Formal Description of Network Management Issues*

Gregor v. Bochmann[a], Pierre Mondain-Monval[b] and Louis Lecomte[c]

[a,b]Département I.R.O, Université de Montréal, CP 6128, Succursale A, Montréal, Québec, Canada, H3C 3J7

[c]Computer Research Institute of Montreal (CRIM), 3744, rue Jean-Brillant, Bureau 500, Montréal, Québec, Canada, H3T 1P1

Abstract

This paper presents an object-oriented design methodology and a supporting object-oriented language for the specification of applications in the field of network management. A general object-oriented software design methodology is presented. An example of application of this methodology for the specification of fault management functions in the case of a real, industrial, transmission system is then presented. Last, we present the desired features of an object-oriented language, MONDEL, that we are currently defining for the specification of distributed systems.

1. Introduction

Appearances of very large scope networks, and the possibility of interconnecting multiple and various networks (telephone networks, private and public packet switching networks, Integrated Services Digital Networks, and LocalArea Networks) allows a great number of applications and users to communicate using a profusion of services. Apart the operational aspects of the network components, current efforts focus on the management aspects of the network: (1) network surveillance; e.g., how to know what is happening in the network, which users are using the services, (2) network control; e.g., how to offer the services in the most efficient way, how to recover from failures.

So far, the efforts of the various telecommunication and information processing standardization groups have defined (1) some abstract representation of a network (components, users, applications, events) to be included in a global, distributed, data base,

* This work is supported by a joint research contract between Bell Northern Research and Computer Research Institute of Montreal.

the Management Information Base (MIB), (2) the different functions to be provided (configuration management, fault management, accounting, and many others), and (3) some basic services to support the previously cited functions (e.g., ISO Common Management Information Service [ISO89]). Considering the great versatility of networks, these standards must achieve a high level of abstraction, genericity, flexibility, and extensibility, as well as large applicability and acceptance among the networking community.

The aim of this paper is to show how these high-level issues can be formally described for real systems while staying in line with the concepts defined by the standardization groups. In particular, we use an object-oriented approach, and represent the standards concepts by means of objects. This paper is organized as follows: in Section 2 we present an object-oriented design methodology; in Section 3 we show how this methodology was applied to a real, industrial, transmission system, and Section 4 deals with the language issues involved in such an exercise. Section 5 provides our conclusions both on the methodological and language aspects.

2. Methodology for specification development

This section presents an object-oriented software design methodology for the specification of applications in the field of network management and other areas. The goal is to provide some simple, efficient, means to structure the process leading from some informal application specification to a formal object-oriented design.

This methodology is based on the concept of objects; i.e., that software applications can be designed based on the principle of aggregating data items together with operations performed on them. The application can be seen as a composition of such objects. Advantages of this design approach are numerous, though it still stays a very informal, intuitive, process based on human expertise. This methodology was defined [Mond90] with a specific object-oriented target language, MONDEL, whose main features are described in Section 4. However, we believe it is general enough to be applied to any object-oriented language.

Second, this methodology 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 already widely applied to software design.

A last key point is to specialize this methodology for the specific case of applications related to network management. Considering the starting point of the design process, we take in account the type (and amount!) of information usually available to designers of such applications; especially, it seems highly desirable to stay as close as possible to the practices of telecommunication and information processing standardization groups [T1M189c ,ODP, ISOnm1, ISOnm2, ISOnm3 ,ISOnm4].

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, all come up with approximately the same philosophy [Booc86, Meye87, Bail88, Bail89, Jalo89, Bail90].We identified four major steps which are explained in more detail below. They 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 we want to handle, 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. A second step 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, that can be easily mapped onto object-oriented languages constructs.

3. The third step is concerned with the allocation of the functions to be provided 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. The last step is concerned with the definition of the behaviours of the objects; these behaviours 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.

2.1. Step 1: Problem definition

This preliminary step is necessary to help the designer to consider some important points.

Network Operation, Administration, Maintenance, and Provisioning (OAMP) is a very complex field due to the intertwinning of very different functionalities: data transmission, configuration management, error recovery, service optimization, security control, accounting, and many others. This step should help to state the intended functionalities of the application to be designed and to focus on the relevant aspects.

Complexity is also due to the large variety and number of components: various pieces of equipment, multiple services, numerous users and applications. Also, each component must present some of the different functionalities previously stated. With respect to the intended functionalities, not all components need to be specified. Also, the level of details to be considered for a given component may vary.

Due to the geographical distribution of networks, as well as evolution over time, application specifications must achieve a high-level of abstraction and genericity: the specifications must stay valid for a wide variety of equipment from different manufacturers, as well as for various configurations and services which evolve 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 added. Such an approach

promotes re-use of specifications since it does not isolate a given application form existing and future ones.

The last point to consider is the intended purposes of the specifications.We use the term system model to denote the specification of a domain together with its functionalities. Such a model may be used for various purposes:

- formal verification, to check the consistency of the design, and/or the correctness of algorithms;
- 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.

Different formalisms may be required to achieve these different goals. Even with the same formalism, different specification styles may be adopted. For instance, "intentional" or "extensional" styles might be preferred. Also, the level of details to be included in the model greatly depends on its intended use. The results of this preliminary step are more of a set of guidelines for the following steps than formal results. They should help the designer to focus on relevant purposes and to determine the level of abstraction required for each component. Though this is presented as a preliminary step, our experience showed that this should be constantly considered throughout the entire design process.

2.2. Step 2: 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. Various entities exist, and are usually characterized with specific properties covering different aspects such as state, value, and structure. These entities can be represented as objects and attributes whatever the target language is.
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:
    - structuring aspect, represented by the aggregation relationship, often stated as "is-part-of" or "is-made-of" relationships; this relationship helps to define appropriate

abstraction levels; i.e., whether a given entity must be handled as a whole or as a set of components; aggregation can be static or dynamic;

- typing aspect, represented by the inheritance relationship, often stated as "is-a" relationship; different entities in the domain may share some common characteristics which can be specified in a generic template (or type); templates can be specialized for various purposes; e.g., to specialize inherited features or to add some new ones;

- functional aspects: many identified relationships stem from the functionalities the designer intends to specify; also, some of these relationships are static while some are dynamic.

Object-oriented languages allow a designer to formally specify these relationships: aggregation and functional relationships are represented by means of attributes, while subtyping is well captured by the inheritance mechanism.

3. The next sub-step is a step of consistency checking. Consistency checking applies to both entities and relationships:

   - entities having common attributes might be specialized instances of more generic types; common characteristics may be grouped within common templates;

   - an entity might include orthogonal aspects; therefore, a given entity may be considered as the combination of two or more generic ones separately specified;

   - entities must have attributes representing their relationships with others; specific entities may also be defined to represent these relationships;

   - relationships may lead to define entities functionalities;

   - relationships must be considered with respect to the inheritance lattice; a relationship stated for general templates might not hold as such when coming to more specialized entities, and thus should be more precisely defined.

   The principle here is to provide some basic means (1) to structure the domain (inheritance and aggregation relationships), and (2) to examine the domain according to the identified functionalities (specific relationships).

4. A last sub-step is to (re)write a design documentation where the identified components appear clearly. Since a formal model may not be suitable for human purposes, a textual, informal, documentation should closely match the system model.

2.3. Step 3: Functions definition

The function allocation step intends to distribute the required functionalities among the identified entities. There are four 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. For each operation, the designer must consider which entity seems the "most natural" one to offer the operation. Several points are to be considered:

   - when an operation can be offered by several entities, it might be better to allocate it to a common ancestor in the inheritance lattice;

   - when an operation does not seem to fit a particular entity, or seems to naturally belong to very different entities, or seems to correspond to some cooperative processing by several entities, it 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. The same process as in Step 2 should be reapplied; this favors the re-use of software specifications since the resulting structured domain can be re-used when defining future applications in the same field.

4. Since this step leads to define new entities and operations they offer, the textual specification can be rewritten so all identified entities appear clearly and their interactions are expressed in terms of operations.

2.4. Step 4: Behaviours definition

This last step consists of the definition of the behaviours 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 behaviour 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 behaviour 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 Step 1 to 4, until all components are fully specified.

3. An example: Network Fault management

We present in this section an application of the previously defined methodology to a real, industrial, transmission system.

3.1. Step 1: Problem definition

This example aims at specifying some standard network management functions [T1M189b] in the case of a real, industrial, transmission system [FD86, Leco90]. Also, conformance with a general network model defined by some telecommunication standards group [T1M189a] is required (Fig. 1).

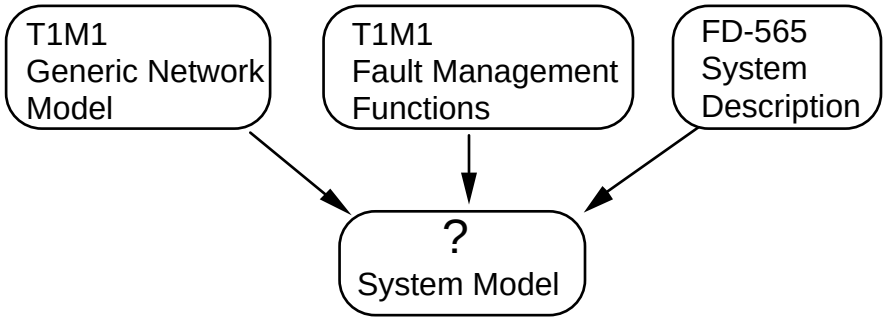


Figure 1: Problem definition

Though this example applies to a specific system, it has a significant scope since it includes all the components designers of network management applications have to deal with:

- an abstract, standardized, description of the domain [T1M189a],

- an abstract, standardized, description of the functions [T1M189b], and

- a real, industrial, system description [FD 86].

The general problem of network management is defined in [T1M189a]. The set of nodes (called *Network Elements* or NE's for short) providing the transmission service is connected to a set of surveillance systems (called *Operation and Surveillance Systems* or OS's for short) through a *Network Management Network*. The real system to be specified is an instance of a *Network Element*. Only aspects related to fault management, as defined between a *Network Element* and an *Operation and Surveillance System* (Fig. 2), by the relevant standards [T1M189b], are to be defined. Fault management functions in [T1M189b] are defined as a set of messages exchanged between the NE and the OS. They include standard parameters, and the functions to be implemented are denoted by the expected sequences of sent and received messages, and the behaviours in reaction to them.

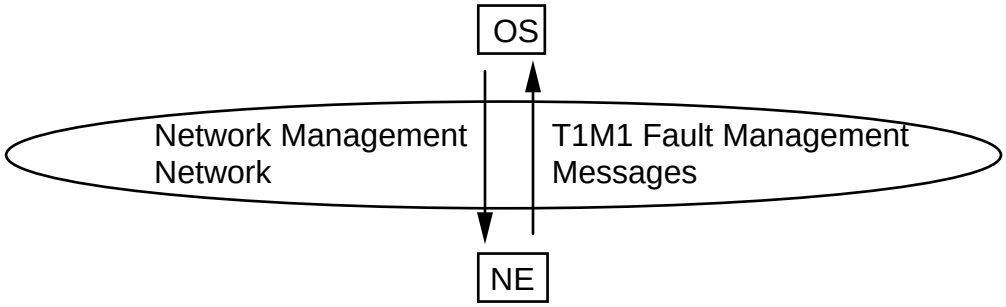


Figure 2: Fault management functions

Therefore, we did not consider in our specification the "normal" behaviour of the transmission system (such as connection establishment, data transfer, ...) but only (1) the physical structure of the system, as a specialized instance of a more general, abstract, network element, and (2) the "faulty" behaviour.

3.2. Step 2: Domain definition

The result of the second step is the description of the physical structure of the system, as an instance of the generic network element, in terms of its sub-components which also are instances of the generic model (Fig. 3). In particular, we included as the leaves of the aggregation tree the so-called *AlarmPoints* of the real system, which actually are part of the basic components, the so-called *Units*. But since we are concerned only with fault management, we can "project" these specific parts onto the domain we are interested in. Therefore, other aspects (e.g., data transmission) are not modelled. On this figure, aggregation is shown by a plain arrow while inheritance is shown by a dashed arrow.

FD-565 Transmission System **T1M1 Generic Network Model**

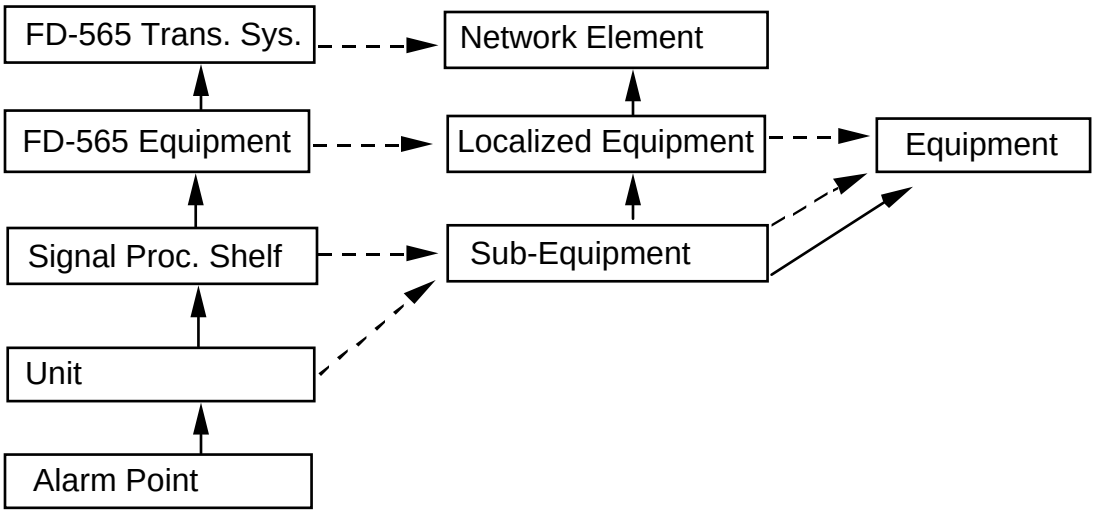


Figure 3: Domain definition

3.3. Step 3: Functions definition

The results of this third step is the definition of support objects accepting operations corresponding to the messages they might receive. The operations parameters strictly correspond to those of the messages. For instance,when considering the alarm reporting functionalities, we defined two support objects, the *NEAlarmReportControl* and *OSAlarmReportControl* objects. One of them represents the interface of the network element and the other one represents the interface of the surveillance and operation system.

Since we described the physical structure as an aggregation tree, we also defined the *NEAlarmReportControl* object as a hierarchy of components matching the physical structure (Fig. 4). We foresaw that messages exchanged between the NE and OS would also have to be exchanged among the components of the system hierarchy. The resulting support objects are then integrated within the application domain as suggested on Fig. 4. Each level of the *NEAlarmReportControl* has a specific, functional, relationship with the corresponding level of the system physical structure.

In our object-oriented language, there are several ways to represent such a functional relationship:

1. by use of attributes, which allows each object to call operations offered by others,

2. through inheritance, by defining that a *RealAlarmPoint* object will inherit attributes, operations, and behaviours of the *AlarmPoint* and *AlarmPointReportControl* objects, or

3. by use of a specific *control* construct which allows one object to define more constraints on acceptance of operations and more processing in response to operations calls of another object declared as a *controlled* attribute [Boch90a].

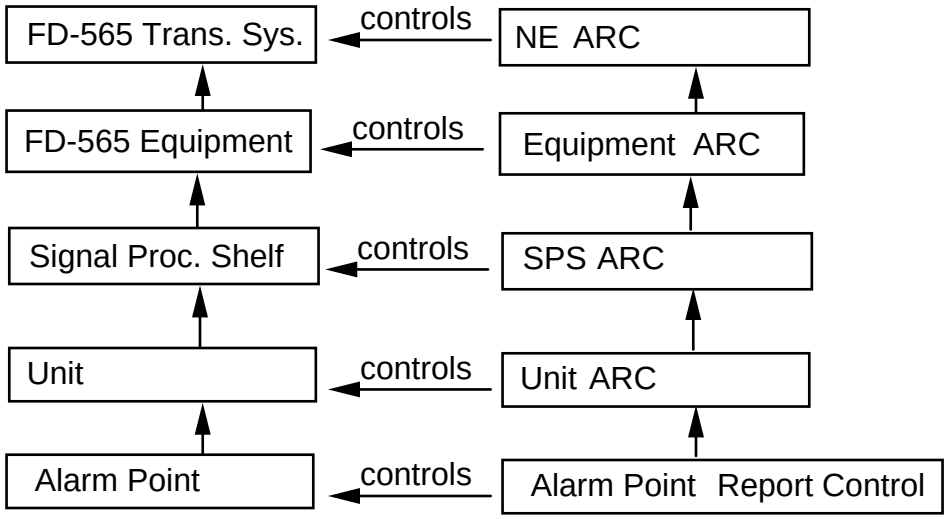
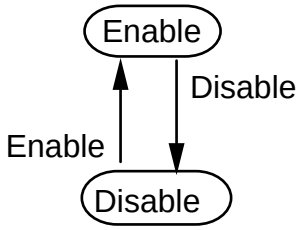


Figure 4: Alarm Report Control (ARC) support objects

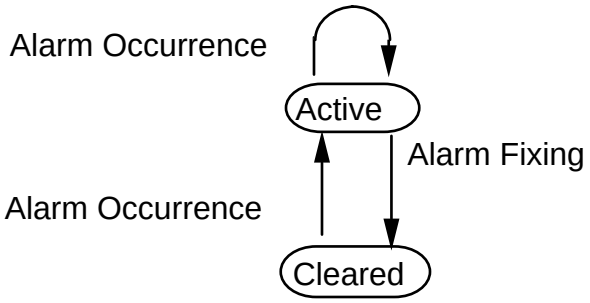
Considering the functionalities described in the standards, we also allocated two operations, *Enable* and *Disable*, both supported by the *AlarmPointReportControl* object, in order to enable and disable the alarm reporting capability of the *AlarmPoint* object.

3.4 Step 4: Behaviours definitions

The last step is the most difficult one since (1) the system description does not include any behaviour specification related to T1M1 fault management functions, and (2) the T1M1 standards do not describe any detailed behaviour of an object such as an alarm control point. We include in that paper the behaviour we defined for the *AlarmPoint* and *AlarmPointReportControl* objects. Basically, the *AlarmPoint* and *AlarmPointReportControl* objects behave as described on Fig. 5.



Operations acceptance for Alarm Point Report Control



Fault model for Alarm Point

Figure 5: Alarm Point and Alarm Point Report Control behaviour

Their behaviours are represented by two finite state machines running in parallel: the first one relates to the occurrence of an exception within an *AlarmPoint* object (the two states are *active* and *cleared*) while the second one relates to the alarm reporting capability of the *AlarmPointReportControl* object (the two states are *enabled* and *disabled*). The combined behaviour is quite simple: an alarm must be reported when the *AlarmPoin*t object is in the *Active* state the AlarmPointReportControl object is in the Enable state. The mapping of such finite state machines onto object-oriented languages constructs is shown in [Boch90a].

4. Issues concerning specification languages and development tools

We have developed an object-oriented specification language called MONDEL [Boch89] which supports the development methodology described above. In many respects, MONDEL ressembles other existing object-oriented languages. It has, however, certain properties which make it particularly suitable for describing real-time control systems, such as the example of Section 3. One of these properties is the fact that object instances and classes can be naturally represented in the language. Another aspect is its formal definition [Barb90a] which provides the basis for the construction of development tools, while certain other aspects are discussed below.

4.1. Overview of MONDEL

In MONDEL, everything is an object. Therefore the entities, their attributes and the relationships identified during Step 2 of the methodology are represented in MONDEL as objects. Relationships may be represented in different ways, as discussed in [Boch90a]. 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 2 are usually represented as persistent objects.

Concerning the design information related to Step 3 of the methodology, MONDEL has the well-known concept of operations (sometimes called "methods") which are defined for a given object class and 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 of higher level of abstraction, since cross-over of messages at interfaces can be largely avoided [Boch90b].

Concerning Step 4 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".

4.2. Relation with other specification formalisms

For applications, such as network management, where many standard 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, is close to the formalism used in the standard documents, and 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, as explained in the following (for more details, see [Boch90a].

The ASN.1 notation [ISO87] is commonly used for the description of the application layer PDU's and their parameters. The ASN.1 constructs of SEQUENCE (or SET), SEQUENCE (or SET) OF, CHOICE as well as most predefined ASN.1 types have direct counterparts in MONDEL.

The notation for Remote Operations (ROSE) [ISO88] corresponds to the definition of operations that can be invoked on objects. Depending on the intention of the specifier, a remote operation including the return of a result can be presented in MONDEL as a single synchronous operation call, or as two operation calls corresponding to the two PDU's exchanged between the calling and called objects according to the ROSE protocol. The exception handling notation of MONDEL may be used to describe the error situations and error handling in the calling object.

For the description of OSI management, a notation for describing object classes has been developed in the standardization community [ISOnm4]. Most of its concepts can be directly related to the MONDEL notation, although certain concepts have no direct counterpart. It is important to note, however, that the OSI notation does not provide a formal framework for defining the behaviour of objects (i.e. Step 4 of the methodology).

Finite state diagrams, often found in communication protocol and service specification, can be translated into MONDEL using various schemes, one of which is shown in the example of Section 3.

4.3. Use of formal descriptions and related development tools

Formal, as well as informal, specifications are used in various ways. The specification of a system is the basis for the implementation of that system. It is also the basis for the selection of test cases and the reference for the analysis of test results. But first of all, the specification itself must be validated, possibly against a more abstract specification and requirements document. In the case of protocol specifications, these issues are further discussed in [Boch90c].

In order to partially automate the above development activities, various support tools can be used. It is important to note, however, that specification languages without a formal definition (in particular, natural language) make the construction of automated support tools very difficult. A survey of tools developed for use with protocol specifications can be found in [Boch87]. Many of these tools are intended for specifications written in one of the so-called Formal Description Techniques (FDT's). For the MONDEL language, a formal language syntax and semantics has been developed and is the basis for several development tools which are shortly described below.

A MONDEL compiler verifies the syntax and static semantics of MONDEL specifications, including the type checking rules. It also translates the specification into an intermediate form which can be used for the execution by an interpreter written in Prolog [Will90]. This allows a user-guided or automatic execution of MONDEL specifications in a simulated environment which is very useful for interactively validating specifications.

While simulated executions of specifications are helpful for finding errors, they are not able to show the absence of errors. A complementary approach of exhaustive validation may prove the absence of errors, but is usually much more difficult to realize. Work on exhaustive validation of MONDEL specifications is in progress [Barb90b]. It is restricted to a (useful) subset of the language and exploits the fact that this subset can be translated into Petri nets. The validation methods and algorithms available for Petri nets can therefore be used on the translated specifications, or be adapted to operate directly on the MONDEL specifications.

5. Conclusions

Our conclusions during that experiment are manifold; they deal with standards representation issues, with general software design issues, as well as with the required features for a formal language aiming at supporting such an area.

- On the standardization issues, it appeared that the generic models provided by the standardization groups are greatly improved by the use of a formal, common, representation. The object-oriented model achieves such a purpose. For instance, many extensions to ASN.1 have been defined to provide some formal, high-level, and precise definition of management issues. The point to be considered here is that the language we defined easily supports these ASN.1 extensions and can be used throughout the entire design process.

- On the design process itself, we believe that a common design model powerful enough to represent aspects such as Entity-Relationship concepts, Object, and Functions is absolutely necessary to provide formal models matching the various standard documents. The design methodology presented in Section 2 takes into account efforts of standardization groups.

- Beside its capability to represent standardization aspects, the model must be formal; i.e., its semantics must be formally defined if one wants to use it for software design, formal verification, simulation, and automated implementation. Such a formal model must also be supported by adequate tools.

Acknowledgments

The authors want to thank M. Shurtleff, J.M. Serre, A. Bean, and D. Wood from Bell Northern Research for their useful comments and advice.

6. References

[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 1989.

[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 *Verification of high-level language specifications: a Petri net based approach*, 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 1989.

[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*, Computer Network and ISDN Systems, April 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 Engineering, February 1986.

[FD86] *FD-565 Optical Fiber Digital Transmission System: System description*, Northern Telecom Limited, 1986.

[ISOnm1] DP 10165-2 *Systems Management - Object Management Function* .

[ISOnm2] DP 10165-2 *Structure of Management Information -Definition of Support Objects.*

[ISOnm3] DP 10165-3 *Structure of Management Information -Definition of Management Attributes.*

[ISOnm4] DP 10165-4 *Structure of Management Information -Guidelines for Managed Object Definition.*

[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.

[Leco90] 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.

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

[Mond90] P. Mondain-Monval, G. v. Bochmann, *An object-oriented software design methodology*, Progress Report Document N. 7 for CRIM/BNR project, June 1990.

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

[T1M189a] Committee T1-Telecommunications Standards Contribution, *Operation, Administration, Maintenance, and Provisioning (OAMP): A generic network model for interfaces between operations systems and network elements*, Doc. Number T1M1.5/89-010R2, July 1989.

[T1M189b] Committee T1-Telecommunications Standards Contribution, *Fault Management Messages*, Doc. Number T1M1.5/89-011R2, July 1989.

[T1M189c] Committee T1-Telecommunications Standards Contribution, *Modelling guidelines*, February 1989.

[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.