A METHODOLOGY FOR PROTOCOL DESIGN AND SPECIFICATION BASED ON AN EXTENDED STATE TRANSITION MODEL

Richard Chung
CNCP Telecommunications
Toronto, Canada

A B S T R A C T

A lot of effort is being dedicated worldwide to defining Open Systems Interconnection (OSI) protocol standards and tools for formally describing them. However, little work is done in developing methods for applying these tools to the development of new protocols.

This paper presents such a methodology based on OSI and software engineering principles. It provides general guidelines for designing protocols based on an extended state transition formal description technique (FDT). It emphasizes a systematic, analytical and algorithmic approach to achieve completeness of protocol specification during the protocol design process. Its possible application to the development of OSI protocol standards is discussed and suggested for study by the standards community.

1. Introduction

Due to the complexity of designing and specifying a complete set of standards for OSI [Zimm], there has been a growing interest in both the International Telegraph and Telephone Consultative Committee (CCITT) and the International Organization for Stardardization (ISO) in developing methods for formally describing the services and protocols of OSI. Of these methods, the Extended Finite State Machine Specification, also called the Extended State Transition (EST) model [ISO/TC97/SC16/ N1347], has reached the higher level of maturity and acceptance because of its familiarity to protocol designers and implementors. The EST method has two forms of representation: a graphical form and a program-like linear form. This paper presents some user guidelines for applying the linear form to the design and specification of protocols.

The S.70 Teletex transport protocol [S70] was chosen as the example in this paper because a) it is a well-known protocol which is integrated in the OSI transport protocol standard as Class 0, b) transport protocol Class 0 is used within the CCITT and ISO FDT groups as basic trial example specification, and c) it is complex enough to demonstrate how the user guidelines can be applied. Reference is also made to OSI transport protocol class 1 [Tran] when required.

Section 2 presents a short description of the extended state transition FDT used.

Section 3 presents the major steps in the design of OSI protocols as the basis for a methodology. These steps are service analysis and specification, and protocol specification.

Section 4 elaborates on a protocol design methodology which consists of methods for a) partitioning service and protocol elements, b) developing a formal service specification, c) deriving protocol elements from the service specification, d) stepwise refinement of the protocol specification to represent the relationships between protocol elements and service primitives, and e) choosing data structures. It suggests that completeness of protocol specification should be achieved during the design process and not through post-design analysis and maintenance. An algorithmic approach to achieve the completeness of the stepwise refinement process and hence the protocol specification is presented.

Section 5 outlines how the extended state transition model FDT could possibly be applied to the development of OSI protocol standards. It is finally suggested that such a methodology be studied and defined within the standards community.

2. Extended State Transition Model FDT

This paper is restricted to an FDT for protocol specification. The more general applications of FDT are covered in [Boch 82]. An FDT provides the means for precisely specifying communication protocols in the form of an abstract representation of their essential requirements and aspects. It should allow the production of accurate, unambiguous, sufficiently complete, implementation-independent specifications which could hopefully be automatically verified and implemented.

The FDT used in this paper is a preliminary version of the extended state transition model developed by Subgroup B of the ISO/TC97/SC16 WG1 ad hoc group on FDT [ISO/TC97/SC16/N1347]. This is a descriptive method which combines the event-driven finite state machine model of protocols with the power of the high-level programming language Pascal. Pascal is used to express the

processing routines and data structure operations associated with each state transition.

In the extended state transition model, a system is viewed as consisting of interconnected modules, each of which is an extended finite state transition machine. For example, Figure 1 shows a model of a transport layer service provider (or "system") comprising of a transport entity module, a mapping module and a local buffer module.

A module may participate in two types of interactions, namely:

a) an exchange of control information with a local module to initiate or coordinate the execution of some function in a user/provider relationship (a "service primitive" interaction);

b) an exchange of a Protocol Data Unit (PDU) with a remote peer module to cooperatively execute some procedure (a PDU interaction).

The set of interactions (or interaction primitives) between a given module and another module in its environment is called a channel. The specification of a module includes an enumeration of the interaction points through which it interacts with its environment.

Each module is defined in terms of:

a) a state space, which defines all the (internal) states in which the module may be at any possible point in time;

b) the possible transitions from each state to a "next" state.

The state space of a module is specified by a set of variables. A possible state is characterized by the values of these variables. One of these variables is called "STATE". It represents the "major state" of the module.

In a given major state, the possible transitions which may occur depend on the values of some of the state space variables (an enabling predicate), and possibly on the reception of an input interaction. If several transitions are possible at some given time, the model assumes that an implementation will execute only one transition at the time. The choice is not specified but the transitions may be assigned a relative priority. The operation of executing a transition performs some action which may change the values of variables and may initiate some output interaction(s) with the environment.

The possible transitions of a module are defined by the specification of a number of transition types. Each transition type is characterized by:

a) the enabling condition - this includes:
- the present major state (FROM clause)
- an optional input interaction (WHEN clause)
- an optional additional enabling predicate (PROVIDED clause)

b) the operation of the transition - this includes:
- the next major state (TO clause)
- the action (BEGIN statement which may include the generation of output interaction(s))

The syntax of the FDT gives the specification writer total flexibility in presenting the definition of transition types. However, there are two major approaches to organizing the transitions, namely:

a) an interaction-oriented approach, in which transitions are ordered according to the input interaction. Transitions with the same WHEN clause are collected and possibly nested together;

b) a state-oriented approach, in which the transitions are ordered according to their present major state, that is the FROM clause.

The interaction-oriented approach provides a disciplined approach to the stepwise refinement of the protocol specification as elaborated below.

This paper provides some user guidelines for developing a complete specification of the transition types. They are based on the state diagrams shown in Figure 2 for the error-free connection establishment procedure of the S.70 transport layer procedure (class 0). Separate diagrams are shown for the calling end and called end for simplicity of presentation.

3. Elements of Protocol Design and Specification Methodology

The major work on FDT in the OSI context has so far been in defining the model, concepts and language for transcribing and formally representing or analysing existing protocols such as X.25 or S.70. There has been little work on using the FDT as an aid in the design and development of new protocols. We believe that it should be possible to start formalizing protocol design through an FDT and OSI principles.

Protocol design encompasses all these activities concerned with refining the definition of some communications service requirements into the specification of a protocol whose implementation will satisfy those requirements. It is usually preceded by a service definition step which consists of identifying and producing a formal statement of the service requirements from the user point of view. Protocol design consists of two major steps: a) service analysis and specification, and b) protocol specification.

The service analysis and specification step focuses on the protocol functions which would be required to realize the user requirements. It identifies in more technical details what is to be done but not how it is to be accomplished. The service specification defines all the functions from the point of view of the system as a whole independently of its internal layered structure.

The protocol specification step completes the analysis and produces a protocol description in terms in which it can be implemented and tested. In this respect, protocol specification is an inherent aspect of protocol design. The protocol specification defines the distribution of functions to the various entities in each layer of the system and the way they operate.

There is a close analogy between these protocol design steps and certain stages of the software life cycle [Wass]. The service analysis and specification step corresponds to the software requirements analysis and definition stage. The protocol specification step covers both the software architectural (or preliminary) and part of the detailed design stages.

During the design of pre-OSI protocols (such as HDLC, X.25, S.62, S.70), no formal service specifications were available. At best, there might have been a definition of the service in user-oriented terms (e.g. Recommendation F200) for which the protocol (e.g. Recommendation S62) was designed. The practice typically consisted of defining the functions and protocol elements in a

piecemeal, independent manner, then consolidating them together, if required, in a procedural description. State diagrams would typically be left out (e.g. HDLC or S.70) or added as an afterthought! Then, the rigorous analysis of the protocol or implementation experience would uncover inconsistencies and ambiguities between the different parts of the specification. Much time and effort would be spent in enhancing, unifying and harmonizing these parts.

Due to the ever-increasing complexity of OSI protocols, there is a need for a more disciplined approach to the protocol design process. The objective is to produce a better unified and harmonized protocol right from the start. This could be accomplished by a methodology which is based on sound OSI principles. It should follow rigorously the steps in the protocol design process and require specific deliverables for in-depth review at the completion of each step.

The components of such a methodology are:

a) partitioning principles;

b) rules for formally defining protocol functional requirements in terms of service primitives. A service primitive is "an abstract, implementation independent element of an interaction between a service-user and the service-provider at a service-access-point". The resulting deliverable is a service specification [Boch 80], for example the transport service definition document [Trans].

c) rules for deriving protocol elements from the service specification, and completely describing the behaviour of the interacting modules in terms of the relationships between protocol elements, service primitives and possible orders of execution. A protocol element may be a protocol data unit, which is interchanged between peer communication entities, or a mapping to a local function, e.g. a service primitive. The resulting deliverable is a protocol specification [Boch 80], for example the transport protocol specification document [Tran].

4. FDT User Guidelines for Protocol Design and Specification

The guidelines are based on existing practices and principles. Some of the proposed rules have actually been informally used in the development of new protocols such as for access to message handling systems [MHS2]. However, the major examples given in this paper are in terms of how the S.70 transport protocol could have been formally developed because S.70 is a well-known simple protocol suitable for illustrating the principles.

4.1 Partitioning Methods

Partitioning is a key technique which is extensively used to master the complexity of protocol design. It is the conceptual decomposition of a system or specification into smaller components in order to facilitate its understanding, validation and implementation. Two major partitioning methods used in protocol design are: logical phase partitioning and functional partitioning.

Logical phase partitioning is the basis of the proposed methodology for the design of OSI protocols. It consists of grouping the service primitives to derive protocol elements. It also provides the framework for the stepwise refinement of the protocol specification as presented below.

However, in order to effectively apply the stepwise refinement process, the internal structure of the service provider needs to be elaborated. This requires a form of functional partitioning which decomposes a system into relatively independent components. These components could also be modelled as individual finite state machines. Service primitives could be defined to express the interactions between the components.

An appropriate functional partitioning structure is normally evident from the functional analysis inherent in the process of deriving protocol elements after logical phase partitioning. A simple functional partitioning structure which is applicable to S.70 and Class 0 is shown in Figure 1. If necessary, further decomposition is possible through a more in-depth analysis of the protocol elements and their interrelationships. An example is the decomposition of the HDLC Classes of procedures into the functional components called link set up, source (of data), sink (of data), PF-control (poll/final bit procedure), clock (timer operation), transmission (FCS error check, buffer management), and checkpointing (poll/final bit error recovery) [Boch 77]. The structuring concepts used in that example seem generally applicable to the specification of OSI protocols whose components are reusable in several contexts, e.g. the transport protocol classes. However, such a level of decomposition is not required for simpler protocols such as those for Teletex (S.62, S.70).

4.2 Developing the Service Specification

It is desirable that the user service requirements definition be presented in, or at least decomposeable into, atomic, user-oriented capabilities called service elements [MHS1]. This identification and definition of the most elementary functions required would ensure complete coverage of the service requirements.

As an example, some service elements which could be defined for the Transport Layer Service [Trans] are: Called Address, Calling Address, Expedited Data Option, Quality of Service, and TS-user data. These are some capabilities which the Transport provider allows the Transport users to interchange.

4.2.1 Service Primitives

The objective of the service analysis step is to produce a more formal representation of the capabilities corresponding to or realizing the service elements. There should be some guidelines for developing this representation in terms of service primitives. One guideline is to define service primitives by logically packaging together the service elements in a manner that the occurrence of the primitive represents a logically indivisible event which has some meaning to the service-user. Effectively, the service elements become parameters in one or more service primitives [MHS2]. For example, the transport layer service elements mentioned above are components of the T-Connect request and T-Connect indication primitives.

4.2.2 Logical Phase Partitioning

In a service specification, the service

primitives should be grouped together according to their types and interrelationships [ISO/TC97/SC16/N897] in order to permit the methodological derivation of protocol elements. Primitives could be of four types: a) request or response by a service-user; b) indication or confirm by a service provider.

The primitives are grouped according to a) whether the service is unconfirmed or confirmed, and then in b) phases of operation. A service is said to be unconfirmed if a request primitive originated by a service-user results only in an indication primitive to the remote service-user. For example, the Transport connection release service is an unconfirmed service consisting of the T-D-req and T-D-ind primitives. A service is said to be confirmed if additionally the remote service-user issues a response primitive which results in a confirm primitive to the originating service-user. For example, the Transport connection establishment service is a confirmed service consisting of the T-C-req, T-C-ind, T-C-resp and T-C-conf primitives.

A phase of operation is a set of service primitives which together perform some logically related services with a well defined meaning to peer service-users. Each service primitive may form part of unconfirmed or confirmed groups or be otherwise unrelated. For example, the transport service is divided into three phases: connection establishment, data transfer and connection release.

4.3 Deriving Protocol Elements from Service Specification

The criteria for deriving protocol elements are based on the functions performed by the layer. These functions are a) those which are visible to the user (i.e. service elements or service primitive parameters), and b) those which are required for the internal management of the layer in order to realize the OSI (N)-service primitives from the (N-1)-service primitives, for example to ensure data integrity, synchronization, error control.

A possible method for deriving protocol elements consists of the following steps:

1) consider each phase in turn;

2) for each phase, consider each service request primitive in turn and

A) map it directly into one or more protocol elements (an E1 set) in association with its service indication primitive, if any.

B) define additional protocol elements which are required to complete the interactions initiated by set E1 in point 2a). Criteria include:

B1) does an E1 protocol element demand a protocol response, e.g two-way handshake? If so, the acceptance and rejection protocol elements need to be defined though they are not part of an end-to-end confirmed service. This is the case, in S.70, of TCC as a rejection of a TCR and, in class 1, the case of DC as an acceptance of DR [Tran].

B2) need for indicating an error in reception of an E1 protocol element e.g. with an incorrect parameter. This is the case of the S.70 Transport Block Reject (TBR) or Class 0 Error (ERR) protocol element.

B3) need for error recovery protocol elements. This is the case of the Class 1 RJ protocol element.

C) if the service request primitive is part of a confirmed service, the related service response primitive is mapped directly into one or more protocol elements in association with its service confirm primitive. This is the case, in S.70, of TCA in response to TCR.

3) for each phase, repeat step 2 for any primitive not considered in step 2, e.g. unrelated indication primitive. This is the case of the N-D-ind initiated within the network layer.

The application of this method to S.70 and Transport protocol classes 0 and 1 is summarized in Figure 3. It also shows the steps under which the protocol elements are derived.

4.4 Stepwise Refinement of Protocol Specification

This is a critical step of the protocol design process. For a design to be a successful solution, it should be simple and closely related to the structure of the problem in a natural manner. A natural solution to the protocol design problem is provided by an interaction-oriented FDT approach which also ensures completeness.

4.4.1 Completeness

A protocol specification is considered to be complete when it has defined the effects of all possible input interactions from the environment to the specified module in all relevant situations. In a specification, the reception of some input interaction (with some specific parameter values) in some module state could: a) result in a defined normal or error transition, or b) be correctly ignored, or c) be unintentionally omitted. The latter case is an "unforeseen error" due to incomplete or incorrect assumptions about the possible errors in the real world environment [Boch 82]. The distinction between the last two cases is not always evident. In a complete protocol specification, the possibility of "unforeseen errors" should not arise, and the occurrence of an undefined context should be intentionally ignored and treated as an irrelevant situation. The refinement algorithm presented below fulfils this condition.

The amount of error situations to be defined is subject to debate. It is argued that some error cases need not be standardized because they may have local significance only. However, it is important that protocol standards identify and suggest actions for all error cases (including local ones) which should not be ignored so as to:

a) serve as an implementation guide and avoid deadlock situations due to unforeseen error situations;

b) allow development of robust and resilient implementations. The implementation of transitions for handling local errors provides an important debugging tool during system integration testing. Such transitions could be selectively enabled or disabled at run-time. Provided that the interfaces (channel type definitions) are not changed, it should be possible to isolate the protocol module from side effects due to modifications, additions or complete redesign of the modules with which it interacts. If each module implements this self-protection mechanism then the system becomes easier to maintain.

4.4.2 Algorithm for Interaction-Oriented Specification Refinement

In an interaction-oriented approach, there is a need to consider all the possible states for each input interaction together with the relevant context variables. These variables are those associated to the sequence of preceding interactions which led to the state under consideration. The proposed method shown in Figure 4 to achieve this completeness has the following basis:

a) by following the logical sequence inherent in the methods described above for logical phase partitioning and protocol element derivation, we have a stepwise algorithm for identifying all possible input interactions. These should include interactions for possible exception conditions. The latter are service primitives used by the (N+1)-entity to reject the (N)-entity's output interactions or by the (N-1)-entity to indicate significant events.

b) the algorithm is refined by providing rules for considering all the situations in which an input interaction may occur. Situations are categorized as: normal, rejection, abnormal. A normal situation exists when the interacting entities cooperatively and correctly execute the interaction in the correct state(s) as defined by the protocol. A rejection situation occurs when an entity receives an input interaction in conformity with the protocol (that is, in the correct state(s)) but cannot execute it. If this is due to some acceptable or defined conditions, the receiver then issues an output interaction to indicate the rejection to the requestor. The rejection conditions include parameter errors in received interaction, internal constraints of the receiving entity, inability of receiver to satisfy requested function. An abnormal situation occurs when an input interaction is received in an incorrect state(s). It may arise due to an incorrect implementation of the protocol/interface (i.e channel) specifications or unforeseen perturbations or failures of the environment.

c) a systematic, disciplined approach to the consideration of each category of situations separately for each input interaction eliminates "unforeseen errors". The algorithmic structure and presentation format of the specification implicitly provide guidelines as to the context variables to be considered.

d) as a global check of completeness, the interaction-oriented specification for an (N)-entity should contain a WHEN clause for all protocol data units (PDU's) and all service primitives, which may be issued by the (N+1) or (N-1) entities. The (N)-entity issues its service primitives as output interactions.

The stepwise protocol specification refinement method illustrated by the algorithm in Figure 4 refers to input interactions only. The service primitives are first partitioned in phases of operation as described above. The major steps of the algorithm for each phase in turn are:

A1) derive or extend the state transition diagram/description for normal operation;

A2) for each interaction occurrence, consider the possible rejection and abnormal situations and note any new interactions or states introduced;

A3) extend the state transition diagram/description with new interactions from step A2;

A4) extend the state transition diagram/description with interactions used for exception conditions.

When all the phases have been considered, the actions corresponding to the occurrence of each interaction in each of the new state, introduced in steps A1 to A4, are defined.

Figure 5 shows a partial specification resulting from the application of this algorithm to the connection establishment and release phases of S.70. It also indicates the order in which the refinement was progressed. Its purpose is to show the mapping between input and output interactions and the clear separation between normal, rejection and abnormal situations. All other details are abstracted out. A cause parameter of the T-D-ind primitive is used to formalize the treatment of local error conditions for the guidance of implementors. Such a usage is possible because the primitive can be invoked at any time by the TS-provider.

Step A1 is performed by a straightforward refinement of the service specification according to the groups of primitives within the phase as defined above. An unconfirmed service group is refined by formally defining the mapping of its service request primitive into local functions and PDU's and the mapping of these PDU's into service indication primitives. A confirmed service group is refined by formally defining the mapping of its additional service response primitive and associated PDU's into service confirmation primitives. This is illustrated by substeps 1 to 4 in Figure 5.

Step A2 may result in new interactions, typically the PDU's defined during protocol element derivation for rejecting other PDU's or indicating errors. This is illustrated by substeps 5 to 8 in Figure 5.

In step A3 the handling of these new input interactions (substeps 9 to 11 in Figure 5) may lead to new states. The impact of these new states on existing interactions is postponed to the end of the phase refinement process in order to simplify and minimize the number of iterations.

In step A4 (substeps 12 to 14 in Figure 5), care should be taken on the possibility that the service primitive used for rejection could also be used for another function, such as entering a new phase. This provides a natural basis for choosing the order in which phases are to be considered. For S.70, it is natural to consider the release phase (substeps 13 to 14 in Figure 5) after the establishment phase.

4.5 Choice of Data Structure Representations

During the interaction-oriented refinement process, new context variables need to be defined to represent the states. While defining the semantics (actions) of the transitions, the protocol syntax (data representation, coding) has to be developed in parallel. In particular there is a need to choose intermediate data structures, such as data buffers. These are used in mapping the user data representation, given in the service primitives, into the protocol data representation which is transferred in the PDU. Such a choice could also be made in the partitioning step.

The formalism of abstract data types

developed for software engineering seems to be applicable to the FDT based on the extended state transition model [Boch 82]. General guidelines for selecting appropriate data structures include:

a) the chosen data structure should give the user a clear appreciation in the simplest manner possible of the capabilities available in the protocol and their possible usage;

b) the operations on the chosen data structure should closely reflect the functions invoked by the PDU's. As an example, in a Teletex document protocol formal specification, a Command Document Start may cause the receiver to open a document file in which subsequent text is written. A Command Document Continue after a document interruption may cause the receiver to reopen a selected document file and position it to the point where the document is to be linked and overwritten, as needed;

c) the chosen data structures should help a protocol implementor in understanding the protocol and deriving a straightforward data mapping to his memory and file management systems;

d) they should be implementation-independent and accommodate a wide range of implementation environments. As an example, in a Teletex document protocol formal specification, a document may be implemented as a file of records in which each record represents a Teletex page or a set of linked files each of which represents a Teletex page.

5. Application to OSI Protocol Development

The development of any protocol of some complexity is highly iterative. Protocol standards are usually developed by sizable committees. A committee would typically design a protocol by successively defining the functions (from the service definition), semantics, syntax and state diagrams. Each step may be developed by several parallel independent subgroups and the steps may overlap. An editor is appointed to consolidate the natural language descriptions produced by all the subgroups into a single document. His role is to ensure a uniformity of style as well as identify any inconsistency between the different texts or any omission and ambiguity. However, the imprecise nature of natural languages and the complexity of the protocol make the editor's task rather difficult.

The extended state transition model FDT provides a useful tool for the editor to formally analyze and record the agreements made by the different groups. The resulting formal specification could act as a catalyst for uniform understanding of the agreements. An interaction-oriented format would provide a check on the correctness of the relationships between the service specification and the agreements. This parallel development of natural language and FDT specifications would reduce inconsistencies and promote completeness.

In order to benefit most from the methodology, it is suggested that:

a) sub-groups be assigned to study, in parallel, functions for each specific phase;

b) sub-groups record their agreements in a standard format such as the following:

- Name of protocol element
- Description of function
- Condition for and action to be taken on sending
- Condition for and action to be taken on reception
- Special considerations

c) editor integrates the different subgroup reports by applying the interaction-oriented specification refinement algorithm to produce a formalized specification which identifies the abnormal situations;

d) whole group reviews the formalized specification for consistency, accuracy and completeness. It decides whether abnormal situations can be ignored and, if not, identifies them specifically with possibly a suggested action to guide the implementor.

6. Conclusion

A lot of effort is being dedicated to defining OSI protocol standards and tools for formally specifying them. However, little work is done in developing methods for applying these tools to the design and specification of complete, accurate and correct protocols.

International standards development committees should give deeper consideration to this topic so that compatible, as well as truly robust and resilient, implementations could be developed. The methodology proposed in this paper could be an input to such a study. It emphasizes a systematic analytical approach to achieve completeness during the protocol design process. The desirable result of the study should be user guidelines which could be part of the standards on formal description techniques. Guidelines on the formalization of protocol functions, such as multiplexing and flow control, within this framework are also required.

The systematic and consistent application of these user guidelines to the design and specification of OSI protocol standards would ensure a uniform understanding by all interested parties. This would promote their wider acceptance and timelier implementation, thus bringing OSI a step closer to realization.

Acknowledgement

The author expresses his deep appreciation to G.V. Bochmann for his valuable comments on the preliminary outline which gave birth to this paper, and his continuous support.

References

[Boch 77] G.V. Bochmann and R.J. Chung, "A formalized specification of HDLC classes of procedure". Proc. Nat. Telecommun. Conf., Los Angeles, CA, Dec. 1977, Paper 3A.2.

[Boch 80] G.V. Bochmann and C.A. Sunshine, "Formal methods in communication protocol design". IEEE Trans. Commun., vol. COM-28,pp. 624-631, Apr. 1980.

[Boch 82] G.V. Bochmann et al., "Experience with formal specifications using an extended state transition model". IEEE Trans. Commun., vol. COM-30,pp. 2506-2513, Dec. 1982.

[ISO/TC97/SC16/N897] "Data Processing-Open Systems Interconnection-Service conventions".

[ISO/TC97/SC16/N1347] "A FDT based on an extended state transition model (Working Document, November 1982)".

[MHS1] "Draft Recommendation X.400: Message Handling Systems: System Model-Service Elements (Version 2)". Dec. 1982.

[MHS2] "Draft Recommendation X.411: Message Handling Systems: Message Transfer Layer". Dec. 1982.

[S70] "Recommendation S.70, Network-independent basic transport service for Teletex". CCITT, Yellow Book, Fascicle VII.2, 1980.

[Tran] "Information processing systems-Open systems interconnection-Transport protocol specification". ISO/ DP8073.

[Trans] "Information processing systems-Open systems interconnection-Transport services definition". ISO/DP8072.

[Wass] A.I. Wasserman, "Information system design methodology". In: P. Freeman, A.I. Wasserman (Ed.), "Tutorial on software design techniques (third edition)", IEEE, pp.25-44, 1980

[Zimm] H. Zimmermann, "OSI reference model - the ISO model of architecture for Open Systems Interconnection", IEEE Trans. Commun., Vol. COM-28, pp. 425-432, Apr. 1980.

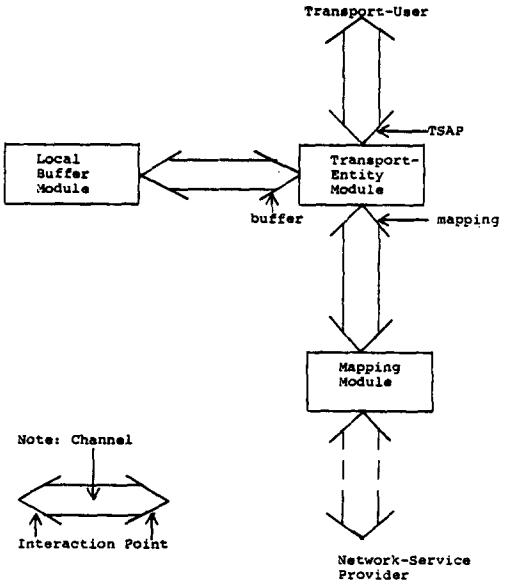

Figure 1 - Model of Transport Service Provider

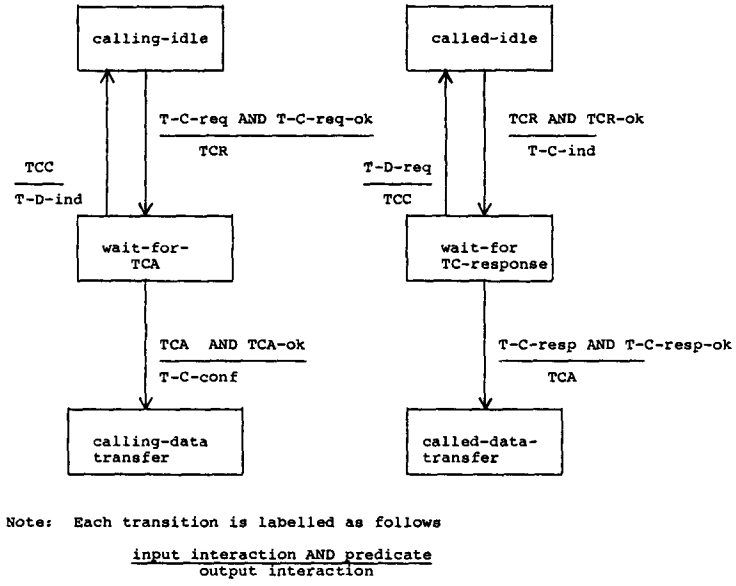

Figure 2 - State Diagrams for S.70 Error Free Connection Establishment Procedure

Phase	Service Primitives		Protocol Elements		
			S.70	Class 0	Class 1
Connection Establishment	T-CONNECT request	T-C-req	TCR(A),TBR(B2)	CR(A),ERR(B2)	CR(A),ERR(B2)
	indication	T-C-ind	TCR(A)	CR(A)	CR(A)
	response	T-C-resp	TCA(C),TBR(B2)	CC(C),ERR(B2)	CC(C),ERR(B2)
	confirm	T-C-conf	TCA(C)	CC(C)	CC(C)
Data Transfer	T-DATA request	T-DT-req	TDT(A),TBR(B2)	DT(A),ERR(B2)	DT(A),ERR(B2),AK(B1), RJ(B1,B3)
	indication	T-DT-ind	TDT(A)	DT(A)	DT(A)
Connection Release	T-DISCONNECT request	T-D-req	TCC(B1),TBR(B2), N-D-req(A)	DR(B1),ERR(B2), N-D-req(A)	DR(B1,A),ERR(B2), DC(B1)
	indication	T-D-ind	TCC(B1),N-D-ind(A,3)	DR(B1),N-D-ind (A,3)	DR(B1,A)

Note: The references in parenthesis correspond to the steps in section 4.3

Figure 3 - Derivation of Protocol Elements from Service Primitives

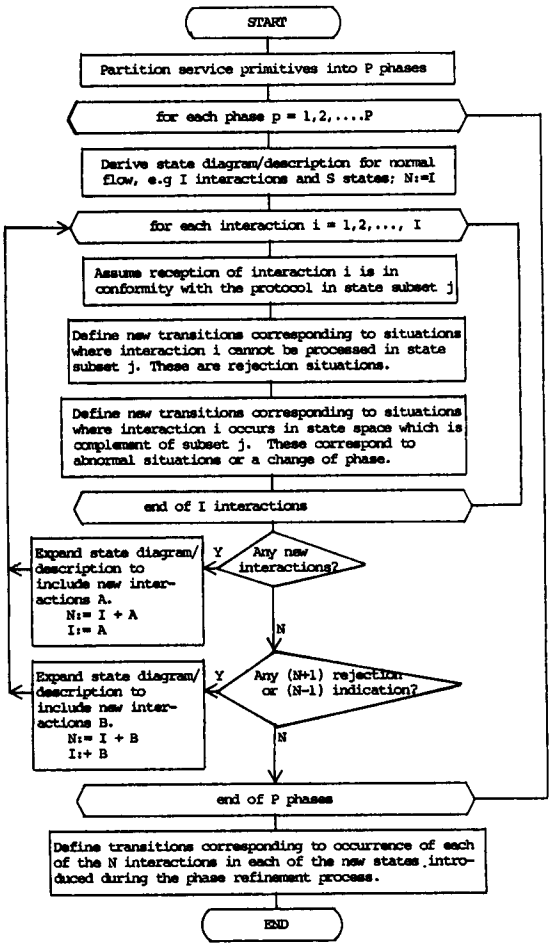

Figure 4 - Interaction-Oriented Refinement Algorithm

```
1.(   WHEN TSAP.T-C-req
  (    FROM calling-idle
  (    PROVIDED T-C-req-ok TO wait-for-TCA
  (    BEGIN mapping.TCR;... END;

5.(    PROVIDED NOT T-C-req-ok TO SAME (* parameter error*)
  (    BEGIN TSAP.T-D-ind (T-C-req-error);... END; (*suggestion*)
  (    FROM [wait-for-TCA, calling-data-transfer, called-idle,
  (         wait-for-TC-response, called-data-transfer]
  (    TO idle (* initial state, not in Figure 2*)
  (    BEGIN mapping.N-D-req (local-error-type); ...
  (         TSAP.T-D-ind (local-error-type);
  (
  (    END;
////////////////////////////////////////////////////////////////

2.(   WHEN mapping.TCR
  (    FROM called-idle
  (    PROVIDED TCR-ok TO wait-for-TC-response
  (    BEGIN TSAP.T-C-ind (connection-request); ... END;

6.(    PROVIDED NOT TCR-ok TO SAME
  (    BEGIN mapping.TCC (cause);... END; (* S.70, 5.5.6.1 *)
  (    FROM [calling-idle, wait-for-TCA, calling-data-transfer,
  (         wait-for-TC-response, called-data-transfer]
  (    TO SAME
  (    BEGIN mapping.TBR;... END;
////////////////////////////////////////////////////////////////

3.(   WHEN TSAP.T-C-resp
  (    FROM wait-for-TC-response
  (    PROVIDED T-C-resp-ok TO called-data-transfer
  (    BEGIN mapping.TCA; ... END;


7.(    PROVIDED NOT T-C-resp-ok TO called-idle
  (    BEGIN mapping.TCC (cause);...
  (         TSAP.T-D-ind (T-C-resp-error); (*suggestion*)
  (    END;
  (    FROM [calling-idle, wait-for-TCA, calling-data-transfer,
  (         called-idle, called-data-transfer]
  (    TO idle
  (    BEGIN mapping.N-D-req (local-error-type);...
  (         TSAP.T-D-ind (local-error-type);
  (    END;
////////////////////////////////////////////////////////////////

4.(   WHEN mapping.TCA
  (    FROM wait-for-TCA
  (    PROVIDED TCA-ok TO calling-data-transfer
  (    BEGIN TSAP.T-C-conf;... END;

8.(    PROVIDED NOT TCA-ok TO calling-idle (*may retry or clear*)
  (    BEGIN TSAP.T-D-ind (parameter-mismatch);...END;
  (    FROM NOT wait-for-TCA TO SAME
  (    BEGIN mapping. TBR;...
  (         (* activate error recovery *)
  (    END;
////////////////////////////////////////////////////////////////

9.(   WHEN mapping.TCC (cause)
  (    FROM wait-for-TCA TO calling-idle (*may retry or clear*)
  (    BEGIN TSAP.T-D-ind (cause);... END;

11.(  FROM NOT wait-for-TCA TO SAME
  (    BEGIN mapping.TBR;...
  (         (*activate error recovery*)
  (    END;
////////////////////////////////////////////////////////////////

10.(  WHEN mapping.TBR
  (    FROM ANYSTATE TO SAME
  (    BEGIN...
  (         (*examine reject cause and determine action*)
  (    END;
////////////////////////////////////////////////////////////////

12.(  WHEN TSAP.T-D-req
  (    FROM wait-for-TC-response TO called-idle
  (    BEGIN mapping.TCC (cause);... END; (*rejection of T-C-ind*)

13.(  FROM NOT wait-for-TC-response TO idle
  (    BEGIN mapping.N-D-req (normal-clear);...END; (*phase change*)
////////////////////////////////////////////////////////////////

14.(  WHEN mapping.N-D-ind (cause)
  (    FROM ANYSTATE TO idle
  (    BEGIN TSAP.T-D-ind (cause);...END;
////////////////////////////////////////////////////////////////
```

Note: Numbers to the left indicate the order in which refinement was progressed in accordance with Figure 4 into associated set of transitions.

Figure 5 - Partial Specification for S.70 Transport Connection Establishment and Release Phases