bestEffort
scope	wholeSubtree
filter	*omitted*
attribute ID list	{}

4.1.2 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 discovering all the managed objects below the selected position it discovers only those immediately beneath it. In MIB-speak, it finds all the managed objects contained by the base object. In computer-speak, it finds the children. See Table 2 for details of the request.

Table 2: CMIS M-GET request for the “expand one level” operation

Field	Value
base managed object class	<taken from browser context>
base managed object instance	<taken from browser context>
access control	*omitted*
synchronization	bestEffort
scope	firstLevelOnly
filter	*omitted*
attribute ID list	{}

4.1.3 Expand by class

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, as discussed in section 7.2. The user can select those managed object classes of interest or can scan the list and choose likely candidates – prior knowledge is helpful but not essential. The choices are used to parameterise the request, full details of which are shown in Table 3, that finds all occurrences of these classes below the previously selected starting point.

Table 3: CMIS M-GET request for the “expand by class” operation

Field	Value
base managed object class	<taken from browser context>
base managed object instance	<taken from browser context>
access control	*omitted*
synchronization	bestEffort
scope	wholeSubtree

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

attribute ID list	{}

4.1.4 All attributes

This operation is the only drill-down operation that the user can execute from the MIB Browser. It targets a single managed object that was discovered by an earlier navigation. The values of all the object's attributes are obtained. See Table 4 for details.

Table 4: CMIS M-GET request for the "all attributes" operation

Field	Value
base managed object class	<taken from browser context>
base managed object instance	<taken from browser context>
access control	*omitted*
synchronization	bestEffort
scope	baseObject
filter	*omitted*
attribute ID list	*omitted*

4.1.5 Find class

This operation can be invoked on a managed object whose existence and name has been inferred from the results of an earlier operation but whose class is unknown. See Table 5.

Table 5: CMIS M-GET request for the "find class" operation

Field	Value
base managed object class	actualClass [5]
base managed object instance	<taken from browser context>
access control	*omitted*
synchronization	bestEffort
scope	baseObject
filter	*omitted*
attribute ID list	{}

5 Operation Definer

The MIB Browser provides its 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 – the protocols and information models used – the Operation Definer gives full flexibility in the construction of CMISE requests. Operations defined this way can be use to extend the browser's repertoire.

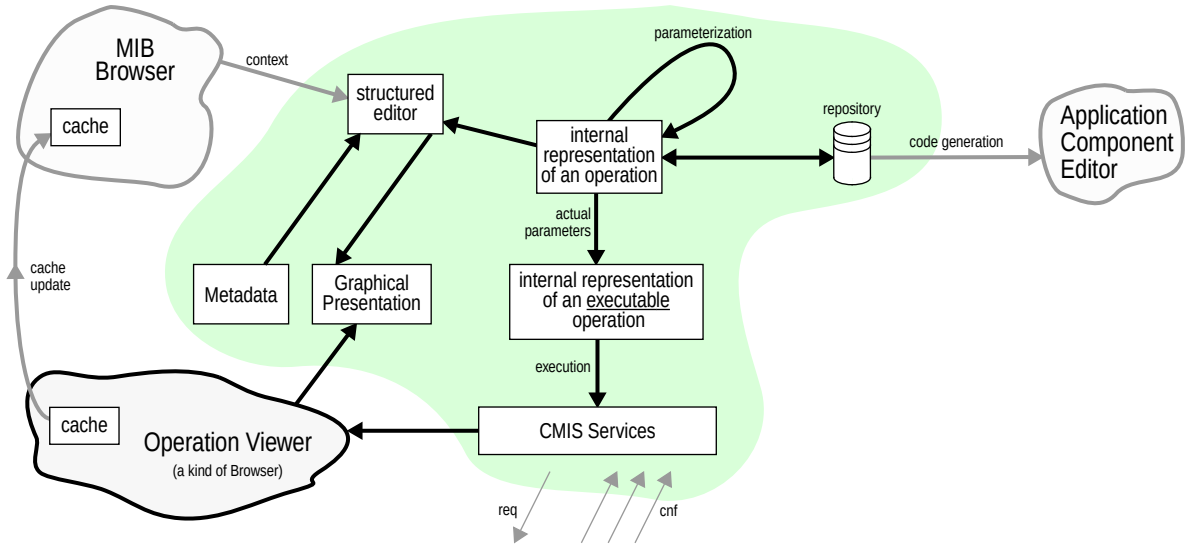

Figure 4: Design of the Operation Definer

Details of how the different elements of the Operation Definer are related are shown in Figure 4.

As the name implies the Operation Definer helps the user to specify an operation they want 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 them in a similar way to the MIB Browser. The user can examine the results and if necessary modify the operation and re-issue it. This cycle can continue until the user is satisfied with the operation.

The results obtained by executing an operation can be added to a MIB Browser, in order 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 Definer might be the operation itself, rather than the results of executing it once. The results returned will be useful, in that they can help prove it works correctly, but the specification is the real output. The user might want to store such an operation so that it can be used in the future and the Operation Definer maintains a repository for this purpose. By adding to this repository the user can build up a "toolbox" of useful operations, rather in the way a system administrator builds up a library of shellscripts.

For an operation to be executable all aspects of its specification must be fixed. The Definer picks up the starting point, the base managed object and class, from the browser context. Other aspects 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 in the future 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 parameterised, meaning that each time it is used the values of those aspects 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.

6 Application Component Editor

It is our belief that telecoms applications of the future will not be monolithic but will be composed of a number of large-grained, distributed objects, which we call application components.

Some of these 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. The Application Component Editor, Figure 5, is the result.

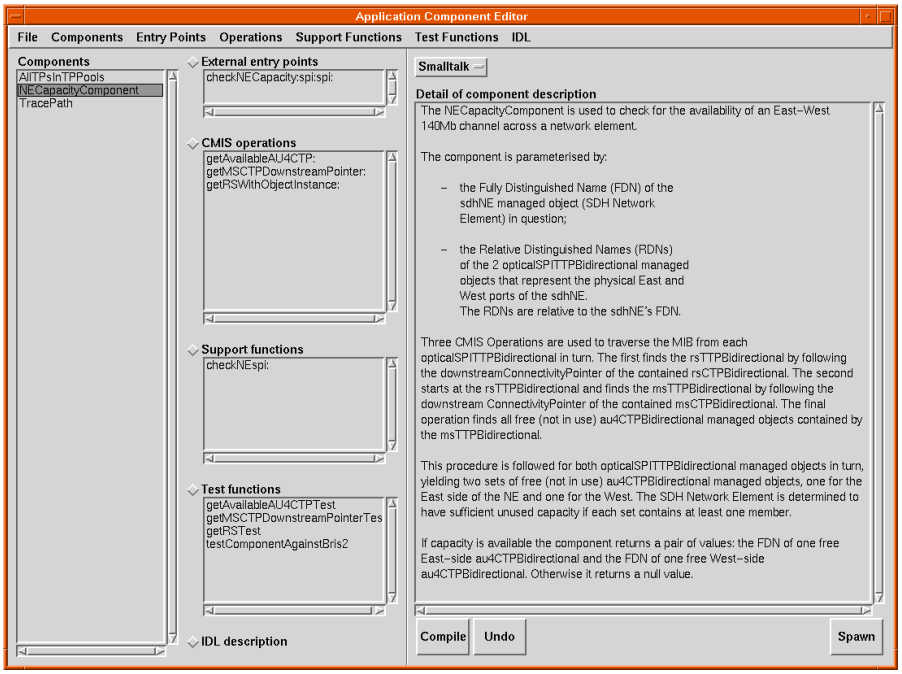

Figure 5: Window dump of the Application Component Editor

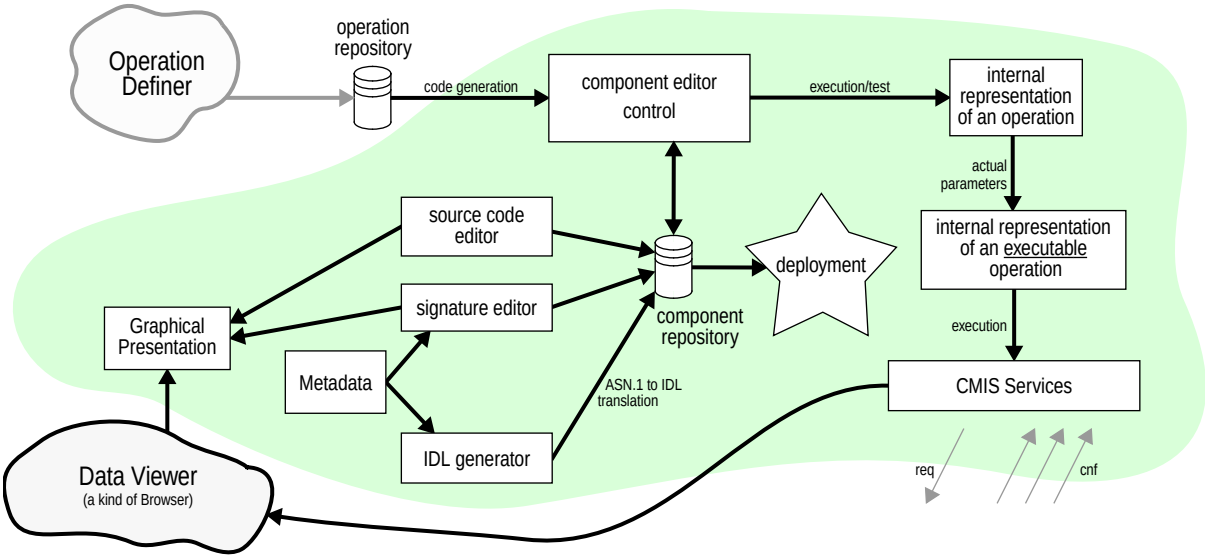

Figure 6: Design of the Application Component Editor

A component has 4 parts:

- a number of entry points it presents to the outside world. Together these make up its visible interface. Other parts of an application make use of the component by making calls against this interface.
- a number of operations that interact with the MIB. These issue 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 component as a whole.

As shown in Figure 6 the Component Editor makes use of the Operation Definer's repository. Operations stored there can be selected for inclusion in components. They are translated into source code, as a method, that, when executed, will perform the same operation. The fields that were identified as parameters when the operation was stored away are now treated as formal parameters to the method. In addition, simple test functions are automatically generated that test the operation with its original parameter values. These too are realised as source code. The results obtained from running the test functions are presented to the developer in a similar style as the MIB Browser.

Although source code generation is not strictly necessary – interpretation of previously built operations, such as is performed by the Operation Definer, would do perfectly well – the ability to generate it helps make the tools acceptable to the more traditional telecoms OSS developer market.

The Component Editor is not restricted to editing new and existing components – it also provides help in deployment. There are several ways in which a completed component can be made available for inclusion in an application:

- 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 behaviour

We have concentrated on the first option. The signature editor, shown in Figure 6, 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 generator which produces equivalent CORBA/IDL interfaces using a standard translation algorithm [8][9]. These 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, now a ParcPlace® product, to automate the entire process of deployment as a CORBA object. 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++.

7 Architectural framework

The three tools are built on top of a shared framework, as shown in Figure 7. This contains a number of components that provide useful services. By building on top of this framework we were able to increase commonality of implementation and of appearance and behaviour, making the tools behave as members of an integrated environment rather than as independent, standalone tools.

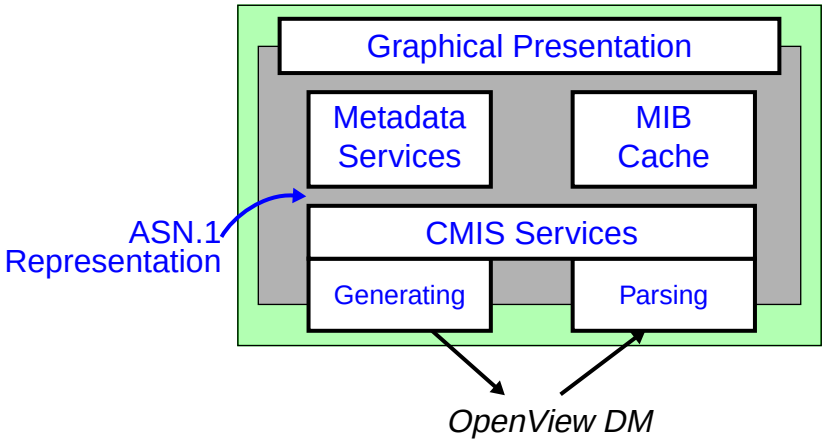

Figure 7: Architectural framework

7.1 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 specialise them to 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 itself 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 shared a common look and feel – because much of the code was common to them all. Figure 8 shows the inheritance hierarchy.

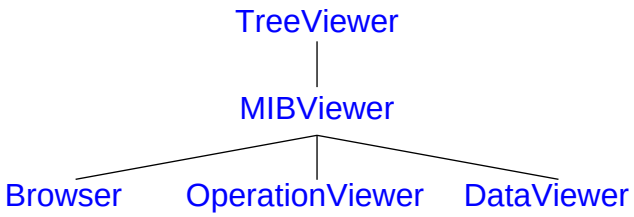

Figure 8: Inheritance hierarchy of the Graphical Presentation classes

The managed objects in the MIB are organised into a tree, called the containment tree in TMN because the tree's edges represent a containment relationship. So networks contain devices which contain equipment which contain software. This containment-oriented view shows the equipment-oriented origins of the TMN recommendations. The MIB Browser enables its user 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 cause the browser to execute CMIS operations to extend the browse in the way the user directs. We showed a view of the tree as it was discovered during the browse. We felt that focusing on a single managed object at a time, or presenting a view only of the path through the tree to that object would hide the larger picture from the user. This larger picture often provides the context that helps the user to understand the smaller picture.

One lesson we have learnt from the 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 their 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 their screen rather than throw the onus on to them to "do better next time".

7.2 Metadata services

There are many sorts of metadata used by the tools:

- GDMO that describes the kinds of data that may be stored in the MIB and how they can be structured;
- ASN.1 that 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 parameterised;
- descriptions of each application component.

The tools obtain the GDMO and ASN.1 metadata via the OpenView Metadata Services. It 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 and makes 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. While the user wants to refer to the "*network*" class and the "*downstreamConnectivityPointer*" attribute and the "*internalTimingSource*", "*speech*", "*locked*" and "*sunday*" data values the CMIP protocol will expect to see { 0 0 13 3100 0 3 1 } and { 0 0 13 3100 0 7 19 } and 0, 0, 0 and 0. Our metadata services perform these context-sensitive translations automatically, in both directions.

7.3 MIB cache

Operations sent by the tools result in responses being received from the MIB. A response equates to a single managed object that was involved in the operation. Some responses indicate errors, 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 learnt about the MIB by the tools. The cache can be pre-loaded ("seeded") on start up, 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 on to the MIB cache.

We did not attempt to maintain consistency between the cache and the real MIB. One way of increasing the timeliness of the data would be to register to receive appropriate types of events that would inform the browser of attribute value changes and object creation and deletion.

7.4 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 on Smalltalk's support for multiple thread execution and synchronization.

Smalltalk classes were defined that represent CMIS requests and confirmations. 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. These are then sent to the Smalltalk process that made the request.

7.5 OpenView Distributed Management platform

The tools connect to an intermediary program called the CMIS Interpreter which in turn uses OpenView DM 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, rather like a symbolic form of CMIS.

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

7.6 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. Their types are represented in the metadata.

8 Conclusion

We have described the prototype of a software environment that aids the construction of telecoms management applications. It comprises three tools that together address many aspects of the development lifecycle, from investigation of the problem through to deployment of the solution.

By addressing a well-defined subset of the overall application area – application components that interact with the TMN MIB – we have been able to build tools that partially automate the task. We believe this could greatly increase developer productivity. The tools' usefulness is not restricted to the development of applications: the MIB Browser, combined with operations and components built using the other tools, is a powerful environment for exploring, understanding and trouble-shooting the MIB.

Acknowledgements

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

References

[1] CCITT M.3010, “Maintenance - Principles for a Telecommunications Network: Principles for a Telecommunications Management Network”, 1992.

[2] ITU-T M.3000, “Maintenance - Principles for a Telecommunications Network: Overview of TMN Recommendations”, 1994.

[3] CCITT X.701 | ISO/IEC 10040, “Information technology - Open Systems Interconnection - Systems management overview”, 1992.

[4] CCITT X.720 | ISO/IEC 10165-1, “Information technology - Open Systems Interconnection - Structure of management information: Management information model”, 1992.

[5] CCITT X.722 | ISO/IEC 10165-4, “Information technology - Open Systems Interconnection - Structure of management information: Guidelines for the definition of managed objects”, 1992.

[6] CCITT X.710 | ISO/IEC 9595, “Information technology - Open Systems Interconnection - Common management information service definition”, 1991.

[7] CCITT X.711 | ISO/IEC 9596-1, “Information technology - Open Systems Interconnection - Common management information protocol - Part 1: Specification”, 1991.

[8] X/Open Company Ltd., “Inter-Domain Management: Specification Translation”, Preliminary Specification, 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.