Modelling OSI Management Information Base
With Object Oriented Analysis

E. G. KOTSAKIS, AND B. H. PARDOE, FELLOW, IEEE

Electrical & Electronic Engineering Dept.
University Of Salford
Salford M5 4WT
UK.
E-mail: e.kotsakis@eee.salford.ac.uk

Abstract

The Management Information Base (MIB) is the conceptual repository containing information about the resources and elements to be managed in an OSI-based management system. OSI systems management is based on the concept of object oriented design. This type of system may include performance management, accounting management, security management configuration management, and fault management. The analysis of such a system reveals data complexity associated with both the behaviour and the operation of the MIB.

This Paper highlights issues involved in MIB modelling and definition of managed objects. Aside the object oriented modelling framework an important consideration is addressed to the Guidelines for the Definition of Managed Objects (GDMO). The GDMO provides a way for bridging the gap between the Management Information Model and the concrete specification of a management object class.

Introduction

The MIB is a conceptual scheme which embodies relationships between objects and determines the operation which may be performed on them. The MIB encompasses an abstract model and an implementation model of management information. The abstract model [1] among other defines

- Principles of naming objects
- The logical structure of management information
- Concepts related with management object classes and the relationship between them.

The implementation model [2] defines

- The platform for hosting a MIB.
- The architectural principles for partitioning MIB information
- Database type (Object oriented or relational)
- Translation of MIB object model into a schema

Why Object Orientation?

The applications in the area of networking require different data manipulation. This is due to differences in the characteristics of the application domain. Not only does the stored data has different characteristics, but there are also noticeable differences in the kind of relationships between data, user access patterns to the data, and variety of users

---

1 The term user includes any software application or hardware device that may use data stored in a management system.
who access the data. The aim of designing a database is to produce a representation that resemble the real world as much as possible. Therefore the database must provide mechanisms for ensuring that a representation of the real world is feasible. This implies that facilities must be provided to model the different kinds of entity found in a real system and the complexity of the relationships defined between entities. The easy manipulation of different entities implies the integration of structuring facilities with the methods that act to the primitive structures. The object oriented (OO) paradigm may be seen as a way to implement this integration, providing mechanisms for processing behavioural models. The integration is accomplished by incorporating both operations and attributes in a single module called object.

An other paradigm that may be used for modelling a MIB is the Entity Relationship (ER) paradigm [5]. Part of the OO paradigm could be modelled using ER models. However the ER notion of paradigm is not suitable for modelling OO concepts since the former introduces the idea of generic relationship between entities and it is lacking in power to model objects. Therefore it seems that the OO modelling is more suitable than any other approach for data representation.

**MIB Information Model**

The information model defines the logical structure of managed objects. Managed objects are abstract representations of communication resources (e.g. routers, PBX, protocol state machines, modems, etc.). An object is described by using four kinds of characteristics.

- **Attributes**, that denote specific characteristics of the resource.
- **Operations**, that are performed on a set of attributes
- **Behaviour**, that specifies how the object reacts to operations performed on it.
- **Notifications**, that may be emitted to the managing station through a protocol as a reaction in an external event or as a frequently action.

The characteristics above are those that are visible to managing entity at the managed object boundary. The managed object may be viewed as a opaque sphere surrounding the real resource with a window through which management information may pass [3]. The internal functioning of the resource represented by the managed object is not visible to managing station.

The **managed object class** provides a way to specify a family of managed object. A managed object class is a template for managed objects that share the same attributes, operations, notifications and behaviour. A managed object is an instantiation of the managed object class.

**Management Operations**

The Management Information Model [1] defines two types of management operations

- operations intended to be applied to the object attributes
- operations intended to be applied to the management object as a whole.

Attribute oriented operations are as follows

- get attribute value
- replace attribute value
- replace with default value
- add member
- remove member

Any operation may request to be performed on a list of attributes. The operations may also be performed atomically (either all operations succeed or none is performed). Operations that may be applied to the managed object as a whole are the following

- create
- delete
- action

An action operation requests the managed object to perform the specified action and to indicate the result of this action.

**MIB Object Orientation**

The fundamental idea behind the modelling of a MIB is concentrated in the following concepts.

- Allomorphism
- Encapsulation
- Specialisation & Inheritance
- Containment

**Allomorphism** is a property of a managed object in which the object of a specific class may be managed as if it were an instance of another object class. Allomorphism refers to the ability of the instance of a subclass to resemble the behaviour of an instance of the superclass. In this case the superclass and
subclass are said to be the allomorphic superclass and the allomorphic subclass respectively.

The essence of encapsulation is that the operations form the interface through which a user of the managed object can access the attributes and retrieves or changes the values of some of them. Hence the user of the managed object has no way to access those attributes other than through the defined set of operations. This is very like to a “black box” approach, where an object provides a fixed set of services to another object. Therefore the purpose of the encapsulation is to define for each complex managed object not only the structure type, but also the operations through which a user can access and manipulate the managed object.

Specialisation is used to hide the similarities among managed object classes and to emphasise their difference. The terms of specialisation and generalisation are used some times to express the same concept. Specialisation refers to the derived or lower-level class (subclass) and generalisation refers to the base or higher-level class (superclass). Each subclass is an extension of the superclass since inherits some or all of its characteristics and includes some additional characteristics such as new attributes, new management operations, new notifications and new behaviour. The higher-level class emphasises the commonality between managed object classes. The specialisation may be viewed as a containment relationship that exists between the higher-level class and one or more lower-level classes. The subclass can modify (override) some of the definitions of the superclass in order to satisfy its requirements. The ability of a subclass to inherit the attributes, notifications, operations and behaviour from more than one superclass is called multiple inheritance. An example of an inheritance hierarchy is shown in Figure 1.

Containment is a relationship between managed object instances (not classes) in which an object instance may contain one or more other object instances. Containment relationship is used to model real world hierarchies and it is different from inheritance relationship as follows:

The subordinate (contained) object instance does not inherit any property from the superior (containing) object instance. This means that any property of the superior object cannot be accessed from within a subordinate object. In the other hand we may access the properties of a subordinate object from the superior object. This is exactly the opposite of that in the superclass-subclass relationship in which superclass properties may be accessed from within the subclass and subclass properties are not allowed to be accessed from within the superclass. In the superclass-subclass relationship, each object instance of either the superclass or the subclass exists independently. Unlike this, in the superior-subordinate relationship, the existence of the subordinate object instance depends on the existence of its superior. The subordinate managed object instance may be contained in only one superior object instance.

**Definition of Managed Objects**

The Management Information Model describes the conceptual framework that is used for the purpose of OSI management. However, this model in isolation is insufficient to allow the development of managed object definition. The GDMO provides the link between conceptual scheme and managed object definition [7]. GDMO is concerned with the language that definers are required to use to document their managed object class definitions. The notation used for defining object classes is based on templates. A template is a standard form for the specification of a managed object class. It defines the type of the class, the syntax, and the order of its component. Each template is uniquely identified by a template-label. Whenever a such a label appears in the body of a template definition, it can be replaced by the entire text of the referenced template.

A managed object class may be defined by the following elements:

![Figure 1. Inheritance hierarchy](image-url)
- Attributes and their value ranges
- Attribute oriented operations
- Managed object oriented operations
- Notifications
- Behaviour definition
- Attribute groups
- Packages
- Naming

An attribute group is a means through which a reference to a collection of attributes within a managed object is feasible. An attribute group may be fixed (its attributes may not change) or extensible (attributes may be added as a result of specialisation).

A package is an integral module of a managed object class definition consisting of a collection of characteristics (attributes, operations, notifications, behaviour). The same characteristic may be present in more than one package. Packages may be mandatory (present in all instances of a given class) or conditional (present in a managed object for which a associated condition is true).

The specifications of the elements above implies nine templates as follows:
- Managed object class
- Package
- Parameter
- Attribute
- Attribute group
- Behaviour
- Action
- Notification
- Name binding

The core template of the object class definition is the MANAGED OBJECT CLASS. This template is depicted in Figure 2. Elements within dashed boxes may be omitted. Multiple boxes depict that their elements may appear one or more times. For example in the CHARACTERIZED BY construct one or more packages may appear. The <class_label> is a string that serves as a textual identifier of the managed object class. The DERIVED FROM construct indicates the superclasses from which the current managed object class inherits their characteristics. The CHARACTERIZED BY construct lists any mandatory packages that are to be included in all instances of the managed object class. The CONDITIONAL PACKAGE construct lists any conditional packages, that is any packages whose inclusion in an instance of the class depends on a condition which is evaluated at instantiation time. The REGISTER AS construct specifies a unique global identifier used to identify the management object class. Inheritance and packages provide the means for the definition of managed objects. Both inheritance and packages provide a natural mechanism of reusing existing specifications.

The notification template is shown in Figure 3. The PARAMETERS construct identifies event information. The WITH INFORMATION SYNTAX and the WITH

---

**Figure 2. The managed object class template**

**Figure 3. The notification template**
REPLY SYNTAX constructs identify the ASN.1 [8] data type that is to be carried by the Common Management Information Protocol (CMIP) [15].

The package template is shown in Figure 4. The BEHAVIOUR construct is a textual description of the behaviour of the managed object. The ATTRIBUTES, ACTIONS, and NOTIFICATIONS constructs list all the attributes, actions, and notifications of this package. The propertylist specifies some properties about the value of an attribute.

The attribute template is shown in Figure 5. The attribute template is used to define the syntax of an attribute type. The MATCHES FOR construct permits matching rules to be applied to an attribute value. The PARAMETERS construct permits parameters to be included in the definition. These parameters may be used by the CMIP for error checking. The DERIVED FROM construct indicates from where this attribute is derived. If the attribute is not derived from any other, the WITH ATTRIBUTE SYNTAX construct is used to define the ASN.1 data type.

The behaviour template, shown in Figure 6, is used to define a human readable text describing the behaviour of the managed object. It contains just one construct (DEFINED AS) which specifies the behaviour by a readable string.

Example of a Managed Object Class Definition

The following example illustrates how a managed object class may be built by using templates. Figure 7 shows the structure of the eventForwardingDiscriminator managed object class and the relationships between
the component templates. The definition of the eventForwardingDiscriminator is derived from the superclass discriminator [14] which is used to define the criteria for controlling management services (event reports etc.). A eventForwardingDiscriminator is a managed object that allows the control of the event reports generated from other objects.

The eventForwardingDiscriminator includes the mandatory package efdpackage which is characterised by the destination attribute, which indicates the primary address of an application entity to which the discriminator forwards events. The managed object class also conditionally includes the backUpDestinationListPackage, which is characterised by the backUpDestinationList attribute, which indicates a list of application entities that may be used as an event destination if the primary address fails. The modePackage conditional package defines the mode (confirmed or nonconfirmed) for reporting events. The eventForwardingDiscriminatorBehaviour is used to define a string describing the behaviour of an instance of the managed object class eventForwardingDiscriminator.

**Conclusion**

As the networks become more complex and integrate new technologies the management design becomes more essential. Structural models based on object oriented paradigm are used to provide a global approach in network management modelling. The object oriented approach of a management system provides a natural way to model the behaviour of the managed objects. The model obtained by using object oriented analysis will be a more direct representation of the management task, providing a better framework for understanding and manipulating the complex relationships between managed objects. The object oriented formulation allows new managed object classes to be defined as an extension of the management system and it also provides mechanisms such as encapsulation, allomorphism, containment, and inheritance that approach the conceptual requirements of a real management system more naturally.
References


