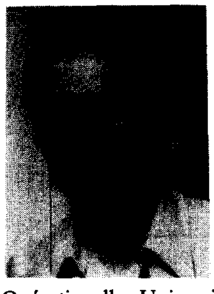
Protocol Specification for OSI *

Gregor v. BOCHMANN
Département d'informatique et de recherche opérationnelle, Université de Montréal, Montréal, Quebec, Canada H3C 3J7

Abstract. The collection of Open Systems Interconnection (OSI) standards are intended to allow the connection of heterogeneous computer systems for a variety of applications. In this context, the protocol specifications are of particular importance, since they represent the standards which are the basis for the implementation and testing of compatible OSI systems. This paper has been written as a tutorial on questions related to protocol specifications. It provides certain basic definitions related to protocol specifications and specification languages. Special attention is given to the specification formalisms used for OSI protocol and service descriptions, including semi-formal languages such as state tables, ASN.1 and TTCN, and formal description techniques (FDTs) such as Estelle, LOTOS, and SDL. The presentation is placed within the context of the general protocol and software development life cycle. An outlook to available methods and tools for partially automating the activities during this cycle is given, and ongoing research directions are discussed.

Keywords. Communication protocols, specifications, open systems interconnection, formal specifications, specification languages, protocol validation, conformance testing, protocol design, formal description techniques, protocol standards.



Gregor v. Bochmann (M'82-SM'85) received the Diploma degree in Physics from the University of Munich, Munich, West Germany, in 1968 and the Ph.D. degree from McGill University, Montreal, P.Q., Canada, in 1971.
He has worked in the areas of programming languages, compiler design, communication protocols, and software engineering and has published many papers in these areas. He is currently a Professor in the Département d'Informatique et de Recherche Opérationelle, Université de Montréal, Montréal. His present work is aimed at design models for communication protocols and distributed systems. He has been actively involved in the standardization of formal description techniques for OSI. From 1977 to 1978 he was a Visiting Professor at the Ecole Polytechnique Federale, Lausanne, Switzerland. From 1979 to 1980 he was a Visiting Professor in the Computer Systems Laboratory. Stanford University, Stanford, CA. From 1986 to 1987 he was a Visiting Researcher at Siemens, Munich.

* Work supported in part by the Natural Sciences and Engineering Research Council of Canada.

1. Overview

1.1. Introduction

The interworking between the different components of a distributed system is controlled by the protocols used for the communication between the different system components. These components must be compatible with one another, that is, satisfy the defined communication protocols. In order to facilitate the implementation of compatible system components, it is important to have a precise definition of the communication protocol to be used. The protocol specification is used for this purpose.

A collection of standards of communication protocols and services are being developed for Open Systems Interconnection (OSI) [53] which is intended to allow the interworking of heterogeneous computer systems for a variety of applications. In this context, the protocol specifications are of particular importance, since they represent the standards which are the basis for the implementation and testing of compatible OSI systems. The standard specifications are usually written in natural language, augmented with certain formalisms. In addition, formal description techniques have been developed for application to OSI (see Section 3) and have been used for the description of certain OSI protocols and services. The formal nature of the specifications make it possible to apply certain automated tools during the protocol development life cycle.

This paper has been written as a tutorial on the questions related to protocol specifications. This first section provides some definitions and addresses some general questions related to protocol specifications and specification languages. It is important to note that specifications are not only important for communication protocols, but also in general for software and hardware development. Therefore the system development life cycle is discussed in Section 2, first in general terms and then in the specific context of protocol and service specifications. Section 3 gives an introduction to the various specification formalisms that are used for OSI protocol and services. Section 4 provides an overview of the kinds of tools that may be used for automating some of the activities of the protocol development cycle, based on formal and semi-formal specifications. This is a very broad subject area which is covered in more detail in other publications. Section 5, finally, gives an outlook on ongoing research related to specification languages and their use, and suggests certain areas which seem to be of particular importance to protocol specifications.

1.2. *What is to Be Specified?*

Communication protocols within a distributed system are usually organized in a hierarchy of several layers. As an example, Fig. 1 shows the 7-layer OSI architecture [26]. In this context, specifications are required for the following objects:

(a) The *communication service* of a particular protocol layer. This is the behavior of the black box shown in Fig. 2, including *local* properties pertaining to a single service access point and *global* properties relating the interactions at different access points,

(b) The *protocol* to be followed by the protocol entities of a given protocol layer. This is the required behavior for the protocol entity, represented by the black box shown in Fig. 3, and

(c) The *interface* through which the communication service is provided to the user at a particular service access point.

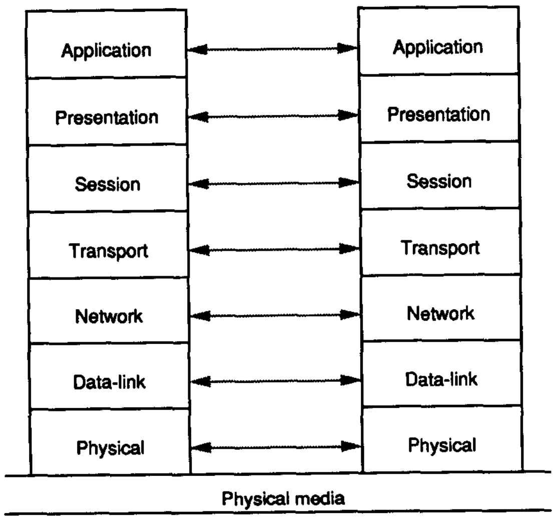


Fig. 1. Layered OSI protocol architecture.

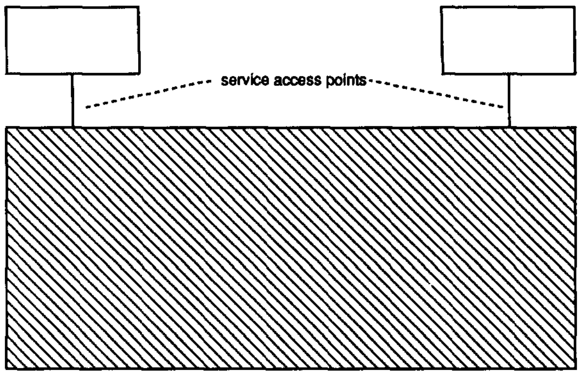


Fig. 2. Communication service.

The specifications for the communication services and protocols have a global relevance. The interface properties, on the other hand, could be different for different service access points; in each case, however, they must be consistent with the local properties included in the corresponding service specification.

The primitive interactions between a protocol entity providing a service and a service user, at a high level of abstraction, are called *service primitives* [26]. They are invoked at the service access points between the user and the communication service. In the more detailed view of the protocol specification (see Fig. 3) the protocol entity plays the role of the communication service to be provided; at the same time, it is the user of the communication service below. Each service primitive usually includes several parameters which are exchanged between the service user and the protocol entity during the execution of the service primitive. Some of these parameters may represent *user data* which is transmitted by the communication service to the remote user without interpreta-

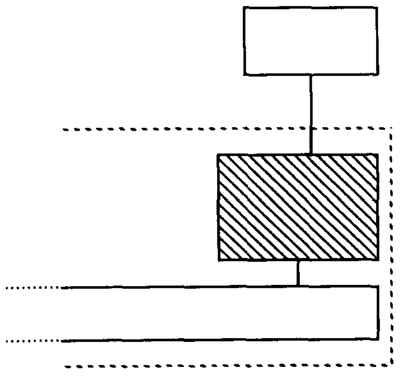


Fig. 3. Communication protocol entity.

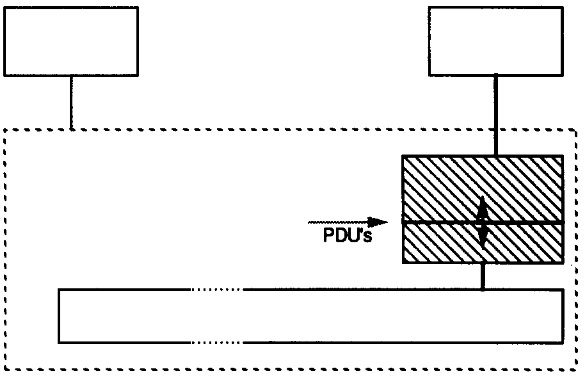


Fig. 4. Communication protocol entity: role of PDUs.

tion. The interactions exchanged through the underlying communication service between the peer protocol entities are called *protocol data units* (PDUs). It is important to note that abstract properties of a protocol entity are often described in terms of the exchange of PDUs with the peer entity and service primitives with the user. However, the PDUs are not exchanged directly with the peer but by coding the PDUs and their parameters in the user data exchanged through the underlying communication service. This is indicated in Fig. 4, were the upper part of the box representing the protocol entity corresponds to its abstract properties, and the lower part to the mapping of the PDUs to the service primitives of the communication service below.

As an example, we consider in the following the OSI Transport layer. Figure 5 shows a system including two Transport entities providing the Transport service through the access points *T*1 and *T*2, and communicating through the underlying Network service using the service access points *N*1 and *N*2, respectively. Figure 6, called a *time sequence diagram*, shows an example of possible exchanges of Transport service primitives and Transport PDUs.

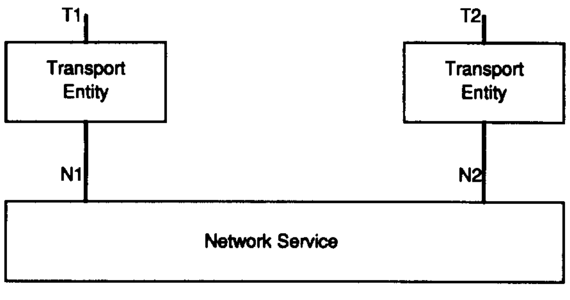


Fig. 5. Architecture of system providing the transport service.

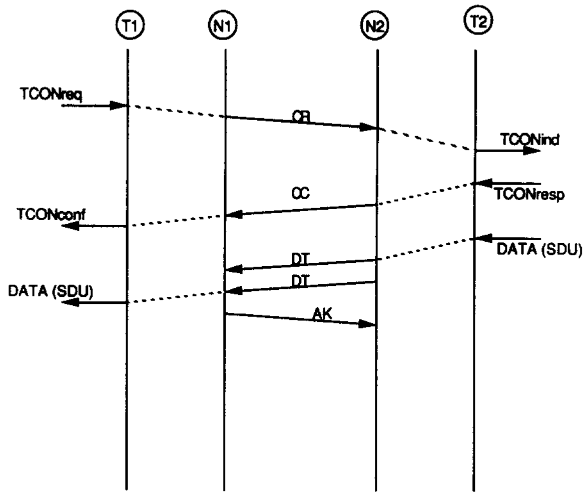


Fig. 6. Time sequence diagram showing an example sequence of interactions.

Figure 6 is useful for providing an understanding of the basic exchanges leading to the establishment of a Transport connection between two users of the Transport service. The left side is called "initiator" and the right side "responder". The initiating user invokes a *TCONreq* (Transport connection request) service primitive and waits for a confirmation. The initiating Transport entity sends a *CR PDU* and waits for return of a *CC PDU*. After this exchange, the connection is established and data can be sent in both directions. As shown in the figure, one "service data unit" (SDU) provided by the user may be fragmented into several *DT PDUs* for transmission between the two protocol entities.

1.3. Behavior Aspects to Be Specified

The specification of a communication service, a protocol, or another system component should define all the relevant behavior aspects of the specified object, but nothing more. The other aspects should be left undefined, such that they could be chosen to fit particular implementation requirements.

For the subsequent discussion, it seems useful to distinguish the following aspects of behavior (the term "interaction" stands for service primitives and PDUs):

(a) *Temporal ordering of interactions*: This aspect defines the order in which the different interactions may be executed. This includes the distinction between the different types of service primitives and PDUs, and the allowed temporal ordering for these interactions. A distinction is also made whether the interaction is initiated by the protocol entity or "received" from its partner.

(b) *Range of possible values for interaction parameters*: This is the range that these parameters of service primitives and PDUs may take.

(c) *Rules for interpreting and selecting values of interaction parameters*: This aspect represents a refinement of aspect (b); it defines (i) for each interaction that may be initiated by the protocol entity, the allowed parameter values as a function of the interaction history of the entity, and (ii) for each interaction received, the significance of the received parameter values for the subsequent processing by the protocol entity.

(d) *Coding of PDUs*: While aspect (c) defines possible parameter values for the exchanged PDUs, this aspect defines the format in which the PDUs and their parameters are coded for transmission through the underlying communication service.

(e) *Liveness properties*: Liveness properties are defined in contrast to *safeness* properties [3]. The latter defines the rules that must be satisfied for all interactions that occur. In this sense, the aspects (a) through (d) are safeness properties. Liveness properties define that certain interactions will actually happen, usually in the sense that "something useful will eventually happen" [43]. A simple example is the statement that an implementation of a transport protocol will not refuse all connection requests. (Aspects (a) through (d) of the specification only state that an implementation may accept or refuse a connection request.)

(f) *Real-time properties*: While liveness properties state that certain interactions will actually happen (qualitative properties), real-time properties provide quantitative measures. For communication protocols and services, they are usually related to communication delay and attainable throughput. Other real-time properties are related to reliability, mean-time between failures and probabilities for refusing service requests.

1.4. Specification Languages

An important issue related to "specification" is the question in which language a specification is written. Most specifications are written in natural language, since this language is easily understandable by humans. However, in many cases it is advantageous to define certain aspects of the specified behavior in a more formal manner.

While specifications written in natural language are readily accessible to the human reader, they present the following disadvantages compared to specifications written in some formal language:

(a) Natural language specifications often contain ambiguities and are difficult to check for completeness and consistency.

(b) They are not helpful for the automation of the implementation or conformance testing processes, because they cannot be processed by automated tools. A further discussion of these issues is given in Sections 2 and 4.

Many different formal specification languages have been developed for various purposes, and many of them have been applied to the description of distributed systems and communication protocols. The most important approaches and their suitability for the different behavior aspects defined above can be summarized as follows:

(1) finite state machines (aspect (a)),

(2) formal grammars (aspect (a)),

(3) Petri nets (aspect (a)),

(4) algebraic calculi, e.g. CCS (aspect (a)),

(5) high level programming languages (aspects (a), (b) and (c)),

(6) abstract data types (aspects (b) and (c)),

(7) temporal logic (aspects (a) and (e)).

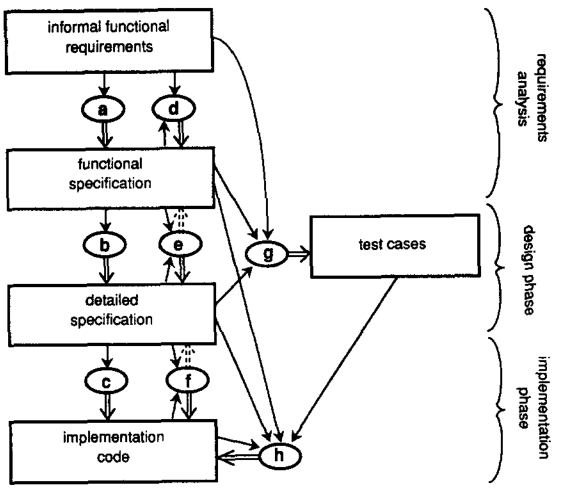
Various extensions of the approaches (1) through (4) have been defined by combining them with programming language or abstract data type approaches for the description of parameter values (aspects (b) and (c)). Various extensions have also been developed for handling real-time aspects. The aspect of PDU coding is not addressed directly by any of the methods, except by ASN.1 (see Section 3.1.3). However, it can be expressed by most methods, though often in a very clumsy manner.

Section 3 contains a more detailed discussion of the specification languages used for OSI communication services and protocols.

1.5. Historical Perspective

The principle of layered protocol architectures was clearly stated in the work on the French Cyclade network in the early seventies [45]. This also lead to the recognition of the importance of service specifications (see for instance [58]). Most of the formal specification approaches mentioned above are much older. Since the mid seventies, they have been applied in various research projects to the specification of communication protocols and their validation, implementation or testing (for an early review, see [13]).

With the beginning work on the standardization for OSI at the end of the seventies, some people recognized that formal specifications could be useful in the development of OSI standards, since the specifications of protocol standards should be unambiguous and are used by many people for the implementation and testing of protocol implementations. Special groups discussing "formal description techniques" (FDT) for application to OSI were formed within ISO and CCITT in 1980 [47,57]. These groups developed three specification languages, Estelle, LOTOS and SDL, which are further discussed in Section 3. Some trial specifications of OSI protocols using Estelle and LOTOS were also developed within ISO. The present version of SDL, developed within CCITT, is based on a version already defined in 1980 (without support for formally describing parameter values) which has been used for the description of switching systems in certain CCITT recommendations.



(a) creation of functional specification
(b) creation of detailed specification
(c) creation of implementation code
(d) validation of functional specification
(e) validation of detailed specification
(f) validation of implementation code by informal walk-throughs and debugging tests
(g) design of test cases
(h) validation of implementation code through formal testing procedures

Notation:	
▭	design/implementation document
◯	design/implementation activity
A → B	information of (A) is used for activity (B)
B ⇒ A	activity (B) creates or modifies document (B)

Fig. 7. Software development activities.

2. The System Development Life Cycle

2.1. System Development Life Cycle in General

Software development is usually supported by a methodology which is a systematic procedure that can be followed to produce the required software product (see for instance [23]). A methodology can be characterized in several ways: major types of activities that must be performed, fundamental concepts, and techniques/tools utilized with the methodology. Usually, the following phases of development are foreseen:

- requirement analysis: definition of the problem to be solved and development of the functional specification,
- design phase: development of a detailed specification,
- system implementation, and
- system maintenance.

During the requirement analysis, the functional specification of the system is created from an informal description of the system, which is given in plain text, possibly augmented by diagrams or flowcharts. As shown in Fig. 7, this phase also includes a validation activity which has the objective of ensuring that the functional specification corresponds to the requirements, and that the specifications are internally consistent. This validation activity is very important, since errors in the specifications are very expensive to eliminate if they are found during the later phases of the system life-cycle.

The following phases represent step-wise refinements of the specifications until the system implementation is reached. Usually, there is at least one intermediate, more detailed system description, called "detailed specification", as shown in Fig. 7. Each of these more detailed descriptions must be validated in respect to the more abstract description. The validation of the implementation code is mainly performed through testing. In addition to informally applied debugging tests performed by the implementation team, larger software development projects usually involve the development of test cases from the specifications and the application of these test cases to the implementation by a separate team of people.

It is important to note that each validation activity may lead to some update of the more abstract specification which serves as a reference, since errors in the latter may be detected. Also during the system maintenance phase, which corresponds to the usage of the system, errors are detected occasionally and must be corrected. In addition, after longer periods of usage, the requirements for the systems may change. This may lead to a revision of the whole system, which would be performed through separate phases of analysis of the changed requirements, changes to the design, and implementation.

2.2. *The Use of Formal Specifications*

The role of the specifications goes far beyond a particular development activity. As shown in Fig. 7, the functional and detailed specifications are used in most development activities during the system development life-cycle. Therefore the description techniques and languages used for these specifications also have a strong impact on the system development life-cycle. The use of formal specification languages makes it possible to partially automate some of the activities in the development cycle.

The following paragraphs indicate how the different activities can profit from the use of formal specifications. These approaches are quite general in nature and can be applied outside the protocol area. A more detailed discussion in the context of OSI protocols is given in Section 4.

(1) Specification validation: In this paper, the term "validation" is used for any activity which serves for obtaining a valid specification or implementation, that is, a specification or implementation which satisfies its more abstract requirements and does not contain any internal errors or inconsistancies. It includes approaches through testing and verification. For the validation of specifications through testing, the approaches described under points (3) and (4) below can be applied if the specification is executable, at least in a simulated mode. Through the testing approach, unfortunately, one is never sure whether all errors have been detected. On the contrary, validation through verification implies the consideration of all possible execution paths. Depending on the language used for the specification, various tools exist for automating, at least partially, the verification process (see also [44]).

(2) Step-wise refinement of specifications and implementations: Implementation-oriented specifications may be developed from the original system specification, using the same language. The refined specification may be used for semi-automated code generation. (See Section 4.3 or [52] for more details).

(3) Test case selection: Partly automated methods exist for the selection of test cases based on the formal specification of the system to be tested (see Section 4.4).

(4) Test result analysis: The results of tests performed on the implementation must be analysed and compared with the specification of the system, or the implementation-oriented specification (if implementation-dependent characteristics are tested). In the case of formal specifications, this comparison can be largely automated (see Section 4.5).

2.3. *The Role of Service and Protocol Specifications*

In the case of communication system development, there is an additional important aspect to be considered: At the level of detail where the communication between different systems is considered, two kinds of specifications are of prime importance, namely the communication service and protocol specifications. As shown in Fig. 2, the specification of the service of layer N describes the communication service provided to the user entities residing in the layer $(N+1)$. The protocol specification for layer N describes certain rules for the behavior of each protocol entity in layer N. The protocol specification should define those aspects of the entities' behavior which are required for compatible communication among the different communicating systems and the provision of the defined communication service. Additional aspects of their behavior are usually left unspecified in the protocol specification; they can be chosen differently in each system implementing the protocol.

In the context of the software life-cycle discussed in Section 2.1, the service and protocol specifications can both be considered as functional specifications; they define the behavior of different parts of the overall system. The *protocol specification* defines the behavior of a protocol entity, which is located in each communicating system component. It is therefore of prime interest for any implementation project. The protocol specification is the most abstract specification from which the implementation proceeds, usually through more detailed specifications taking into account various aspects of the particular system to be implemented, such as performance objectives and interface requirements. It is also the reference for the selection of test cases for conformance testing and the analysis of test results.

The corresponding *service specification* is important for the following reasons:

(a) *Abstract specification of the interface at a service access point*: The service specification for layer N includes the definition of the service primitives, which are the basis for the protocol specifications for layers N and $(N+1)$. It also includes the local rules (for a given service access point) about the temporal order of their execution and the allowed parameter values. The service primitives and these local rules provide an abstract specification of the local interface between the layers N and $(N+1)$, and is therefore the basis for the design of the real layer interfaces within the systems implementing the protocol hierarchy.

(b) *Validation of the protocol specifications*: The service specifications for layers N and $(N-1)$ serve as a reference for the validation of the protocol specification for layer N. As shown in Fig. 8, the protocol specification may be considered as a refinement of the service specification, exhibiting the internal structure shown in Fig. 8(b). As shown in the figure, one can distinguish four concerns for validation:

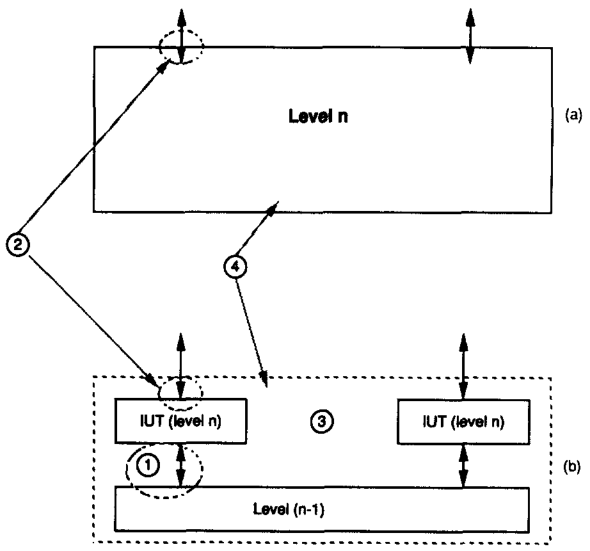
(1) Verifying that the use of the underlying communication service by the protocol entity is consistent with the local rules of the underlying service specification for layer $(N-1)$.

(2) Verifying that the interactions specified for the protocol entity with its user are consistent with the local rules of the service specification for layer N.

(3) Verifying that the protocol system consisting of the protocol entities communicating through the underlying communication service (as shown in Fig. 8(b)) satisfies the usual "nice" properties, such as absence of deadlocks, absence of non-specified receptions or undesired blocking, and absence of loops without progress. This is sometimes called the validation of "general properties" [13].

(4) Verifying that the global rules of the layer N service specification (relating the execution of service primitives at the different service access points) are satisfied by the protocol system.

(c) *Development of protocol conversion strategies*: The service specifications of two different distributed systems using different protocols for similar functions are the natural basis for designing interworking strategies between the incompatible systems. If similar communication services can be identified within the two systems, it is possible to design a communication gateway providing a protocol conversion function (see [6,24] for more detail). If the services are compatible, the design of the protocol converter can be derived from the two protocol specifications [7].



LEGEND
(a) Service specification.
(b) Corresponding system specification including protocol entities and underlying service.

(1) Consistency of protocol with underlying service.
(2) Protocol validation (general properties).
(3) Consistency of protocol with corresponding service.
(4) Protocol validation (service properties).

Fig. 8. Different aspects of protocol validation.

3. Specification Languages for OSI

The standardization of OSI involves the definition of a large number of communication protocols and services. The corresponding ISO standards and CCITT recommendations are mainly written in natural language, sometimes complemented with formal and/or semi-formal elements. This section gives an overview of different specification languages which are being used in the OSI standardization context.

We use in the following the distinction between informal, semi-formal and formal specification techniques. This distinction is somehow arbitrary. In Section 3.2, we discuss the so-called "formal description techniques" (FDTs) which were developed for the description of OSI communication protocols and services. These languages have a well-defined syntax, and for each specification satisfying the syntax rules, the language semantics defines, in a formal manner, the meaning of the specification. Several of the languages mentioned in Section 1.4 also satisfy the criterion. Those techniques which have a precisely defined syntax, but no formally defined semantics are in the following called "semi-formal". Some existing programming languages also belong to this class. Other techniques used within the context of OSI, finally, are only defined in an informal manner.

3.1. Informal and Semi-formal Methods

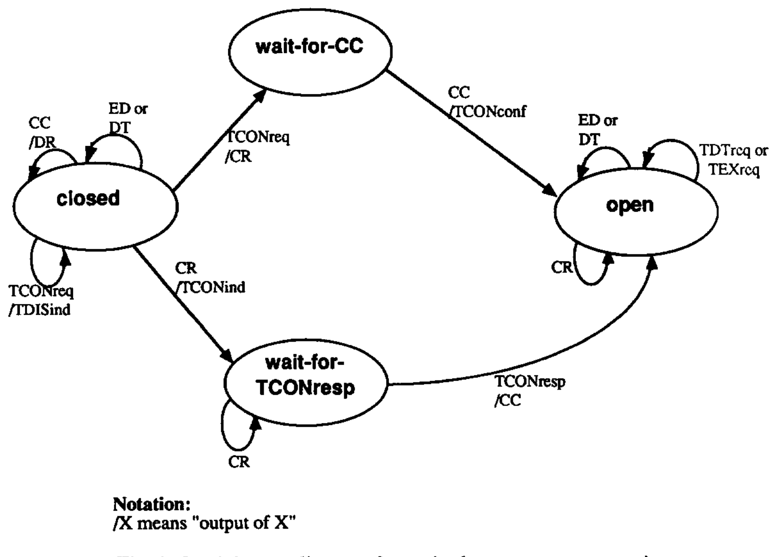
Most OSI service and protocol specifications are written in natural language augmented with tables and other forms of organized information. The following notations are used in many cases.

3.1.1. Service Primitives

The primitive interactions between layers are called *service primitives*. Their names and parameters are defined in the service specifications. Service primitives are grouped into *service elements*, as for example the *TCON* primitives for the establishment of a Transport connection. The different primitives within a group are distinguished by name suffixes such as "request", "indication", "response" or "confirmation". Certain general rules apply to the order in which these different primitives are executed by the initiating and responding protocol entities [27]. Sometimes such rules are described by *time-sequence diagrams*, such as shown in Fig. 6. It is important to note that such diagrams define only certain examples of sequences. This is not sufficient for a specification which should define all possible sequences of interactions.

3.1.2. State Tables

The specification model of a state transition machine is often used for describing systems. In



Notation:
/X means "output of X"

Fig. 9. Partial state diagram for a single transport connection.

event \ state	WFNC	WFCC	WBLC (class 2 only)	OPEN	CLOSING (class 2 only)	WFTRESP	CLOSED
TCONreq							PO:TDISind CLOSED; P2:NCONreq WFNC; P3:CR WFCC; P4:WFNC
TCONresp						CC OPEN	
TDTreq				[3] OPEN			
TEXreq				P7:[4] OPEN			
NCOconf	CR WFCC						
CR				P9:open	P9:closing	P9: WFTRESP	P1:DR(1) CLOSED NOT P1: TCONind WTRESP
CC		P8:TCONconf OPEN; P6 and P5: NDISreq CLOSED; P6 and P7: TDSind DR CLOSING	P5:(3) NDISreq CLOSED; TDISind P7: DR CLOSING				DR CLOSED
ED	Does not Exist in Class 0 (2)			[4] OPEN	CLOSING		CLOSED
DT				[3] OPEN	CLOSING		CLOSED

NOTES
1- An ER TPDU shall be sent in certain cases - see 6.6
2- If received it shall be processed as a protocol error - see 6.22.
3- A CR with class 2 has been sent and a CC class 0 is received.
4- If DC is not available (i.e. class 0 only implemented) of SRC-REF is zero.

Specific actions for classes 0 and 2

NAME	DESCRIPTION
[1]	If the network connection is not used by an other transport connection assigned to it, it may be disconneted
[2]	See 6.22 (receipt of an ER TPDU)
[3]	See data transfer procedure of the class
[4]	See expedited data transfer procedure of the class
[5]	An N-RESET response has to be issued once for the network connection if the network connection has not been released. In class 0, an N-DISCONNECT request has to be issued.

Predicates for classes 0 and 2

NAME	DESCRIPTION
P0	T-CONNECT request unacceptable
P1	Unacceptable CR TPDU
P2	No Network connection available
P3	Network connection available and open
P4	Network connection available and open in progress
P5	Class is class 0 (class selected in CC)
P6	Unacceptable CC
P7	Class is class 2
P8	Acceptable CC
P9	Class 4 CR

Fig. 10. Extract of state table from transport protocol standard (classes 0 and 2).

this model, a component of the system is described by defining the possible states of the component and the state transitions the component will do in relation with interactions with other components in the system. The simplest such model is a finite state machine (FSM) for which the number of possible states and the number of different interactions (ignoring parameters) is finite. Various notations can be used to represent the transitions of a FSM, such as transition diagrams, regular grammars, or transition tables. Figures 9 and 10 show such a diagram for a simplified Transport protocol and part of the state tables included in the OSI standard. The table also refers to certain predicates and actions which are listed in a separate table and defined informally.

3.1.3. ASN.1 Notation for Interaction Parameters

The ASN.1 notation [33] was originally developed in conjunction with the CCITT recommendations of 1984 on message handling systems. It 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 data types, such as integers, reals, booleans, bit strings, octet strings and various kinds of character strings. It also allows the definition of composed data types, such as groups of elements

```
TRANSPORT-PDU DEFINITIONS ::=

BEGIN

TPDU ::= CHOICE {
             cr-pdu      CR,
             cc-pdu      CC,
             dr-pdu      DR,
             dc-pdu      DC,
             dt-pdu      DT,
             ed-pdu      ED,
             ak-pdu      AK,
             ea-pdu      EA,
             rj-pdu      RJ,
             er-pdu      ER }

CR ::= SEQUENCE {
        code            [0] CR-CODE,
        credit          [1] BIT STRING,    --only 4 bits
        dest-ref        [2] ADDRESS-TYPE DEFAULT {0000000000000000B},
        source-ref      [3] ADDRESS-TYPE, --2 octets indicating source address
        class           [4] CLASS-TYPE,
        option          [5] OPTION-TYPE,
        var-part        [6] VARIABLE-PART-TYPE OPTIONAL,
        user-data       [7] OTCET STRING DEFAULT {}
                }
-- Similar for other TPDUs

CR-CODE ::= BIT STRING DEFAULT {1110B} --TPDU code for CR
-- Similar for other TPDU-CODE.

ADDRESS-TYPE ::= OCTET STRING --2 octets only

CLASS-TYPE ::= INTEGER {class0 (0), class1 (1), class2 (2),
                        class3 (3), class4 (4)} --only 4 bits

OPTION-TYPE ::= BIT STRING {extended-format (2), explicit-flow-ctl (1)}
                        --4 bits only

VARIABLE-PART-TYPE ::= SET {
                        tsap-id         [0] OCTET STRING OPTIONAL,
                        tpdu-size       [1] OCTET STRING OPTIONAL,
                        version-number  [2] OCTET STRING OPTIONAL,
                        security        [3] OCTET STRING OPTIONAL,
                        checksum        [4] OCTET STRING OPTIONAL,
                        add-opt-select  [5] OCTET STRING OPTIONAL,
                        alt-class       [6] OCTET STRING OPTIONAL,
                        ack-time        [7] OCTET STRING OPTIONAL,
                        throughput      [8] OCTET STRING OPTIONAL,
                        residual-error  [9] OCTET STRING OPTIONAL,
                        priority        [10] OCTET STRING OPTIONAL,
                        transit-delay   [11] OCTET STRING OPTIONAL,
                        reassign-time   [12] OCTET STRING OPTIONAL
                          }

END -- of TP_PDU definitions --
```

Fig. 11. TPDUs expressed in ASN.1 (Note: This definition reflects the logical structure of the transport PDUs, but not the coding used by the protocol.)

(called SEQUENCE, corresponding to RECORD in Pascal), sequence of identical types (called SEQUENCE OF), a type of alternatives (called CHOICE, corresponding to Pascal's variant records), a TAG defining a code for distinguishing between different alternatives, and others.

This notation therefore covers the specification aspect (b) mentioned in Section 1.3, i.e. the definition of the range of possible values of parameters. It is mainly used for defining the range of values for the PDU parameters of OSI Application layer protocols. But it could also be used for describing the parameters of service primitives. As an example, Fig. 11 shows a possible definition of the data structure of Transport PDUs.

The main reason for the success of ASN.1 as specification language is probably the fact that it is combined with a standard encoding scheme for PDUs [34] which has been adopted for OSI Application layer protocols. Based on the information contained in the ASN.1 definition of the PDU structure, this scheme completely determines the PDU encoding, and can be used for implementing the coding and decoding functions in a systematic manner (see also Section 4.3).

ASN.1 also includes a macro definition facility which allows the definition of additional notations. One such notation is for instance defined for specifying the parameter data types of remote operations [36] which are used for many Application layer protocols.

3.1.4. TTCN Notation for Test Case Specification

The TTCN (Tree Table Combined Notation) is relatively recent, and has been developed for the description of test cases for OSI conformance test suites [31]. As its name indicates, the language includes several different notations. The overall organization of the language is in terms of a collection of tables defining different aspects of a test, such as service primitives, PDUs, and their parameters, order of interactions, and constraints on parameter values. The interaction ordering is defined in terms of a conceptual tree where each branch represents a possible execution order. In addition to the tabular notation, a linear form of TTCN is being developed for the exchange of test cases in machine-readable form. The ASN.1 notation can also be used for certain aspects of the test descriptions.

TTCN is being used for the description of OSI test cases. In the opinion of the author, the language is not well structured and in many respects quite "ad hoc". The semantics of the language is defined informally without reference to other specification languages. In order to formally relate the defined test cases to the corresponding protocol specification, a definition of the semantics in terms of one of the FDTs discussed in Section 3.2 would be useful [50].

3.2. Formal Description Techniques

In addition to the semi-formal methods discussed above, CCITT and ISO have developed so-called Formal Description Techniques (FDTs) for the description of OSI protocols and services, namely Estelle [29], LOTOS [30] and SDL [17] . (For a tutorial introduction to these languages, see [16], [14] and [47], respectively.) Although these languages are specifically intended for the description of OSI protocols and services, they have potentially a much broader scope of application. However, their effective use in the OSI area, so far, has been relatively slow. This may be partly explained by the competition between these three languages, which each have certain advantages, and by the difficulty many people have in learning a new language.

3.2.1. Characterization of the Techniques

In Estelle, a specification module is modelled by an extended FSM. The extensions concerning the aspects (b) and (c) of Section 1.3 are covered by type definitions, expressions and statements of the Pascal programming language. In addition, certain "Estelle statements" cover aspects related to the creation of the overall system structure consisting in general of a hierarchy of module instances. Communication between modules takes place through the interaction points of the modules which have been interconnected by the parent module. Communication is asynchronous, that is, an output message is stored in an input queue of the receiving module before it is processed.

SDL, which has the longest history, is also based on an extended FSM model. For the aspects (b) and (c) it uses the concept of abstract data types with the addition of a notation of program variables and data structures, similar to what is included in Estelle. However, the notation for the latter aspects is not related to Pascal, but to CHILL, the programming language recommended by CCITT. The communication is asynchronous and the destination process of an output message can be identified by various means, including process identifiers or channel names.

LOTOS is based on an algebraic calculus for communicating systems (CCS [41]) which includes the concepts of finite state machines plus parallel processes which communicate through a rendezvous mechanism which allows the specification of rendezvous between two or more processes. Asynchronous communication can be modelled by introducing queues explicitly as data types. The interactions are associated with gates which can be passed as parameters to other processes participating in the interactions. These gates play a role similar to the interaction points in Estelle. The aspects (b) and (c) are covered by an algebraic notation for abstract data types, called ACT ONE [21], which is quite powerful, but would benefit from the introduction of certain abbreviated notations [9,28] for the description of common data structures.

In contrast to the other FDTs, SDL was developed, right from the beginning, with an orientation towards a graphical representation. The language includes graphical elements for the FSM aspects of a process and the overall structure of a specification. The aspects (b) and (c) are only represented in the usual linear, program-like form. In addition, a completely program-like form is also defined called (SDL-PR) which is mainly used for the exchange of specifications between different SDL support systems. Presently, there is also a joint work item in ISO and CCITT for the development of a graphical representation of LOTOS.

A comparative evaluation of the three FDTs is difficult to do. The following subjective statements address some of the issues: It seems that Estelle and SDL have the advantage of using well-known concepts of FSM and programming languages which make the initial understanding of the languages easier. The graphics aspects of SDL are also helpful in this respect. On the other hand, LOTOS has relatively few, but powerful language constructs which makes the learning of the complete language easier. LOTOS specifications often tend to be more abstract than specifications written in Estelle or SDL, which are often implementation-oriented. The concepts in the latter languages can be more directly related to typical implementation constructs. For the description of service access points, the rendezvous mechanism of LOTOS is better than the asynchronous message passing of Estelle and SDL, since the latter do not allow a complete specification without including implementation choices [12]. The LOTOS syntax seems to be more natural than the FSM-oriented syntax for the description of test cases. The formal definition of Estelle and LOTOS seem to be more readily usable for the construction of tools than the formal definition of SDL (which is given as an annex of the Recommentation). An attempt of a critical evaluation and comparison of the three languages can be found in [55].

This discussion only covers specification methods which are used by ISO and CCITT for the description of communication protocols and services. There are a number of other important specification methods and associated tools. Many tools are based on Petri nets and their extensions, or logic-based methods. Other languages are important because of their application in large scale projects (e.g. FAPL [51]).

3.2.2. Specification of OSI Concepts, Services and Protocols

It is clear that a given system can be described, using the same language, in several different equivalent manners [59]. With three different FDTs, there are even more possibilities. In order to orient the development effort for formal OSI specifications and for providing a better basis for comparing the three FDTs, ISO and CCITT have jointly developed a document [32] which contains guidelines for the application of the three FDTs.

This document includes in particular a discussion of how the different OSI concepts can be modelled in the Estelle, LOTOS, and SDL languages. This includes concepts, such as service access point, connection end point, service primitives, protocol entities, multiplexing, concatenation and backpressure flow control. In a second part of the document, a number of examples are described in each of the FDTs. These examples are

(a) The "deamon game", a simple self-contained system which can be taken as a lead-in to the more complex examples that follow.

Table 1
Formal specifications of OSI communication protocols and services

This is a (necessarily incomplete) list of formal specifications which are intended to precisely reflect the requirements defined in the OSI standards.

Link Layer
LAP-D of ISDN (FSM aspects) in SDL: CCITT Recommendation Q.931
LAN Logical Link Control (IS 8802/2) in ADA: Annex of standard
Idem, in LOTOS: ESPRIT project (see [22])

Network Layer
OSI Network Service in LOTOS: ESPRIT project (see [22])
Idem, in Estelle: ESPRIT project (see [20])
X.25 Packet Level in SDL: [25]
Connectionless Internet Protocol (IS 8473) in Estelle: Annex of standard

Transport Layer
Transport Service (ISO 8072) in Estelle: ESPRIT project (see [20])
Idem, in LOTOS: ESPRIT project (see ISO TC97/SC6/WG4 N-138, see also [22])
Transport Protocol (ISO 8073), classes 2/4 in Estelle: ESPRIT project (see [20])
Idem, in Estelle: ISO TC97/SC6 N3576
Idem, in LOTOS: ISO Technical Report ISO TC97/SC6/W64 N-117 (see also [22])
Teletex Transport protocol (OSI class 0) in Estelle: ESPRIT project (see [20])

Session Layer
Session Service (ISO 8326) in Estelle: ESPRIT project (see [20])
Idem, in LOTOS: ISO Technical Report ISO-SC21 DTR-9571 (see also [22])
Session Protocol (ISO 8327) in Estelle: ESPRIT project (see [20])
Idem, in LOTOS: ISO Technical Report ISO-SC21 DTR-9572 (see also [22])

Presentation Layer
Presentation Service (ISO 8822) in ESTELLE: ESPRIT project (see [20])
Idem, in LOTOS: ESPRIT Project (see [22])
Presentation Protocol (ISO 8823) in Estelle: ESPRIT project (see [20])
Idem, in LOTOS: ESPRIT Project (see [22])

Application Layer
Virtual Terminal Protocol (IS 9041.2) in Estelle: [4]
File Transfer, Access and Management (IS 8571) in Estelle: in the context of a conformance test system developed in Estelle [54]
Idem, in Estelle: ESPRIT project (see [20])
Message Handling System (MHS) protocols in Estelle: in the context of a conformance test system developed in Estelle [37]
Transaction Processing(ISO/DP 10026-3) in Estelle: Annex of DP
Idem, in LOTOS: ESPRIT Project (see [22])

(b) A sliding window protocol which demonstrates flow control and error recovery techniques present in many real protocols. The relation with the underlying service is described.

(c) The "Abracadabra" protocol and service includes the well-known alternating-bit protocol and demonstrates in addition the features of a connection-oriented communication service.

(d) The Teletex Transport protocol (similar to the OSI Transport protocol class 0) based on CCITT Recommendation T.70.

These examples can be helpful for a better understanding of these languages and for their comparison. Unfortunately, the specifications in the different languages do not always describe the example at the same level of detail. Another comparative example is given in [8] where an effort has been made to present similar specifications in the different languages.

In addition to the tutorial specifications mentioned above a number of formal specifications of OSI protocols and services have been developed with the purpose of precisely reflecting the natural language description given in the standard documents defining these protocols and services. The standardization community has stated that formal specifications of standard OSI protocols and services could be included in the standard documents as annexes, which in some cases may be included without being formally part of the standard, or in other cases could play the role of the standard reference.

Table 1 lists a number of presently available formal specifications of OSI protocols and services, some of which were developed within the standardization committees.

4. Tools for Using Formal and Semi-formal Specifications

While Section 2.1 gave an overview of the role the specification plays during the system development life cycle, this section gives an overview of the tools that can be used for the different development activities. A summary is shown in Table 2. The interested reader may find a more detailed survey about techniques and existing tools in [5].

4.1. Creation of Specifications

Specifications are written in some form of natural or formal language. Their creation cannot be

Table 2
Automated tools and methods depending on description language

Specification language used	Specification aspects handled (see Section 1.3)	Tools for the validation of specifications	Tools for code creation	Tools for testing
Natural language (1)	All	Cross referencing	Support packages	–
FSM (2)	(a)	Dynamic analysis: – exhaustive (FSM) – simulation (FSM)	Creation of program skeleton	Trace analysis (FSM) test case selection (FSM)
ASN.1 (3)	(b),(d)	Static analysis	(De-)coding routines data structures	PDU creation and viewing trace analysis (coding only)
FDTs (4)	(a),(b),(c)	Static analysis and dynamic analysis: – exhaustive – simulation	Complete translation of specification into executable form	Trace analysis (complete) test case selection

Notes: The terms used are explained in Section 4.
(1) Tools and methods available if specifications are informal.
(2) Tools and methods available (in addition to case (1)) if state tables (state diagrams, or other equivalent FSM notation) are used to specify certains aspects of the protocol. The automated tools/methods apply only to the aspects covered by the FSM formalism.
(3) Tools and methods available (in addition to case (1)) if the PDU structure is specified in ASN.1.
(4) Tools and methods available (in addition to those for cases (2) and (3)) if the full protocol specification is written in an FDT, such as Estelle, LOTOS, SDL. (It is noted that many existing protocol specifications written in SDL only cover the FSM aspects formally, the other aspects being covered in an informal manner.)

automated, however, certain computer based tools can be useful, such as text editors, spelling checkers and cross-reference tools. For the case that graphic representations are used, for example in the case of SDL, specialized graphic editors are very useful.

4.2. Validation of Specifications

Once a specification has been created in its initial form, it must be validated. This is usually a difficult task. In the case of formal specifications the following methods and tools can be used for this purpose.

(a) *Static analysis*: Based on the text of the specification, static analysis is quite useful to find clerical errors. Tools, normally related to a particular specification language, exist for the checking of context-free syntax, scope rules, type conformance, and other semantic conditions. The analysis corresponds to a part of what compilers do for programming languages.

(b) *Dynamic analysis*: In contrast to static analysis, the dynamic analysis of specification considers some kind of "execution" of the specified system. Because of the large number of possible situations that may occur during an execution of the system, dynamic analysis is usually much more difficult to do than static analysis. However, it can detect errors which are not detectable by static methods. A more detailed tutorial of this topic can be found in [44].

The dynamic methods can be classified into *exhaustive* and *simulation* methods. The exhaustive methods consider all possible situations that may occur during the execution of the specified system. In most cases, however there are too many cases to be considered. Therefore these methods are usually applied to a simplified description of the system. The best-known methods are related to the exhaustive reachability analysis for systems specified as a collection of finite state machines. The verification of programs and assertions, and other methods involving theorem proving, also belong to this class. They can be applied to the complete specifications; they are, however, difficult to apply to specifications of the size that are found in most practical applications.

The simulation methods validate only certain selected paths among all the possible executions.

However, they can be applied to real-size specifications, provided that the specification language allows some form of simulated execution (which is the case for the FDTs mentioned in Section 3.2). A number of simulation tools exists for this purpose (e.g. [38,39]). In order to reduce the large stage space to be explored by exhaustive reachability analysis, certain authors have proposed random and probability-based exploration procedures [40,61].

Methods and tools for performance evaluation also belong to the area of dynamic analysis. In the area of OSI, this aspect is of secondary importance, since the OSI standards are primarily concerned with *compatibility* between heterogeneous systems.

4.3. Developing Implementations

As mentioned in Section 2.1, the implementation development proceeds in several steps of refinement starting with the specification of the protocol to be implemented. For the validation of the more detailed system descriptions, some of the methods mentioned above could be used. In addition, the code generation process can also be partly automated. The following types of tools can be used:

(a) *Automatic generation of program skeletons from finite-state-oriented descriptions*: Many code generation tools for SDL are of this nature.

(b) *Automatic generation of program source code from FDT specifications*: As explained in [11], the abstract formal protocol specification must usually be refined before the program generation tool can be applied. Large parts of the implementation code can be automatically generated from detailed formal specifications [52,60].

(c) *Automatic generation of coding and decoding routines from ASN.1 specification of PDUs*: Because of the regular coding scheme used with ASN.1, the (de-)coding function can be automated. Existing tools either interpret the given ASN.1 description of the protocol dynamically, or generate, in source code, the specialized coding and decoding routines for the given protocol.

(d) *Support packages*: Software packages for buffer management, inter-task communication and other run-time support are very useful, if available, for a protocol implementation project. However, they are usually not very portable.

4.4. Selecting Test Cases

For the validation of an implementation (or specification) through testing, it is important to select appropriate test cases in order to cover all aspects of the behavior to be tested. The resulting set of tests, sometimes called "test suite", should also be as small as possible. In the case that an implementation is tested for conformance with its specification, the tests are usually determined based on the specification.

While in the area of OSI, standard test suites are developed for testing implementations for conformance to the protocol standards [46], the selection of test cases remains an important issue since additional requirements, performance and robustness of implementations must be asserted, aspects which are not covered by protocol conformance tests. Most existing tools for automating the selection of test cases for given protocol specifications are based on finite-state-machine specifications (e.g. [2,48]). They do not take into account the testing of the parameter values of service primitives and PDUs. Software test methods, based on data flow analysis, can be adapted for this purpose and combined with the former methods [18,49,56].

4.5. Analysing Test Results

Each test case defines the input to be applied to the system under test, which sometimes may also depend on observed outputs. The purpose of test result analysis is to determine whether the trace of interactions observed during a test satisfies the requirements of the reference specification. This result, sometimes called verdict, is not easy to obtain in general; one sometimes refers to an "oracle" to provide an answer. Clearly, such an oracle should be automated in order to make conformance testing manageable.

In the context of OSI, standardized test suites have been developed which also include verdicts for different test outcomes foreseen [31]. Nevertheless, the automation of test result analysis remains an issue in the OSI area because the determination of these verdicts in the first place, and the analysis of test results in the case that other kinds of tests are applied, is not easy. The automatic comparison of the observed test trace with the specification of the system under test can be performed if the specification is given in a formal language (see for instance [10]). However, it is not clear to what extent such analysis tools are capable of handling the complexity and speed of protocol implementations in real time.

It is expected that similar tools will also be available to validate, in respect to the protocol specification, the verdicts of the conformance test cases which were developed in an ad hoc manner. This requires, however, a well defined translation between the language used for the definition of the test cases (e.g. TTCN) and the FDT used for the formal protocol specification.

4.6. Overview of Tools for OSI Specification Techniques

As mentioned earlier, an advantage of using formal specifications is the possibility of using semi-automated methods in the system development life cycle. Among the different specification languages mentioned in Section 3, several do not cover all specification aspects and therefore do not provide automation of those aspects not covered. Table 2 gives an overview of specification aspects covered by the different languages and the kind of automated tools that can be used.

It is important to note that the FSM models and the ASN.1 notation cover different aspects. The former describe "major" states and kinds of input/output interactions, ignoring parameters, while the latter describes the possible range of values (data types) of the interaction parameters and the coding of these parameters in the PDUs. However, even taken together (as for instance in the OSI FTAM specification [35]), they do not cover the aspect (c) (see Section 1.3) of defining allowed parameter values for particular instances of communication and the interpretation of these values. For example, they cannot describe how the parameter value of an output interaction of a protocol entity depends on its state and/or previously received interaction parameters. All these aspects are however covered by the FDTs.

5. Research Directions

It is difficult to describe all directions of research related to protocol specifications. Much work is presently undertaken for developing methods and tools for the automation of the protocol development life cycle, and the application of these methods and tools in the context of real development projects. In addition, much work goes into the development of better specification languages. Further details can be found in existing surveys (e.g. [5]) and in the proceedings of an IFIP-sponsored conference series (e.g. [1,15]). The following paragraphs comment on the development of specification languages.

In the OSI context, ongoing research centers around the semi-formal specification languages ASN.1 and TTCN and the FDTs Estelle, LOTOS, and SDL. Main issues are the relations between the semi-formal languages with the FDTs, and their use for the formal specification of the protocols and services. Based on experience with the different FDTs, it is also expected that some pragmatic decisions on the use within OSI of one or the other FDT will be made.

In the meantime certain researchers propose improvements to the existing, standardized FDTs. Such proposals include for instance the introduction of rendezvous interactions to Estelle together with certain simplifications concerning the parallel processes within Estelle specifications [19], the introduction of abbreviated notations for defining common data structures, such as enumerations, records and arrays, in LOTOS [28,9], and the extension of SDL to handle non-determinism and separate input queues [42]. It is not clear what impact these proposals will have on the use of the respective FDTs.

In a larger context, the use of formal specification for software and hardware development for various kinds of applications is an active research area. It seems that the following concepts developed in this more general context will also have an impact on future applications of formal methods to protocol specifications:

(1) *Object-oriented specifications*: Object-oriented programming languages have recently had a strong impact on development methods for intelligent workstation software and corresponding knowledge representations. For adapting existing databases of real-time transaction systems for such kinds of applications, the concept of object-oriented databases has been developed. These concepts seem also to become important for the standardization work on distributed processing and system management.

(2) *The consideration of liveness properties*: Most specification languages, including the FDTs, do not address liveness properties explicitly. In fact, many properties concerning fairness (or the absence of starvation) cannot be expressed in these languages. Temporal logic contains an operator (with the meaning "will happen eventually") which is useful for these purposes (see point (e), Section 1.3). However, it is not clear how this formalism can best be combined with the existing specification languages.

(3) *Procedural versus declarative specifications*: This distinction is partly a question of specification style [59], but also a question of the language. For example, programming languages are designed for supporting an algorithmic, procedural specification style and are sometimes combined with a declarative specification language which allows the definition of input/output assertions. This latter style is also supported by the algebraic formalism for abstract data types in LOTOS and SDL. Research is oriented towards the (semi-)automatic translation of non-executable declarative specifications into equivalent specifications in an executable, more algorithmic form, and the use of certain logic-oriented languages that are directly executable, such as Prolog.

References

[1] S. Aggarwal and K. Sabnani, eds., *Protocol Specification, Testing and Verification VIII* (North-Holland, Amsterdam, 1989).

[2] A.V. Aho, A.T. Dahbura et al., An Optimization Technique for Protocol Conformance Test Generation Based on UIO Sequences and Rural Chinese Postman Tours, in: *Proc. IFIP Symposium on Protocol Specification, Testing and Verification*, Atlantic City (1988).

[3] B. Alpern and F.B. Schneider, Recognizing Safety and Liveness, *Distr. Comput.* **2** (1987) 117–126.

[4] P. Amer and Çeçeli, Estelle Formal Specification of the ISO Virtual Terminal, *Comput. Standards Interfaces* **9** (2) (1989) 85–104.

[5] G. v. Bochmann, Usage of Protocol Development Tools: The Results of a Survey (invited paper), in: *Proc. IFIP Symposium on Protocol Specification, Testing and Verification*, Zurich (1987) 139–161.

[6] G. v. Bochmann and P. Mondain-Monval, Design Principles for Communication Gateways, *IEEE Trans. Select. Areas Comm.*, to appear.

[7] G. v. Bochmann, Deriving Protocol Adapters for Communication Gateways, *IEEE Trans. Comm.*, to appear.

[8] G. v. Bochmann, Specification of a simplified Transport Protocol Using Different Formal Description Techniques, *Comput. Networks ISDN Systems*, to appear.

[9] G. v. Bochmann and M. Deslauriers, Combining ASN1 Support with the LOTOS Language, in: *Proc. IFIP Symposium on Protocol Specification, Testing and Verification IX* (1989).

[10] G. v. Bochmann, R. Dssouli and J.R. Zhao, Trace Analysis for Conformance and Arbitration Testing, *IEEE Trans. on Software Engrg.* **15** (11) (1989) 1347–1356.

[11] G. v. Bochmann, G. Gerber and J.M. Serre, Semiautomatic Implementation of Communication protocols, *IEEE Trans. Software Engrg.* **13** (9) (1987) 989–1000; reprinted in: D.P. Sidhu, ed. *Automatic Implementation and Conformance Testing of OSI Protocols* (IEEE, New York, 1989).

[12] G. v. Bochmann and A. Finkel, Impact of Queued Interaction on Protocol Specification and Verification, in: *Proc. Internat. Symp. on Interoperable Information Systems (ISIIS)*, Tokyo (1988) 371–382.

[13] G. v. Bochmann and C.A. Sunshine, Formal Methods in Communication Protocol Design (invited paper), *IEEE Trans. Comm.* **28** (4) (1980) 624–631.

[14] T. Bolognesi and E. Brinksma, Introduction to the ISO Specification Language Lotos, *Comput. Networks ISDN Systems* **14** (1) (1987) 25–59.

[15] E. Brinksma et al., eds., *Protocol Specification, Testing and Verification IX* (North-Holland, Amsterdam, 1990).

[16] S. Budkowski and P. Dembinski, An Introduction to Estelle: A Specification Language for Distributed Systems, *Comput. Networks ISDN Systems* **14** (1) (1987) 3–23.

[17] CCITT SG X, Recommendation Z.100, 1987.

[18] E. Cerny and G.v. Bochmann, Testing Implementations of an Application-level Communication Protocol: Interlibrary Loan, in: *Proc. IEEE FTCS '85* (1985).

[19] J.P. Courtiat, "Estelle*: A Powerful Dialect of Estelle for OSI Protocol Description", in: *Proc. IFIP Symposium on Protocol Specification, Testing and Verification*, Atlantic City (1988).

[20] M. Diaz et al., *The Formal Description Technique Estelle* (North-Holland, Amsterdam, 1989).

[21] H. Ehrig and B. Mahr, *Fundamentals of Algebraic Specifications 1* (Springer, Berlin, 1985).

[22] P.H.J. van Eijk et al., *The Formal Description Technique LOTOS* (North-Holland, Amsterdam, 1989).

[23] C. Gane and T. Sarson, *Structured Systems Analysis: Tools and Techniques* (Prentice-Hall, Englewood Cliffs, NJ, 1979).

[24] P.E. Green, Protocol Conversion, *IEEE Trans. Comm.* **34** (3) (1986) 257–268.

[25] D. Hogrefe, Protocol and Service Specification with SDL: The X.25 Case Study, Technical Report FBI-HH-B-134/88, Universität Hamburg, 1988.

[26] ISO IS 7498, Reference Model for OSI, 1982.

[27] ISO TC97/SC16, Service Conventions, TR 8509.

[28] ISO 97/21 N1540, Potential Enhancements to Lotos, 1986.

[29] ISO IS9074, Estelle: A formal description technique based on an extended state transition model, 1989.

[30] ISO IS8807, LOTOS: A Formal Description Technique, 1989.

[31] ISO TC97/SC21, DIS 9646/1, DIS 9646/2, DIS 9646/3, OSI Conformance Methodology and Framework, Part 1: General Concept, Part 2: Abstract Test Suite Specification, Part 3: The Tree and Tabular Combined Notation (TTCN), 1989.

[32] ISO Technical Report DTR-10167, Guidelines for the Use of Formal Description Techniques for OSI Specifications, 1989.

[33] ISO IS 8824, Information Processing - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1).

[34] ISO IS 8825, Information Processing - Open Systems Interconnection - Basic Encoding Rules for Abstract Syntax Notation One (ASN.1).

[35] ISO TC97/SC21, File Transfer, Access and Management, IS 8571.

[36] ISO IS 9072, Remote Operations.

[37] A. Iwabuchi, R.J. Linn and J.P. Favreau, Application of Formal Specification Techniques to the Specification of the MHS Test System, in: *Proc. International Symposium on Interoperable Information Systems* (INTAP), Tokyo (1988) 255–262.

[38] C. Jard, R. Groz and J.F. Monin, VEDA: A Software Simulator for the Validation of Protocol Specifications, in: *Proc. COMNET '85 (IFIP), Computer Network Usage: Recent Experiences* (North-Holland, Amsterdam, 1985).

[39] L. Logrippo et al., An Interpreter for LOTOS: A Specification Language for Distributed Systems, *Software Practice and Experience* **18** (4) (1988) 365–385.

[40] N.F. Maxemchuk and K. Sabnani, Probabilistic Verification of Communication Protocols, in: *Proc. IFIP Symposium on Protocol Specification, Testing and Verification VIII* (1988) 307–320.

[41] R. Milner, *A calculus of Communicating Systems*, Lecture Notes in Computer Science **92** (Springer, Berlin, 1980).

[42] F. Orava, Formal Semantics of SDL Specifications, in: *Proc. IFIP Symposium on Protocol Specification, Testing and Verification VIII* (1988).

[43] S. Owicki and L. Lamport, Proving Liveness Properties of Concurrent Programs, *ACM Trans. Progr. Languages and Systems* **4** (3) (1982) 455–495.

[44] B. Pehrson, Protocol Verification for OSI, *Comput. Networks ISDN Systems* **18** (1989/90) 185–201, this issue.

[45] L. Pouzin, Presentation and Major Design Aspects of the CYCLADES Computer Network, in *Proc. 3rd ACM-IEEE Communication Symposium*, Tampa, FL (1973) 80–87.

[46] D. Rayner, OSI Conformance Testing, *Comput. Networks ISDN Systems* **14** (1987) 79–98.

[47] R. Saraco and P.A.J. Tilanus, CCITT SDL: Overview of the Language and its Applications, *Comput. Networks ISDN Systems* **13** (1987) 65–74.

[48] B. Sarikaya and G. v. Bochmann, Synchronization and Specification Issues in Protocol Testing, *IEEE Trans. Comm.* **32** (4) (1984) 389–395.

[49] B. Sarikaya, G. v. Bochmann and E. Cerny, A Test Design Methodology for Protocol Testing, *IEEE Trans. Software Engrg.* **13** (April 1987) 518–531.

[50] B. Sarikaya and Q. Gao, Translation of Test Specifications in TTCN to Lotos, in: *Proc. IFIP Symposium on Protocol Specification, Testing and Verification*, Atlantic City (1988).

[51] G.D. Schultz, D.B. Rose, C.H. West and J.P. Gray, Executable Description and validation of SNA, *IEEE Trans. Comm.* **28** (4) (1980) 661–677.

[52] D.P. Sidhu and T.P. Blumer, Semi-automatic Implementation of OSI Protocols, *Comput. Networks ISDN Systems* **18** (1989/90) 221–238, this issue.

[53] Special Issue on Open Systems Interworking, *Proc IEEE* (December 1983).

[54] Test System for Implementations of FTAM/FTP Gateways, Final Report, National Inst. of Standards (US), ICST, 1988.

[55] The SPECS Consortum and J. Bruijning, Evaluation and Integration of Specification Languages, *Comput. Networks ISDN Systems* **13** (1987) 75–89.

[56] H. Ural, A Test Derivation Method for Protocol Conformance Testing, in: *Proc. IFIP Symposium on Protocol Specification, Testing and Verification VIII* (1988) 347–358.

[57] C. Vissers, Formal Description Techniques for OSI, in: *Proc. IFIP Congress 1986*, Dublin (1986).

[58] C. Vissers and L. Logrippo, The Importance of the Concept of service in the design of Data Communications Protocols, in: *Proc. IFIP Workshop on Protocol Specification, Verification and Testing*, Toulouse (1985).

[59] C. Vissers, G. Scollo and M.v. Sinderen, Architecture and Specification Style in Formal Descriptions of Distributed Systems, in: *Proc. IFIP Symposium on Protocol Specification, Testing and Verification*, Atlantic City (1988).

[60] S.T. Vuong et al., An Estelle–C Compiler for Automatic Protocol Implementation, in: *Proc. IFIP Symposium on Protocol Specification, Testing and Verification*, Atlantic City (1988).

[61] C.H. West, Protocol Validation by Random State Exploration, in: B. Sarikaya and G. Bochmann, eds., *Protocol Specification, Testing and Verification VI* (North-Holland, Amsterdam, 1986).