Object-oriented modelling and development of distributed systems

Gregor v. Bochmann

Université de Montréal and Centre de Recherche Informatique de Montréal

Abstract

The object-oriented approach to the analysis and design of information processing systems enjoys presently much popularity. For applications to distributed systems, certain specific requirements must be met. The paper reviews object-oriented analysis and design methodologies (OOADM's) and other specifications languages that have been proposed for the development and standardization of distributed systems. After presenting, as an example, the description of the OSI directory system, various methods and languages are discussed which were developed within the standardization community, such as ASN.1 and its extensions for the description of distributed systems management, the ODP methodology, and the formal description techniques Estelle, LOTOS and SDL. A comparison of these different methods and languages can only be performed through the consideration of their respective specification concepts. The paper presents the results of a study which compares the concepts found in these different OOADM's and specification languages. The possible future convergence of these different concepts into a more coherent object-oriented framework is also discussed.

Keywords

Objec-oriented analysis and design, distributed systems, protocol engineering, formal methods, open distributed processing

1. Introduction

The development of distributed computer systems is a complex task. In addition to the complexity of the information processing tasks, difficulties arise in connection with concurrency, resource contention, optimal distribution of functions and compatibility between the different system components. In order to get a good understanding about the dynamic behavior of a distributed system, often models of the system are constructed during the early phases of the development process. They may take the form of formal specifications, simulation models or protytyping systems. This allows the validation of the system design before the different parts of the system are implemented. Modelling techniques used for this purpose include Petri nets and so-called Formal Description Techniques (see Section 4.3) which have been developed for the specification of communication protocols and services [Boch 90g].

Recently, the object-oriented approach to the analysis and design of information processing systems has witnessed an overwhelming popularity, as evidenced by the large number of articles and books that have been published. It is suggested that the object-oriented approach has advantages over the traditional methodologies, such as structured analysis and design. However, the object-oriented analysis and design methodology (OOADM) is still in its growing phase and continues to evolve. There are already many popular OOADM's that have been widely circulated [Rumb 91], [Shla 88], [Wirf 90], [Booc 92], [Coad 91], [Cole 93], but no single one has gained general acceptance.

"Given this early state of the development of the OOADM, an organization that is planning for a transition to object-oriented technology faces one critical question: *which OOADM should be chosen?* The answer to this question demands a systematic comparison of the available OOADM's before any commitment can be made to one of them. Unfortunately, the comparison of these methodologies is complicated by the fact that the currently popular OOADM'S are composed of informal descriptions [Cham 92], and each methodology has its own set of definitions of concepts, techniques and notations. A comparison of the methodologies is, therefore, subject to the perceptions and interpretations of the person who performs the task." [Hong 93]. Several such comparisons of OOADM's have already been published [Goor 92], [Hong 93], [Cham 92], [Mano 93].

The purpose of this paper is to give an introduction to the modelling of distributed systems using an object-oriented approach and to discuss the various methods and formalisms that can be used for this purpose. While the OOADM's mentioned above are useful for this purpose, other methods and languages have been specifically developed for this purpose in the area of communications protocols.

After a short characterization of OOADM's in Section 2, we present in Section 3 an example of the development of an object-oriented specification of the OSI directory system. In Section 4, we present various methods and languages which have been developed by the international standardization committees for the description of distributed systems and communication protocols, and their relation with the object-oriented concepts is discussed. In Section 5 finally, we present some results on the comparison of object-oriented methodologies and languages which were obtained within a collaborative research project on object-oriented modelling of distributed systems, called IGLOO. The discussion deals with the question which concepts are required for the complete description of object-oriented systems and which of these concepts are supported by the different methodologies and languages. We close the paper with some conclusions.

2. Object-oriented analysis and design

Much literature exists about the advantages of the object-oriented approach to system analysis, design and implementation [Meye 88]. In general, we believe that the following characteristics of object-orientation are of particular importance:

"Object" and "relationship": In the object oriented approach, the application domain is conceptualized as a set of dynamic entities called objects or object instances. This concept leads to software that is easier to understand because objects naturally correspond to the concepts of the application domain. The set of object instances is structured into a set of object classes. An object class groups objects of the same kind or objects with a similar structure. The properties of the objects within a given class are described by a corresponding object type definition. Object instances are normally related with one another through different kinds of relationships. The notion of entity-relationship diagrams [Chen 76] can be used to show the types of objects and their relationships.

Inheritance of properties: Classes of objects are organized into a class hierarchy, or an inheritance structure, which describes how each object type is related to the other types through specialization or/and a generalization. This inheritance structure facilitates the reuse of specifications and code, which is an important notation in software engineering. The entity-relationship diagrams can be extended to include the representation of inheritance.

Encapsulation: Every object normally offers a fixed set of services to be invoked by other objects. Services are modeled as attributes or operations offered by objects. At a lower level of abstraction, a given object may be seen to be composed of several component objects. This decomposition leads to a second type of hierarchy, called composition or aggregation hierarchy, which can be represented by a "is-part-of" relationship. The operations of a composite object are defined such that the object encapsulates all related operations on its internal components. Using (composite) objects as building blocks for specifications allows hiding of certain aspects which are related to the implementation of the provided operations and the internal structure of the object, as already advocated in [Parn 72].

The **software development cycle**, which traditionally goes through the steps of analysis, functional specification, design and implementation according to the so-called waterfall model, may also require adjustments in order to take advantage of the characteristics of the object-oriented approach.

It is important to note that during the early phases of the object-oriented development cycle the descriptions of the system under study is not complete. In fact, most analysis and design methodologies prescribe the order in which the different aspects of the system specification should be defined. Although the practitioners do not agree on the number and the denomination of the different steps involved in this process, all come up with approximately the same philosophy (for a detailed comparison see [Goor 92], [Hong 93]). In our previous work [Boch 93f], we identified the following four major steps, which can be iterated until a satisfactory design is obtained:

(a) A **preliminary step** consists of the identification of the general problem area, of the specific aspects to be handled, and of the purpose of the expected design; this step is an informal one, but it should lead the designer to a separation of the different aspects included in the application, and also to a precise definition of the intended purposes of the model to be elaborated.

(b) Step 1 (**Domain definition**): The first step of the specification process 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 object types, so-called entities, together with the relationships among them.

(c) Step 2 (**Function definition**): A subsequent step is concerned with the allocation of the functions as operations offered by the objects identified in the previous step; certain functions may also uncover new objects which must be integrated into the application domain.

(d) Step 3 (**Behaviors definition**): The last step is concerned with the definition of the behaviors of the objects types; 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 compositions, and the entire development process should be recursively applied again to each such composition.

3. An example: The OSI directory system

The OSI directory system [CCITT Rec. 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).

We present in this section an example of an object-oriented specification of the directory system following the development methodology mentioned above and using as specification language the language Mondel [Boch 90l] which has been developed for the description of reactive, distributed systems.

Concerning the preliminary phase of the development process, it is to be noted that a relatively detailed system design is already given in the OSI standard X.500. The purpose of our effort was to study how easy this design could be represented using the Mondel language and the object-oriented design methodology mentioned above. In this context we studied the translation of certain formalized notations used in the OSI standard into Mondel and developed an executable Mondel specification which could be used for the validation of the design or as a prototype. Further details can be found in [Boch 93f].

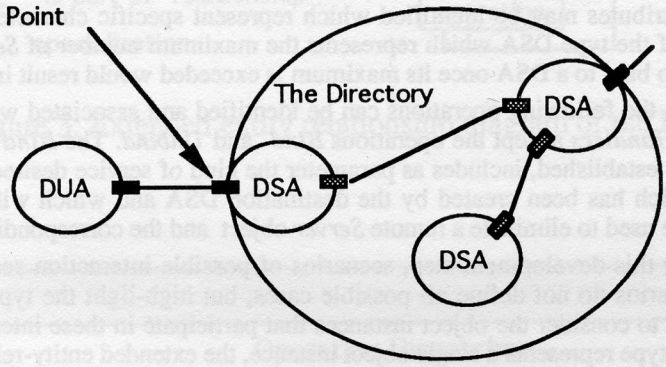

Figure 1: Subsystems in the OSI directory system [X.500]

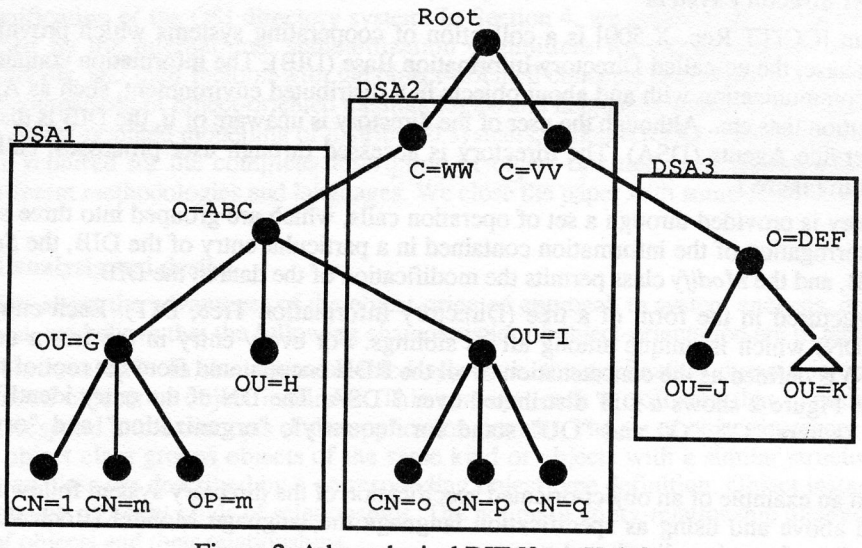

Figure 2: A hypothetical DIT [from X.518]

During Step 1 of domain definition, we identified the entities and relations shown in Figure 3, which uses an easily readable graphic notation. A DSA is composed of a single connection handler (called *BindHandler*) and a *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 domain definition, object attributes may be identified which represent specific characteristics of an object. An example is the attribute *maxServer* of the type DSA which represents the maximum number of *Server* objects that can be supported at any given time. Trying to bind to a DSA once its maximum is exceeded would result in an error condition.

During Step 2 of function definition, 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 *Server* object which has been created by the destination DSA and which will provide the requested service. The operation *Unbind* can be used to eliminate a remote *Server* object and the corresponding connection.

It is often useful to consider, during this development step, scenarios of possible interaction sequences that may occur during system execution. Such scenarios do not define all possible cases, but high-light the typical situations that may occur. In this context, it is important to consider the object instances that participate in these interaction scenarios. In the more simple cases where each object type represents a single object instance, the extended entity-relationship diagram (such as Figure 3) may be used to identify the object instances. In cases where it is important to distinguish the different object instances that belong to the same type, an more detailed "instance diagram" may be use (for an example, see [Boch 93f]).

During Step 3 of behavior definition, we had to describe the behavior of the DUA and DSA connection handler and the different specializations of the *Server* type. Each operation of the *Server* types 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 so-called *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. These actions are represented in the X.500 standard as shown in Figure 4. The corresponding representation in Mondel is discussed in [Boch 93f].

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 requires a more detailed representation of the *DIBFragment* object, including the structure of the DIT and related information. This leads to a recursive application of the development steps in order to represent the *DIBFragment* in terms of the contained component objects, their operations and respective behavior, as explained below.

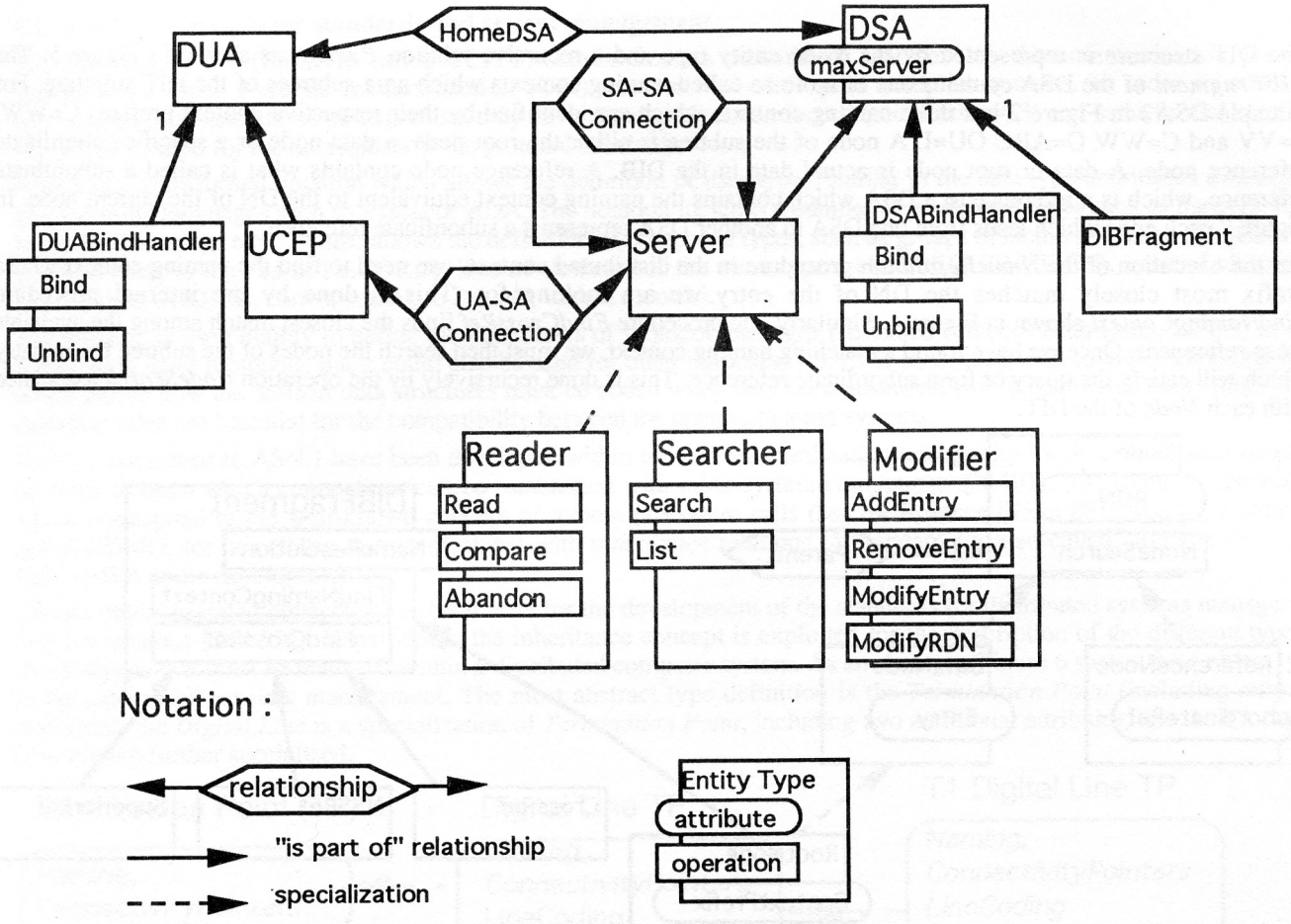

Figure 3: Extended entity relationship diagram of the Directory System

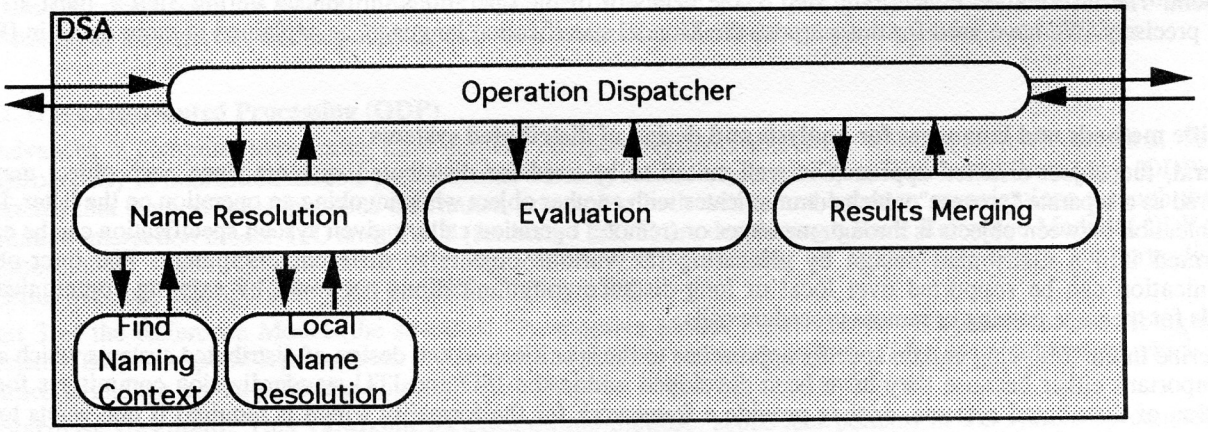

Figure 4: DSA behavior - internal view [from X.118]

The DIT structure is represented by the *Node* entity type and a recursive relation *Parent*, as shown in Figure 5. The *DIBFragment* of the DSA contains one or more so-called naming contexts which aqre subtrees 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 4. 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.

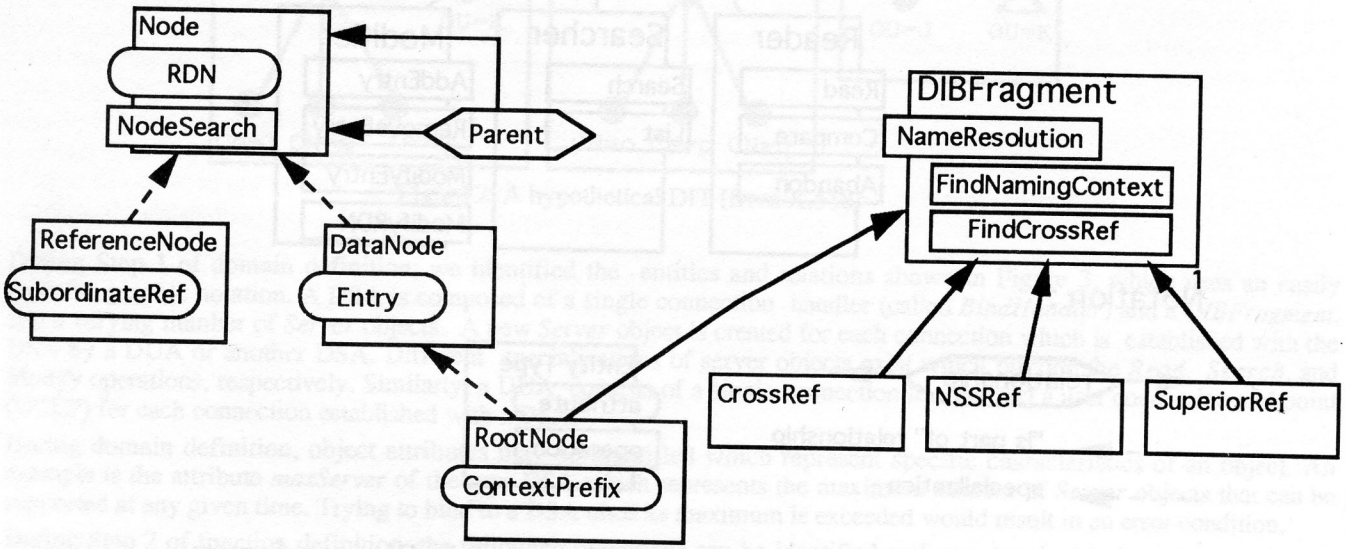

Figure 5: Information held in a DSA

We conclude by noting that Figure 5 contains the information concerning Step 1 and Step 2 of the *DIBFragment* refinement. The information concerning Step 3, the behavior of the operations introduced during Step 2, must also be defined precisely. We have used for this purpose the Mondel specification language, as explained in more detail in [Boch 93f].

4. Specific methods and languages for analysis and design of distributed systems

In general, the object-oriented approach is well suited to be used for distributed systems since each object may be considered as a separate "process" which communicates with another object when invoking an operation on the latter. If the communication between objects is through messages or (remote) operation calls, a given system specification can be easily transformed into a distributed design by allocating the various objects to different computers. The inter-object communication can be supported over local or long-distance communications networks by existing communication protocols for message passing or remote procedure calls.

We describe in the following certain specific approaches to the specification and design of distributed systems which seem to be important either because they have been developed within the ISO and ITU standardization committees for the description of distributed systems, or they provide a framework for the formality which is required if one wants to use automated tools for the validation of the specifications and for implementation development (for further discussion of these issues, see for instance [Boch 90g]).

4.1. OSI Application layer standards and system management

The ISO and ITU-T (formerly called CCITT) standardization committees that develop the communication protocol standards for Open Systems Interconnection (OSI) have to deal with quite complex systems. In order to make the descriptions of the standards more precise, they have developed certain formal (see Section 4.3) and semi-formal notations which can be used for the description of the protocol standards.

ASN.1 is a semi-formal 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), lists 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 the parameters of protocol messages (so-called Protocol Data Units, PDU's) of OSI Application layer protocols, but it could also be used for other purposes. ASN.1 is associated with so-called encoding rules which define how the defined data structures must be coded when they are transmitted over a communications link. These encoding rules are essential for the compatibility between the communicating systems.

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" which correspond to the well-known concept of remote procedure calls (see for instance [Tann 88]), and (c) a notation, called GDMO, for describing classes of object with inheritance relations. A standard for a so-called "Remote Procedure Call" is also under development.

Certain object-oriented concepts have been used for the development of the standards for distributed systems management (see for instance [Stal 93]). In particular, the inheritance concept is exploited for the description of the different types of (real) objects that must be managed within a distributed computer system. As an example, Figure 6 shows three object types in the context of network management. The most abstract type definition is the *Termination Point* (including only two attributes), the *Digital Line* is a specialization of *Termination Point*, including two additional attributes, and the *T1 Digital Line* is even further specialized.

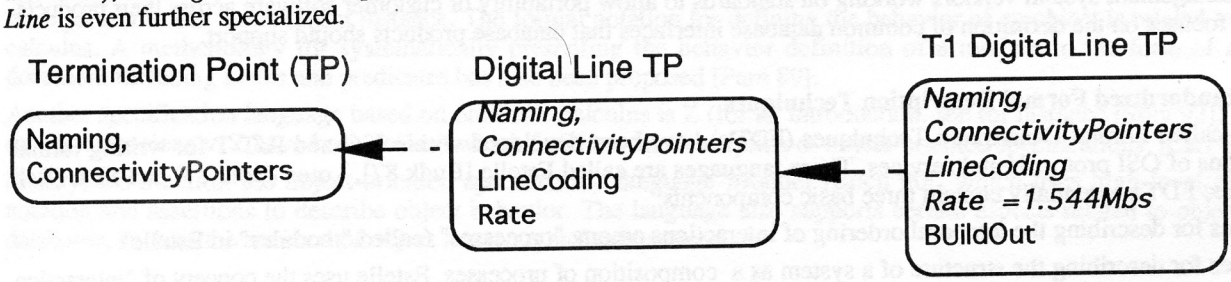

Figure 6: Three types of managed objects: An example of inheritance

The properties of the managed objects concerning different functionalities of system management such as shown in Figure 6, are described using the GDMO notation. However, it is to be noted that this notation includes no formalism for the definition of the behavior of these objects, only their attributes and offered operations are defined formally.

4.2. Open Distributed Processing (ODP)

"Advances in computer networking have allowed computer systems across the world to be interconnected. Despite this, heterogeneity in interaction models prevents interworking between systems. Open Distributed Processing (ODP) describes systems that support heterogeneous distributed processing both within and between organizations through the use of a common interaction model. ISO and ITU-T (formerly CCITT) are developing a Basic Reference Model of ODP to provide a coordinated framework for the standardization of ODP by creating an architecture which supports distribution, internetworking, interoperability and portability." [Raym 93]

Part 3 of the Reference Model (the so-called "Prescriptive Model") defines five viewpoints for describing distributed systems. Each viewpoint represents an abstraction and focuses on certain aspects of the specified system which may be of particular interest from a particular poinrt of view. They can be characterized as follows:

Enterprise viewpoint: This viewpoint focusses on the purpose, scope and policies of the system by considering such concepts as agents, artifacts (passive objects), communities (classes of objects belonging together), roles (which include permissions, prohibitions and obligations).

Information viewpoint: This viewpoint focusses on the semantics of information and information processing activities by considering such concepts as schema, relations and integrity rules. There are three types of schema:

(1) A static schema corresponds largely to the commonly used entity-relationship diagrams used for the analysis of information systems and is similar to the domain definition step of the development methodology mentioned in Section 2.

(2) An invariant schema defines contraints that must be satisfied at all times between the actions that are executed within the system. They correspond to the integrity constraints of database systems.

(3) A dynamic schema specifies possible state changes of an object or a group of objects.

Computational viewpoint: This viewpoint focusses on the functional decomposition of the system into objects which are candidates for distribution. In this view the interfaces between the different objects are defined. A distinction between operational, transactional (a special type of operational interface), and stream interfaces is made. The latter type of interface is concerned, for instance, with stream communication for continuous media, such as voice and video.

Engineering viewpoint: This viewpoint focusses on the infrastructure required to support distribution and introduces concepts related to the allocation of resources, the communication and binding between objects, and object migration.

Technological viewpoint: This viewpoint focusses on the choice of technology to support the system, such as computer hardware, networks, operating systems, and software modules.

Among these viewpoints, the Information, Computational and Engineering viewpoints are the most elaborated. Part 2 of the Reference Model describes object-oriented concepts and notations to be used for the descriptions of distributed systems according to these viewpoints.

In addition, the Reference Model on ODP describes a trader object which provides the so-called trading function. This function allows a server object to advertize its service by exporting it to the trader, and allows a potential client to request a required service from the trader. The trader will try to match the request with a service offer in the trader's database. If a successful match has been found, the client may obtain the requested service by communicating directly with the identified service provider.

It is interesting to note that many of the ODP concepts are taken up by the industrial consortia Object Management Group (OMG) and Object Database Management Group (ODMG). The latter is an informal consortium of object-oriented database management system vendors working on standards to allow portability of customer software across their products. The group focuses on the definition of common database interfaces that database products should support.

4.3. Standardized Formal Description Techniques

Several so-called Formal Description Techniques (FDT's) have been developed within ISO and IUT-T for writing formal specifications of OSI protocols and services. These languages are called Estelle [Budk 87], Lotos [Bolo 87] and SDL [Beli 89]. 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 system 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 the interconnection of interaction points of two or more processes.

(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 assignable variables, but including process and interaction parameters) and uses abstract data types.

In the following we discuss how the object-oriented 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 90c]). The same 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, there is no support for the concept of classes for gates. Various ways to introduce the concept of objects into Lotos have been discussed in the literature [Cusa 90].

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 no explicit concept of specialization. The situation is similar for SDL, although some form of specialization is available for abstract data types. Lotos also provides such a facility, and in addition allows for the use of the so-called constraint-oriented specification style

[Viss 89] which allows the definition of specializations of behaviors by specifying additional constraints about the allowed order of interactions.

It is to be noted that SDL has recently been extended to include a number of object-oriented features in its 1992 version. The extension includes operation calls, 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 since 1967.

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 in addition 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 90a]. 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.

4.4. Other Object-Oriented Specification Languages

In the context of programming languages, the definition of behaviors of operation (Step 3 of the development methodology described in Section 2) is often considered an implementation issue. But it is important to note that a complete specification of a system component must include the definition of its behaviors. Besides the FDT's discussed above, there are a number of other specification languages that provide facilities for the definition of behaviors. In the following, we mention a few such languages that are object-oriented.

The specification formalism called Traces [Hoff 88] 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-based 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 [Parn 89].

Another specification language based on predicate calculus is Z (for an introduction, see for instance [Spiv 92]). Recently, certain extensions to Z have been defined which facilitate the description of object-oriented specifications [Carr 89].

Finally, we mention the object-oriented specification language Mondel [Boch 90l] which uses high-level algorithmic notation and assertions to describe object behavior. The language also supports certain aspects related to object-oriented databases, such as the retrieval of object instances and transactions.

5. Common concepts for object-oriented modelling

5.1. Comparison of concepts in the IGLOO project

The IGLOO project is an ongoing research collaboration between the Centre de Recherche Informatique de Montréal (CRIM), three universities (Université de Montréal, UQAM and Université de Sherbrooke), and six industries (BNR, CAE, DEC, Machina Sapiens, Servacom, Teleglobe). The research activities of the IGLOO project are divided into four axis, namely (1) object-oriented analysis and design methodologies, (2) implementation development tools, (3) software reuse, and (4) applications in distributed systems management. The direct costs of this research are funded by the industrial participants and the Quebec Government through the so-called "Synergie" program.

Whereas much research on object-oriented methods has been done in the past in relation with object-oriented programming and in the context of sequential progamming languages, the objective of the IGLOO project is to advance the state of the art in the earlier stages of the software development cycle, specifically in requirements analysis and specification development, and to give special attention to the modelling of active object behaviors and concurrency, aspects that are important to the specification of reactive systems and real-time applications. The main results of the project are expected to be new methods and approaches, and related prototype tools, for developing and validating specifications, for developing prototype implementations from formal specifications (including support from object-oriented databases and providing user-friendly interfaces), for preparing, storing, and retrieving reusable components, and for building network management applications.

Common concept / Notation	1	2	3	4	5	6	7	8	9	10	11	12	13
2.1.1 - what is an object / general	A	A	A	A+	A-	A	A		A	A	A	A	
2.1.1.1 - object with state	A	A	A	A+	A	A	A		A	A	A	A	A
2.1.1.2.- object representation		A-	A	A-	-	A	A	A-	A	A		A-	A
2.1.2.1 - object identity	A-		A	A	-	A	A		A	A	A-	A	
2.1.2.2 - equality / equivalence					-		A		A	A	A		
2.1.2.3 - shallow equality					-		A						
2.1.2.4 - properties of equality					-		A						
2.1.3 - operations and attributes: "na" no attributes; "a" attributes , "nd" no dist.	a	a	nd	a	a	a	a	nd	a	a	a	a	a
2.1.4.1 - immutable objects				A	-	A-	A				A-		A-
2.1.4.2 - distinguish side-effects		A	A	A-	-	A-	-		A	A		A	
2.1.4.3 - active objects explicitly supported	A+				-		-		A	A+	A-		
2.1.4.4 - parallelism within the same object	A+	A	A-		-	A-	-		A	A	A-		
2.2.1 - definition of "type"	A	A-			A-	A	A+		A	A	A	A	A
2.2.2.1 - operation signature	A-	A	A+		A	A	A		A	A	A	A	A
2.2.2.2 - object signature		A	A		-		A		A	A		A+	A
2.2.2.3 - relationship signature			A		-		A						
2.2.2.4 - parameterized types	A-				-	A	A		A	A	A	A	
2.2.3.1 - definition of "subtyping"	A-	A	A-		A-	A	A		A+	A	A+	A	A
2.2.3.2 - syntactic subtyping					-		A		A	A			A
2.2.3.3 - semantic subtyping		A-			-		A		A+	A-			
2.2.3.4 - implementation subtyping					A		A						
2.2.3.5 - explicit subtyping				A	-	A	A		A	A	A	A	
2.3.1 - definition of "relationship"	A+	A-	A+	A	-		A-	A+			A-	A	
2.3.2 - elements describing relationships	A	A-	A+	A+	-		A-	A			A-	A	
2.4.1 - approach to describing behavior: "s" state machine model, "alg"orithmic, "ax"iomatic, "all"	s	s	s	s	-		-	??	ax	alg, ax	s	ax	alg, ax
2.4.4.1 - object creation and deletion	A+			A	-	A	-		A	A	A	A	
2.4.4.2 - access to (search through) persistent objects	A			A	-	A	A-		A-	A		A	
2.4.4.3 - behavior relating to other objects		A		A	A	A	-			A	A	A	A+
2.4.4.4 - kind of inter-object communication: "m"essages, "rpc" (remote) procedure calls, "c" conditions depending on other object's state, "r"endezvous	m, r	m		m	-	c	-			rpc, r	rpc, m	?	rpc, s
2.4.4.5 - behavior supporting relations				A	-	A	A-						
2.4.5.1 - interpretation of "illegal traces": "b"locking, "i"gnored input, "u"ndefined behavior, "s"equential behavior ,any order	b, u	d		b	-		-			b, u		s	u
2.4.5.2 - nondeterminism: "d" systems are assumed to be deterministic, "o"utput nondeterminism, "s"tate nondeterminism (can be detected through blocking tests)	s			d	-		-		o	o, s	o,s	s	d
2.4.5.3 - semantic model with real paralellism: blank means interleaved operations		A			-	A	-		A	A	A		
2.4.5.4 - liveness properties supported					-		-					A-	A+
2.5 - interobject behavior				A+	A+	A+	-			A+			A

In the context of this project, we first did a review of existing object-oriented analysis and design methodologies (OOADM's, see Introduction) as well as certain programming languages (such as Smalltalk, C++, Eiffel) and specification languages for distributed systems [Dini 93a]. Then a study was performed to identify the common concepts that underlie the different OOADM's and specification languages, and which allow the specification of those properties of module interfaces that are required for supporting the reuse of module specifications and implementations [Boch 94b]. An overview of the obtained results is shown in the enclosed table. (Note: The colomns refer to the following notations: 1 - Booch, 2 - Object Charts, 3 - Rumbaugh, 4 - Shlaer-Mellor, 5 - Wirf-Brock, 6 - ODP methodology, 7 - ODMG, 8 - IUT Man-Machine-Interface language, 9 - LarchC++, 10 - Mondel, 11- SDL, 12 - Z and object-oriented Z; 13 - Contracts). These results should be helpful to facilitate the interworking between software components that have been developed using different specification paradigms, for instance class libraries, object-oriented databases, network management applications using standardized object views of the managed objects and others.

During a subsequent phase, it is intended to define the correspondence between the concepts and the notations used in certain important design methods and specification languages in order to facilitate the integration of tools originally developed for these different existing notations. For this purpose, it is necessary to provide precise definitions of the relevant concepts and develop a formalization of their semantics.

5.2. Partial views

During different phases of the system development process, and for different purposes, various partial views of a complete system specification may be considered. For instance, the 3-step methodology mentioned in Section 2 considers during the first step a partial view containing only the object types and relationships, during the second step the names and syntactic signatures of the operations are included, and during the third step the semantics of these operations (i.e. the object behavior) is added. Another example is the partial behavior specification in the form of scenarios (using for instance the notation of Message Sequence Charts standardized by ITU-T) during the analysis phase which must be completed during a later phase in order to obtain a complete functional specification. We consider the following partial views:

(1) **Object types**: This includes the concepts: type name, informal meaning of the objects, object identity, object equality, object state, immutable objects, and possibly active and parameterized object types (for further details, see [Boch 94b]).

(2) **Relationship types**: This view requires the definition of the object types and, in addition, includes the concepts: relationship name, informal meaning of relationship, roles, and possibly relationship signature and multiplicities.

(3) **Operations**: This view requires the definition of the object types and, in addition, includes the concepts: operation name, informal meaning of operation, parameters, returned results, presence or absence of side-effects, and possibly operation signature, object signature, syntactic subtyping and explicit subtyping.

(4) **Object behavior:** This view requires the definition of the operations and includes, in addition, the concept of "behavior" which may be described using one of the following approaches:

(a) Simple state machine model: The behavior is defined in the form of a standard finite state machine, or in some extended notation, such as for instance State Charts [Hare 87].

(b) Model with abstract variables/objects (extended state machine model): The "state" of the object is determined by the values of several (abstract) variables. The state determines whether certain operations are possible or not. The simple state machine model under (a) may be considered a special case where the object contains only a single "state" variable. The model may be used for objects with "sequential" behavior (i.e. each operation may be executed in any state) as well as for objects with sequencing constraints, such as expressed by the simple state machine model. There are two variants to this style:

(b1) Algorithmic style (e.g. Mondel language)

(b2) Axiomatic style (e.g. Z)

Additional concepts for the object behavior are: non-determinism, semantic subtyping, parallelism within an object, refusal semantics, and liveness.

(5) **Inter-object behavior:** This view requires the definition of the behavior of single objects and their relationships, and includes the additional concepts: object interactions (e.g. message passing, remote operation calls, rendezvous interactions), behavior relating to other objects, behavior supporting relations, data flow between different objects, and possibly object creation and deletion, access to persistent objects, liveness and fairness, as well as object interfaces/ports.

5.3. Discussion of comparisons

As the summary of the table shows, the views of object types and operations are supported by practically all notations. The situation is different for the other views. For instance, the view of relationship types seems to be best supported by the Rumbaugh methodology.

Concerning the object behavior view, it is either supported only informally, or through a specific style. Several analysis and design methodologies support the state machine model. The algorithmic style is supported by Mondel and SDL, the axiomatic style by Larch and Z. Contracts [Helm 90], a notation for the specification of inter-object behavior, use mainly the axiomatic style, but their sequencing aspect is rather algorithmic.

Many methodologies use the State Charts approach for the description of object behavior. Some of them do not support parallel state charts (with rendezvous coupling, or with state-conditional constraints). Most systems support embedded (hierarchical) charts. Several of these systems use "extended state charts" including local abstract variables. One of the open issues is how to formalize the description of the transitions (as far as updating the local state and interactions with other objects is concerned). An algorithmic or axiomatic style may be used for this purpose.

While some languages only cover the behavior of single objects (e.g. Larch), different approaches to inter-object behavior have been proposed. One particular issue in this context is the identification of the interacting object instances. This issue can be addressed through different approaches, such as the following:

(a) Abstract variables "pointing" to other objects (like in object-oriented programming languages; e.g. SDL, Mondel)

(b) Interconnection of ports (e.g. Estelle channels, SDL channels and the the SDL "VIA" construct, Object-Time)

(c) Inter-object behavior based on relationships (e.g. so-called "calling relationships"; see also [Kurk 91])

It may be feasable to combine the best approaches for the different views, as supported by existing notations, in order to obtain some kind of "super method" which supports all the different views in a coherent manner. This is the subject for future research.

6. Conclusions

A large number of object-oriented analysis and design methodologies (OOADM's) have been proposed in the literature and many specification languages support object-oriented specifications. Within the IGLOO project, we tried to identify the key concepts underlying these methods and languages and to determine whether a common set of concepts can be identified which is supported by most (if not by all) methodologies and languages.

It turns out that the concepts related to the meaning of an object and of operations are quite uniformly supported by the different existing approaches. The concepts related to relationships are also relatively uniform, but are only supported by a subset of the methodologies and not explicitly by the languages. The concepts related to the description of the behavior of objects are less uniform and are not supported by all approaches. Different specification styles have been proposed for the behavior of objects: state machines (e.g. State Charts) or model-based descriptions with variables using an algorithmic or axiomatic approach.

It can be expected that future developements will lead to a better integration of the different aspects of specifications, which include (a) the traditional aspects of state machine, data flow graphs, and entity-relationship diagram defining the object domain, and (b) the specific "object-oriented" aspects of encapsulation, specialization, and dynamic object creation and deletion.

It is important to note that the OOADM's and their specification concepts should not be seen in isolation. These concepts should be easily translatable into the concepts of object-oriented implementation languages, databases for the development of implementations, and into formal specification languages for the validation of system specifications and designs. In fact, only formal specifications allow the systematic validation of the system description through simulated execution of the specification or through the use of verification tools that consider automatically all possible interaction scenarios exhaustively [Pehr 90]. The concepts should also be closely related to the notations used for the standards of communication protocols and other distributed systems in order to simplify the implementation of the latter.

As the OOADM's are still in the growing phase and continue to evolve, it can be expected that some convergence of the concepts and notations used for the object-oriented description of distributed systems will occur in the coming years. Such convergence is important to avoid a duplication of efforts. It will also stimulate the development of automated tools, not only for the editing of nicely looking specification diagrams, but also for the validation of specifications and the development of conforming implementations.

Acknowledgements

This content of this paper is based on collaboration with many people. I would like to thank in particular M. Barbeau, P. Dini, R. Dssouli, P. Mondain-Monval, S. Poirier, J.M. Serre and A. Vogel for the discussions and joint work that led to this article. I would also like to thank all collaborators in the IGLOO project for their contributions and in particular the co-authors of the method comparison, namely P. Dini, M. Barbeau, P. Colagrosso, R. Keller, D. Ramazani and S. Somé. This work is supported by the IGLOO project (see Section 5.1).

References

[Beli 89] F. Belina and D. Hogrefe, The CCITT-Specification and Description Language SDL, Computer Networks and ISDN Systems, Vol. 16, pp.311-341, 1989.

[Boch 90a] G. v. Bochmann, Specifications of a simplified Transport protocol using different formal description techniques, Computer Networks and ISDN Systems, Vol. 18, no.5, June 1990, pp. 335-377.

[Boch 90g] G. v. Bochmann, Protocol specification for OSI, Computer Networks and ISDN Systems 18 (April 1990), pp.167-184.

[Boch 90l] G. v. Bochmann, M. Barbeau, M. Erradi, L. Lecomte, P. Mondain-Monval and N. Williams, Mondel: An object-oriented specification language, publication départementale no. 748, Dépt. IRO, Université de Montréal, 1990.

[Boch 94b] G. v. Bochmann, P. Dini and e. al., Common concepts for object-oriented analysis and design, Deliverable M.a.2, IGLOO project, Centre de Recherche Informatique de Montreal, Jan. 1994.

[Boch 93f] G. v. Bochmann, S. Poirier and P. Mondain-Monval, Object-oriented design for distributed systems: The OSI directory example, to appear in Computer Networks and ISDN Systems.

[Bolo 87] T. Bolognesi and E. Brinksma, Introduction to the ISO Specification Language Lotos, Computer Networks and ISDN Systems, vol. 14, no. 1, pp.25-59, 1987.

[Booc 92] G. Booch, The Booch Method: Notation, Computer Language, Sept. 1992, pp. 47-70 and Oct. pp. 37-55.

[Budk 87] S. Budkowski and P. Dembinski, An introduction to Estelle: a specification language for distributed systems, Computer Networks and ISDN Systems, vol. 14, no. 1, pp.3-23, 1987.

[Carr 89] D. Carrington, D. Duke, R. Duke, P. King, G. Rose and P. Smith, Object-Z: An object-oriented extension to Z, Proc. FORTE'89 (Vancouver), North Holland Publ.

[Cham 92] D. d. Champeaux and P. Faure, A comparative study of object-oriented analysis methods, Journal of Object-Oriented Programming, March/April 1992, pp. 21-33.

[Chen 76] P. P. Chen, The Entity-Relationship model - Toward a unified view of data, ACM Trans. on Database Systems, Vol. 1, No. 1, March 1976, pp.9-36.

[Coad 91] P. Coad and E. Yourdon, Object Oriented Analysis, Yourdon Press, Englewood Cliffs, N.J. 1991.

[Cole 93] D. Coleman and e. al., Object-Oriented Development: The FUSION Method, Prentice Hall, OO Series, 1993.

[Dini 93a] P. Dini and e. al., Methods for object-oriented analysis and design: A survey, Technical Report, CRIM (Deliverable M.a.1.1 of IGLOO project), 1993.

[Goor 92] G. v. d. Goor, S. Hong and S. Brinkkemper, A comparison of six object-oriented analysis and design methods, Technical report, Twente University, Enschede, the Netherlands.

[Hare 87] D. Harel, Statec harts: A visual formalism for complex systems, Science of Computer Programming 8, 19987, pp. 231-274.

[Helm 90] R. Helm, I. M. Holland and D. Gangopadhyay, Contracts: Specifying behavioral composition in object-oriented systems, Proc. OOPSLA/ECOOP'90, Ottawa, Oct. 1990, pp. 169-180.

[Hoff 88] D. Hoffman and R. Snodgrass, Trace specifications: Methodology and models, IEEE Tr. SE 14, No. 9 (Sept. 1988), pp. 1243-1252.

[Hong 93] S. Hong, G. v. d. Goor and S. Brinkkemper, A formal approach to the comparison of object-oriented analysis and design methodologies, Proc. HICSS-26 (Hawaii), IEEE Computer Society Press, 1993.

[Kurk 91] R. Kurkio-Sunio, Stepwise design of real-time systems, Proc. ACM Conf. on Software for Critical Systems, Software Eng. Notes, Vo. 16, 5 (Dec . 1991), pp. 120-131.

[Mano 93] F. Manola(Editor), X3H7 Object Model Feature Matrix, Draft standardization document of ANSI X3H7, Nov. 1993.

[Meye 88] B. Meyer, Object Oriented Software Construction, C.A.R. Hoare Series Editor, Prentice Hall, 1988.

[Parn 72] D. Parnas, on the criteria to be used in decomposing systems into modules, Comm. ACM, Vo. 15, No. 2 (1972), pp. 1053-1058.

[Parn 89] D. L. Parnas and Y. Wang, The Trace Assertion Method of Module Interface Specification, Technical Report 89-261, Queen's University, Kingston, Canada; submitted to IEEE Transactions on Software Engineering.

[Pehr 90] B. Pehrson, Protocol verification for OSI, Computer Networks and ISDN Systems, Vol. 18 (1989/90), pp. 185-201.