G.v. Bochmann

Proceedings of the 1992 IFIP
International Conference on

Upper Layer Protocols, Architectures and Applications

ULPAA' 92

May 27-29, 1992
Vancouver, Canada

Object-Oriented Design for Distributed Systems and OSI Standards

Gregor v. Bochmann, Stéphane Poirier and Pierre Mondain-Monval

Département I.R.O, Université de Montréal, C.P. 6128, Succursale A,
Montréal, Québec, Canada, H3C 3J7.

Keywords

Object-oriented specification, directory service, Open Systems Interworking, entity-relationship model, object-oriented databases, distributed systems design, design methodologies, formal description techniques, specification languages.

Abstract

For an object-oriented design methodology to be effective, it is important to provide methods and tools for validating the design specification before going into the implementation phase. The paper proposes a design methodology and a related object-oriented specification language which allows the validation of specifications through simulated execution, or through automatic exhaustive simulation for a certain subset of the language. The paper also discusses the relation of this design methodology and language to other design methodologies which are in wide use, such as the entity-relationship model for databases, the ASN.1 notation used for Open Systems Interconnection (OSI) communication protocols, as well as methodologies used in the standardization committees for the elaboration and description of various kinds of distributed systems standards. It is shown how these different approaches can be integrated into a single methodology and language, using the OSI Directory System as an example.

1. Introduction

The goal of a design methodology is to provide simple, efficient means to structure the process leading from an informal application description to a precise design specification. Using an object-oriented approach, one applies the principle of aggregating data items together with operations performed on them, calling the whole an "object". An application can be seen as a composition of such objects. One of the advantages of this approach is the information hiding principle, which is naturally supported. Another advantage is the fact that at the informal level of description, the concept of an "object" is very common; therefore a design and implementation framework directly supporting this concept leads to more readable specifications. A third important property, usually associated with the object-oriented approach, is "inheritance" which, at the specification level, should mean the inheritance of properties. Properties defined for some more general type of objects will be inherited by objects belonging to a more specialized type of object, sometimes called "subclass".

Although these principles of object-oriented design are well known, it still remains a very informal, intuitive process based on human expertise. Within a joint industry-university research project we have developed a specific methodology [Mond90], and a related object-oriented specification language, MONDEL [Boch89] which have been applied to a number of examples (e.g. [Boch90i]). Although our main interest has been in the area of distributed system management and communication protocols, we believe that our approach is general enough to be applied to many other areas.

For applications relating to communications and other standards, it is important to include, within the system design cycle, the integration of the relevant standards. The amount of information available in these documents is generally huge, and they are mostly written in natural language. In the area of OSI, the standardization groups have developed a certain number of specification practices, guidelines, and more or less formal description techniques and languages (for an overview, see for instance [Boch 90c]). It is important that a design methodology used in this area be able to naturally integrate these existing specification elements into the overall system description.

The purpose of this paper is to demonstrate these issues by explaining our experience with a particular example, namely the OSI directory system [X.500]. In Section 2, we give an overview of our design methodology and the MONDEL language. Section 3 describes the application of this methodology and language to the description of the OSI directory system. In Section 4, we discuss how our design methodology and specification language relate to various other description techniques used for the directory standard and in other areas. Based on these relationships, it is possible to derive certain parts of the design specification from the existing partially formalized documents.

2. Object-oriented design methodology and specification language

2.1. Design methodology

Our design methodology [Boch90d] acknowledges the need to relate object-oriented concepts to the more classical, function oriented, design practices [Ward89, Bois89]. Thus our methodology uses some Entity-Relationship concepts [Chen 76, ISO89] already widely applied to software design. In the area of distributed systems, it is also important that the design methodology relates to existing documentation, such as standardization documents or other existing requirements. This issues are further discussed in Section 4.

Current object-oriented design methods usually comprise a certain number of design steps. Though not all practitioners agree on the number and the denomination of

these steps, but all come up with approximately the same philosophy [Booc86, Meye87, Bail88, Bail89, Jalo89, Bail90].We identified the following four major steps, which can be iterated until a satisfactory design is obtained:

1. A preliminary step first consists of the identification of the general problem area, of the specific aspects to be handled, and also on the aim of the expected design; this step is an informal one, but it should lead the designer to separate the different aspects included in the application, and also to precisely define the intended purposes of the model to be elaborated.
2. Step 1: The first step of the design development consists of the identification of the key components of the so-called application domain; this step takes as input any information describing the functionalities to be provided, and should produce as a result a set of so-called entities, together with relationships among them, which can be easily mapped onto object-oriented languages constructs, as discussed in Section 4.1.
3. Step 2: A subsequent step is concerned with the allocation of the functions as operations offered by objects identified in the previous step; some functions will also uncover some new objects which must be integrated in the application domain.
4. Step 3: The last step is concerned with the definition of the behaviors of the objects; these behaviors consider the possible sequences of operations calls as well as the necessary processing associated with each operation; complex objects can be specified as smaller "applications" i.e., the entire design process can be applied to each individual object as it is applied to the global application.

2.2. The specification language MONDEL

We have developed an object-oriented specification language called MONDEL [Boch89] which supports the development methodology mentioned above. In many respects, MONDEL resembles other existing object-oriented languages. It has, however, certain properties which make it particularly suitable for describing real-time distributed systems, such as the example described later.

In MONDEL, everything is an object. Therefore the entities, their attributes and the relationships identified during Step 1 of the methodology are represented in MONDEL as types, as further discussed in Section 4.1. The multiple inheritance scheme of MONDEL supports in a direct manner the "is-a" relations identified in the design.

In contrast to many other object-oriented languages, MONDEL distinguishes between persistent and non-persistent objects. Persistent objects are like entries in a data base; they remain present until they are explicitly deleted, and they can be interrogated by database-oriented statements which identify the object instances in the database which belong to a given class and have specified properties. Entities and relationships identified during Step 1 are usually represented as persistent objects.

Concerning the design information related to Step 2 of the methodology, MONDEL has the well-known concept of operations (sometimes called "methods") which are defined for a given object class, and are provided by each instance of that class to be called by other object instances. In order to call an operation, the calling object has to know the identity of the called object. The type checking feature of MONDEL is able to detect many design errors related to the parameters of the called operations and the returned results.

In contrast to many object-oriented languages, communication between objects is synchronous, that is, a calling object is blocked until the called object executes a RETURN statement, which may include the delivery of a result. Synchronous communication is better suited for specifications at higher level of abstraction, since cross-over of messages at interfaces can be largely avoided [Boch 90a].

Concerning Step 3 of the methodology, MONDEL provides a number of statements to express the order in which the operations can be accepted by an object, or for specifying the actions to be performed when an operation call is accepted by the object. The concept of an ATOMIC operation is introduced which represents a "transaction" in the sense of databases, that is, it represents a set of actions which are either all performed without interference from other "transactions", or undone if an exception condition is encountered. The language has exception handling similar to ADA. The actions of different objects are usually performed in parallel; it is also possible to define several parallel activities within a single object.

The statements of the language have the flavor of a high-level programming language, except for the database-oriented statements mentioned above. However, it is also possible to write assertional specifications by defining input and output assertions for operations, or by defining INVARIANT assertions which must be satisfied at the end each "transaction".

MONDEL has a formally defined semantics [Barb 90a] which is the basis for a development environment which includes a compiler and simulator which can be used to execute MONDEL specifications. In addition, a verification tool automatically checks for certain design errors, such as deadlocks for not too large specifications (for more details, see [Boch 90, Will 90, Barb 90b]).

3. An example: The OSI directory system

3.1. Overview of the OSI directory system

The OSI directory system (DS) [X.500] is a collection of cooperating systems which provide information and manage a distributed database, the co-called Directory Information Base (DIB). The information contained in the database is used to facilitate the communication with and about objects in a distributed environment, such as Application Entities, people, terminals, distribution lists etc.. Although the user of the directory is unaware of it, the DIB is distributed over many sites called Directory Service Agents (DSA). The directory is accessed through user processes, called Directory User Agents (DUA), as shown in Figure 1.

The service of the directory is provided through a set of operation calls, which are grouped into three service classes: The Read class allows the interrogation of the information contained in a particular entry of the DIB, the Search class permits the exploration of the DIB, and the Modify class permits the modification of the data in the DIB.

The DIB is logically structured in the form of a tree (Directory Information Tree, DIT). Each entry has a Relatively Distinguished Name (RDN) which is unique among all its siblings. For every entry in the tree a unique name, called Distinguished Name (DN) is defined as the concatenation of all the RDN encountered from the root of the DIT to the entry in question. For example Figure 2 shows a DIT distributed over 3 DSA. The DN of the entry identified by a triangle is C=VV O=DEF OU=K (where "C", "O" and "OU" stand for "country", "organization" and "organizational unit", respectively).

3.2. Preliminary step: Problem definition

This preliminary step is necessary to guide the designer during the following steps, and to focus on the relevant aspects.The results of this preliminary step are more of a set of guidelines for the following steps than formal results.

During this first step, the intended functionalities of the application to be designed should be stated. Usually, not all components of a complex system need to be specified; also the level of details to be considered for a given component may vary. Often a high-level of abstraction and genericity is desirable: the specifications must stay valid for a wide variety of configurations and withstand changes to be introduced over time.

The application to be specified may concern an existing domain. For instance, network management applications are usually defined for existing networks. Therefore, the application domain already exist and the new functions must cope, at the appropriate abstraction level, with existing components. Generally, the design of an application starts with a (possibly empty) given domain, and new components are to be added. Such an approach promotes re-use of specifications since it does not isolate a given application form existing and future ones.

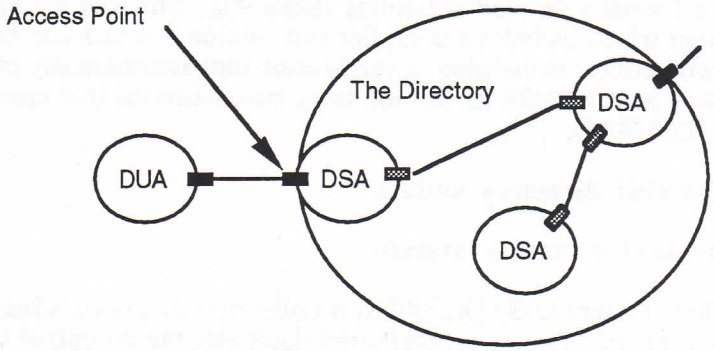

Figure 1: Subsystems in the OSI directory system [X.500]

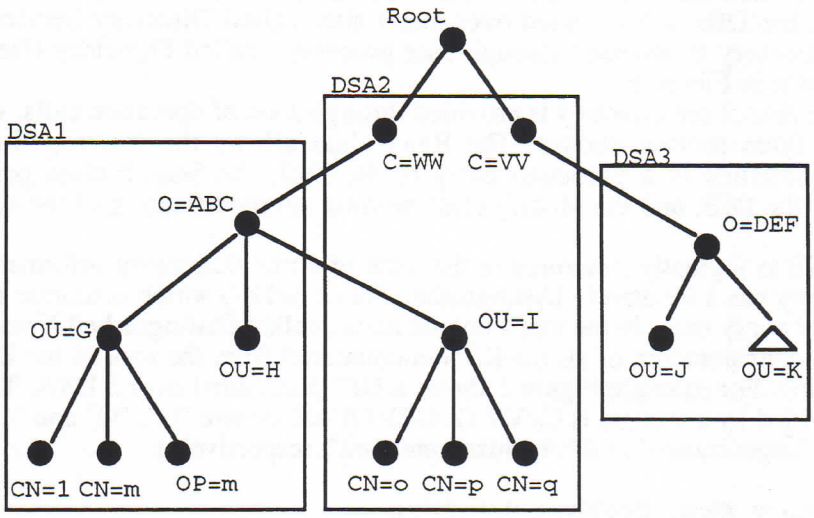

Figure 2: Hypothetical DIT [X.518, Figure 5]

Finally, the intended purpose of the specification must be considered. The specification may be used for various purposes:

- design validation (internal consistency and relation to requirements);
- documentation for some hardware and/or software architecture;
- simulation, for user training, for future system development analysis, or as a prototype before building a larger scale system;
- performance analysis;
- (automated) software production.

In the case of the directory system, a relatively detailed system design was already given in the form of the OSI standard [X.500]. The purpose of our Directory Project [Poir 91] was to study how easy this design could be represented using the MONDEL language and the object-oriented design methodology discussed here. We were therefore also concerned with the translation of certain formalized notations used in the OSI standard into MONDEL, as discussed in Section 4. The second purpose was to obtain an executable MONDEL specification of the directory which could be used for the validation of the design or as a prototype.

3.3. Design Step 1: Domain definition

The domain definition step is intended to capture the relevant elements of the existing or foreseen domain, together with their essential characteristics. The result should be a description of the domain as a set of entities together with the various relationships among them that are relevant to the functionalities and purpose of the model. This step can be refined in the following sub-steps:

1. The first one is to identify entities of interest, together with their specific characteristics.
2. A second sub-step is to identify the various relationships existing among these entities. They usually cover different aspects of the application to specify:
   - specialization of object types, represented by the inheritance relationship, often written "is-a"; different entities in the domain may share some common characteristics which can be specified as a general class (or type), called superclass, which may be inherited by more specialized subclasses
   - structuring aspect, represented by the aggregation relationship, often stated as "is-part-of" or "is-made-of" relationships
   - functional aspects: many identified relationships stem from the functionalities the designer intends to specify

In the case of the OSI directory system, we can identify the entities and relations shown in Figure 3, using a common graphic notation. A DSA is composed of a single connection handler (called Bind Handler) and DIBFragment, and a varying number of Server objects. A new Server object is created for each connection which is established with the DSA by a DUA or another DSA. Different specializations of server objects exist which provide the Read, Search, and Modify operations, respectively. Similarly, a DUA consists of a single connection handler and a user connection end point (UCEP) for each connection established with a DSA.

During the domain definition, object attributes may be identified. These attributes may represent specific characteristics of an object, such as the attribute "maxServer" of the type DSA which represents the maximum number of "Server" objects that can be supported at any time. Trying to bind to a DSA once its maximum is exceeded would result in an error condition.

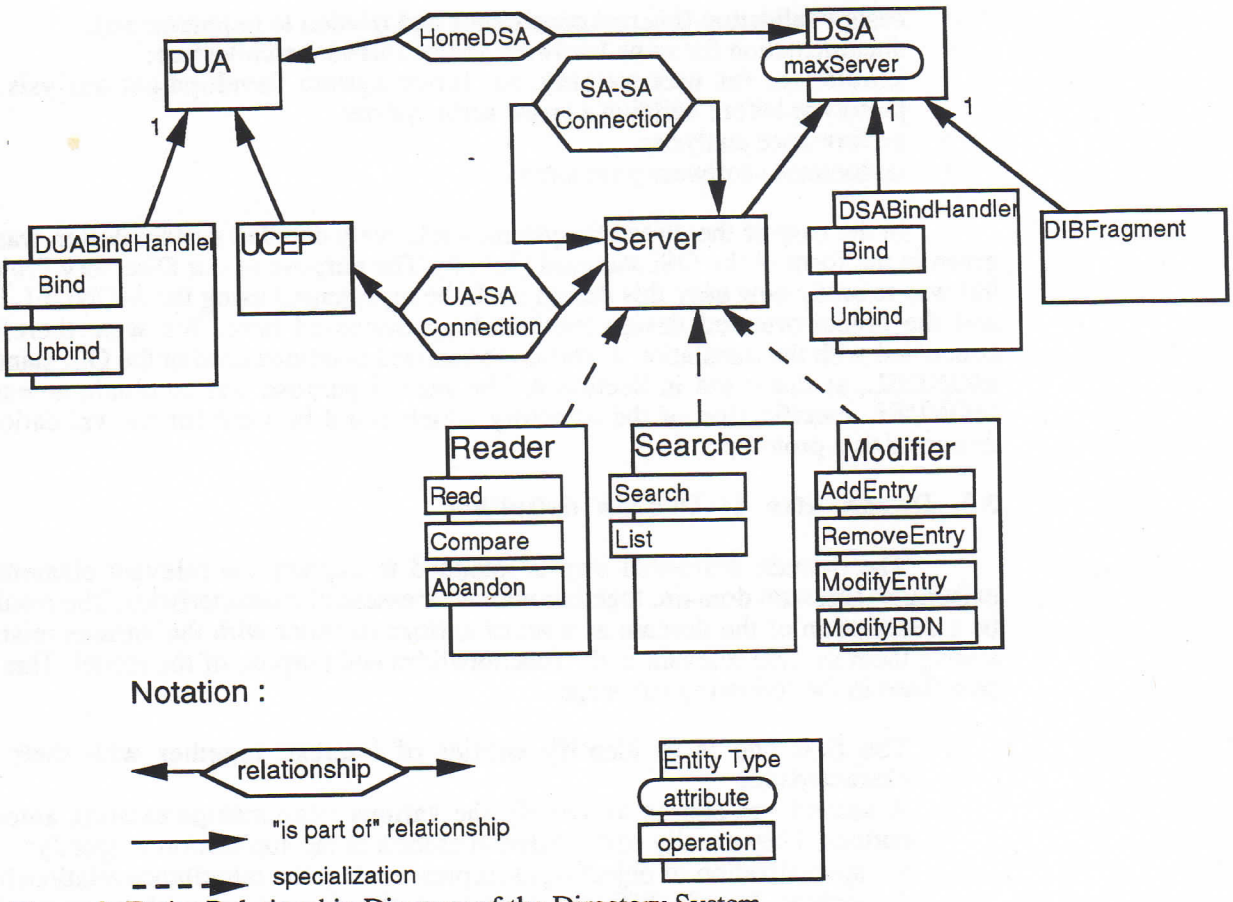

Figure 3: Entity Relationship Diagram of the Directory System

The graphic notation used in Figure 3 and the following is not part of our design methodology. However, we think that graphic representations of designs are important for making the designs understandable. The notation of Figure 3, or any other suitable graphic notation may be used in relation with our design methodology.

3.4. Design Step 2: Functions definition

The function allocation step intends to distribute the required functionalities among the identified entities. There are several important aspects to this process:

1. Since entities are represented as objects, the functions they have to perform are defined as operations they must offer. An operation is formally defined as a procedure or a function; i.e., with some input and output parameters, which must be objects too. These parameters must be defined if possible.
2. Having identified the operations, the designer must allocate them to some objects.
   - An operation sometimes might be convenient to allocate it to a new "support" entity introduced for the purpose of allocating the operation;
   - operations and parameters types should be specified with respect to the entities offering them; also, when operations are inherited, operations names and parameters types may have to be refined.

3. The next point is to consider the support entities introduced during the allocation process and to integrate them within the application domain.

In the case of the OSI directory system, the following operations can be identified and associated with the objects shown in Figure 3. The DUA and DSA BindHandlers accept the operations Bind and Unbind. The Bind operation invoked on a DSA, to which a connection is to be established, includes as parameter the kind of service desired. It returns as result the identification of a connection with a Server object which has been created by the destination DSA and which will provide the service over the created connection. The operation Unbind can be used to eliminate a remote Server object and the corresponding connection.

Figure 4 shows a possible scenario of operation executions involving the establishment of a Read connection by a user with a Server of a DSA. This is done by first invoking a "Bind" on the object "DUABindHandler1" (1). This creates an object "UCEP" and invokes the operation "Bind" on its "HomeDSA", which is the object "DSABindHandler1". This will cause an object "Reader1" of type "Reader" to be created and its reference returned to "DUABindHandler1" which may now complete the relation "UA-SA". Note: This scenario is similar to the connection establishment scenario used in [Mond90b] for the description of the OSI Reference Model. Now the user has the possibility of invoking a "Read" operation (3) which will be treated at "DSA1" by the "Reader1" object (4). Assuming that the operation cannot be satisfied at this DSA, a reference to another DSA will be found and permit to invoke a "Bind" operation on "DSABindHandler2" (5) to create a "ChainedReader" object. Once the reference to this new object is returned, the Read operation is forwarded to it (6).

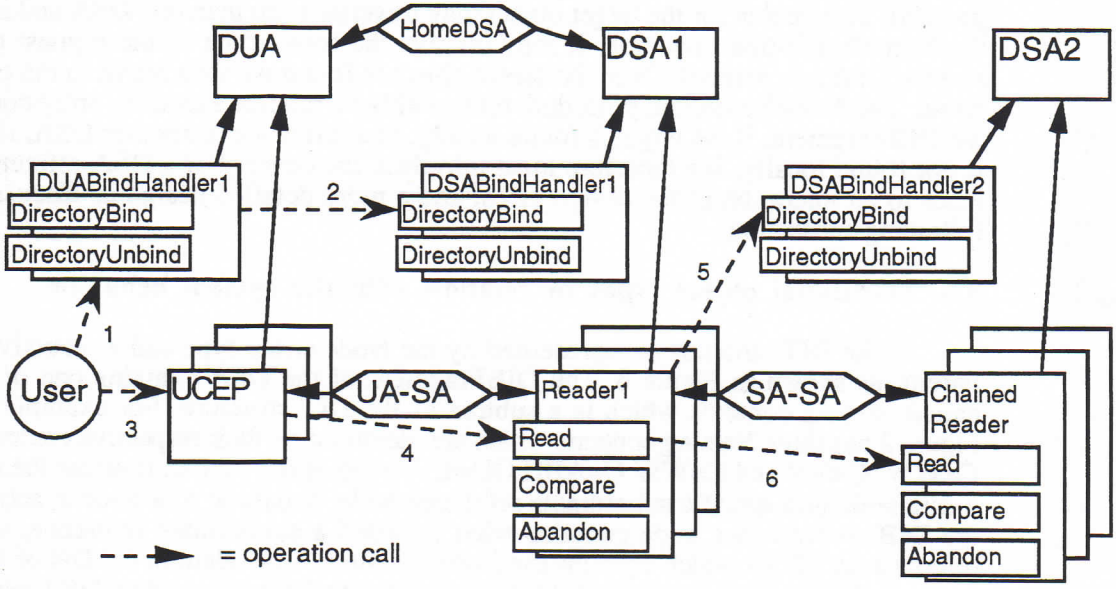

Figure 4: Object Instances in the Directory System

It is important to note that, in contrast to Figure 3 which shows the classes of objects in the directory system, Figure 4 shows instances of such objects and instances of their relations.

The Server objects accept the Read, Search or Modify operations, depending on their specialization. The corresponding operations of a chained Server are the same, except that the parameter and result types contain additional information concerning the chaining of the requests.

3.5. Design Step 3: Behaviors definition

This last step consists of the definition of the behaviors of the various entities for the allocated operations. This process mainly depends on the knowledge and expertise of the designer in the specific field. However, some general principles can be applied:

1. For each object, it is first necessary to define the accepted sequences of operations; a general skeleton can be defined for that purpose.
2. For each operation offered by a given object, there are two possibilities:
   - the behavior for that operation is simple enough so it can be easily specified with the language statements; the necessary processing is described in terms of state changes, attributes modifications, and interactions with other objects;
   - the behavior is complex, and a refinement process can be applied to it; the technique is to consider the processing to be performed as a restricted application which can be specified by applying the design process from the Preliminary step to Step 3, until all components are fully specified.

In the case of the OSI directory system, we have to describe the behavior of the DUA and DSA connection handler and the different specializations of the Server class. Each operations of the Server classes is executed sequentially. The first phase of the execution consist of finding a target object in the DIT from which the evaluation will proceed. This search for the target object may traverse more than one DSA and is done by the NameResolution operation. It may involve the forwarding of the request to several remote DSA's in parallel. Once the target object is found we then move to the evaluation phase. The NameResolution procedure returns either a reference to some entry contained in the DIBFragment, if the target is found locally, or a reference to another DSA, if the entry is not found locally. We therefore must introduce the concept of a DSA reference which leads to an interation of the design steps, and a more detailed design as discussed in the following subsection.

3.6. Additional object types in relation with the system behavior

The DIT structure is represented by the Node entity type and a recursive relation Parent, as shown in Figure 6. The DIBFragment of the DSA contains one or more so-called naming contexts which is a subtree of the DIT structure. For example DSA2 in Figure 2 has three Naming contexts which are identified by their respective context prefixes C=WW, C=VV and C=WW O=ABC OU=I. A node of the subtree is either the root node, a data node or a specific subordinate reference node. A data or root node is actual data in the DIB. A reference node contains what is called a subordinate reference, which is a reference to a DSA which contains the naming context equivalent to the DN of the current node. In Figure 2 each edge which leads from one DSA to another DSA represents a subordinate reference.

For the execution of the NameResolution procedure in the distributed context, we need to find the naming context whose prefix most closely matches the DN of the entry we are looking for. This is done by the internal procedure "FindNamingContext" shown in Figure 6. Similarly, the procedure "FindCrossRef" finds the closest match among the available cross references. Once we have found a matching naming context we must then

search the nodes of the subtree for an entry which will satisfy the query or for a subordinate reference. This is done recursively by the operation "NodeSearch" associated with each Node of the DIT (see Figure 6).

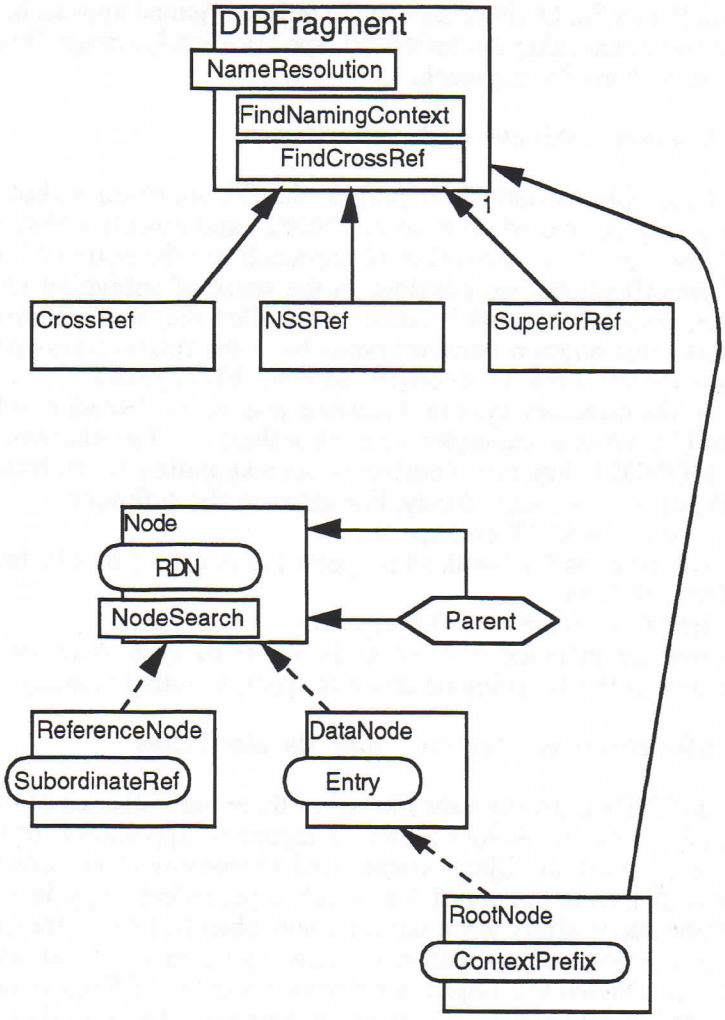

Figure 6: Information Held in a DSA

4. Discussion of various description formalisms

For applications, such as the directory system discussed in this paper, where many standards and other documents must be considered during the system design, it is important to find a common description formalism which is suitable to express all relevant design issues, which is close to the formalism used in the standard and other documents, and which is formal enough to allow computer-aided design tools. We have tried to make MONDEL compatible with several notations used in the context of OSI standardization and other areas, such as explained in the following subsections (see [Boch90a] for further details).

4.1. The entity-relational model and object-oriented specifications

We discuss in this subsection how the concepts of the entity-relationship (ER) approach [Chen 76, ODP] relates to the object-oriented approach, and how these concepts can be represented using the MONDEL specification language. It is assumed that the reader is familiar with the ER approach.

4.1.1. Classes, objects and inheritance

Like object-oriented languages, the ER approach makes the distinction between classes of objects, called types in MONDEL, and objects which are the instances of such types. However, the object-oriented approach has the notion of inheritance, which in the area of specifications, we consider in the sense of subtyping [Amer90, Boch89b]. This notion corresponds to what is sometimes called the "is-a" relationship. It is important to distinguish this relation between types from the relationships between object instances, which are the usual relations considered in the ER approach.

In the directory system discussed above, the "Reader" inherits from the "Server" type; the UA-SA is an example of a relationship; To express the relation between types, MONDEL has two constructs corresponding to multiple inheritance and non-deterministic choice, respectively. For instance, the definition

type S = A and B endtype S

means that the class S inherits all properties of A and of B; it is therefore a specialization of both. The definition

type G = choice A or B entype G

means that an instance of type G is either of type A or of type B. This construct corresponds to the discriminate union of types, as found in many programming languages.

4.1.2. Modelling an "entity" and its attributes

MONDEL distinguishes between three basic classes of objects: actors, persistent, and passive objects. Actors typically represent application programs, users, or system interfaces; persistent objects correspond to instants of entities and relations, which are involved in transactions and for which a persistent copy is retained until the current transaction successfully terminates; passive objects, finally, include the predefined objects of integers, strings, etc. as well as common data structures and other user-defined objects.

Each MONDEL object has a certain number of fixed acquaintances (usually called "attributes" in MONDEL). Their value does not change during the entire lifetime of the object. It seems natural to model an ER entity type as a persistent MONDEL type where the entity attributes are modelled as inactive acquaintances. In the case that the entity attribute value may change, an acquaintance of type VAR [T] may be used which means that the acquaintance is a variable, to which new values of type T may be assigned during system evolution. For example, the following definition

```
type RootNode = DataNode
   contextPrefix : contextPrefix
endtype RootNode
```

describes the entity "RootNode" of Figure 6. "contextPrefix" is its specific attribute, while it also inherits the attributes "Entry" and "RDN" from its superclasses.

4.1.3. Modelling relationships using the "database approach"

A relationship between object instances can be most directly be modelled in MONDEL as an object type which has an acquaintance for each role of the relation. For instance, the definition

```
type UA-SA = persistent with
  ucep : UCEP
  server : Server
endtype UA-SA
```

describes a relationship between the types "UCEP" and "Server" of Figure 3

This approach of modelling relationships has the following properties:

(1) Referential integrity (i): An instance of a relationship can not be created without the existence of the related entities.
(2) Referential integrity (ii): If one of the related entities is deleted, the corresponding instance(s) of a relationship is (are) also deleted.
(3) Dynamic relationship: The creation and deletion of instances of a relationship do not affect the existence of the related entities.
(4) Independence: The relationship and the related entities are specified as separate types.
(5) Specialization and relationship attributes: A basic relationship (as exemplified above) may be refined by adding attributes or other pertinent information to the definition of the relation type. For example:

It is possible to specify further properties by defining a subset of the acquaintances to form a key, or by adding an invariant condition which must be satisfied during the creation of each occurrence of the type. An example is shown in Section 4.1.5 below.

4.1.4. Modelling relationships using the "data structure approach"

In the context of high-level programming languages, relationships may be represented using various data structures. MONDEL provides certain predefined structures, such as VAR, SET, BAG, and SEQUENCE, which may be used for this purpose. However, the symmetry between the different roles of the relationship is lost in these representations.

For example, the first relationship "is part of" in Figure 6 could be represented by an acquaintance (of the DIBFragment) which is of type SET [CrossRef]. With this approach, the aspects (2), (4) and (5) of Section 4.1.3 are lost. In the case that at most one "CrossRef" exists for each instance of DIBFragment, the acquaintance may be of type VAR [], instead of a SET [], which simplifies its use and implementation; the value NIL can be used to indicate the absense of a related "CrossRef". This representation is the same as for a variable entity attribute.

Inversely, if the "is part of" relationship is fixed for each component, the relationship may be represented by a fixed acquaintance of the component. For example, the above relationship could be represented by an acquaintance (of the "CrossRef" entity) which is of type "DIBFragment". In MONDEL, this representation also implies referencial integrity, that is, if a "DIBFragment" is deleted then all "CrossRef"s related to it will also be deleted.

4.1.5. Modelling aggregation relationships

An aggregation relationship, such as "an entity instance of type AType is composed of entities of type BType", is a particular case of a functional relation from BType to AType, and can be modelled as explained above. For example, the relation that a Part may be (recursively) composed of several subparts may be described as

```
type Node = persistent with
  superior : Node OPT; {or "superior : VAR Node ;" }
        endtype Node
```

The superior acquaintance assumes the value NIL if the Node instance is the root node. We may want to force this condition by the use of the MONDEL invariant clause.

```
type RootNode = DataNode with
  contextPrefix : DistinguishedName
  invariant
        superior = nil
endtype RootNode
```

4.2. Standard data structure definitions in ASN.1 and Remote Operations

ASN.1 is a notation which allows the definition of data types , similar to the data type definitions available in programming languages such as Pascal or ADA. The notation includes a number of predefined types such as integers, reals, booleans, bit strings etc ... It also allows the definition of composed types , such as groups of elements (called SEQUENCE, corresponding to the Pascal RECORD), groups of elements of the same type (called SEQUENCE OF), a list of alternative types (called CHOICE, corresponding to Pascal's variant records).

The notation is used mainly to define protocol data units (PDU) parameters of OSI Application layer protocols.. But it could also be used for describing the parameters of service primitives and other data structures. The systematic translation from ASN.1 to data structures in specification languages, such as Estelle and LOTOS, or implementation languages, such a C (see for instance [Boch89c]) have been defined. A similar translation can be defined into MONDEL, as discussed in more detail in [Boch 90e].

Various extensions to ASN.1 have been developed within the OSI standardization community for describing such issues as: (a) ports through which connections can be established with other system components [X.407], (b) "Remote Operations" [ISO88] which correspond to the well-known concept of remote procedure calls, and (c) classes of object with inheritance relations [ISO89]. The latter notation supports concepts similar to those discussed here (see [Boch90a] for a more detailed comparison). In contrast to the other notations, it is not used for the description of the OSI directory standard.

The "remote operations" can be modelled naturally in MONDEL in the form of operation calls. In MONDEL, as in any object-oriented language, all communication between objects proceeds through the call of operations which are associated with an object, which must be known to the object that executes the call. In MONDEL, the calling object has to wait for the return of the operation, which may include the result of the operation, or may occur immediately after the acceptance of the operation by the called object. This allows the modelling, in MONDEL, of the synchronous and asynchronous communication foreseen for "remote operations" (see [Boch90a] for more details).

4.3. The description of object behavior

For the description of the behavior of operations, one may use algorithmic specifications based on a high-level programming language (an approach taken for MONDEL) or based graphic notations, such as Nassy-Shneiderman diagrams. In an alternate approach, the specifier defines input and output assertions for the operations. These approaches are sufficient in the case of "sequential" objects, that is, for objects that accept one operation call after another, in an arbitrary order.

In the context of distributed systems, however, the order of execution of different operations is often of prime importance. In order to specify the allowed sequences of operation executions, there are a number of approaches, varying between algorithmic specifications (e.g. LOTOS [Loto 89] or MONDEL) defining directly the possible execution orders, and assertional methods imposing constraints on the allowed execution sequences and their parameters (e.g. [Hoff88]).

A finite state machine (FSM) model is often used for the specification of ordering constraints for communication protocols and other applications, however, it usually only provides for a simplified model of the specified system. This model is directly supported by certain description techniques (e.g. Estelle [Este 89] and SDL [SDL 87]), and can also be translated in a straightforward manner into an algorithmic form. One translation scheme foresees one procedure per state; it models the FSM in a direct manner. Another translation scheme tries to obtain an algorithmic description which follows, as much as possible, the style of structured programming [Boch 87g]. More details are given in [Boch 90a], and an example for the Directory system is discussed in [Boch 90e].

5. Concluding discussion

We have presented in this paper an object-oriented design and formal specification of the OSI Directory System. This specification was developed using an object-oriented design methodology and related specification language, called MONDEL. Since the OSI Directory standard includes already most of the design choices included in our specification, the main interest of this work is to demonstrate how various different design paradigms and description techniques can be integrated into an object-oriented approach and supported by a single object-oriented specification language.

The Directory specification in MONDEL discussed in this paper has been validated through execution in the exsting MONDEL simulator [Will 90]. It has also been the basis for an implementation in C++ following a systematic translation approach from MONDEL into C++, as described in [Nadé 92].

The design of a distributed application, such as the directory, involves different paradigms and techniques addressing different aspects of the design. Summarizing the discussion of Sections 3 and 4, we identify the following major aspects:

(a) The identification of entity types and relationships within the application domain, as an early phase within the design process. This is related closely to the entity-relationship model for databases, and some description standards and design methodologies for distributed systems [ODP]. The integration of these concepts into an object-oriented paradigm was discussed in Section 4.1.
(b) Specialization of object classes, a concept largely developed within object-oriented languages and directly supported in this context.

(c) The description of operations that may be invoked on objects, and of data structures that may be used as parameters or result types. This relates to the ASN.1 notation used in the OSI standardization context and corresponding notations used in many programming languages.
(d) The description of the behavior of objects, which should determine the effect and results of operations and any constraints on the order and/or parameter values of operation invokations. This relates to various description paradigms, such as finite state machines (or extensions, such as SDL or Estelle), structured programming notations and flowcharts (such as Nassy-Sheiderman), or assertional methods using input/output assertions. This aspect is the most difficult one for any system description.

The integration of these different aspects into a single design methodology, supported by a suitable specification language, and associated with suitable tools for the validatin of design specifications, the development of implementations from the design, and the generation and management of appropriate test cases, is clearly a desirable goal. However, it is not clear how best to develop an overall system/application development environment which makes the development process more efficient and reliable. We think that the here described example of the OSI Directory System demonstrates that the first part of the goal, namely the integration of existing design paradigms and description techniques into a single object-oriented methodology and specification language, is feasable.

Acknowledgements

The here described methodology and specification language was developed within a joint research project on "Object-oriented databases: application modelling and speciifcation for telecommunications network management" by Bell-Northern-Research and the Computer Research Institute of Montreal (CRIM). The authors thank all members of this research group for many fruitful discussions. The work on the Directory System was supported by the IDACOM-NSERC-CWARC Industrial Research Chair on Communication Protocols.

6. References

[Amer90] P. America, *A Behavioral Approach to subtyping in object-oriented programming languages*, Philips Journal of Research, Vol. 44, Nos 2/3, pp. 365-383, 1990.

[Bail88] S. Bailin, *An object-oriented specification method for ADA*, in ACM *Proceedings of the Fifth Washington Ada Symposium*, June 1988.

[Bail89] S. Bailin, *An object-oriented requirements specification method*, CACM, Vol. 32, N. 5, May 89.

[Bail90] S. Bailin, *Remarks on object-oriented requirements specification*, 1990.

[Barb90a] M. Barbeau, G. v. Bochmann *Formal semantics of MONDEL*, Progress Report Document N. 11 for CRIM/BNR project, June 1990.

[Barb90b] M. Barbeau, G. v. Bochmann *Formal Verification of Object Oriented Specification in MONDEL*, Progress Report Document N. 12 for CRIM/BNR project, June 1990.

[Boch87] G. v. Bochmann, *Usage of protocols development tools: the results of a survey*, Invited Paper, 7th IFIP Symposium on Protocol Specification, Verification, and Testing, Zurich, May 1987.

[Boch89] G. v. Bochmann, M. Barbeau, A. Bean, M. Erradi,L. Lecomte, *CRIM/BNR Project---The specification Language MONDEL* Progress Report Document N. 2 for CRIM/BNR project, April 89.

[Boch89b] G. v. Bochmann, *Inheritance for objects with concurrency* , Université de Montréal publication #687a, March 1989.

[Boch89c] G. v. Bochmann, Michel Deslauriers *COMBINING ASN.1 support with the Lotos language*, Université de Montréal publication #684, Mai 1989.

[Boch 90] G. v. Bochmann, M. Barbeau, M. Erradi, L. Lecomte, P. Mondain-Monval and N. Williams, *Mondel: An Object-Oriented Specification Language*, Publication departementale #748, Departement IRO, Université de Montréal, November 90.

[Boch90a] G. v. Bochmann, S. Poirier, P. Mondain-Monval, M. Barbeau, *System specification with MONDEL and relation with other formalisms*, Progress Report Document N. 13 for CRIM/BNR project, June 1990.

[Boch90b] G. v. Bochmann, *Specifications of a simplified Transport protocol using different formal description techniques*, to be published in Computer Network and ISDN Systems.

[Boch90c] G.v. Bochmann, *Protocol specifications for OSI*, Comp. Network & ISDN Systems, April 90.

[Boch90d] P. Mondain-Monval, G. v. Bochmann, *An object-oriented software design methodology*, Progress Report Document N. 7 for CRIM/BNR project, June 1990.

[Boch 90e] G.v.Bochmann, S.Poirier, P.Mondain-Monval, Object-oriented design for distributed systems: The directory service example, Techn. Report, 1990.

[Boch87g] G. v. Bochmann and J. P. Verjus, *Some comments on*, transition-oriented" vs. "structured" specification of distributed algorithms and protocols", IEEE Trans. on SE Vol SE-13, No 4, April 1987, pp. 501-505.

[Boch90i] L. Lecomte, P. Mondain-Monval, G. v. Bochmann, *Un modèle orienté objet pour le système de transmission NT FD-565*, Progress Report Document N. 8 for CRIM/BNR project, June 1990.

[Bois89] H. Bois, *Une methode de developpement de logiciels fondee sur le concept d'objet et exploitant le langage ADA*, Universite Paul Sabatier, Toulouse, France, October 1989.

[Booc86] G. Booch, *Object-oriented development*, IEEE Transactions on Software Eng., February 1986.

[Card88b] L. Cardelli, J. Donahue, L. Glassman, M. Jordan, B. Kalsow, G. Nelson, *Modula 3 Report*, D.E.C., 1988

[Chen76] P. P. Chen, *The entity-Relationship Model - Toward a Unified View of Data"* , ACM Trans. on Database Systems, 1, 1 (March 1976), pp. 9-36

[Este89] ISO, *IS9074 (1989), Estelle: A formal description technique based on an extended state transition model*, 1989

[Hoff88] D. Hoffman, R Snodgrass, *Trace specifications: Methodology and models* , IEEE Tr. SE 14, No.9 (Sept. 1988), pp.1243-1252.

[ISO87] DIS 8824 *Specification of Abstract Syntax Notation One (ASN.1)*, 1987.

[ISO88] DIS 9072-1 *Remote Operations, Part 1: Model, Notation, and Service Definition*, 1988.

[ISO89] DIS 9595 *Common Management Information Service Definition*, 1989.

[Jalo89] P. Jalotte, *Functional refinement and nested objects for object-oriented design*, IEEE Transactions on Software Engineering, Vol. 15, N. 3, March 1989.

[Loto89] ISO, *ISO, IS8807 (1989), LOTOS: a formal description technique, Based on the Temporal Ordering of Observational Behavior.*, 1989

[Meye87] B. Meyer, *Reusability: the case for object-oriented design*, IEEE Software, March 1987.

[Mond90b] P. Mondain-Monval, *Object-oriented Model for the OSI Reference Model* Université de Montréal, January 1990.Submitted to IFIP Symposium on Protocol Specification, Testing and Verification, 1990.

[Nade 92] R. Nadeau, MSc thesis, Université de Montréal, 1992.

[ODP] *Working Document on Topic 6.1 - Modeling techniques and their use in ODP*, ISO/IEC JTC1/SC21 N 3196, December 1988.

[Poir91] S. Poirier, *Évaluation du langage de spécification MONDEL à la description de protocoles de communication*, Masters thesis Université de Montréal, 1991

[SDL87] CCITT SG XI, Recommendation Z.100, 1987

[T1M189] Committee T1-Telecommunications Standards Contribution, *Modelling guidelines*, February 89.

[Ward89] P.T. Ward, *How to integrate Object orientation with structured analysis and design*, IEEE Software, march 1989.

[Will90] N. Williams, G. v. Bochmann *Description technique d'un simulateur pour le langage MONDEL*, Progress Report Document N. 10 for CRIM/BNR project, June 1990.

[X.407] CCITT, ISO 8505-4, *Message Handling Systems Abstract Service Definition Conventions*, 1988

[X.500] CCITT, ISO 9594-1 *The Directory - Overview of concepts, Models and Services*, 1988.

[X.501] CCITT, ISO 9594-2 *The Directory - Models*, 1988.

[X.511] CCITT, ISO 9594-3 *The Directory - Abstract Service Definition*,1988.

[X.518] CCITT, ISO 9594-4 *The Directory - Procedures for Distributed Operation*, 1988.