A GENERIC MANAGEMENT INFORMATION BASE BROWSER

G. Pavlou, J. Cowan, J. Crowcroft

Department of Computer Science
University College London
Gower Street, London WC1E 6BT
England

Abstract

A *Management Information Base browser* is a very important application in any management environment as it enables a human manager to browse through the Management Information Tree of a remote managed system, looking at and possibly modifying management information. Such an application can be designed in a *generic* way, without prior knowledge of the managed object classes present in a managed system. This results in a tool that can cope with different versions of standard MIBs, proprietary extensions or even newly introduced ones. The issues behind the concepts and the design of the generic browser are presented in this paper; special attention is given to the issue of coping with changes in the remote MIB in a generic fashion.

Keyword Codes: C.2.3; C.2.4
Keywords: Network Operations; Distributed Systems

1. Introduction

The OSI Network Management model is based on the object oriented paradigm: management of physical or logical real resources is enabled through abstractions of them known as Managed Objects (MOs) [1]. These and their properties are formally specified in an abstract form [2] and this specification together with the access management protocol CMIS/P [3] [4] identify uniquely the interoperable interface for management applications. The managed object classes are organised in an inheritance hierarchy while the managed objects constituting a Management Information Base (MIB) instance in a managed system are organised in a Management Information Tree (MIT), according to containment relationships [5]. In simple terms, an inheritance relationship specifies that an object *is a kind of* another object, while a containment one specifies that an object *is a part of* another object.

The OSI Network Management standards specify facilities that are both powerful and flexible. One of the most powerful concepts is that of allomorphism. This enables either managed or managing systems to be extended, while at the same time allowing existing ones that are unaware of these changes to continue functioning as normal. This flexibility is very important as the OSI management standards will evolve over a long period of time, possibly

resulting in discrepancies between the managed object classes understood by managing and managed systems. This will be true, for example, where new generations of standard Management Information Bases have been implemented, proprietary changes to standard MIBs have been made and new managed object classes have been introduced. Version control problems in MIBs is known to be one of the most important management problems.

A very important application in any management environment is a Management Information Base browser which enables the human manager to browse through the Management Information Tree of a remote managed system, looking at and possibly modifying information to implement management decisions. It should be emphasised here the fact that the browser enables one to browse through a MIB *instance*. This is in contrast with browsers that allow one to browse through *definitions* of managed object classes, in a similar fashion to object-oriented programming language browsers as pioneered by Smalltalk.

The browser as an application can be designed in a *generic* way, assuming no prior knowledge of the managed object classes present in the target managed system; it thus can cope with the evolution of MIBs. This provides a solution to the problem of version mismatch, proprietary extensions or completely new classes. The generic browser uses the powerful features of the OSI Management model to enable one to move around in the management information base and detect changes in it. It has no hard-wired knowledge of the object classes in the target managed system. All it uses is minimal information stored in a local database which describes mappings between the class, attribute and event identifiers and their names and abstract syntaxes as defined by the OSI management standards. This database should be updated regularly with information in addenda, proprietary extensions etc. This concept can be taken a step further using the global OSI directory as a repository for this information which could be searched when information cannot be locally found.

Another important aspect of such a tool is to allow a human manager to detect changes in the management information. The obvious solution to this is to use polling which is a necessary management facility but hardly satisfactory as it results in wastage of network resources. Event reporting is better suited to the nature of the OSI Management model which provides a rich set of facilities [6] [7]. The problem of how the browser uses these generically is described later in detail.

The rest of the paper has the following structure: in Section 2 we look in detail in the most important features of the OSI management model that are used to implement the generic MIB browser. In Section 3 we present its design in terms of features and functionality and look in detail at the two different modes of detecting changes, polling and event reporting. In Section 4 we discuss the implementation models, concentrating on the structure and content of the information held in the local database and the possible usage of the OSI directory as a repository for this. Finally we present our conclusions.

2. The OSI Management Information Model

We describe here the general features of the OSI Management Information model and in particular those used by the browser to manipulate the remote MIBs in a generic fashion. The model is based on the object oriented paradigm and allows one to manage real resources through abstractions of them known as managed objects. Managed Object Classes (MOCs)

related to various communications resources are specified by standards groups in a formal language whose acronym is GDMO, standing for Guidelines for the Definition of Managed Objects [2]. The management information schema, its inheritance and containment hierarchies together with the management attributes, operations, actions, notifications and behaviour of each managed object class are formally specified in that language.

Management of real resources takes place through instances of managed object classes i.e. managed objects. The collection of managed objects related to the management of an open system is known as a managed system. These managed objects are organised in a Management Information Tree according to containment relationships. This is necessary for naming purposes, so that managing applications can unambiguously access them. In the MIT schema, the containment relationships between classes are specified by Name Bindings [5]. Each object class has a naming attribute in a name binding with a containing class, which together with its value identifies the object uniquely within the scope of the latter. This is known as a Relative Distinguished Name (RDN). The sequence of all RDNs from the top of the tree to a managed object constitute its Distinguished Name (DN), which is unique within the scope of the managed system. A simple example of an MIT instance and its schema are given in Figure 1.

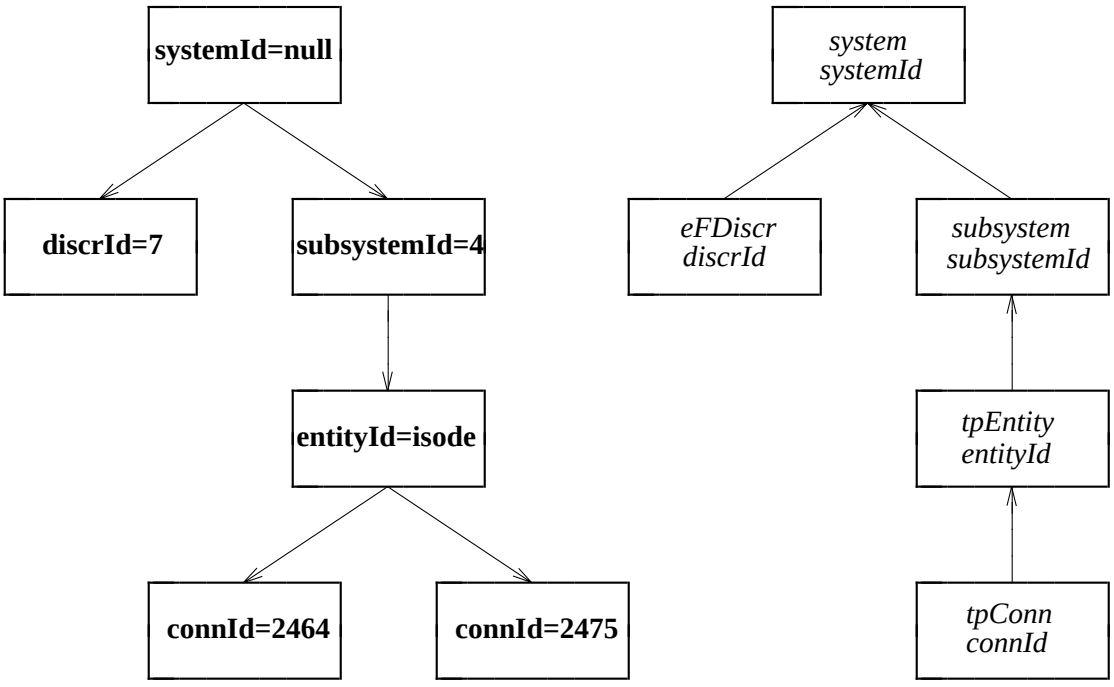

Figure 1. Example of an MIT Instance and Schema

A managed object instance may belong to many *allomorphic* classes, enabling it to be managed as any of them [5]. This is a very important concept as it allows one to extend a managed object class according to specific needs without preventing *other* systems from managing it. All managed object classes are derived from the generic class *top*, which has as its attributes the actual class, the list of allomorphic classes, the name binding and the list of

packages realised in that object instance. A package is a collection of attributes, notifications, operations and behaviour and there may be conditional (optional) ones, increasing even further the management flexibility.

The management service/protocol CMIS/P [3] [4] enables access to management information. It provides a rich set of operations which together with the managed object class specifications uniquely identify the management interface between a managing and a managed system. These services namely are:

- **M-GET** which reads management attributes
- **M-SET** which modifies management attributes
- **M-ACTION** which performs an action on a managed object
- **M-CREATE** which creates a managed object
- **M-DELETE** which deletes a managed object
- **M-EVENT-REPORT** which enables a managed system to notify a managing application that an event has occurred.

All these operations have as common parameters the distinguished name and the managed object class. The former enables a particular object to be addressed while the latter enables it to be managed as a particular allomorphic class. The M-GET, M-SET, M-ACTION and M-DELETE operations may be applied to more than one objects using scoping: this enables one to address subordinate objects of a particular level, down to a particular level or the whole subtree. The selection of objects may be controlled by a filter parameter, containing assertions on attribute values connected by logical operators.

Let’s look now at those issues relevant to the generic nature of the browser. First it is possible to get the attributes of a managed object without knowing what these are in advance. This is achieved by specifying no attribute list in the M-GET operation which results in all attributes being returned. Also using scoping, the subordinates of an object can be found. Usually when searching for subordinates one specifies just one well known attribute e.g. the "objectClass", to reduce processing in the agent. Every result will contain the object class and distinguished name, so that any of the subordinate objects can subsequently be addressed.

The key point is that in order to bootstrap this searching process, there must be a well known top object in the MIT. The standards specify that the top object in every managed system will be an instance of class *system*, which can have any relative distinguished name. The local distinguished name form is used in order to access this, which enables one to access objects relative to the top. If the system object’s distinguished name is e.g. {systemId=athena}, its local name would be {}, an empty DN. In this way the top object in a managed system can always be accessed [5].

The other main issue relates to event reporting, whose model is described in [6]. A managing system, such as the browser, needs to create a special management support object called an *eventForwardingDiscriminator* in the remote managed system in order to receive event reports. The main attributes of that object are a destination address, which specifies where an event report should be sent, a discriminator construct i.e. a filter which specifies the conditions under which the event report should be sent and an administrative state, which

may be used to turn the discriminator off and on. The latter may be used to avoid deleting the discriminator in order to temporarily suspend event reporting. Via the event discriminator, one can receive either all event reports from the remote MIB when no filter expression is specified, or particular event reports depending on the assertions in the filter.

Finally it should be stated that the managed object class, attribute and event identifiers are ASN.1 [8] **OBJECT IDENTIFIER**s i.e. a series of numbers according to the registration to the global ISO/CCITT naming tree. Attribute values and event report information are presentation streams i.e. instances of the ASN.1 type **ANY** [4]. These can only be meaningfully manipulated if their syntax is known. This is defined in standard or other ASN.1 modules and specified in the managed object class specification. A local database is used to describe the mapping of class, attribute and event identifiers to their names and abstract syntaxes. It is this information the browser uses to handle attribute and event information in a generic fashion, as described in Section 4.

3. The Generic Browser

Designing the generic MIB browser according to the principles described above raised two main issues: the first was how to organise the display to allow the human manager to move around in the MIT. The second was how to ensure that the local display reflected exactly the state of the remote MIB.

3.1 Operations and Functionality

Initially we started with the idea of displaying the whole of the containment hierarchy of the remote MIB graphically, as a tree structure. For each node of the tree we would display the class and relative distinguished name, allowing the user to select a specific object to look at all its attributes. The problem with this approach is that it requires one to read the whole management information tree and it only gives a picture of it at the time the information was requested i.e. at the browser start-up time. Transient objects such as connections may be created and deleted reflecting the operation of the real resource they represent. Other managed objects, such as event discriminators may be created through CMIS by another manager. Requesting this information periodically in order to update the displayed MIT does not guarantee to capture all changes and is quite an expensive operation in terms of both agent processing and network resources.

An alternative approach is to request all event reports from the remote MIB in order to update the displayed MIT: managed objects should emit event notifications associated with object creation and deletion. As event reports include the distinguished name of the emitting object, the policy to update the displayed MIT is simple: if the distinguished name is unknown to the browser it means that an object has just been created and the display is updated. If the DN is known, one single attribute is requested to find out if the object was possibly deleted. The problem with this approach is that requesting all event reports and - for most of them - subsequently performing a M-GET operation may potentially create even more traffic than periodically requesting the whole MIB!

Instead of displaying the entire MIT we adopted the notion of a current object: this is an object whose attributes are displayed along with the class/relative distinguished names of its immediate subordinates; the position of this object in the containment hierarchy is

indicated by displaying its distinguished name. The human manager may then either select a subordinate object in order to move **DOWN** the MIT or move **UP** to the object above, making it the new current object.

A **REFRESH** operation allows a manager to refresh the values of the current object without having to move back up and down the containment hierarchy. When refreshing an object or even when trying to move up or down the MIT, there is the possibility that the target object no longer exists e.g. a connection has been closed resulting in the deletion of the corresponding managed object. In that case, the browser automatically moves up the hierarchy until it finds an object that exists.

As one moves up and down the MIT, it is also possible to **MODIFY** the value of an attribute. As long as its syntax is known, the value entered from the keyboard may be generically parsed. An M-SET operation is always attempted at the remote agent as the browser does not know which operations are valid for each attribute of the current class; this introduces a possible waste of network resources but this is the only way the generic browser can ascertain if the operation is invalid, its value wrong etc. and then display this to the user.

Looking at the mechanics of the generic handling of these operations, they are all based on the features of the OSI management model as explained in section 2. When the browser is first started it attempts to establish a management association to the remote managed system and to subsequently get the top object of class *system* using an empty local distinguished name. If any of these steps fail, the browser abandons communication with that managed system. Otherwise, the state information that the browser keeps about the current MIT position is:

1. a list of all the managed object classes from the top to the current object
2. the current object’s distinguished name
3. its attribute identifiers/types and their position in the display
4. the managed object classes and (relative) distinguished names of its subordinates

This information is adequate to enable the browser to move up and down the MIT as the appropriate distinguished name can be constructed and the managed object class is known. The attribute identifiers of the current object enable one to request the modification of an attribute value. Every time the current position is changed, two M-GET requests are issued: one for the new current object with an empty attribute list i.e. requesting all attributes and no scope information and another one for the same object with a single attribute and scope specifying the first level subordinates.

In order to display the current object in a meaningful way, the browser looks up in the local directory the mapping of class and attribute identifiers to their respective names and syntaxes. Even if the database does not contain information about objects in the MIB, the browser is still able to move up and down the MIT. However the attribute and class identifiers are only displayed in the numeric "dot notation" form e.g. 2.9.3.2.41.3 and no attribute values are displayed. Even when an attribute is found through the directory, its syntax may be a proprietary extension not known to the browser i.e. has no precompiled knowledge, in which case that attribute value cannot be displayed.

3.2 Monitoring by Polling

The concept of the current object is powerful enough but it only allows one to look at a single object at a time. One may express the need to examine more than one object at once. Also there may be a need to poll these objects regularly, in which case refreshing each of them becomes cumbersome. This led to the idea of a **MONITOR** operation: it is possible to monitor an object in a separate window which refreshes itself periodically according to a default time interval that can be changed.

This operation applies to the current object which may become a monitored one. Subsequent changes of the MIT position do not affect any monitored objects. This way there may be many objects monitored in separate windows, in addition to the current one. When a monitored object dies, the relevant window is closed and an appropriate message is displayed. It must be noted that monitoring by polling is a rather wasteful operation since unchanged objects may be retrieved. Nevertheless, it is important for the network manager and the browser provides this flexibility.

An instance of the generic browser display is shown in Figure 2. The basic window contains the current object's class, distinguished name, attributes and subordinates. The actual object on the display is a transport entity and has two transport connections as subordinates. In the one window, a modify operation is attempted on the *remoteUnsuccessfulConnections* attribute while in the other one of the connections is being monitored. The MIT instance is the same as that of Figure 1.

3.3 Event Handling

Rather than using polling, the OSI Management model encourages the use of event handling to report important changes in the MIB. Ideally, one would like to point to a particular managed object or even an attribute within that object and request to be informed when it changes. Unfortunately, there is no way to generically identify which event reports are associated with a particular managed object class and what are their semantics. This information is specified in the managed object class description and even if there was a way for the latter to be determined on the fly, its semantics are only specified in a natural language such as English!

An event report associated with a managed object class may convey any information pertaining to the event, possibly comprising (some) attribute values of the emitting object among other relevant information. Even though the syntax of an event report can be dynamically determined in the same way as that of the attributes, applying this information to update the displayed object could only happen if knowledge of that particular object class, its event reports and their semantics were hard-wired in the browser. The latter is obviously undesirable as it defies its generic nature.

An alternative approach would be to receive all event reports emitted by a managed object instance and subsequently refresh the local copy of that object by requesting all its attributes again. This introduces the overhead of one more management operation (namely a M-GET) per event report but it is still better than periodically polling the object in terms of network resource usage and management information timeliness. This is the approach taken in the browser.

Refreshing the local copy of an object through event reporting does not guarantee that all the changes in the management information the object comprises will be captured. Most of them will take place silently, without the emission of event reports. It will only be significant changes, classified as events, that will be reported if requested. Examples of these are state changes, error conditions, threshold violations etc. These should be adequate to cover important changes in the operation of the real resource the managed object represents. Refreshing the local managed object copy after the received report will, most of the time, give a picture of what happened by highlighting the changes to attribute values.

Even if the browser cannot determine which event reports are likely to occur and under which conditions, the human manager may be partly able to determine that from other contextual information e.g. from the name of an attribute. If there is for example a protocol machine object with one attribute named "protocolErrors" (a counter) and another one named "protocolErrorsThreshold" (obviously a threshold applied to that counter), it is very likely that event reports will be emitted when the threshold is violated. However, the browser program itself is unable to make this semantic leap by examining just a sequence of ASCII characters.

The current browser's implementation of event handling relates to the current object being examined or to any other monitored object. The monitor operation has actually two modes: monitor by polling and monitor by event reporting. The two modes can co-exist and the human manager may select whichever mode s/he finds appropriate.

Let's now take a look at the mechanics of this generic event handling. The first time the browser is requested to do anything with event reporting, it creates an event forwarding discriminator managed object in the remote managed system. This contains a filtering expression specifying under which conditions an event report should be forwarded. In the case of an object monitored by event reporting, it is enough to assert on its name:

```
objectInstance = <monitoredObjectInstance>
```

This guarantees that only events related to the particular monitored object will be received. Every time this happens, the object is refreshed and a message about the change is displayed.

When the current object is being monitored, one is interested in both the current object and its immediate subordinates. Supplying a filter comprising a logical OR of expressions as above becomes too expensive to evaluate when there are a lot of subordinates. CMIS filtering is powerful enough to allow simplify this using the *GreaterThanOrEqual* operator:

```
objectInstance >= <currentObjectInstance>
```

This involves only one distinguished name comparison, which alleviates processing in the agent. When a report pertaining to the current object occurs, the object is refreshed. When the report pertains to any of its subordinates, its distinguished name on the display is highlighted to show that something happened. The human manager may subsequently move down to that object.

When there are many event monitored objects, the filtering expression is a logical OR of all the expressions and the browser keeps track of the current one to which it adds or subtracts accordingly. Every time this expression changes, the discriminator construct attribute is set through a M-SET operation in the remote agent in order to change the event reporting mode.

4. Implementation Models

We previously referred to a local database which is used to map attribute and event identifiers to their respective names and syntaxes. The information in this database is as defined in the various documents specifying management information related to communications resources. This will be gradually enriched with proprietary or other information as it becomes available. The exact information in the database is:

- the object identifier of the attribute, event or class
- its name as an ASCII string e.g. activeConnections
- the name of the managed object class it belongs
- the object identifier of the ASN.1 module where its syntax is defined
- the actual syntax as in that module e.g. ObservedValue

This information allows the browser to map the object identifier of the classes, attributes and events to their name and syntax. The browser has precompiled knowledge of various syntaxes, which it uses to manipulate meaningfully the attribute and event values. This knowledge consists of the following methods related to a particular syntax:

- *encode* which converts an internal representation of a syntax to a ASN.1/BER [8] [9] presentation stream
- *decode* which performs exactly the inverse operation
- *parse* which converts an ASCII string to an internal representation and
- *print* which performs exactly the inverse operation

The term internal representation above means a data structure as manipulated by a program. When a presentation stream comes from the network i.e. an attribute or event report value, decode and print are used to generate the value to be displayed while parse and encode are used for the modify operation to convert the typed-in value to a presentation stream.

In the current implementation, a simple ASCII file is used as the database and all the information is read in memory for fast access. This of course has an impact on the size of the browser but for the time being the amount of the information loaded is quite small. A more scalable model would be to load in this information as the browser encounters new objects during its operation, possibly from a real database. This may result in slower access to objects the first time they are encountered. Some example entries for the ASCII database used at present are given in Table 1.

name	oid	class	module	syntax
discriminator	2.9.3.2.3.3			
discrId	2.9.3.2.7.1	discriminator	2.9.3.2.2.1	SimpleNameType
discrConstruct	2.9.3.2.7.56	discriminator	2.9.1.1.3	CMISFilter

TABLE 1. Example database entries

Information which is not stored locally may be obtained by accessing the global X.500 directory service [10]. This assumes that the directory information model will be extended to accommodate this. The browser will then contact the local Directory Service Agent (DSA) to request information that it cannot find in its local database. This feature is not yet implemented but is planned for the future. The browser will then connect to the local DSA upon initialisation in order to be able to pass requests for information later. These will take place asynchronously, without affecting its normal operation. Any information obtained from the directory will be added to the local database, to be available in future invocations of the browser.

A generic browser based on the above principles could be implemented using any programming paradigm. Our implementation is based on the object oriented one as both the OSI Management Model and Graphical User Interfaces exhibit a similar nature. This enhanced the software structure and enabled reusability of many generic components. It made it particularly easy to separate the browser's application engine from the display manager so that in the future it would be possible to adapt it to different user interface technology.

The programming language used for the implementation was C++ [11] and we have used the excellent InterViews graphical object library [12] based on the X-Window system. The underlying OSI stack was provided by the ISODE software [13] [14], enabling the browser to run both on pure OSI network technology and TCP/IP using the RFC1006 method. The CMIS/P management protocol was as provided by the OSIMIS package [15], according to the latest version 2 international standard. The browser is now an integral part of OSIMIS.

5. Conclusions

The MIB browser is a very useful application and its generic nature makes it a vital part of a management platform as it can be used to browse through existing or future MIBs of diverse nature. It is a particularly useful tool in a management environment, allowing the human manager to take a microscopic view of the operation of real resources through the corresponding managed objects. We have also found it a very useful debugging aid tool during the development of management information bases.

In the future, the browser will be enhanced to cope with some of the limitations outlined in this paper. The X.500 Directory Service will be used in addition to the local database. There is also much scope for improving the event handling in the browser. Finally, we intend to provide a more macroscopic view of the MIB structure using a graphical representation.

The browser has actually been used in three completely different environments: in the RACE NEMESYS project (NEtwork Management using Expert SYStems) as part of a prototype for traffic and quality of service management in Integrated Broadband Communications [16], in the ESPRIT PROOF project (Primary Rate OSI Office Facilities) to manage gateways between IP/X.25 and Primary Rate ISDN [17] and in the OSIMIS package where it is used to manage a MIB related to the ISO Transport Protocol [15]. In all these applications the same browser has been used without the need to modify a single line of code!

Figure 2. The Browser in Operation

Acknowledgements

The work described in this paper was accomplished under the NEMESYS research project (NEtwork Management using Expert SYStems) as part of the RACE research programme (Research in Advanced Communications in Europe).

REFERENCES

[1] ISO/IS 10040, *Information Technology - Open Systems Interconnection - Systems Management Overview*, August 1991

[2] ISO/IS 10165-4, *Information Technology - Structure of Management Information - Part 4: Guidelines for the Definition of Managed Objects*, August 1991

[3] ISO/IS 9595, *Information Technology - Open Systems Interconnection - Common Management Information Service Definition, Version 2*, July 1991

[4] ISO/IS 9595, *Information Technology - Open Systems Interconnection - Common Management Information Protocol Specification, Version 2*, July 1991

[5] ISO/IS 10165-4, *Information Technology - Structure of Management Information - Part 1: Management Information Model*, August 1991

[6] ISO/IS 10164-5, *Information Technology - Open Systems Interconnection - Systems Management - Part 5: Event Report Management Function*, August 1991

[7] ISO/IS 10164-6, *Information Technology - Open Systems Interconnection - Systems Management - Part 6: Log Control Function*, August 1991

[8] ISO/IS 8824, *Information Processing - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1)*, 1987

[9] ISO/IS 8825, *Information Processing - Open Systems Interconnection - Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)*, 1987

[10] ISO/IS 9594-1, *Information Processing - Open Systems Interconnection - The Directory: Overview of Concepts, Models and Service*, 1988

[11] Bjarne Stroustrup, *The C++ Programming Language*, Addison-Wesley, Reading, MA, 1986

[12] M.A.Linton, P.R.Calder, J.M.Vlissides, *InterViews: A C++ Graphical Interface Toolkit*, Technical Report CSL-TR-88-358, Stanford University, July 1988

[13] M.T.Rose, J.P.Onions, C.J.Robbins, *The ISO Development Environment User's Manual Version 7.0*, PSI Inc / X-Tel Services Ltd, July 1991

[14] M.T.Rose, *The Open Book - A Practical Perspective on OSI*, Prentice Hall, New Jersey, 1990

[15] G.Pavlou, G.Knight, S.Walton, *Experience of Implementing OSI Management Facilities*, Proceedings of the IFIP Second International Symposium on Integrated Network Management, Washington, April 1991

[16] A. Mann, G.Pavlou, *Quality of Service Management in IBC: an OSI Management Based Prototype*, Proceedings of the Fifth RACE Telecommunications Management Network Conference, London, November 1991

[17] S. Walton, G.Knight, R.Carbonell, M.Hardie, *The Construction of a Connectionless Ethernet/ISDN Gateway*, Microprocessors and Microsystems, Vol. 15, No. 1, Jan-Feb 1991