Formal Description Techniques

CHRIS A. VISSERS, RICHARD L. TENNEY, MEMBER, IEEE, AND GREGOR V. BOCHMANN, MEMBER, IEEE

Invited Paper

Abstract—**Early in the development of OSI, it was recognized that formal description techniques (FDT's) would be required to accomplish the goals of OSI. This paper is a brief history and a report on the status of the work of ISO / TC97 / SC16 / WG1 ad hoc Group on Formal Description Techniques. The group comprises three subgroups: the first working on architectural concepts; the second, on an FDT based on extended finite-state machines; and the third, on an FDT based on temporal ordering of interaction primitives. An overview of the techniques developed by each of these groups, as of December, 1982, is presented.**

I. INTRODUCTION

A. OSI Background and Framework

EARLY IN the development of OSI, it was recognized that if goals of OSI were to be accomplished, formal description techniques (FDT's) would be required to ensure that implementors anywhere in the world can develop correct and compatible implementations.

As eminently described by John Day, the first Rapporteur on FDT, in Annex E of an early version of the Reference Model for Open Systems Interconnection (SC16/N227), FDT's are necessary to provide:

- an *unambiguous*, *clear*, and *concise* specification of a service, protocol, interface, or the Reference Model;
- a basis for determining the *completeness* of the specification;
- a foundation for the *analysis* of the specification as to its correctness, efficiency, etc.;
- a basis for the *verification* that the specification meets all of the requirements of the OSI;
- a basis for determining the *conformance* of implementations to the OSI standard specification;
- a basis for determining the *consistency* of OSI standards with each other;
- a basis for developing *implementation* support.

Many of these goals are crucial to the development of compatible implementations. If formal techniques are not available, the usefulness of OSI standards will be greatly reduced.

The issues involved with FDT are rather abstract in nature and not intended to produce the contents of the technical specification of the Reference Model or the protocols and services for each layer. This was the reason Working Group 1 (WG1) formed a separate group on FDT.

The first meeting of the FDT group was in Chicago, IL, in January 1980. Since then meetings have taken place in Amsterdam, The Netherlands, in June 1980; Berlin, Germany, in February 1981; Washington, DC, in October 1981; Enschede, The Netherlands, in April 1982; Catania, Italy, in November 1982; and Paris, France, in February 1983. Minutes, Recommendations, and other relevant working documents have been distributed as regular SC16 documents.

This paper gives an overview of the work on FDT's for OSI specifications as carried out by the ISO TC97/SC16/WG1—ad hoc group on Formal Description Techniques as of December 1982. A companion article in this issue, by G. Dickson and P. de Chazal, gives an overview of corresponding activities by the CCITT.

B. Early Work

The FDT group quite rapidly observed that the Reference Model and associated protocols and services presented a major challenge for formal specification and concluded that the group could not simply adapt an existing technique to fit OSI requirements. In order to provide immediate support for the technical working groups while formal methods were being developed, the group began by drafting guidelines for the informal specification of protocols and services. These guidelines served two purposes: on one hand as an aid to the technical working groups to promote structure, consistency, and completeness, and on the other hand to achieve homogeneously structured specifications among different layers which would be much easier to specify formally once adequate formal techniques were available.

Although the informal guidelines absorbed most of the first two meetings, the group also produced a work plan, a set of criteria for evaluation of an FDT, and a first categorization of different FDT's.

C. Forming of Subgroups

One of the difficult problems confronting the FDT group was to converge the many divergent views on what FDT concept(s) should form the basis for OSI application, in particular in the light of the likely possibility that a certain concept may be appropriate for one layer but inadequate for other layers. The FDT group has solved this problem by forming Subgroups around corresponding FDT concepts and charging each Subgroup to come up with one proposal, in much the same way that a technical working group converges to a common solution. The FDT group expects that different FDT's are likely to be used together, complementing each other, rather than competing with each other. Moreover, the FDT group assumes that growing insight may raise the need for even newer techniques and Subgroups, whose results may stepwise replace techniques now under development.

Manuscript received January 15, 1983; revised September 25, 1983.

C. A. Vissers is with the Twente University of Technology, Dept. Informatics, 7500 AE Enschede, The Netherlands.

R. L. Tenney is with the Department of Mathematical Sciences, University of Massachusetts, Boston, MA 02125.

G. V. Bochmann is with Département d'Informatique et de Recherche Opérationelle (IRO), Université de Montréal, Montréal, P.Q., Canada H3C 3J7.

0018-9219/83/1200-1356$01.00

The three Subgroups thus formed during the Berlin (February 1981) meeting are:

1) Subgroup A (chaired by Gregor V. Bochmann), dealing with architectural specification concepts which (may) serve as the architectural basis for the description languages developed by Subgroups B and C.

2) Subgroup B (chaired by Richard L. Tenney), dealing with the development of an FDT based on the Extended Finite State Machine concept.

3) Subgroup C (chaired by Chris A. Vissers), dealing with the development of an FDT based on the Temporal Ordering of Interaction Primitives.

Liaison is maintained with appropriate groups (VII/Q27, VII/Q39, XI/3-1) within the CCITT. This applies in particular to Subgroup B, as its FDT exhibits features similar to the SDL technique developed by the CCITT.

A more detailed summary of each Subgroup's activity follows this introduction. For a detailed insight in the technical concepts the reader is referred to the tutorial documents of the Subgroups.

D. Tutorial Documents

In order to work in a coherent way, the Subgroups have agreed to produce, for each FDT, a so-called "tutorial" document, each of which follows the same outline. In addition, Subgroup A has the charge of promoting the use of identical architectural concepts by each Subgroup, in so far as this is possible. The tutorial documents should be self-contained and provide sufficient material for an OSI expert to learn, appreciate, and apply the FDT. The outline of the tutorial documents is as follows:

1) *Introduction*. This chapter introduces the main concepts, purpose, and philosophy underlying the FDT.

2) *Model*. This chapter provides the fundamental basis of the language in an informal, i.e., intuitive, way and provides the platform for the other chapters.

3) *Language Elements*. This chapter gives the FDT language by giving the notation and the informal semantics of the language elements. It illustrates the use of the language with small examples. In addition, short guidelines for the implementor as well as for the specifier are given.

4) *Formal Syntax*. The syntax summary of Chapter 3.

5) *Formal Semantics*. The formal semantics provides the formal and mathematical basis for the FDT and the ultimate arbitration for interpreting the language elements. It provides rules of mathematical manipulation that are used in Chapters 6 and 7.

6) *Integration Rules*. This chapter provides proof rules (verification rules) for checking that an (N)-service is rendered by an $(N-1)$-service and an (N)-protocol.

7) *Conformity Rules*. This chapter provides the proof rules for checking that an implementation complies with a specification.

8) *Terminology*.

Annex 1: *User Guidelines*. These guidelines give detailed hints to the implementor to interpret specific language constructs in the light of simplifying implementations, etc. In the same way hints may be given to the specifier to represent frequently used technical constructs.

Annex 2: *Applications to Draft Standards*. The FDT group has agreed to provide at least some trial specifications of Transport Layer standards to demonstrate the use of the FDT in complex specifications.

Annex 3: *Language Support Tools*. In the long term, an FDT may be supported by automated tools for designing, manipulating, and assessing specifications and implementations.

Annex 4: *Check Against Evaluation Criteria*. This Annex is used to assess the FDT against the set of evaluation criteria.

Annex 5: *Relation to Alternative Models*. This Annex is used to indicate differences, correspondences, and/or compatibility between different FDT's, and may provide translation rules.

Other Annexes if necessary.

E. Status

The Subgroups have produced elaborate tutorial documents, and the proposed FDT's are in a state in which trial specifications can be produced. In addition, the groups themselves have produced and are preparing major examples and applications to draft standards, particularly at Transport level, to demonstrate the features and applicability of the techniques. Some crucial aspects such as the proving that an implementation complies with a specification have not been addressed in full, and work on these issues is continuing.

The following sections briefly introduce the main concepts of each technique. The reader is referred to the most recent tutorial documents for a detailed insight.

II. Subgroup A: Architectural Concepts

A. Objectives and Main Features of FDT Developed

The objectives of the work in Subgroup A are twofold: 1) the development of architectural specification concepts which may serve as a basis for the description languages developed by Subgroups B and C, and 2) the coordination of the developments in Subgroups B and C in order to make it possible that subsystem specifications given in the languages of Subgroups B and C can be combined into a single system specification in a meaningful way. The latter is called in the following "interworking between Subgroup B and C languages."

The main specification concepts defined by Subgroup A are "modules" and "channels." A module is a unit of specification. Its behavior may be defined using the languages of Subgroup B or C, or it may be defined in terms of a "refinement," that is a structure of interconnected submodules. Each submodule, in turn, must be specified. Modules (and/or submodules) interact with each other through channels. The channel specification defines the possible interactions between the modules that use the channel for their interactions. Some examples of channel definitions can be found in Appendix I. An example of the stepwise refinement of a module is shown in Fig. 1.

These concepts, which are being used within the work of Subgroups B and C, can also be directly applied to the OSI reference model. In particular, the meaning of "service specification" and "protocol specification" has been defined more precisely in the context of formal descriptions. Additional objects for possible descriptions have been identified, such as interfaces, entity submodules, etc.

B. Current Status of Work

The concepts outlined above have been successively refined during the meetings of the Subgroup. Their definition, as given in the tutorial document, is relatively stable. The application of these concepts to the Reference Model, as described in the tutorial document, has been one of the first items of work of the Subgroup, and has led to the present definitions.

Work on the interworking of the Subgroup B and C languages has just started. Earlier work on this topic has been difficult

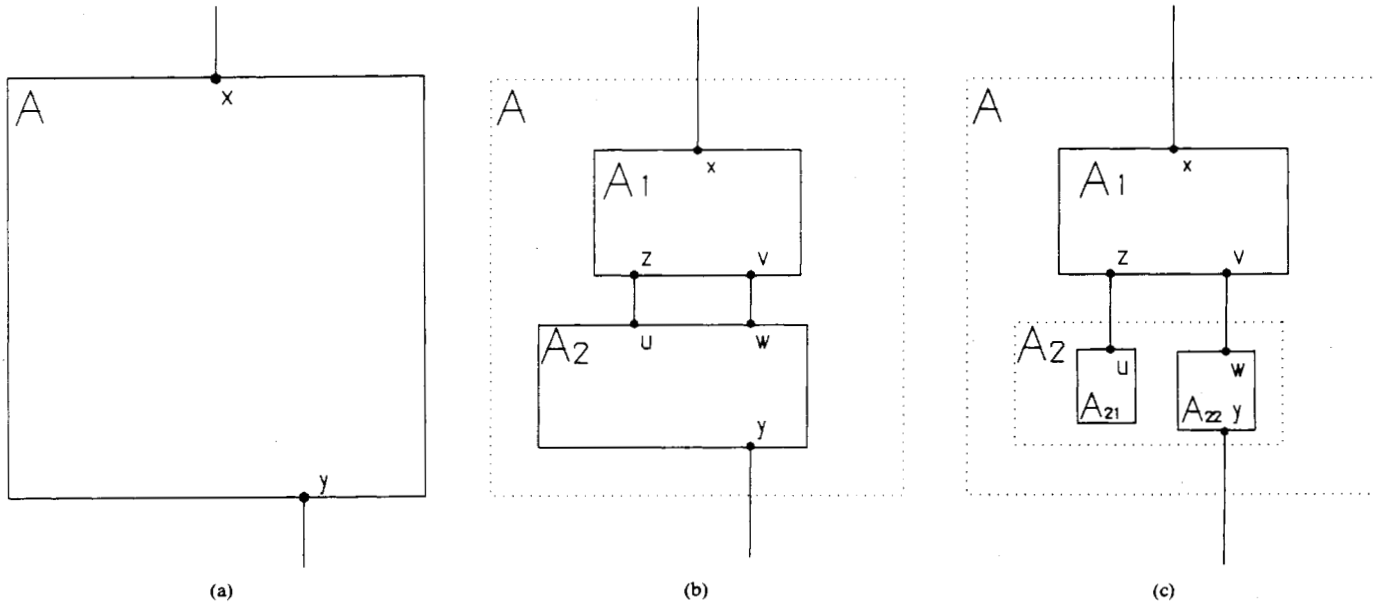


Fig. 1. Example of step-wise refinement. (a) Module A. (b) Submodules A_1 and A_2 representing the substructure of A. (c) Further refinement of submodule A_2.

because of the preliminary nature of the status of the Subgroup B and C languages. This topic was first included in the work program of Subgroup A only during the April 1982 meeting of the ad hoc group on FDT.

C. Future Work

Future work of the group will mainly consist of liaison with groups developing complete FDT's (including Subgroups B and C) for checking possible differences in the interpretation of architectural concepts, and eliminating these differences, if possible.

III. Subgroup B: Extended Finite-State Machine Specification

A. The Model

The underlying model for the technique is an extended finite-state automaton. The state space of a module is determined by a set of variables; a state is determined by the values assumed by each of these variables. One of these variables is a distinguished variable called the "STATE"; it represents what is traditionally thought of as the state of a finite automaton. It is sometimes called the "major state" to distinguish it from the other variables which are sometimes called "context variables." The STATE is often used to encode the status of the connection, e.g., closed, opening, etc., while the context variables are often used to store sequence numbers, grade of service, data, and the like.

Transitions are specified from a major state (or set of major states) to a major state. These transitions may depend on predicates on the context variables, and they may depend on an input. Associated with each transition is an operation to be executed as part of the transition. It may change the values of the context variables, and it may initiate output interactions with the environment. The operation is assumed to be atomic.

Many specifications may indeed by deterministic, but this is not a requirement of the specification technique. Thus for a given state and input interaction, more than one enabling predicate may be true, and several different transitions may be possible. A priority may be assigned to transitions. The transitions of high priority will take precedence over those of lower priority, but if two or more transitions of the same priority are enabled, then exactly one of these will be chosen, nondeterministically, for execution.

Spontaneous transitions, i.e., those that do not depend on any input (and hence have no "when" clause), may include a delay clause with two parameters, d_1 and d_2. The transitions may not occur until the enabling condition has remained true for d_1 time, and it must be considered immediately if the enabling condition remains true continuously for d_2 time. If the delay clause is absent, a delay condition of $d_1 = 0$ and $d_2 =$ infinity is assumed. This means that the transition may occur at any time the enabling condition is true, but possibly with infinite delay (i.e., never). A spontaneous transition marked NO DELAY must be considered immediately whenever its enabling condition is true.

B. The Language

To specify transitions and the operations associated with them, a language that is based mainly on PASCAL, which is now an ISO Draft Standard (ISO DIS7185), was developed. Some additions to PASCAL were found necessary.

A specification comprises three major parts: the channel type definitions, the module definitions, and the system structure.

The channel type definitions serve to name the two players of the channel and define what the interactions of the channel are and which player can initiate them. The channel type definitions describe the potential connectors between the various modules.

The module definitions specify the actual transitions and their operations. Each transition may be listed separately, giving the state(s) in which it may be applied, the resultant state, the input, and other conditions on context variables that enable the transition and the priority of the transition. These various parts of the transition are all optional. Furthermore, transitions may be nested; thus for example, all transitions that apply from the state S_1 may be grouped together without repeating the "from S_1" portion of the transitions.

The system structure portion of the specification is intended to describe the instantiation of various modules, combined to make

a system; its syntax and semantics remain to be defined. This is an area that requires coordination between Subgroup B and Subgroup A.

C. Status of the Work

The first four sections of the tutorial document are substantially complete. Work on the formal semantics of the model is progressing smoothly, and Section 5 is now nearly complete. In addition, major examples of the technique and application of the technique to draft standards already exist or are in preparation; these are to be included in Annex 2 of the tutorial.

At present, there are no language support tools, but some that exist for related FDT's could presumably be adapted to this technique.

IV. Subgroup C: Temporal Ordering Specification

A. Model for Temporal Ordering Specification

The approach of Subgroup C is based on the concept of Temporal Ordering of Interaction Primitives which is characterized by defining the behavior of a subject of specification by describing and ordering only the interactions with its environment as they can be observed from outside (the "black-box" concept). This concept has the inherent advantage of refraining from any statement that is in the realm of the implementor.

The basis of the technique is to specify all possible sequences of information inputs and outputs that are observable at the boundary (i.e., the access points or channels) of the subject of specification (called part or module). For a service specification, for example, this would mean all possible sequences of service primitives.

The technique allows the addition of specification statements such as statements concerning the dependencies of values of information outputs on values of information inputs, and statements concerning the time dependencies between inputs and outputs (which are necessary to express performance characteristics such as transfer delay, throughput, and the like).

One of the crucial aspects that any formal specification language has to deal with is the level of abstraction in which the inputs and outputs of the subject of specification should be described. Corresponding to the approach adopted in OSI, Subgroup C initially has chosen a so-called Interaction Primitive (IP), which, at the service level, is substituted for a Service Primitive, and at the protocol level is substituted for a Protocol Data Unit. The Interaction Primitive is assumed to be a *common activity* that the subject of specification shares with its environment. These common activities take place at (service- or protocol-) access points. During this common activity information is established and exchanged.

A set of "temporal ordering" operators is introduced which serve to place IP's in any desired temporal relationship. Thus Protocol Data Units and Service Primitives are temporally ordered to get "higher order" compositions which again may be ordered to ultimately yield the specification of a service or protocol.

To allow for a realistic description of concurrency of inputs and outputs and an unambiguous semantics when mapping the specification onto the implementation, an IP is assumed to elapse in time, it is not abstracted to an infinitesimally small (atomic) moment in time. The current temporal ordering operators and time performance predicates are based on these characteristics. The temporal ordering operators thus allow for such things as the sequential and parallel ordering of IP's as well as the possibility of mutual disruption of IP's which can be used to express such aspects as collision problems in disconnect phases.

The model is independent of the choice of data types and operations on data types. To promote coherence within the FDT group, Subgroup C will use Pascal for these aspects of the language.

B. Status of the FDT

The first framework of the language was available at the FDT Washington meeting (October 1981) in document WASH-7. The technique has shown promising results. Although improvements to the language elements are possible and necessary, trial specifications have shown that rather complex specifications indeed can be kept concise. The early draft of the tutorial document reflects the status as of mid-1982. It contains the first four chapters and an annex which contains a fairly complete description of the transport service, demonstrating the use of the language. More trial specifications are available and in preparation.

C. Current Work

Since the drafting of the early working documents, Subgroup C has paid considerable attention to the formal semantics of the model and language elements (Chapters 5–7). The formal semantics must provide the mathematical basis of the language, which is necessary for the development of a specification "metatheory" which contains proof rules for such things as proving that a service is rendered by a protocol (plus lower level service) and proving that an implementation complies with a specification. This compliance aspect is extremely important and leads to such considerations as balancing the model and language concepts between conciseness, which is necessary to keep a specification surveyable and perspicuous, and precision, which is necessary to avoid overspecification. Thus the work on formal semantics preeminently provides criteria for judging whether or not the formal model and associated language elements can precisely and satisfactorily specify OSI protocols and services.

This work has provided evidence that, for example, the representation of a Service Primitive by a single IP, i.e., without giving the Service Primitive an inner temporal ordering structure, does not always produce an adequate specification. Experience with the description of data primitives, flow control, segmenting, the precise moment of disconnection, and other problems has shown that at least some of the Service Primitives have to be decomposed into more elementary units of information inputs and outputs that have a (partial) temporal order.

For this elementary unit of information input and output, Subgroup C has adopted the term "event," which can be considered as a simple interaction primitive. The event concept allows more powerful (but when necessary also extremely simple) constructions for such things as Service Primitives: a process of any complexity, rather than just a simple input or output of a message, can be used to model a Service Primitive.

For a more detailed discussion of the use of the event concept the reader is referred to the most recent description of the model in the tutorial document.

Using the same set of temporal ordering operators it is possible to construct Service Primitives and Protocol Data Units from events, and again order these to get "higher order" compositions which will ultimately yield the specification of a service or protocol. Within this framework, Subgroup C is now studying the characteristics and potential refinements of the formal semantics

of the temporal operators in order to express their semantical properties in terms of well-defined mathematical properties. Subgroup C considers the availability of a mathematical theory of the ordering operators of extreme importance for a successful treatment of the difficult problems of compliance. Subgroup C expects that the outcome of this study may affect (the notation of) some language constructs. Moreover, it expects that after completion of this study, the contents of Chapters 5–7 will quickly be stabilized.

Appendix I
Subgroup B Language: A Sample Specification

The following is an example of the Subgroup B method of protocol description in use taken from Section 3.6 of the Subgroup B tutorial document. It is a specification of an alternating bit protocol. Although the example shows many of the basic constructs of the language, simplicity dictates that some of the features of the language not be shown here. It is hoped that the example will serve as enough of an introduction to the technique to make its form clear to all readers and to interest experts in studying the full Subgroup B tutorial.

The first section of the example contains some declarations of constants and types, in a style familiar to a reader of Pascal. One obvious addition is the notation "..." which is used to indicate that the specifier is leaving the interpretation to the implementor. Often this is accompanied by a comment to guide the implementor in his choice.

A notation was needed to indicate the properties of the connections between modules. These are called "channels." Each channel may have players, the roles of which are indicated in parentheses after the channel name. The various interface events of a channel are indicated after the role list. For each role, the events that the player may initiate are listed along with their parameters. These parameters are available within a transition that is initiated by the event.

The module header line includes names for the channels it uses as well as an indication of the role the module plays on that channel. Thus the Alternating_Bit module is the Provider of the U channel, which is a U_access_point channel. The inputs from this channel and from the N channel are placed in a common queue. The U_access_point channel supports three kinds of interface events. Two of these may be initiated by the User (and are thus inputs to the Alternating_Bit module), and one of these is initiated by the Provider (and is thus an output of the module).

Following the module header, variables local to the module are declared. Although not used in the example, if there were any labels or types local to the module, they would precede the variables, as they do in Pascal. Then the major states and major state sets are declared. State sets are a convenient way to specify that a transition may take place from any of several major states.

Next is an initialization section. In this, the major state and the variables are given initial values. This determines the initial state of the module.

Then functions and procedures are declared. In addition to the standard Pascal definitions, either the keyword "primitive" or the notation "..." is used to indicate that the details are left to the implementor. Often, the choice of a data structure and the details of the primitives must be coordinated choices. In the example, the choice of the structure of "buffer_type" will determine the details of the procedures "store," "remove," and "retrieve." Furthermore, the actual details of these structures and the routines that manipulate them are not particularly relevant to the action of the protocol. Output from a module over a channel is specified by the keyword OUT (or perhaps OUTPUT, the choice of the keyword is still under study). The actual channel and event are indicated by naming the channel, followed by a ".", followed by the output interaction with its parameters.

Finally, the transitions are listed. The clauses corresponding to the keywords "from," "to," "when," are all optional, and may appear in any order, and may be nested (though they are not in this example). They describe the major state before the transition and after the transition and the required input, respectively. The "provided" clause describes an enabling predicate that must be satisfied for the transition to take place. An optional "priority" may be assigned to any transition.

Once the input is listed, the parameters associated with the input may be accessed in much the same manner as the fields of a record within the scope of a "with" statement. This enhances the readability of the resultant specification.

Notice the transition from state ESTAB back to itself when an S.TIMER_response occurs. This corresponds to the case in which the retransmit timer expires for data that have already been acknowledged. In this case, clearly nothing need be done.

The specification of the action of the timer is included in the full Subgroup B tutorial document.

```
const
   retran_time = 10; (* retransmission time *)
   empty       = 0;  (* empty buffer *)
   null        = 0;  (* place-holder for
                        sequence in ack *)

type
   data_type   = ...;
   seq_type    = ...; (* for alternating bit,
                         use 0..1 *)
   id_type     = (DATA, ACK);
   timer_type  = (retransmit);
   ndata_type  =
      record
         id: id_type;
         data: data_type;
         seq: seq_type;
      end;
   msg_type    =
      record
         msgdata: data_type;
         msgseq: seq_type;
      end;
   buffer_type = ...;
   int_type    = ...; (* usually 'integer' *)



(* channel definitions *)

channel U_access_point(User, Provider);
   by User:
      SEND_request(UData: data_type);
      RECEIVE_request;
   by Provider:
      RECEIVE_response(UData: data_type);

channel S_access_point(User, Provider);
   by User:
      Timer_request(Name: timer_type;
                    Time:int_type);
   by Provider:
      Timer_response(Name: timer_type);

channel N_access_point(User, Provider);
   by User:
      Data_request(NData: ndata_type);
   by Provider:
      Data_response(NData: ndata_type);

(* Module definition *)

module Alternating_Bit
   (U: U_access_point(Provider) common queue;
    N: N_access_point(User)     common queue;
    S: S_access_point(User) individual queue);
```

```
var
   send_seq:    seq_type;
   recv_seq:    seq_type;
   send_buffer: buffer_type;
   recv_buffer: buffer_type;
   p,q:         msg_type;

state:
   (ACK_WAIT, ESTAB);
   EITHER = [ACK_WAIT, ESTAB];

initialize
   begin
      state to ESTAB;
      send_seq := 0;
      recv_seq := 0;
      send_buffer := empty;
      recv_buffer := empty;
   end;

predicate Ack_OK(NData: ndata_type);
begin
   Ack_OK := NData.id = ACK
             and (NData.seq = send_seq)
end;

procedure deliver_data(msg: msg_type);
begin
   out U.RECEIVE_response(msg.msgdata)
end;

procedure inc_recv_seq;
begin
   recv_seq := (recv_seq + 1) mod 2
end;

procedure inc_send_seq;
begin
   send_seq := (send_seq + 1) mod 2
end;

procedure remove(var buf: buffer_type;
                     msg: msg_type);
primitive;

function retrieve(buf: buffer_type): msg_type;
primitive;

procedure send_ack(msg: msg_type);
var a: ndata_type;
begin
   a.id   := ACK;
   a.data := msg.msgdata;
   a.seq  := null;
   out N.DATA_request(s)
end;

procedure send_data(msg: msg_data);
var s: ndata_type;
begin
   s.id   := DATA;
   s.data := msg.msgdata;
   s.seq  := msg.msgseq;
   out N.DATA_request(s)
end;

procedure store(var buf: buffer_type;
                    msg: msg_type);
primitive;

(* transitions *)

trans

from ESTAB to ACK_WAIT when U.SEND_request
begin
   p.msgdata := UData;
   p.msgseq  := send_seq;
   store(send_buffer,p);
   send_data(p);
   out S.TIMER_request(retransmit, retran_time)
end;

from ACK_WAIT to ACK_WAIT when S.TIMER_response
                      provided Name = retransmit
begin
   p := retrieve(send_buffer);
   send_data(p);
   out S.TIMER_request(retransmit, retran_time)
end;

from ACK_WAIT to ESTAB when N.DATA_response
                      provided Ack_OK(NData)
begin
   remove(send_buffer, NData.msg);
   incr_send_seq;
end;

from ESTAB to ESTAB when S.TIMER_response
                   provided Name = retransmit
begin
   (* do nothing; the message that caused
      this timer to be set has been
      acknowledged. *)
end;

from EITHER to SAME when N.DATA_response
                   provided NData.id = DATA
begin
   q.msgdata := NData.data;
   q.msgseq  := NData.seq;
   send_ack(q);
   if NData.seq = recv_seq then
      begin
         store(recv_buffer, q);
         incr_recv_seq
      end
end;

from EITHER to SAME when U.RECEIVE_request
         provided not buffer_empty(recv_buffer)
begin
   q := retrieve(recv_buffer);
   deliver_data(q);
   remove(recv_buffer, q)
end;
```

APPENDIX II
SUBGROUP C LANGUAGE: A SAMPLE SPECIFICATION

In this Appendix, we give a formal specification of a simplified Transport Service. The example reflects the status of the language as of mid-1982. We hope that the semantics of some of the interactions and interaction parameters can be understood without a detailed elaboration. A specification of the Transport Service using the most recent language definition is contained in the Subgroup C tutorial document of August 1983 [8].

Briefly, the Transport Service (TS) provides transparent transfer of data between Session Entities (SE's), relieving these entities from any concern about how the transfer is to be achieved. The service provided is connection oriented; it is necessary to establish, use, and terminate a two-point Transport Connection (TC). The TS enables the SE's to request TC parameters and a class of service quality selected from a predefined set of classes. The SE's may choose to accept or refuse a request for connection. The established TC represents a two-way simultaneous data path between a pair of SE's.

An SE can terminate this TC and inform the correspondent SE of this termination. If data are undelivered at the time of the termination request, it is not guaranteed that they will be delivered at all. If the TS is unable to maintain the quality of service agreed upon, it terminates the TC and informs both session entities.

There are two types of data that can be transported by the TC: Normal Data (ND) and Expedited Data (ED). ND is of arbitrary size and ED is of limited size. It is required that the ED be delivered at least as fast as the ND.

The part TS describes the service defined above. Each SE is characterized by a distinct transport address (here ap1 and ap2). TA denotes the set of these addresses. Each access point ap:TA.

TS interacts with SE's by means of interactions. There are the following interactions:

Connection request and Connection indication,
connection Acceptance request and connection Acceptance indication (currently called: Connection response and Connection confirmation),
Disconnect request and Disconnect indication,
Normal Data request and Normal Data indication,
Expedited Data request and Expedited Data indication.

In the specification they are represented by the above underlined letters. The direction of information exchange follows the request and indication conventions.

In all interactions, the first pair of indices gives the source and destination transport addresses.

The parameters in Ar and Ai whose range is the set CLASS gives the quality of the service provided by the TC. The value of such a parameter denotes a tuple of values: ⟨throughput rate1, throughput rate 2, average transit delay1, average transit delay2, maximum connection setup time⟩. The numbers 1 and 2 in the above parameters serve to distinguish between the possible values the TC can have in different directions. The parameters with range CAUSE denote the reason why a disconnect interaction takes place; the parameter p3 in Di denotes the originator of the disconnect. The parameters with range NDATA or EDATA denote the data transmitted during the interaction ND and ED.

The simple TS has only two access points, ap1 and ap2, and the connection can be set up only from access point ap1 (address-calling session entity) to access point ap2 (address-called session entity).

The definition of the temporal ordering used in this example is based on the assumptions that the primitive activities to be ordered have no explicit inner temporal structure and have finite, positive duration. Therefore, only the beginning and/or end of an activity may be temporally related to a beginning and/or end of another activity.

To order such activities we use two techniques: 1) temporal ordering primitives, seq, conc, muterm, and select, which specify a regular temporal ordering of phases, and 2) guards which enable an execution of a phase or activity when certain conditions are satisfied and in this way allow irregular temporal orderings. An additional restriction on the temporal ordering in which activities may be executed may be imposed by the parameter value dependence.

The following definitions of the temporal ordering constructs apply when these constructs are enabled and not disabled. The activities a_1 through a_n, and b_1 through b_n denote phases. A phase can be an interaction or a temporal construct of interactions.

seq $(a_1, a_2, a_3, \cdots, a_n)$. Here, seq specifies that phases $a_1, a_2, \cdots, a_n$ are temporally ordered so that a phase a_{i+1} *begins* some time after the phase a_i has *terminated*. seq has not begun when a_1 has not begun. seq terminates when a_n terminates.

conc $(a_1, a_2, a_3, \cdots, a_n)$. Here, conc specifies that phases $a_1, a_2, a_3, \cdots, a_n$ are temporally unrelated so that each may begin and hence terminate independently of each other. conc has not begun when none of the a_i has begun, and is terminated when all a_i have terminated. Note that in the example, the temporal independency of the phases in the conc construct is sometimes restricted by a value reference, as explained hereafter and shown in the example.

select $(a_1, a_2, a_3, \cdots, a_n)$ specifies that any one but only one of the phases $a_1, a_2, a_3, \cdots, a_n$ may begin. The beginning of a phase excludes the execution of other phases. select has begun when one of the phases a_i has begun and terminates when that a_i has terminated.

muterm $(a_1, a_2, a_3, \cdots, a_n)$ specifies a phase composed from phases $a_1, a_2, \cdots, a_n$ ordered in such a way that all a's are allowed to start independently provided no other a has terminated. When an a_i terminates, the start of those a's that are not yet started is prohibited, those a's that are active and can be disrupted are disrupted, and those a's that are active and cannot be disrupted terminate independently. muterm terminates when each of its constituting a_i is either prohibited to begin, disrupted, or terminated. muterm can be considered to be a weak form of select.

The value of function b-time (⟨activity reference⟩), used in guards, depends on the moment of the function evaluation. If at this moment the concerned activity has begun, b-time (⟨activity reference⟩) = begin time of the referred activity. Otherwise it is undefined. e-time (⟨activity reference⟩) is analogous to b-time, only it delivers the termination time of the referred activity.

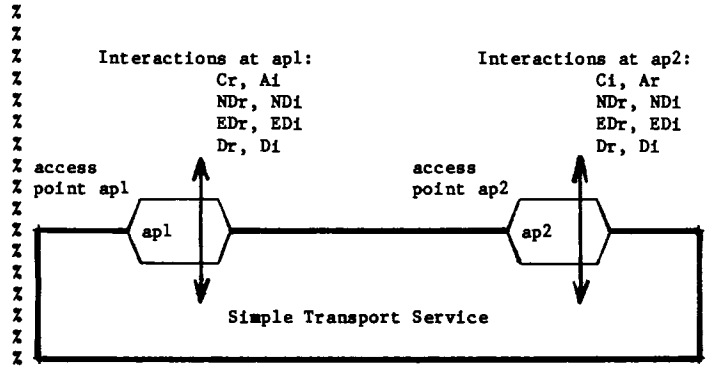
A particular value a parameter (or an n-tuple of parameters) can have during an interaction may be a value (n-tuple of values) that another parameter (n-tuple of parameters) has obtained during some other interaction. The value function, used in a reference to a particular interaction, is performed only when the parameters of the referred interaction are established, in other words when the referred interaction at least partially took place. In this way an implicit restriction is imposed on the temporal ordering of the interaction which uses the value reference.

We can associate with a phase or an ordering primitive a timing specification "$\langle t_1, t_2, t_3 \rangle$," where $0 \leqslant t_1 \leqslant t_2 \leqslant t_3$ and t_1 denotes the lower bound, t_3 the upper bound, and t_2 the average value in some time units that a phase or a temporal ordering primitive might take. For example, if in an interaction, data are exchanged between entities, the t_1, t_2, and t_3 in the notation $a\langle t_1, t_2, t_3 \rangle$ are measures of the throughput.

In the following formal description of the simple TS we added comments hoping that they make the specification self-contained. These comments start with a "%" character.

```
part simple transport service(access points ap1,ap2)

at ap1
disruptible interactions
NDr(data:NDATA),                          % interactions at access point ap1 of calling session
NDi(data:NDATA),                          % entity
EDr(data:EDATA),                          % Normal Data request interaction
EDi(data:EDATA),                          % Normal Data indication interaction
                                          % Expedited Data request interaction
                                          % Expedited Data indication interaction

non disruptible interactions
Cr(calling:{ap1}, called:{ap2}, class:CLASS),   % Connection request interaction
Ai(class:CLASS),                                % Acceptance indication interaction

Dr(reason:CAUSE),                         % Disconnect request interaction
Di((origin,reason):{(p2,p3)|p2="user" & p3:CAUSE   % Disconnect indication interaction
                    or p2="provider" & p3="undefined")}
```

```
at ap2                                                  % interactions at access point ap2 of called session
disruptible interactions                                % entity
NDr(data:NDATA),                                        % Normal Data request interaction
NDi(data:NDATA),                                        % Normal Data indication interaction
EDr(data:EDATA),                                        % Expedited Data request interaction
EDi(data:EDATA),                                        % Expedited Data indication interaction

non disruptible interactions
Ci(calling:{ap1}, called:{ap2}, class:CLASS),           % Connection indication interaction
Ar(class:CLASS),                                        % Acceptance request interaction

Dr(reason:CAUSE),                                       % Disconnect request interaction
Di((origin,reason):{(p2,p3)|p2="user" & p3:CAUSE        % Disconnect indication interaction
                           or p2="provider" & p3="undefined")}

% data parameter denotes normal or expedited data transmitted during an interaction
% class parameters denote the performance parameters
% reason parameter decribes why a user-initiated Disconnect request took place
% origin parameter, which has values "provider" or "user", denotes the originator of a Disconnect indication

define                                                  % define means the beginning of a set of abbreviations

   CLASS <=(thrp1,thrp2,avtrdel1,avtrdel2,maxconsetup)  % <= means 'is abbreviated by'
                                                        % CLASS is the name of an abbreviation; do not confuse
                                                        % CLASS with the set CLASS, which is left undefined

   MAX_CON_SETUP <= value(maxconsetup,Cr at ap1),       % maximum connection setup delay value
   AVERAGE1 <= value(avtrdl1,Ar at ap2),                % average transit delay in one direction
   THROUGHPUT1 <= value(thrp1,Ar at ap2),               % target throughput in one direction
   AVERAGE2 <= value(avtrdl2,Ar at ap2),                % average transit delay in opposite direction
   THROUGHPUT2 <= value(thrp2,Ar at ap2)                % target throughput in opposite direction

   EDATA_FASTER_THAN_NDATA

     <= ALL j:((b-time((j)th EDr at from)               % All those EDr that were accepted before i-th NDr must
               < b-time((i)th NDr at from)              % be delivered before i-th NDi
              )
              impl
              b-time((j)th EDi at to) < time
             )

   AVERAGE-DELAY(k,avtrdelay)
     <= (SUM(j:1..k-1):                                 % the sum of k transit delays for a normal data unit
         (b-time((j)th NDi at to) - b-time((j)th NDr at from)   % to cross the service part, divided by k must be
          )                                             % smaller than the average delay
          + time - b-time((k)th NDr at from)
        )/k <= avtrdelay

TRANSPORT(from,to,thrp,avtrdel)<=                       % abbreviation of normal and expedited data transport
                                                        % in one direction

conc(seq([i:pos int] NDr<,thrp,> at from),              % Normal Data interactions happen in sequence
     seq([i:pos int] EDr at from),                      % Expedited Data request inter. also happen in sequence
                                                        % thrp here as a measure of the throughput

     seq([i:pos int]                                    % Norm. Data indication happen in the same sequence as
           when EDATA_FASTER_THAN_NDATA &               % Norm. Data request but not faster than Exp.Data ind.
                AVERAGE_DELAY(i,avtrdel)                % and satisfying the average delay requirement
           do NDi(value((i)th NDr at from) at to),      % Implicit temp. ordering restriction on conc by
                                                        % parameter value reference, using value
                                                        % use of when guard
     seq([i:pos int] EDi(value((i)th EDr at from)) at to)   % Expedited Data ind's happen in the same sequence as
    )                                                   % Exp. Data requests and with the same values.

end define                                              % end of abbreviations section
```



```
conc
  (at ap1 do
   seq(Cr(ap1,ap2,CLASS),

       muterm
         (seq(when time < MAX_CON_SETUP + b-time(Cr at ap1)
              do Ai(value(Ar at ap2)),

              TRANSPORT(ap1,ap2,THROUGHPUT1,AVERAGE1),
             ),
          select(
                 Dr(reason1),
                 Di("user",value(Dr at ap2)),
                 Di("provider","undefined")
                )
         ),
      ),

  at ap2 do
   seq(Ci(value(Cr at ap1)),
       muterm
         (seq(Ar(value (Ci)),
              TRANSPORT(ap2,ap1,THROUGHPUT2,AVERAGE2),
             ),
          select(
                 Dr(reason2),
                 Di("user",value(Dr at ap1),
                 Di("provider", "undefined"),
                )
         )
      )

  )
end part
```

```
% Concurrent activities at ap1 and ap2 are taking place.
% Specification of activities at access point ap1:
% First interaction that is allowed to take place is
% Connection request interaction. For this very simple
% TS it must hold that source=ap1 and destination=ap2

% Performance specification of connection set up param.
% Interaction Ai can take place after Ar took place,
% an example of temporal ordering restriction by value
% dependence
% Connection use phase. See define section.

% the select allows one of the following interactions:
% local user (at ap1) initiated Disc. req
% remote user (at ap2) initiated Disc. ind
% provider initiated Disc. ind.
% select terminates when one Disc. has taken place.
% Termination of select disrupts seq of Ai and
% TRANSPORT, which terminates muterm
%
% The activities at ap2 are similar to activities at ap1
% Value ref. implies that Ci can take place after Cr
% took place, the parameter values of Ci are the values
% of parameters of Cr.
% Rest is similar to activities at ap1
% In this example no negotiation of Quality of Service
```

ACKNOWLEDGMENT

The authors wish to thank John Day for his work in setting up the OSI FDT group as well as for his helpful comments on a draft of this paper.

BIBLIOGRAPHY

[1] T. Blumer and R. L. Tenney, "An automated formal specification technique for protocols," *Comput. Networks*, vol. 6, pp. 201–217, 1982.

[2] G. V. Bochmann and C. Sunshine, "Formal methods in communication protocol design," *IEEE Trans. Commun.*, vol. COM-28, pp. 624–631, Apr. 1980.

[3] G. V. Bochmann, E. Cerny, M. Gagne, C. Jard, A. Léveillé, C. Lacaille, M. Maksud, K. S. Raghunathan, and B. Sarikaya, "Some experience with the use of formal specifications," in *Protocol Specification, Testing, and Verification*, C. Sunshine, Ed. (Proc. IFIP WG6.1 2nd Int. Workshop on Protocol Specification, Testing, and Verification, Idyllwild, CA, May 1982). Amsterdam, The Netherlands: North Holland, 1982, pp. 171–185.

[4] E. Brinksma, "An algebraic language for the specification of the temporal order of events in services and protocols," in *Proc. European Teleinformatics Conf.*, pp. 533–543, Varese, Oct. 1983.

[5] G. J. Dickson and P. E. de Chazal, "Status of CCITT description techniques and application to protocol specification," this issue, pp. 1346–1355.

[6] International Organization for Standardization, "Draft Tutorial Document: Concepts for describing the OSI architecture," ISO/TC97/SC16/N1346, Dec. 1982.

[7] International Organization for Standardization, "Draft Tutorial Document: Extended finite state machine specification," ISO/TC97/SC16/N1347, Dec. 1982 (latest revision, May 1983).

[8] International Organization for Standardization, "Draft Tutorial Document: Temporal ordering specification language," ISO/TC97/SC16/WGL/N157, Aug. 1983.

[9] International Organization for Standardization, "Draft trial specification of a Class 2 Transport Protocol," ISO/TC97/SC16/WG1/N117, Aug. 1983.

[10] S. Schindler, U. Flasche, and D. Altenkrüger, "The OSA Project: Formal specification of the ISO Transport Service," in *Proc. Computer Network Symp.* (NBS), pp. 136–160, Dec. 1980.

[11] R. L. Tenney, "One formal description technique for ISO OSI," in *Proc. IEEE Int. Conf. Communications* (Boston, MA, June 1983), pp. 1296–1300.

[12] J. Vytopil and C. A. Vissers, "Interaction primitives in the formal specification of distributed systems," Twente Univ. Tech. Rep., Feb. 1981.

[13] R. Schwartz and P. M. Melliar-Smith, "Temporal logic specification of distributed systems," in *Proc. 2nd Int. Conf. on Distributed Systems* (Paris, France, Apr. 1981).

[14] R. Milner, *A Calculus of Communicating Systems*. Berlin:Springer, 1980.