Object-Oriented Specifications in OSI and Distributed Processing[1]

M. Barbeau (Université de Sherbrooke),

G.v. Bochmann (Université de Montréal),

and J.M. Serre (Bell-Northern-Research)

DRAFT (Oct. 7 1991)

1. Introduction

Networks are becoming widely used with new applications appearing one after another. Today, information transfer, and its supporting telecommunication infrastructure is the hearth of any large organization. Yet, the majority of this work is still based on simple client/server architectures. As network size increase by orders of magnitude, simple peer-to-peer communication models, must be replaced by true distributed applications. For example, a simple translation table mechanism sufficient to serve an E-MAIL application on a LAN, may be unsuitable for an E-MAIL international service. Such services requires a distributed and global naming scheme, enabling to find users in different countries.

This paper's objective is to examine the field of distributed processing system specifications, in the context of Open System Interconnection (OSI). Many distributed applications are presently developed in the application layer of the OSI reference model. Two of them are briefly described to give examples of the type of information that needs to be specified for such systems.

Emphasis is put on the Object Oriented paradigm, because this approach seems particularly suited to distributed systems. It enables to define self-contained pieces of data and associated behavior, called objects, which form building blocks for describing distributed components.

Layout of the paper:

[1] Work supported in part by a CRIM-BNR research project and the IDACOM-NSERC-CWARC Instustrial Research Chair on Communications Protocols at the Université de Montréal.

- Section 2 looks at object oriented specification, and introduces concepts that will be used in the rest of the paper (such as subclassing, encapsulation, etc.)
- Section 3 presents an high level design methodology to model systems according to Object-Oriented Paradigms
- Section 4 outlines 2 OSI applications: The X.500 Directory, and the network management standard. The management framework is then use to show an application of the methodology described in Section 3.
- Section 5 examines different formal and semi-formal description technique, and their applicability to distributed systems, and to support Object Oriented concepts.
- Section 6 concludes this paper.

2. What are Object-Oriented specifications

Formal specification techniques have been used for a long time in the protocol specification community [Boch **]. The benefits of using FDTs are:

- formal, unambiguous and readable descriptions of the algorithms,

- verification using several complementary proof methods,

- compilation into real world implementations, and

- consistency checking of the implementations with respect to the formal descriptions using systematic test methods.

In this paper, we are interested in object-oriented techniques. Much literature exists about the advantages of the object-oriented approach [Meye 88]. We believe that the following characteristics of the object-orientation are of particular importance:

The concept of object

In the object oriented approach, the application domain is conceptualized as a set of dynamic entities called object instances. This concept of object leads to software that are easier to understand because objects naturally corresponds to the concepts of the application domain. This set of object instances is structured into a set of object classes. An object class groups objects of the same kind or with similar structure.

Inheritance of properties

Classes of objects are organized into a class hierarchy, or inheritance structure, which shows how every class is related to the others as a specialization or/and generalization of one or more classes. In software engineering, this inheritance structure makes possible reutilisation and extension of descriptions of classes of objects.

Object instances are normally related to one another through different kinds of relationships. Extended Entity-Relationship diagrams can be used to show the classes of objects, the classes of relationships and the inheritance structure.

Encapsulation

At a lower level of abstraction every object may be seen as a structure of components objects. This decomposition leads to a second type of hierarchy called the class composition hierarchy. Every object instance normally offers a fixed set of services to the other objects. Services are modeled as operations on objects. Operations are defined such that every object encapsulates all the operations required for acting on its internal structure. Using objects as building blocks for specifications allows hinding of aspects related to the implementation of services and the structure of an object providing therefore information hinding [Parn 72].

3. Specification Methodology

It is well known that methodologies are required, or at least useful, to guide the designer of an application. Traditional methodologies are either function-oriented [Ward 85] or data structure design oriented [Chen 76]. Object-oriented methodologies integrate both ``orientations''. The basic concept is the one of object. A software application is developed following the principle of aggregating entities together with the operations that can be performed on them. Functions are syntactically grouped at the level of their domain object-type. Entity-relationship diagrams from the field of databases are used to represent the structure of the objects.

The object-oriented methodology used for this experience has originally been proposed in [Mond 90]. It consists of a preliminary analysis step and three modeling steps.

Preliminary Step: Problem Definition

The preliminary step consists of analyzing the requirements and identifying the aspects that need to be handled.

Step 1: Domain Definition

The result of the first step is a conceptual model of the structure of the domain. The conceptual model is described by an entity-relationship diagram which shows the relevant classes of entities together with the relevant classes of relationships among the entities. Entity-relationship diagrams can be translated in a structure of objects. Classes of entities become object types and classes of relationships become either object types or attribute definitions within object types.

Step 2: Identification of Functions

In conformity with the object-orientation, functions, processes or transformations are defined as operations on objects. The second step is concerned with the identification of functions and the allocation of operations to objects. They are allocated such that every object encapsulates all the operations required to modify its internal structure, including its state variables and behavior.

Step 3: Definition of Behaviors

At Step 3, we define the behavior of the objects. At this stage we associate a meaning to the operations. We may also define in which order the operations will be performed.

The design process is iterative. During the development of an application, it is normally necessary to go back and resume a previous step.

Specification languages can be used right from Step 1. The conceptual model becomes clearly defined. Moreover, a certain degree of consistency can be insured by using a compiler that checks the syntax and the static semantics properties of the preliminary design.

Object-oriented software development is both top-down and bottom up. On the one hand, identification of top level objects and their refinement into component objects is a top-down activity, on the other hand, reutilisation of old object definitions in a new design is a bottom-up process.

For example, let us consider the object-oriented design of the database of a small manufacturing company. The problem is defined as follows: Design a database for managing the inventory of the company. The inventory is divided into two categories of parts, namely, base parts and composite parts. The former are imported whereas the latter are manufactured within the company.

A logical model of the database, developed during Step 1, is represented as the Entity-Relationship diagram pictured in Fig. 1. The classes of objects "BasePart" and "CompositePart" inherit from the class "Part". The "UsedIn" relationship shows how a part, either base or composite, can enter in the composition of a composite part. Suppliers of base parts are also represented.

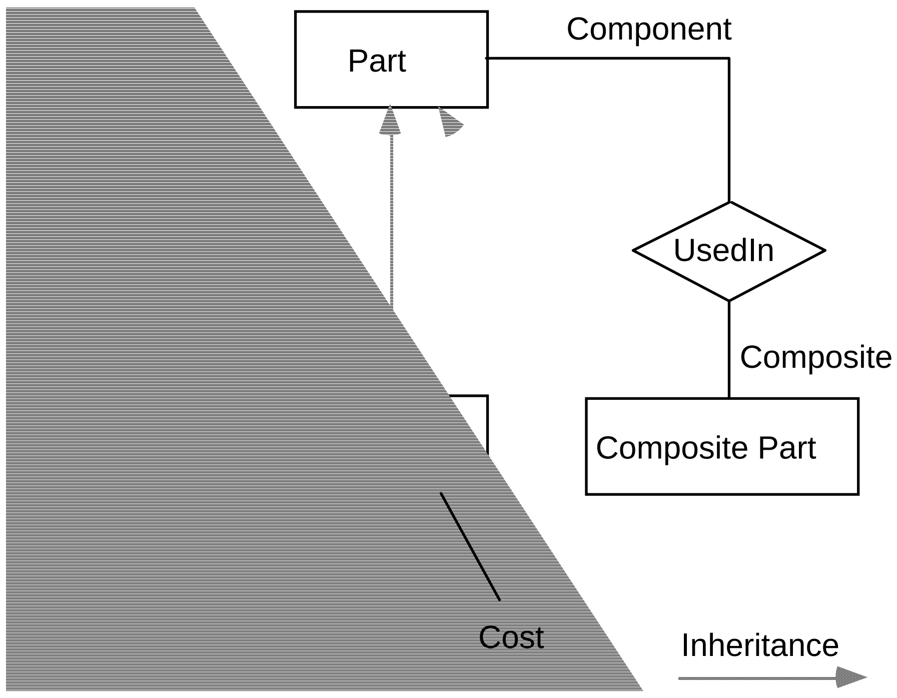


Fig. 2 The Entity-Relationship diagram

Among the required functions, is the capability to compute the total cost of a composite part. In Step 2, we therefore allocate the operation "compute cost" to the class "CompositePart". The actual definition of the operation "compute cost" is developed during Step 3. The cost of every base part is stored in the attribute "cost". For computing the cost, the operation recursively searches through the "UsedIn" relationships until base parts are reached.

4. Application in OSI and distributed processing

This section examines two OSI distributed applications namely, the Directory (X.500) and management of large telecommunication network. The OSI network management framework will also be used to present a small example of an application of the methodology to alarm surveillance.

4.1 The Directory Services and Concepts

The first version of the X.500 series of standards [x1] has been published in 1988. A 1992 version (extending the original) is also in the work. The goal of this standard is to define a global Directory (supporting unique naming) which enables users (processes or humans) to find information about other users, organizations and other entities, etc. The X.500 designers used a top-down approach and started by defining what is the Directory system service (i.e. the external view)

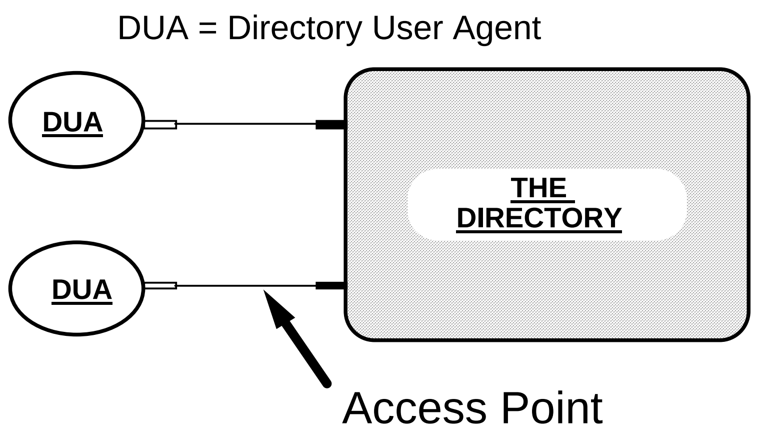


Figure 4.1 User view of the Directory

As shown in Figure 4.1, a user accesses the Directory through an application process called a Directory User Agent (DUA). A user wanting to query the system must first get connected through the bind/unbind services. This step may involve access control and authentification procedures. The user can then use the service read, compare, list and search to find the desired information. From the user point of view, the Directory looks like a large database which contains information. The services are used to look through the *entries* contained in the Directory to find the wanted data. The access to specific entries may also be subject to access control limitations.

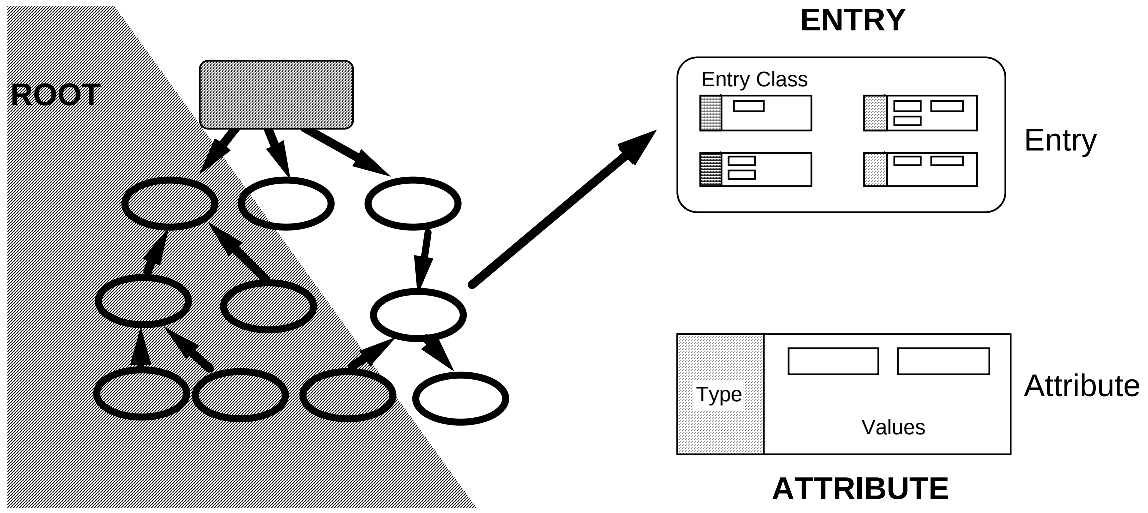


Figure 4.2 Information held in the Directory

As shown by figure 4.2 each entry is of a specific class, and comprises a certain number of (mandatory and optional) attributes. Each attribute is also of a certain type which contains values of a defined syntax. One example of the classes defined in X.500 is the class "organization" which contains the mandatory attribute organizationName, and may contain information such as : business category, postalAddress, description, telephone number, FAX number, E-MAIL address. Recognizing the need for many different types of entries, the standard contains an information model (based of object oriented principles) which defines techniques to structure and represent the information that needs to be stored in the Directory. X.500 uses object oriented techniques to enable users and administrators to define additional classes/attributes. X.500 is thus extensible and may encompass any desired information. These entries are organized into a tree structure called the Directory Information Tree (DIT). Figure 4.3 presents an example hypothetical DIT.

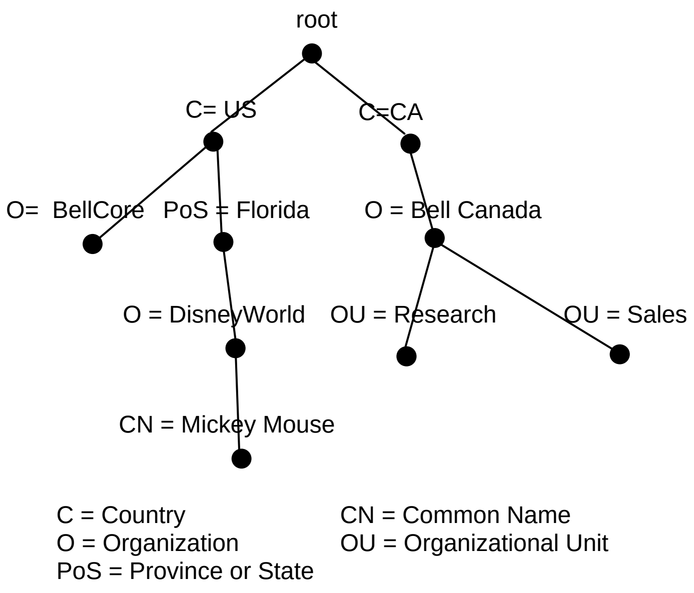


Figure 4.3 Distinguished Names

The tree hierarchy is used to order entries based on natural relationships that exist between them, such as a person working for a company, which is in a StateOrProvince which is in a country. Because of this structure, the name of a single entry is made of a sequence of names, called a Distinguished Name (DN), starting from the root and migrating downard the desired entry. For example, in Figure 4.3 the DN of the Sales unit of Bell Canada is < <C=CA>, <O=Bell Canada>, <O=Sales> >. Under the same superior, entries need to be distinguished by separate names. This last part of the DN is called the Relative Distinguished Name (RDN). This name must be unique relative to the same superior. For example only one entry with <CN= "Mickey Mouse"> can exists under the entry < <C=US>, <PoS= "Florida">, <O = DisneyWorld>>. Applying this principle recursively up to the entries under the root, insures a unique name for each entry[2]. This also enables the fragmentation of the naming space into sub-spaces which can be administered separately (e.g. an entry Mickey Mouse could also exist under < <PoS= California> , <O= DisneyLand> >).

The next part of the Directory standard is the distributed framework. Although the Directory looks like a unified database from the outside, it is not practical (for technical and political reasons) to assume that all that information can be stored into a single system. The distributed model, illustrated in Figure 4.4, introduces the notion of Directory System Agents (DSAs). Each DSA holds a small part of the Directory. The sets of DSAs cooperate together to give the illusion of a global system.

[2]The X.500 series also includes the notion of alias entry. When more than one name is desirable for the same entry an alias can be used. This soften the restrictions imposed by a rigid tree structure.

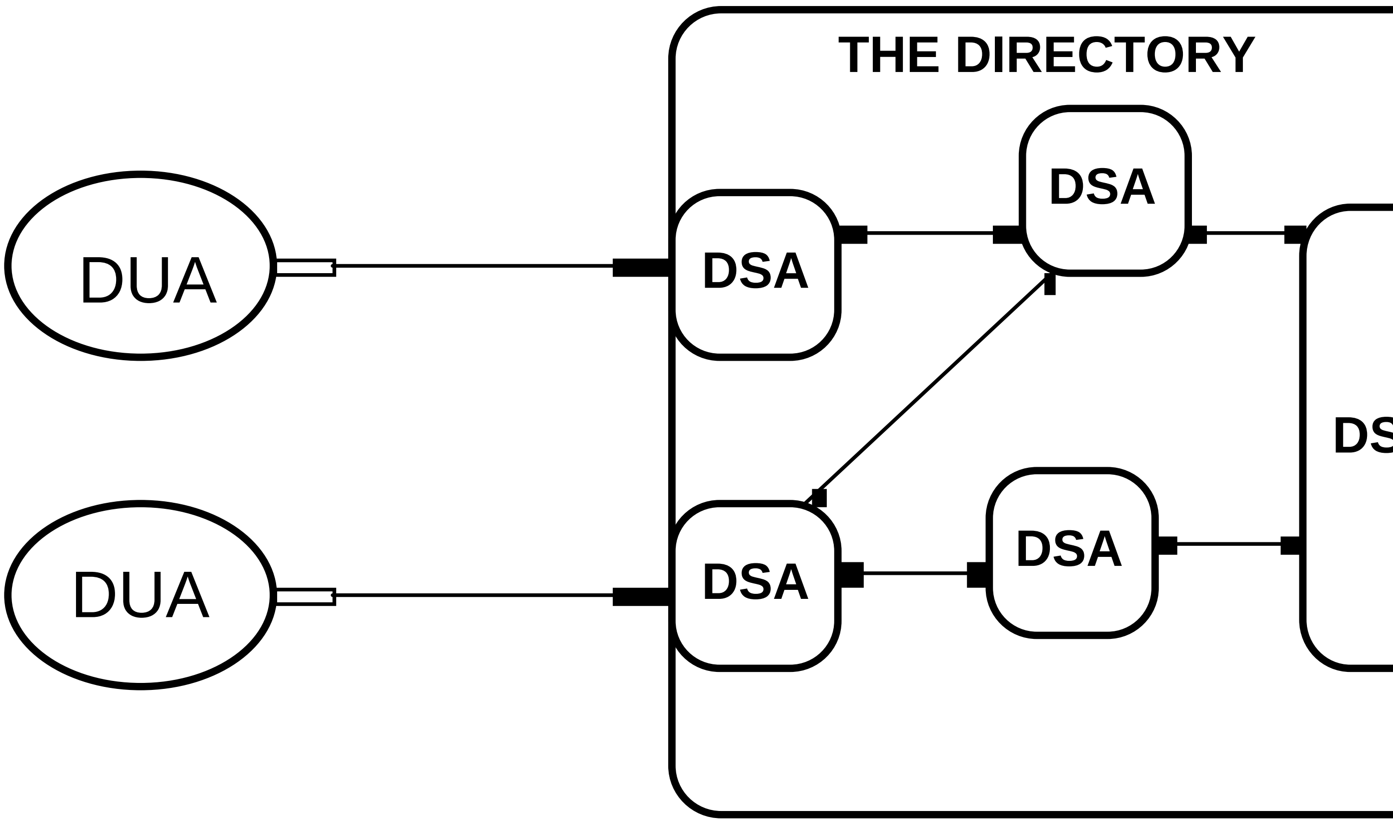


Figure 4.4 Distributed View of the Directory

X.500 defines a Directory System Protocol (DSP) used between the DSA to exchange information. For example a DSA unable to directly answer a query (because it does not hold the desired entry) may "chain" the request to another DSA more susceptible to hold the entry. To perform this delegation, the originator DSA will use its "knowledge" about other DSAs. Minimal knowledge rules for DSA are described in the standard, as well as mechanisms to insure that this knowledge can be updated and that "wrong" (or outdated) knowledge can be corrected.

The Directory has a "static" view of the entries it holds, which is not the case for our next example application: Network Management. But one of the important question in Network management is to be able to identify these object by a global name. It will be made clear in the next section that the X.500 framework was used in network management.

4.2 Network Management

Network management is our second example of an OSI area where distributed applications and the Object Oriented paradigm have been used. In this paper the term "network management" is meant in it's larger sense of managing a complete network, typically owned by a telecommunication provider. In telephony, network operation is often designated by the expression OAM&P (Operation Administration Maintenance & Provisioning). It is a complex field due to the intertwining of different functions, such as: data transmission, configuration management, error recovery, service optimization, alarm detection, alarm correlation, accounting, security and many others.

Due to the geographical distribution of networks, management application must be distributed by definition. The next figure (extracted from North America/CCITT standards on network management) present a summary of the intended TMN (Telecommunication Management Network) architecture.

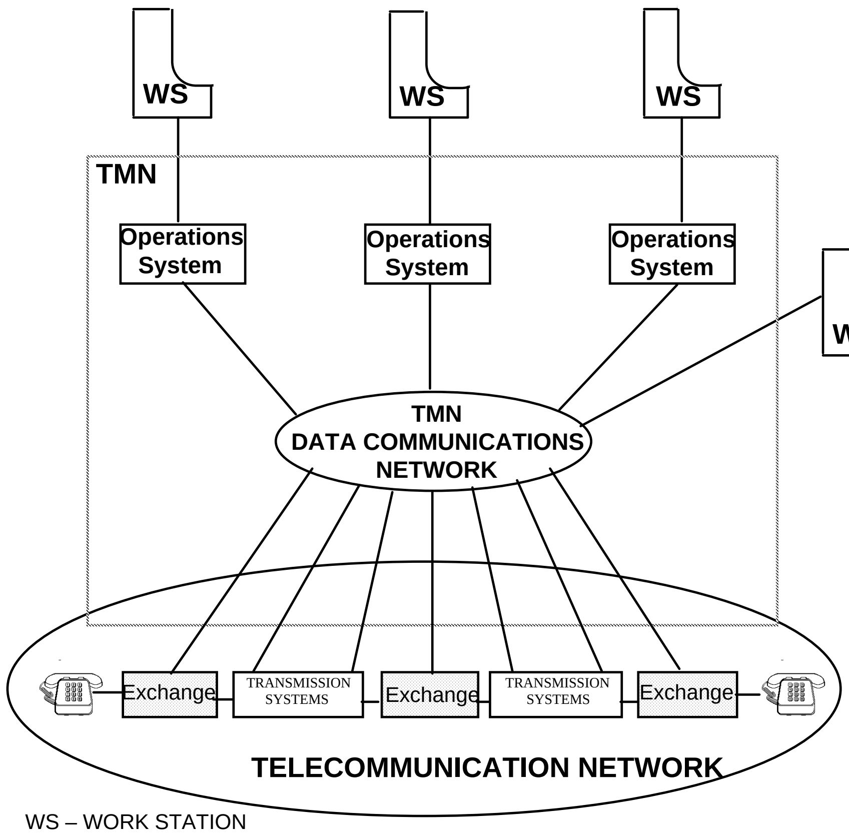


Figure 4.5 Proposed TMN Architecture

As shown by Figure 4.5, it is proposed that the many "Network Elements" (NEs) will communicate through a TMN to various Operation Systems (OSs) which are computerized systems used to manage the network. NEs may communicate management information to one another, and OSs can also inter-communicate and/or be organized in hierarchies. An additional level of complexity in network management, is the fact that these NEs are very different in terms of functionality provided (from a simple repeater to a complex digital switch supporting thousand of communication channels). These NEs are also made by a variety of manufacturers.

For all these reasons, a model of the network used as a base for specifying communication interfaces between various network components must be able to achieve at the same time, a high level of abstraction (for generic management), and be powerful enough to express specific information precisely when detailed information is needed. For example, an application setting up a dedicated "voice" circuit across a number of NEs may not be interested into the detail working of each individual NEs. On the other hand, an application analyzing some alarm report coming from a specific NE may need internal details of that NE to be able to correlate the problem to a specific hardware component.

For these reasons, the Object Oriented approach seems to be talor-made for network management. The network is made of cooperating entities which can be modelled as objects, which can be refined into various level of details. depending of the view needed. Complex NEs can be decomposed into various component objects (e.g. line card, line termination, power supply, etc.), while at the same time higher level "functions" (e.g. service provisioning option such as call forwarding, "dial-in" circuit, etc.) may be represented by higher level objects.

Inheritance is also a powerful tool. A generic concept can be defined as an object class such as a logical "termination point" (TP) where a signal is originated or terminated. This concept can be refined into a point where a modulated signal is terminated, e.g. a line termination (LT). This can be refined further into a point where a specific carrier signal called T1 is terminated: a " T1"LT". Figure 4.6 illustrates this process.

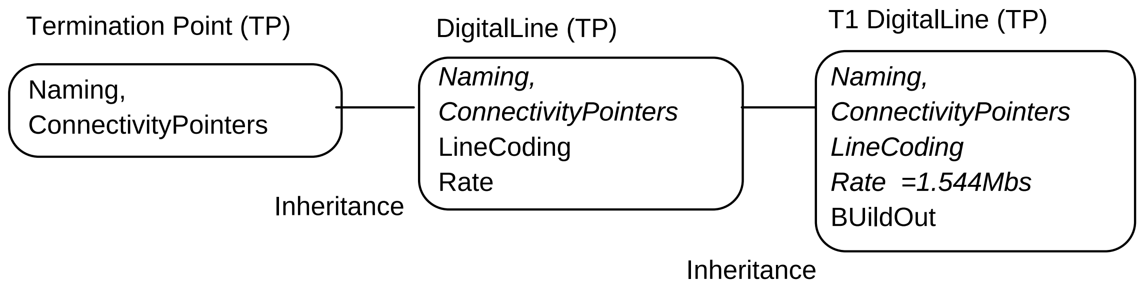


Figure 4.6 Small Inheritance example

Inheritance can be seen as having advantages both from a top/down and a bottom/up design. From a top/down point of view, it permits to obtain high level object classes that can be instantiated in OSs which manages at a generic level. From a bottom/up point of view it permits to define specific classes which express the details of systems at a level useful for full management operations. These specific classes are derived from the generic ones, which maximizes commonality across them.

Because of these advantages, all the bodies involved in network management standardization are now following this approach. ISO SC21/WG4 created the OO framework now used in OSI for network management. The Common Management Information Service (CMIS) and protocol (CMIP) is the "ASE" (Application Service

Element) created by WG4 to transport and access management information. Figure 4.7 illustrates the approach used in ISO system Management.

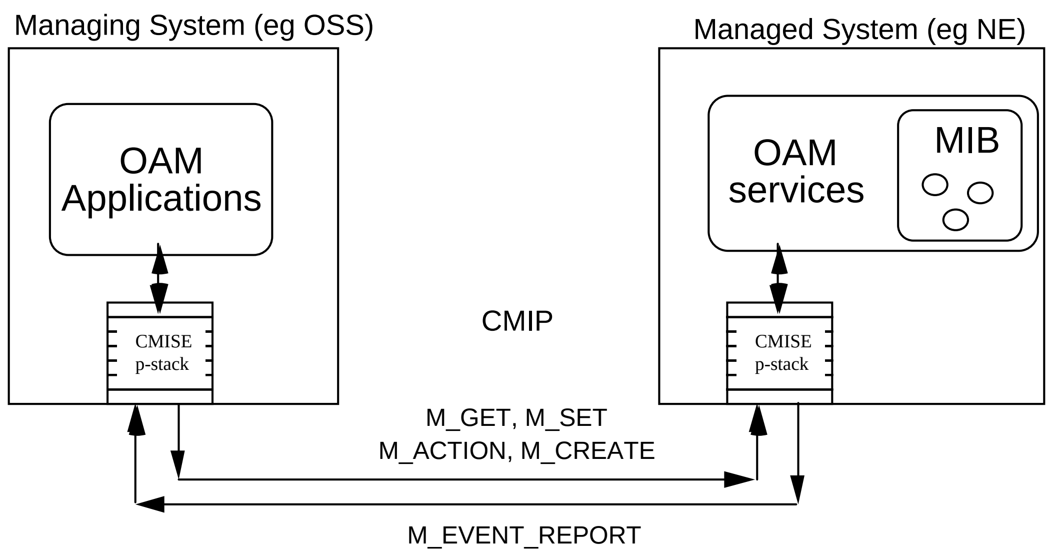


Figure 4.7 CMIS and System Management

CMIS/CMIP is better understood by looking at the system view. CMIS assumes the existence of a Management Information Base containing an object representation of the managed system. All CMIS messages apply to one or more object (instances) contained in the MIB. Example of CMIS services are M_GET to get attribute values of one or more object (read), M_SET to change values. CMIS also includes an asynchronous service M_EVENT_REPORT which is used to signal events happening in the managed system (such as alarms, automated performance report, etc.). Another important factor is that CMIS assumes that the MIB is structured as a tree similar to the Directory approach (see section 4.1).

Assuming the ISO framework, and the use of CMIS/CMIP for network management, the next step is to decide what objects will reside in the MIB for a given system. For the management of OSI protocol stacks, this work is done by WG4 itself. For the telecom network at large, this work is done by standard bodies grouping the telecommunication carriers and equipment vendors. One such body is the committee T1M1.5 (part of ANSI accredited committee T1) which is responsible for developing OAM&P interfaces for the TMN in North America[3]. T1M1 produced a Generic Network Model (GNM) standard [x2] and a fault management [x3].

One author of this paper having been heavily involved into the development of these T1 standards, it is no coïncidence that the methodology proposed in this paper has also been reflected into these documents.

[3] The work of T1M1 is submitted at the international level to CCITT SG IV committee which is working on a M.gnm recommendation for 1992. The work of T1M1 and SG IV are used by technology specific groups to create subclasses. For example, a model of SONET/SDH OAM&P is presently developed.

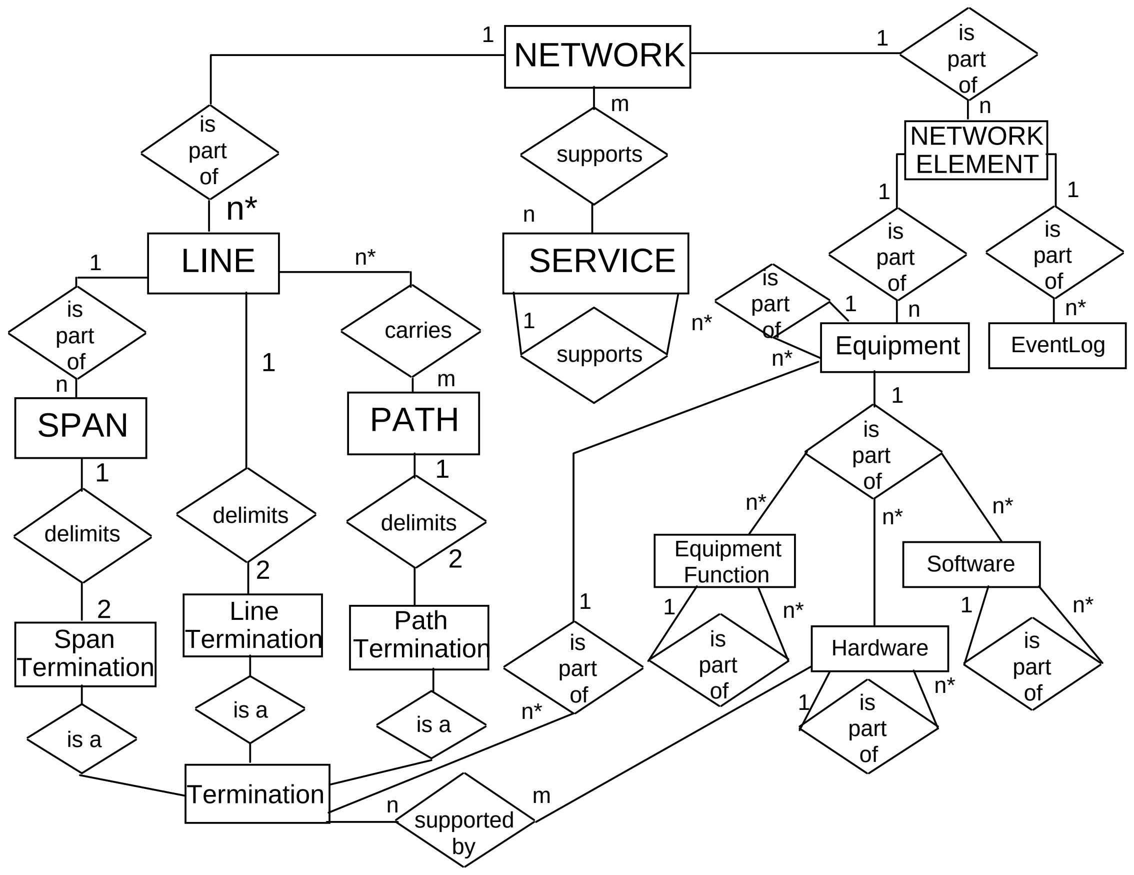


Figure 4.8 Partial E-R Diagram

The T1M1 GNM contains an E-R depiction of the proposed model. During model development, E-R diagrams were used to document an high level view, and to keep track of the needed relationships between object classes. Figure 4.8 presents an extract from the more complete figure contained in the standard. One example of the use of such diagrams is that T1M1 designers were keen on separating a logical view of network equipment (common functionality) from an equipment view which is different from one manufacturer to another. This explains that on this diagram, the function of terminating a line (line termination), is not directly embedded/linked to the hardware which supports it. This relationship is illustrated by the supportedBy diamond between termination and hardware. In a specific equipment, the function of terminating a T1 line may be implemented by 2 circuitPacks, in another, two T1 line terminations may be implemented in one hardware component.

The question of creating a naming tree (as in Section 4.1), is also solved by using existence dependent relationships. For example the simple naming tree of Figure 4.9 can be derived by using the is-part-of (containment) relationships between lineTermination Equipment, NE, and Network. Line can be named by network, but it can also be named by its two end-points (in traditional telephony a dedicated circuit is often named by its two end-points).

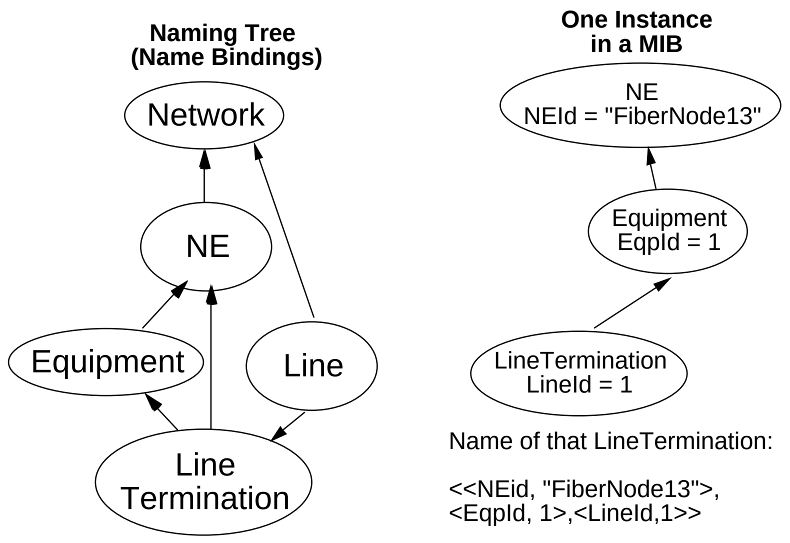


Figure 4.9 Naming Tree example

4.3. Application of the methodology

Using the framework of network management, this section will present a small application of the methodology on the subject of Alarm Reporting.

Preliminary Step: Problem definition:

The objective is to produce a specification covering alarm surveillance between a small network element and a managing system (OS). This specification must include asynchronous alarm reports, log of alarms (history) and filtering of alarms (inside the NE).

Step 1: Domain definition:

Alarm maybe detected/emitted by objects such as those listed in Figure 4.9. But additional objects are required to hold the alarm history, and to filter the alarms going to the managing system.

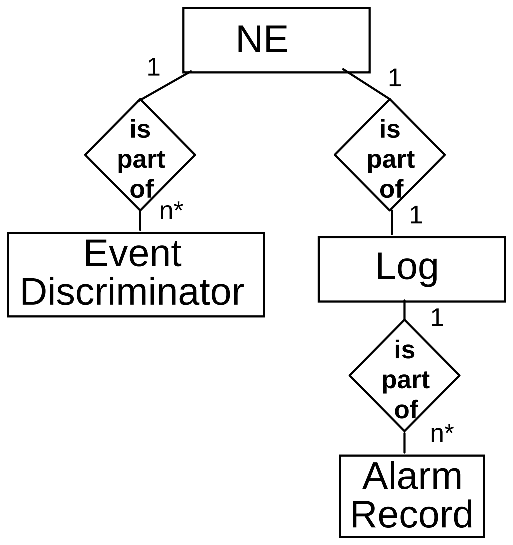


Figure 4.10 Support object for alarm surveillance

The additional entities presented in Figure 4.10 are often called support objects because they are required for OAM&P, but do not represent actual "real" objects inside the managed system. The event discriminator is an object which "filter" outgoing message to the managing system, it contains a "discriminator" construct which is a boolean expression controlled by the managing system. An alarm message respecting the condition expressed in that filter (e.g. severity is critical) will be send to the system The log object is used to hold a number of alarm records and reflects history information.

Step 2: Identification of operations:

Figure 4.11 summarizes the operations exchanged "between objects" (this is not meant to imply any specific implementation), and between the managed system and its manager. The control of the discriminator may be obtained by using a discriminator object.

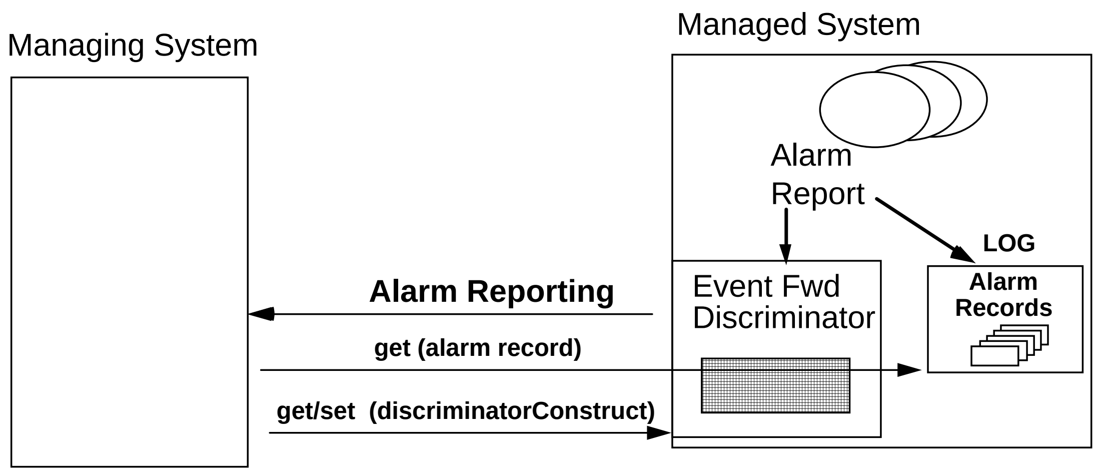


Figure 4.11

The content of the interactions exchanged in Figure 4.11 must also be defined. For the alarm reporting here is an example of some useful information such a message could carry:

Parameter Name	Mandatory/ Optional	Example
ProbableCause	M	TransmissionAlarm: ReceiverFailure
PerceivedSeverity	M	Major
Back Up Status	O	True (failed unit was backed up)
Trend	O	moreSevere
ThresholdInfo	O	Counter: value: 40
MonitoredValue	0	BitErrorRate/Second

Figure 4.12 Some Alarm report parameters

Figure 4.12 presents a non-exhaustive list of useful information for an alarm message. ProbableCause indicates the probable reason for the alarm. The perceivedSeverity indicates the relative importance of the alarm (minor, major, critical) or the clearing of a previous alarm. This attribute can be very useful to filter out all small (minor) alarms. Backed up status shows if the object failed was backed up by another object. When multiple alarms of the same type occurs (on the same object) it may be useful to show that this is a problem that is getting worse (or better). Threshold info and monitored value are used when a threshold crossing is "alarmed" by the OS. In this case, the type of threshold and its value is indicated in threshold info, and the monitored value is indicated by the type of the measured attribute.

Step 3 Behavior of the definition

The following figure illustrates the transition state table for the combination of operational state (disabled/enabled), and of alarm status (cleared, minor, major, critical, underRepair).

The possible combination are Ec (EnabledCleared, Em (EnabledMinor), EM (EnabledMajor), DM (DisabledMajor), DC (DisabledCritical), and Dur (Disabled underRepair).

Res State ----------- Init State	Ec	Em	EM	DM	DC	Dur
Ec		1	2a	2b	3	4
Em	5	1	2a	2b	3	4
EM	5	7	2a	2b	3	4
DM	5	7	8	2b	3	4
DC	5	9	10a	10b	3	4

Dur	6					

Figure 4.13 State Table of Operational State and Alarm Status

The event labelled in the state table are: 1 Minor alarm sent, 2 Major alarm a)stay enabled, b)become disabled, 3 Critical alarm sent, 4 Set to under Repair (either remotely or locally), 5 Last alarm cleared, 6 Repair Terminated, 7 last major alarm cleared, minor remaining, 8 last major alarm causing disabled cleared, other major remaining, 9 last critical alarm cleared minor remaining, 10 last critical cleared major remaining a) enabled b) still disabled.

The reader interested in actual implementation of management network based on the information in this chapter may want to consult [x4, x5]

Discussing a small example such as alarm reporting, implies defining many types of information; object, attributes, operations, behavior, communication. The subject of the next chapter is to explore which techniques are available to describe the information required to specify, verify, implement and maintain such object oriented distributed systems.

5. Object Oriented Specification Language

5.1. Overview

In this section we give an overview of various description techniques that may be used for the development of object-oriented specifications. Description techniques in use vary considerable as far as their formality is concerned. While informal methods have the advantage of being easily adopted, however, the resulting descriptions may lead to difficulty of interpretations, ambiguities and misunderstanding. Therefore formal techniques are advocated. So-called formal languages have a formally defined syntax (rules defining what descriptions are valid) and semantics (rules defining the meaning of a valid description). The standardized "formal description techniques" (FDT) discussed in Section 5.3 belong to this category. Most programming languages have a formally defined syntax, but an informally defined semantics. We call such language semi-formal; the ASN.1 notation described below belongs for instance to this category.

The concepts of object-orientation were first introduced in the context of programming languages. Much after the pioneering design of the Simula language in 1967, Smalltalk was the first widely used language in this respect. Other important examples are Eiffel and C++. Eiffel is a strongly typed language, which facilitates the detection of design errors by the compiler before the execution of a program.

As a specification language for distributed systems, sequential programming languages, such as the above, have the disadvantage that (1) they cannot describe concurrency, (2) the description tend to be implementation-oriented because it is difficult to describe non-determinism or to leave certain aspects undefined. Because of these difficulties, so-called formal and semi-formal specification languages have been introduced, as discussed in more detail in the subsections below.

Certain specification languages only cover certain aspects of a system description. For instance, a finite state machine model only covers those aspects of the behavior of an object which are related to the temporal ordering of input and output interactions. The important aspect of interaction parameters is ignored. Figure XX (#49 revised) shows the major aspects for object-oriented descriptions of distributed systems. It also shows that most specification languages do not cover all aspects. For instance, existing FDT's cover the object-oriented aspects only partly and the ASN.1 extensions described below do not cover the behavior aspect, as discussed in more detail below.

5.2 ASN.1 and extension for Remote Operation and Object Description

ASN.1 (Abstract Notation One) is a specification language defined by CCITT and ISO. It is part of the OSI series of standards [x6]. Its goal is to permit heterogeneous system, having different ways to represent data structures, to exchange information in a standardized format. First the basic ASN.1 will be introduced, then example of extensions to the language (through the use of MACROs and Templates) will be examined.

The intent of this section is not to present a tutorial on ASN.1 (see for instance [Neuf **]), however it may be useful to briefly recall the two main features of ASN.1, the notation, and the encoding of values obtained by using the associated encoding rules.

The ASN.1 language is based on a small number of basic types such as INTEGER, REAL, OCTET STRING, different type of character string, bit string, and others. More complex types are derived by using "constructor" types such as SEQUENCE which is an ordered list of "more" elementary types, and SEQUENCE OF which is an unlimited ordered list made of elements of the same types. The language also consists of operators such as the type assignment operator "::=", and conventions, for example, that type identifier must begin with an uppercase letter and constant must begin with a lowercase letter.

```
PersonelInfo  ::= SET {
                age     INTEGER,
                married     BOOLEAN
                }

Name ::= SEQUENCE {  first IA5String,
                        middle IA5String OPTIONAL,
                        initials IA5String SIZE (1..2) OPTIONAL,
                        last IA5String
                    }
PDU ::= CHOICE { CR, CC, DT, DR, DC, EX, EA }
```

Figure 5.1 Small sample of ASN.1 notation

The second part of ASN.1 consists of a set of rules to encode/decode any value specified using the ASN.1 notation to/from a byte stream that can be transported through high level protocols. The designers of ASN.1 separated the notation from the encoding rules.

Therefore, different encoding rules can be selected according to needs.[4] The next figure illustrates the coding principle used for the Basic Encoding Rules (BER).

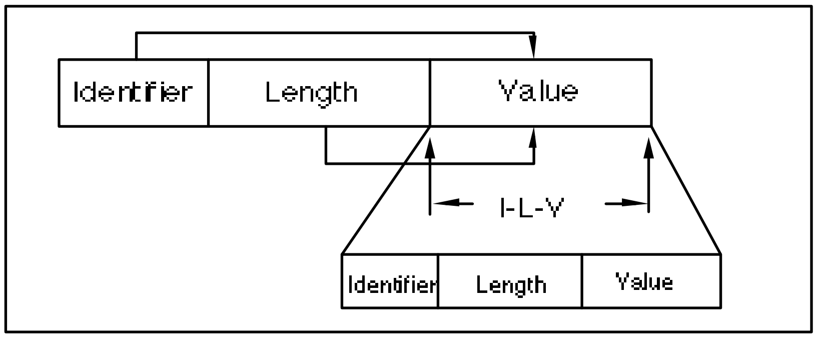


Figure 5.2 ASN.1 BER Encoding

The encoding works on the principle of "Type Length Value" (TLV), as illustrated in Figure 5.2, the value in the case of constructed types can be made (recursively) of other types, until a basic type has been encoded.

ASN.1 is a semi-formal language because it provides means of specifying precisely data structures, as well as the encoding/decoding of values to/from a byte stream (but no formally defined semantics). This process can be automated with the help of an ASN.1 compiler. Many such compilers are available today either as shareware or from commercial sources [**].

The 1988 version of the ASN.1 standard has a facility called MACRO which allows (sophisticated!) users to define additional notation to create language construct tailored to specific needs. An example of such notation is the Remote Operation Service Element (ROSE) defined in [x7].

```
OPERATION MACRO  ::=-
BEGIN
        TYPE NOTATION  ::= Argument Result Errors LinkedOperations
        VALUE NOTATION  ::= value(VALUE CHOICE{
                                  localValue          INTEGER,
                                  globalValue         OBJECT IDENTIFIER})
        Argument                  ::= "ARGUMENT" NamedType | empty
        Result        ::= "RESULT" ResultType | empty
        ResultType ::= NamedType | empty
...........
END
```

Figure 5.3 ROSE Operation Macro

[4] In 1988 the standard contained only the BER described here. The 1992 version will include in addition PER (Packed Encoding Rule), and DER (Distinguished Encoding Rule). Work has also started on a light-weight encoding rule. The selection of the desired encoding rule can be done through the presentation layer protocol at layer 6 of the OSI model.

This Operation Macro defines an ASN.1 Macro called OPERATION which has a value of either type INTEGER, or type OBJECT IDENTIFIER. The notation can be used to define a remote operation (e.g. a remote procedure call) with a list of types as input parameter (in the ARGUMENT clause) and output parameters (in the RESULT clause).

The MACRO notation of 1988 ASN.1 can be used to create useful notation like the Remote Operation. Nevertheless, it is difficult to create a generic parser that can recognize/analyze every possible type of MACRO, and at best this covers only the syntactic aspects. In the example of the ROSE operation, the only thing the operation MACRO really implies is the association a code (integer or object identifier) to an "operation". The semantic consists of the link between this code, and the parameters of the operation, the position of this information in a ROSE APDU, and the effect of the operation itself. Part of this semantic is described in the ROSE standard, the rest is the responsibility of the operation designer. A possible solution is to create compilers which recognizes a few specific MACROS, but this looses the generality searched by the conceptors of ASN.1.

By the end of 1989 the member of the ISO JTC1/SC21/WG4 were struggling with developing different ASN.1 MACROs to represent managed objects. MACROs were needed for object classes, attributes, notifications, actions, name bindings, and the notation was becoming quite complex. There was fear that it was becoming too difficult to built automated tools to support such a notation. Another factor was that in term of a ROSE operation, the semantic related to the operation is still tied with the resulting APDU (plus the semantic of the operation itself). When a MACRO is used to define an object "inheriting" from another object (attributes, behavior, notification/action), this semantic is far from any direct relationship with an APDU[5].

For these reasons it was decided to create a "template" notation which is defined in a document called Guidelines for the Definition of Managed Object (GDMO) [x8]. Because of this, the set of templates defined in this document are often designated as "GDMO" templates. The document defines (informally!) the semantic implied by using these templates. The base of the notation is the managed object class template which is used to define classes made of attributes (mandatory and optional), actions (which can be processed by this class), and notifications (which can be send by this class depending on certain events).

```
<class-label> MANAGED OBJECT CLASS
[DERIVED FROM <class-label>    [,<class-label>]*;
]
[CHARACTERIZED BY     <package-label>      [,<package-label>]*;
]
[CONDITIONAL PACKAGES ,package-label>   PRESENT IF condition-definition
              [,package-label>  PRESENT IF condition-definition]* ;
```

[5] Since then, the ASN.1 MACRO feature has been replaced by the "Information Object Class" mechanism which enables the creation of "template" like construct, and more importantly, which can link part of APDUs to such construct. This seems a great advantage in terms of building tools. The 1992 directory (see section 4.1 will be using this notation. It will be interesting to see if this will also deprecate GDMO templates.

```
]
REGISTERED AS object-identifier ;
```

Figure 5.4 Managed Object Class Template

A managed object class template can refer to another managed object class (inheritance) and is made of mandatory and optional packages. Optional package presence is controlled by a condition which is written in natural language. Although not ASN.1 proper, the GDMO notation refer back to ASN.1 to define elementary components . For example, a managed object class can be registered as an object identifier.

Packages can be used to group a number of related attribute/action/notifications for modularity reasons. Figure 5.4 shows how the different types of templates relates to one another. For example a package template refers to attribute, attribute group, notification, action, behavior and parameter. Is is referred to by the managed object package.

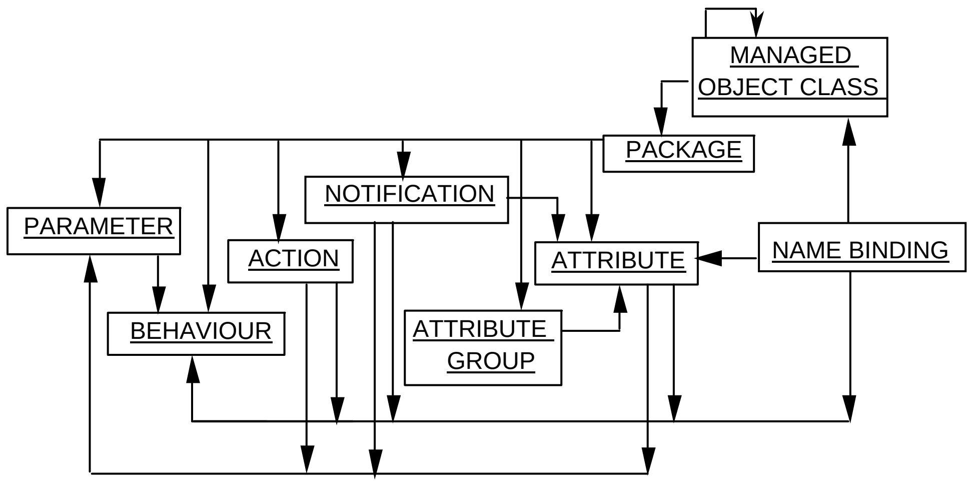
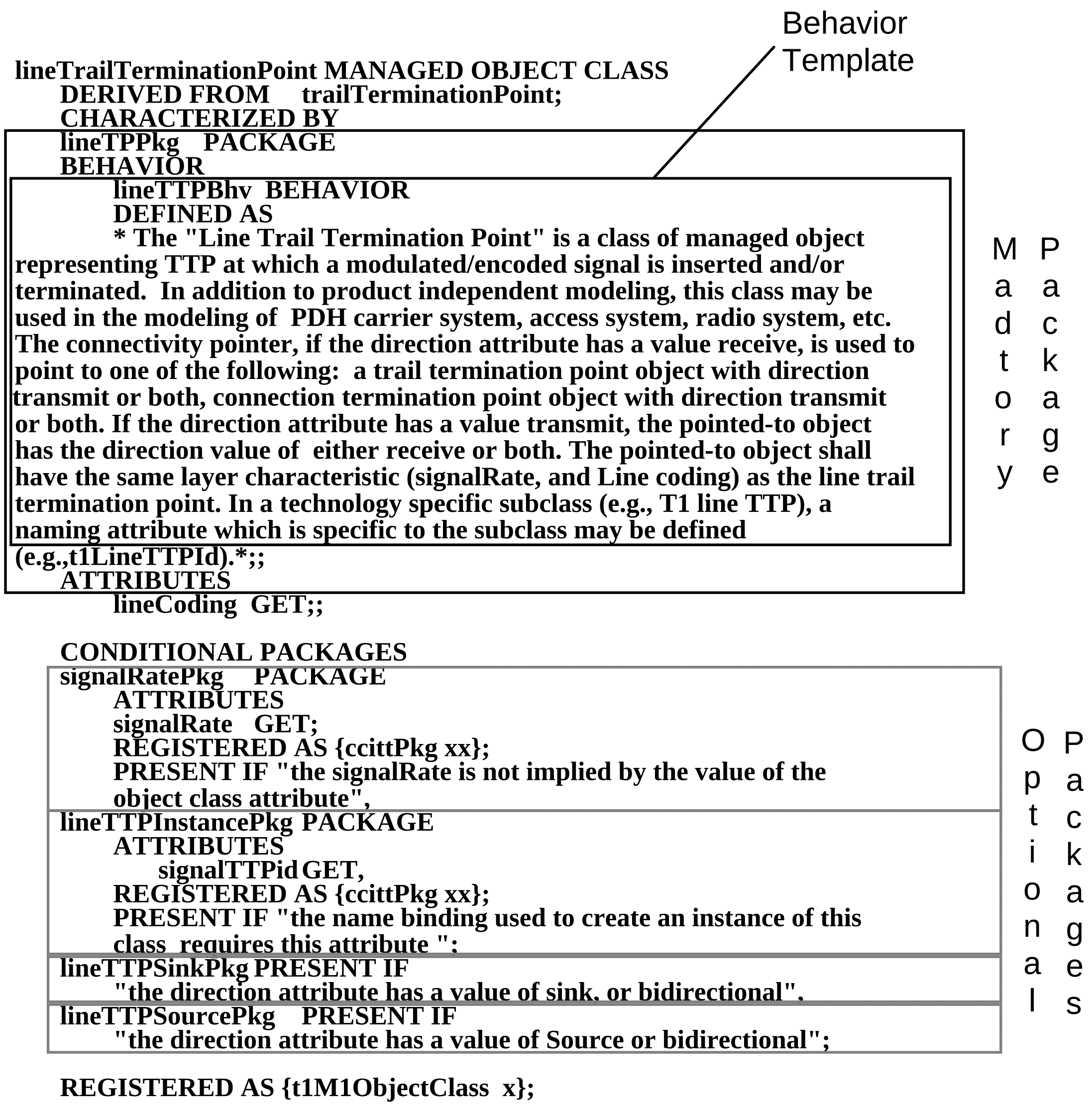


Figure 5.4 GDMO Templates

As an example of the use of the GDMO notation figure 5.5 is a line termination proposal[6] using GDMO. The lineTermination object inherits from another object called "trailTerminationPoint". It contains a mandatory package named lineTTPPkg (defined using the "on-line" style) which contain behavior statements. These statements explain the behavior of the object (in addition to the behavior of trailTerminationPoint). It has also one (mandatory) attribute called "lineCoding". This attribute is qualified with a "GET" qualifier which indicates that a CMIP M_GET message can be used to read the attribute (but an M_SET cannot be used to change it). This object is also made of 4 conditional packages, the first one called signalRatePkg. It contains one optional attribute called signalRate which is there if the condition specified is true. This condition can be verified at implementation or instantiation time, but is not changed after the creation of the instance.

[6] This was part of a Northern Telecom input to T1M1 GNM, at the time of this writing, T1M1 is presently working on the next version of the GNM standard which will be using the GDMO notation.

This notation is limited by its close link with ASN.1. It is very good in term of specifying syntax, but the behaviour is specified outside the template notation itself, currently in natural language. What this means concretely, is that a formalism that can be linked back to ASN.1 will be very useful in the context of OSI, but such formalism should be more powerful than ASN.1 in terms of semantic. This brings us to the issue of behavior descriptions which will be discussed in the next two sections.



```
lineTrailTerminationPoint MANAGED OBJECT CLASS
    DERIVED FROM    trailTerminationPoint;
    CHARACTERIZED BY
    lineTPPkg   PACKAGE
    BEHAVIOR
        lineTTPBhv  BEHAVIOR
        DEFINED AS
        * The "Line Trail Termination Point" is a class of managed object
representing TTP at which a modulated/encoded signal is inserted and/or
terminated.  In addition to product independent modeling, this class may be
used in the modeling of  PDH carrier system, access system, radio system, etc.
The connectivity pointer, if the direction attribute has a value receive, is used to
point to one of the following:  a trail termination point object with direction
transmit or both, connection termination point object with direction transmit
or both. If the direction attribute has a value transmit, the pointed-to object
has the direction value of  either receive or both. The pointed-to object shall
have the same layer characteristic (signalRate, and Line coding) as the line trail
termination point. In a technology specific subclass (e.g., T1 line TTP), a
naming attribute which is specific to the subclass may be defined
(e.g.,t1LineTTPId).*;;
    ATTRIBUTES
        lineCoding  GET;;

    CONDITIONAL PACKAGES
    signalRatePkg    PACKAGE
        ATTRIBUTES
        signalRate   GET;
        REGISTERED AS {ccittPkg xx};
        PRESENT IF "the signalRate is not implied by the value of the
        object class attribute",
    lineTTPInstancePkg PACKAGE
        ATTRIBUTES
            signalTTPid GET,
        REGISTERED AS {ccittPkg xx};
        PRESENT IF "the name binding used to create an instance of this
        class  requires this attribute ";
    lineTTPSinkPkg PRESENT IF
        "the direction attribute has a value of sink, or bidirectional",
    lineTTPSourcePkg    PRESENT IF
        "the direction attribute has a value of Source or bidirectional";

    REGISTERED AS {t1M1ObjectClass  x};
```

Figure 5.5. The line termination object definition.

5.3 Standardized Formal Description Techniques

Several so-called Formal Description Techniques (FDT's) have been developed within ISO and CCITT for writing formal specifications of OSI protocols and services. These languages are called Estelle [Este 89], LOTOS [Loto 89] and SDL [SDL 87]. All three FDT's essentially contain three basic components:

(1) Facilities for describing the temporal ordering of interactions among "processes" (called "modules" in Estelle).

(2) Facilities for describing the structure of a systems as a composition of processes. Estelle uses the concept of "interaction points" connected through "channels". SDL also uses "channels", and in addition the concepts of a "block", which represents a subdivision of a system, and a "route" which is used for describing the routing of messages within a block. LOTOS uses the concept of a "gate" which corresponds to several interconnected interaction points.

(3) Facilities for describing data. Estelle uses elements of the Pascal programming language to describe data structures and operations, including variables and procedures. SDL uses a similar notation, but also provides for the description of abstract data types with operations the semantics of which can be defined through the specification of a set of axioms. LOTOS is a functional language (without variables) and uses abstract data types.

In the following we discuss how the concepts of object instance, encapsulation, class hierarchy and inter-object communication are supported by these FDT's.

In Estelle or SDL, an object instance may be represented by a module instance or process, respectively. In LOTOS, the situation is less clear since LOTOS processes have no identity, and they are also used to represent states or transitions (see for instance [Boch]). These constructs of the FDT's also provide for encapsulation of the inner aspects of the "objects", which only communicate with other "objects" through interactions. In addition, LOTOS uses the concept of gates, which seem to be natural candidates for representing such objects as service access points. However, LOTOS does not support encapsulation, nor the concept of classes for gates. Various ways to introduce the concept of objects into LOTOS have been discussed in the literature [***].

The existing FDT's do not support very well the description of class hierarchies through specialization. In Estelle, several different behaviors (called "body") may be defined for a given type of module, but there is not explicit concept of specialization. The situation is similar for SDL, although some form of specialization is available for abstract data types. LOTOS also provides provides such a facility, and in addition allows for the use of the so- called constraint-oriented specification style [***] allows the definition of specializations of behaviors by specifying additional constraints about the allowed order of interactions.

Within CCITT, under the question of SDL maintenance, an object- oriented version of SDL is being developed [**]. The present proposal includes explicit inheritance for process types and certain aspects of behavior specialization through the specification of additional transitions and states, and the use of unspecified "inner" parts of procedures, as already included in the Simula language.

Concerning the aspect of inter-object communication, there seems to be no generally agreed approach. While some school of thought prefers the message passing paradigm (as used in Estelle and SDL), many object-oriented languages use a (remote) procedure calling mechanism. The object-oriented language Mondel mentioned below also supports rendezvous communication similar to LOTOS. It is to be noted that synchronous communication is preferable to message passing for the development of system descriptions at a high level of abstraction [Boch **]. The main difficulty with message passing is the possibility of message cross-over between two communicating objects. Communication is an important consideration for the comparison of FDT's and other languages.

5.4. Other object-oriented specification languages

As mentioned before, object-oriented languages support encapsulation of modules, which are objects. The interface of the modules of a system are defined in terms of the available operations, as developed during step 2 of the design methodology described in Section **. In the context of programming languages, the behavior of the operations, as considered during step 3 of the design methodology, is often considered an implementation issue, but it is important to note that a complete module specification must include a definition of the behavior. Besides the FDT's discussed above, there are a number of other specification languages that include facilities for the definition of behavior. In the following, we mention a few such languages that are object-oriented.

The specification formalism called Traces [***] emphasizes that the behavior of a module (or object) should be described in terms of the externally visible interactions, more precisely, in terms of the possible sequences (traces) of interactions and their input and result parameters. This approach is object-oriented since all interactions considered within one trace pertain to the same object instance. The formal notation for defining the behavior of an object is related to predicate calculus. A methodology for systematically presenting the behavior definition of a module in the form of a readable document including tables and predicates has also been proposed [***].

Another specification language based on predicate calculus is Z [**]. Recently, certain extensions to Z have been defined which facilitate the description of object-oriented specifications [***]. Applications in the area of ODP have also been presented [**].

Finally, we mention the object-oriented specification language Mondel [**] which uses a high-level algorithmic language and assertions to describe object behavior. During the design of the language, attention was paid to the representation of entity- relationship models with class specialization as used during Step 1 of the design methodology described in Section **, and to a close relation with ASN.1 (as described in Section 5.2) in order to facilitate the translation between standard OSI descriptions and Mondel specifications. The language also supports certain aspects related to object-oriented databases, such as the retrieval of objects instances and transactions. Various applications to OSI protocols have also been done [***].

5.5. Support tools

As shown in Figure 5.6, a system specification is used in the various activities of the system development cycle, such as the validation of the specification, the development of an implementation, the selection of test cases for the implementation, and the analysis of test results. In order to build support tools which may automate part of these different activities, it is necessary to use a specification which is written in some formal notation. The languages discussed above are good candidates for this purpose.

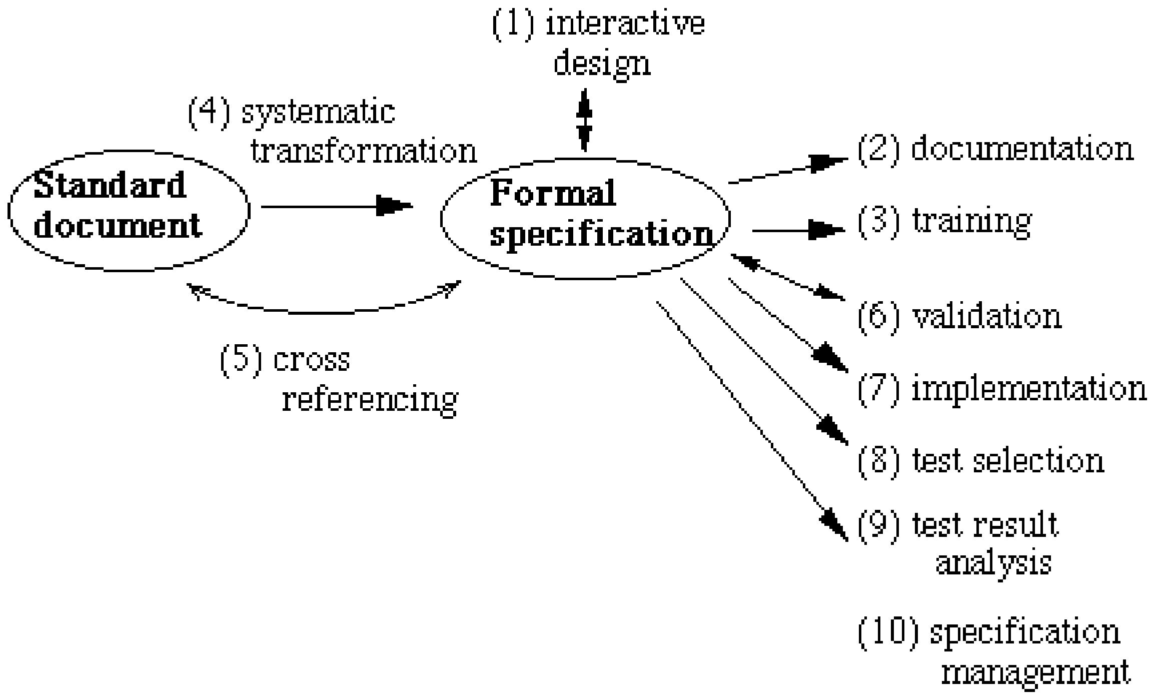


Figure 5.6

A tutorial on specifications for communication protocols and services, formal and informal, is given in [Boch **]. The principles described there remain valid for object-oriented specifications as considered in this paper. Instead of repeating this discussion here, we present in the following shortly the support tools that have been developed for the Mondel language.

The semantics of a Mondel specification is formally defined in terms of an abstract machine which is obtained from the specification through given inference rules which are written in first order logic restricted to Horn clauses. Since the translation of these inference rules to Prolog is straighforward, we were able to obtain a simulation environment in a relatively short time period [Will 90]. This environment allows the interactive and automatic exploration of selected execution paths according to the given specification, and facilitates the debugging of specifications.

For a large subset of the language, it was possible to translate the semantic definition into an equivalent form based on coloured Petri nets [**]. The latter was then used to build a verification tool which makes an exhaustive analysis of all possible execution paths allowed by a given specification. This tool can be used for ****

The automatic translation of Mondel specifications into C++ implementation code is under study. In order to elaborate a translation scheme, we work on the hand translation of a Mondel X.500 specification [***] into a C++ implementation. No Mondel tools related to testing have been developed. However, as for most specification languages, testing tools related to finite state machine models can be adapted if the specification under consideration has an important finite state machine aspect.

7. Conclusion

(To be completed)

Bibliography (incomplete listing)

[x1] Recommendations X.500,501,509,511,518,519,520,521: The Directory, CCITT blue book, 1988.

[x2] T1.214-1990, "American National Standard for Telecommunications - Operations Maintenance & Provisioning (OAM&P) - Generic Network Model for Interfaces between Operations Systems and Network Elements.

[x3] T1.215-1990, "American National Standard for Telecommunications - Operations Maintenance & Provisioning (OAM&P) - Fault Management Messages for Interfaces between Operations Systems and Network Elements.

[x4] "Standard OS-NE Interface (IF2/Q#) Development for the Telecom Canada Integrated Network Surveillance System", C. Morin & T. A. Nguyen, Proceedings of GLOBECOM '90, IEEE, vol. 1 p. 166-70, Dec 1990.

[x5] "First Implementation of an OSI OAM&P Application", W.F. Fairclough, J.M. Serre & D. Wood, Proceedings of ICC 91, IEEE, vol. 1 p. 123-30, June 91.

[x6] X.208,X.209, "Specification of Abstract Syntax Notation One (ASN.1)" & "Specification of Basic Encoding Rules for ASN.1", CCITT blue book, 1988.

[x7] X.219, "Remote Operations Part 1: Model, Notation and Services Definition", CCITT blue book, 1988.

[x8] ISO/IEC 10165-4, "Information Technology - Open Systems Interconnection - Structure of Management Information - Part 4: Guidelines for the Definition of Managed Objects", July 1991.