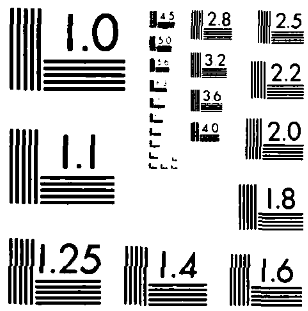
AD-A126 558 PROTOCOL SPECIFICATION REPORT(U) SYSTEM DEVELOPMENT CORP SANTA MONICA CA G A SIMON 29 MAR 82 SDC-TM-7172/301/00 DCA100-82-C-0036 1/1

UNCLASSIFIED F/G 17/2 NL

END
DATE
FILMED
4 83
DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963 A

AD A126558

(2)

SDC

System Development Corporation
2500 Colorado Avenue, Santa Monica, CA 90406, Telephone (213) 820-4111

series | base no./vol./reissue
7172/301/00

author Gerald A. Simon
Gerald A. Simon

technical David J. Kaufman
David J. Kaufman

release Carl M. Switzky
Carl M. Switzky

for Charles A. Savant

date 3/29/82

TM a working paper

This document was produced by System Development Corporation in performance of Contract DCA100-82-C-0036

DTIC FILE COPY

DTIC ELECTE APR 7 1983 H

DCEC PROTOCOLS STANDARDIZATION PROGRAM

PROTOCOL SPECIFICATION REPORT

ABSTRACT

This document presents a set of guidelines for precise specification of communication protocol services and mechanisms. First, the requirements for a complete protocol specification are outlined. Based on these requirements a conceptual framework for a protocol specification is discussed. A description of each of the components of a protocol specification is then presented along with a description of the protocol specification format and syntax requirements. The specification approach incorporates an extended finite state machine technique for formal definition of protocol services and mechanisms.

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

83 04 07 037

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER 7172/301/00	2. GOVT ACCESSION NO. AD-A116558	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Protocol Specification Report		5. TYPE OF REPORT & PERIOD COVERED interim technical report
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Gerald A. Simon		8. CONTRACT OR GRANT NUMBER(s) DCA100-82-C-0036
9. PERFORMING ORGANIZATION NAME AND ADDRESS System Development Corporation 2500 Colorado Ave. Santa Monica, CA 90406		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS P.E. 33126K Task 1053.558
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Communications Engineering Center Switched Networks Engineering Directorate 1860 Wiehle Ave., Reston, VA 22090		12. REPORT DATE 29 Mar 82
		13. NUMBER OF PAGES 45
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) N/A		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for Public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

N/A

18. SUPPLEMENTARY NOTES

This document represents results of interim studies which are continuing at the DCEC of DCA.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Protocols, Data Communications, Data Networks, Protocol Standardization, Protocol Specification

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This document presents a set of guidelines for precise specification of communication protocol services and mechanisms. First, the requirements for a complete protocol specification are outlined. Based on these requirements a conceptual framework for a protocol specification is discussed. A description of each of the components of a protocol specification is then presented along with a description of the protocol specification format and syntax requirements. The specification approach incorporates an extended finite state machine technique for formal definition of protocol services and mechanisms.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

FORWARD

This report is a revised version of the July, 1981 Protocol Specification Report[13]. The major updates and changes incorporated into this revision are centered around several extensions to the basic state machine modelling technique. In addition to these extensions, minor corrections and clarifications to the format rules have been included.

The state machine modelling technique has been enhanced in three major areas. First, the basic data types have been expanded to include data types for queues and stacks in addition to the pre-defined Ada types. Secondly, an optional "split state vector" approach to modelling of delay aspects of protocol services has been added. The third extension provides for the separation of implementation dependent aspects of the protocol into separate action procedures.

The protocol specification techniques and formats presented in this report have been used as the basis for the production of the proposed DoD standard specifications for the Transmission Control Protocol (TCP) [2] and for the Internet Protocol (IP) [1]. These specifications serve as examples of the application of these guidelines to the specification of full scale network protocols. The TCP and IP specifications are therefore considered to be companion documents to this report.

Accession For
NTIS GRA&I
DTIC TAB
Unannounced
Justification
By
Distribution/
Availability Codes
Avail and/or
Dist Special
A

CONTENTS

1. INTRODUCTION.. 2

2. PROTOCOL SPECIFICATION FRAMEWORK.................................. 3

3. SPECIFICATION GUIDELINES.. 8
   3.1 OVERVIEW.. 8
   3.2 SERVICES PROVIDED TO THE UPPER LAYER........................ 9
   3.3 UPPER LAYER SERVICE/INTERFACE SPECIFICATIONS................ 10
   3.4 SERVICES REQUIRED FROM THE LOWER LAYER...................... 11
   3.5 LOWER LAYER SERVICE/INTERFACE SPECIFICATION................. 11
   3.6 PROTOCOL ENTITY SPECIFICATION............................... 12
   3.7 EXECUTION ENVIRONMENT REQUIREMENTS.......................... 13

4. EXTENDED STATE MACHINE MODELLING TECHNIQUE...................... 14
   4.1 BASIC TECHNIQUE FOR EXTENDED STATE MACHINE SPECIFICATIONS.... 14
      4.1.1 Machine Instantiation Identifier 15
      4.1.2 State Diagram 15
      4.1.3 State Vector 15
      4.1.4 Data Structures 16
      4.1.5 Event List 16
      4.1.6 Events and Actions 17
   4.2 SUBORDINATE STATE MACHINES................................... 22
      4.2.1 Invocation of Subordinate Machines 23
      4.2.2 Specification of Subordinate Machines 26
   4.3 EXTENSIONS FOR SERVICE/INTERFACE SPECIFICATION............... 26
      4.3.1 Non-Deterministic Actions 26
      4.3.2 Partitioned State Vectors 27

5. PROTOCOL SPECIFICATION FORMAT................................... 29
   5.1 BASIC STRUCTURE... 29
      5.1.1 Overview 29
      5.1.2 Services Provided to Upper Layer 29
      5.1.3 Upper Layer Service/Interface Specification 29
      5.1.4 Services Required From Lower Layer 32
      5.1.5 Lower Layer Service/Interface Specification 32
      5.1.6 Protocol Entity Specification 33
      5.1.7 Execution Environment Requirements 34
      5.1.8 Glossary 34
      5.1.9 Bibliography 34
   5.2 EXTENDED STATE MACHINE FORMATS............................... 35
      5.2.1 Format for Simple (Non-hierarchical) Extended State Machines 37
      5.2.2 Format for Hierarchical Extended State Machines 39
   5.3 CORRESPONDENCE REQUIREMENTS.................................. 41
      5.3.1 Upper Layer Service/Interface Specification 41
      5.3.2 Lower Layer Service/Interface Specification 42
      5.3.3 Protocol Entity Specification 43
      5.3.4 Execution Environment Requirements 44

6. BIBLIOGRAPHY.. 45

1. INTRODUCTION

As the development of distributed systems and computer networks increases in importance, the need for unambiguous and complete specification of communication protocols becomes more apparent. This need is particularly important with respect to protocol standards which must be implemented by wide communities of users with diverse equipment. The purpose of this document is to provide a format, and set of specification techniques, for the production of complete and unambiguous protocol specifications. Unfortunately, use of such a document cannot completely force production of perfect specifications. This document is intended, however, to encourage increased specification completeness and clarity by providing appropriate techniques and requirements for the specification of each portion of the protocol design.

One of the goals of this protocol specification report is to provide for precise specification of both protocol services and mechanisms. A service specification defines the functional requirements which the corresponding protocol design must satisfy, without implying any specific protocol design or set of mechanisms. The service specification provides a focus for the protocol design and thus a standard against which the protocol is measured. In a similar manner, the protocol design specifies requirements for the operation of protocol implementations, not a particular implementation. Naturally, the protocol specification must define enough about the protocol's operation to insure that separate implementations of the protocol will interoperate. Thus while the protocol specification should not needlessly restrict implementation choices, it is better to overly limit the implementation than to omit needed requirements and possibly produce protocol implementations unable to communicate. The techniques presented in these guidelines are intended to allow precise specification of services and mechanisms while minimizing restrictions on the protocol design and implementation respectively.

This report defines the format for a protocol specification document, describes the syntax for specification of the protocol design, and summarizes the key concepts underlying the format and syntax requirements. These items are addressed in the four sections which follow. The first section provides motivation for each of the components of a protocol specification, and defines the conceptual framework for composition of those components to form a complete specification document. The second section further describes each of the major parts of a protocol specification. The third section provides a detailed description of the basic specification vehicle--extended state machines. The fourth section summarizes the format requirements for a protocol specification document.

2. PROTOCOL SPECIFICATION FRAMEWORK

These protocol specification guidelines are based upon the commonly accepted model of a distributed system as a layered hierarchy of protocols[7, 10] (see Figure 1). In this model, each individual protocol is said to reside at a particular layer N. The users of the layer N protocol, the layer N+1 protocols, rely upon services provided by the layer N protocol. In a similar manner, the layer N protocol itself relies upon a composite of the services provided by all of the lower layers. These services are presented to it by the layer N-1 protocol. The module or process which implements an instance of the protocol is called the protocol entity. A peer relationship exists between these entities, but actual communication is accomplished via the layer N-1 protocol entities. Figure 1 illustrates this layered model.

```
        ------                             ------
       | USER |                           | USER |     LAYER
        ------                             ------       N+1
           \                                  /
 - - - - - -\- - - - - - - - - - - - - - - - /- - - - - - - -
        ---------------------------------------
       |   \  COMPOSITE SERVICE OF LAYERS /    |
       |    \     N, N-1, AND BELOW      /     |
       |     \                          /      |
       |   ----------           ----------     |     LAYER
       |  | PROTOCOL |         | PROTOCOL |    |       N
       |  |  ENTITY  |         |  ENTITY  |    |
       |   ----------           ----------     |
       |         \                  /          |
 - - - | - - - - -\- - - - - - - - /- - - - - -|- - - - -
       |           \              /            |
       |        ------------------             |
       |       |   COMPOSITE OF   |            |
       |       | ALL LOWER LAYERS |            |     LAYER
       |       |  (N-1 AND BELOW) |            |      N-1
       |        ------------------             |
       |                                       |
        ---------------------------------------
```

Figure 1. Protocol Layering--Services and Mechanisms

A protocol specification should define the protocol from a set of perspectives derived from this conceptual model[5]. It is necessary, for example, to specify the relationship between the layer N protocol being specified, and the overall system protocol architecture. Within the context of this protocol architecture, the layer N protocol is to provide certain services to the layer N+1 users. In a similar manner, the layer N-1 protocol is expected to provide certain services to the layer N protocol. These services should be explicitly defined. In addition, it is necessary to define the manner in which these services are accessed. The interface specifications define such access interactions. The specification of the mechanisms constituting a layer N entity have typically received nearly exclusive attention in protocol specifications. Naturally, the mechanisms must be specified in sufficient detail to allow implementors to create implementations of the protocol which can interoperate effectively.

Thus the major sections of a protocol specification as defined in this report evolve directly from the model. Furthermore, these individual perspectives are inter-related in a manner which is consistent with the model. As each section of the protocol specification is described below, this relationship is examined further.

One component of the protocol's operation which is frequently ignored in specifying a protocol is the execution environment in which the protocol exists. It is necessary to specify the minimum requirements which the protocol imposes upon the execution environment. This requirement is based upon the fact that the protocol implementation will utilize system resources and that its operation may be limited by the availability of resources such as buffer space and timers.

A protocol specification document produced in accordance with this report should therefore contain the following sections: 1) Overview, 2) Services Provided to Upper Layer, 3) Upper Layer Service/Interface Specifications, 4) Services Required from Lower Layer, 5) Lower Layer Service/Interface Specifications, 6) Protocol Entity Specification, and 7) Execution Environment Requirements. A glossary of terms and a bibliography complete the basic document. It is expected that companion reports to a protocol specification document will be developed over time. These reports may address implementation strategies, experimental results, or suggestions for associating real values with variables parameterized in the specification. Such additional reports are not further discussed in this guideline document.

In general, the sections of a protocol specification consist of a combination of informal, English language descriptions, and formalized descriptions using extended state machine models. This mix is intended to provide an overview and understanding to the reader of the specification, as well as encouraging the specifier to provide sufficient detail to allow independent implementation teams to produce versions of the protocol which will indeed interoperate. The terminology used here is drawn from both the DoD and ISO communities. The paragraphs which follow briefly summarize the requirements for each section of a protocol specification document. Section 3 of this document provides further detail on the requirements for each protocol specification part. The extended state machine modelling technique is described in Section 4.

A. Overview

This section is intended to provide an informal introduction to the protocol and to convey a general understanding of the protocol's operation. Hence, it describes the relationship of the protocol to the protocol system architecture and defines a model of operation for the protocol utilizing English language descriptions. The overview section is, in fact, a summary of the entire specification and is not required to define the protocol. Thus if any inconsistencies between the overview and other sections of the document exist, the formal specification is to take precedence over the overview section.

B. Services Provided to the Upper Layer

This section describes informally the global properties of the system as viewed by the users of the layer N protocol (i.e., the layer N+1 entities). As such, it defines a set of requirements to be fulfilled by the layer N protocol in conjunction with a composite of the layer N-1 and other lower protocol layers. It is important to note that this section defines requirements and not the manner in which those requirements are fulfilled. This is necessary in order to insure that the service specification provides functional requirements for the protocol mechanisms without placing undue restrictions on the design of the mechanisms.

C. Upper Layer Service/Interface Specification

This section defines the services provided to upper layer (N+1) entities in a more formal manner than in Section B, and defines the way in which these services are accessed. The upper layer interface/service is defined as an abstract machine. This machine specifies the actions of the composite of the layer N protocol entities and the lower layer protocols as viewed at the interface between that composite and the users of the layer N protocol. Thus the events of this abstract machine are user (i.e. layer N+1 entity) requests, and the actions are responses to the users. The abstract machine is specified as a specialized form of the extended state machine method described in Section 4 of this document. In part, this section of a protocol specification provides a foundation for demonstration of a correspondence between protocol services required and the mechanisms defined to provide them.

D. Services Required from Lower Layers

This section is analogous to the description of the services provided to the upper layer. The section defines the services required by the layer N protocol of the composite of all lower layer protocols (i.e. layer N-1 and below). The distinction between services required and services provided is significant. The services identified in this section need only be a subset of the services actually provided by the aggregation of the lower layer protocols, since the N layer protocol need not utilize all of the services available from lower layers.

E. Lower Layer Service/Interface Specification

This section is analogous to the upper layer service/interface specification. It defines the interface between entities at layer N and the composite of the lower layer protocols. Again, the global properties of the system, in this case the layer N-1 and lower layer protocols, are defined.

F. Protocol Entity Specification

This section defines the internal operation of a layer N protocol entity. Exact formats of the messages exchanged by peer entities at this protocol layer are given in order to insure that logically separate layer N entities can intercommunicate. The protocol entity is formally defined utilizing the extended state machine method described in Section 4. The events of this state machine are taken from the description of its three interfaces--the requests of the local layer N+1 entity (i.e., a subset of the events from the upper layer service/interface machine), the responses from the lower layer service/interface machine, and signals from the execution environment (see G below). Similarly, the actions of this machine generate the responses, requests, and calls found in the upper layer service/interface machine, lower layer service/interface machine, and execution environment definition respectively. Thus a composite of an appropriate number (typically two) of the layer N protocol entity machines, the lower layer service/interface machine, and the execution environment signals and calls, should be equivalent to the upper layer service/interface specification. This potential composition capability should assist in validating that the protocol specified meets its service requirements. The relationship between the various formal descriptions is explored in further detail in Sections 3, 4, and 5 of this document.

G. Execution Environment Requirements

This section defines the various system resource requirements which the protocol needs for proper operation. This section is not intended to define a generic operating system. Instead, it is an attempt to extract the minimum necessary requirements to support the protocol. Frequently, for example, a protocol mechanism may rely on the execution environment to support timing related functions via system calls and signaled responses such as interrupts. Naturally, the exact formulations defined in the execution environment section may not exist for all systems on which the protocol is implemented. If they do not exist though, the implementor must create an equivalent service within his execution environment since the protocol's operation depends upon that resource.

H. Glossary

A protocol specification document will undoubtedly have its own specialized set of terms, rely upon precise definitions for terms used elsewhere, and utilize acronyms in place of key phrases or identifiers. A glossary of such terms is therefore required as a convenience to readers of the specification.

I. Bibliography

Typically, a protocol design will be based upon the results of previous design and analysis efforts. Furthermore additional background material or analysis of particular mechanisms may be available in other documents. A bibliography is required in order to identify further reference material and to give credit to other related efforts.

3. SPECIFICATION GUIDELINES

This section describes the requirements for a protocol specification document and provides some general guidance on what type of information should be included in each section of a protocol specification document. As indicated above, each document should contain seven major sections: Overview, Services Provided to Upper Layer, Upper Layer Service/Interface Specifications, Services Required from Lower Layer, Lower Layer Service/Interface Specifications, Protocol Entity Specification, and Execution Environment Requirements. In addition, a glossary and list of references should be included. This section focuses on the major portions of a protocol specification; each of the seven subsections which follow defines the purpose and intended contents of a part of a specification document. Section 4 of this document describes the extended state machine technique used for formal specification of the interfaces and the protocol entity. Section 5 summarizes the format requirements for an entire protocol specification document. Section 5 also defines the relationship between each of the major parts of a protocol specification and describes the necessary correspondences between sections of the protocol document resulting from these relationships.

Production of a protocol specification requires not only an underst[illegible]ng of the required specification syntax, but also an understanding of what [illegible]uld be included in each section of the specification. The following des[illegible]tions attempt to provide such guidance by referring to examples from proto[illegible]ith which the reader may be familiar such as the DoD TCP and IP protocols

3.1 OVERVIEW

The Overview section provides an introduction to the specification document. As a prose summary of the protocol specification, the overview is intended to convey a general understanding of the protocol and its operation. The overview is, however, not intended to provide a detailed definition of the protocol and should be kept brief. In order to keep this section short, concepts may be presented in a simplified form, even though further exceptions to these concepts or exceptions to a generalized rule are identified in the detailed protocol specification.

The introduction should include an architectural context for the protocol to be specified and should highlight the important aspects of the protocol services provided. Additionally, key mechanisms employed to provide these services may be introduced. The architectural context portion of the Overview defines the correspondence between the general services provided by the protocol layer to which this protocol belongs (as defined in the protocol system architecture document), and the services for the protocol to be specified. In defining this architectural context, appropriate sections of the system protocol architecture document should be referenced. A general model of operation for the protocol should be provided to present the highlights of the important protocol services and mechanisms. This model should describe the relationship of the protocol to interfacing protocols and include high level scenarios illustrating the operation of the protocol within the overall system framework. The overview should also include references to documents describing previous work utilized in the protocol design and to reports providing additional material on design issues addressed by the protocol.

3.2 SERVICES PROVIDED TO THE UPPER LAYER

The purpose of this section is to provide an informal description of the services which the protocol will provide to upper layer protocols. These services help to provide a set of functional requirements for the protocol mechanisms, define the services which users of the protocol (i.e., N+1 protocol entities) can expect, and describe general performance goals and performance trade off guidance. As shown in Figure 2, the services provided to users

```
 | LAYER N+1 |               | LAYER N+1 |
 |  ENTITY   |               |  ENTITY   |
  -----------                 -----------
   |       ^                   |       ^
   |       |                   |       |
   V       |                   V       |
 ------------------------------------------ <--- SERVICES PROVIDED TO UPPER
|  |       ^                   |       ^  |      LAYER OR LAYER N SERVICES
|  |       |                   |       |  |
|  V       |                   V       |  |
|  -----------                 -----------|
| |  LAYER N  |               |  LAYER N ||
| |  ENTITY   |               |  ENTITY  ||
|  -----------                 -----------|
|    |     ^                     |     ^  |
|    |     |                     |     |  |
|    V     |                     V     |  |
|    ---------------------------------    |
|   |       COMPOSITE SERVICE OF      |   |
|   |         ALL LOWER LAYERS        |   |
|    ---------------------------------    |
|                                         |
 ------------------------------------------
```

Figure 2. Services Provided by Layer N Protocol

represent the aggregate of the layer N and all lower layer protocols, without any explicit reference to the mechanisms used to provide these services. For example, a service provided by the Internet Protocol might be insulation from packet size restrictions of intervening subnetworks. The mechanisms used to provide this service, fragmentation and reassembly, would not appear in the service specification.

In general, this section describes services in terms of the value added by the protocol. That is, the services identified are either not provided by the lower layer protocols, i.e. layer N-1 and below, or are needed to compensate for problems introduced by the lower layers. For example, a reliable transport protocol specification might list reordering of data and duplicate detection as services, thereby emphasizing that the lower levels may introduce duplicates. In the Internet Protocol example above, the insulation from packet size restriction service underscores the fact that lower layer protocols may have packet size limitations.

Identification of performance criteria within the service specification is consistent with delineation of the value added by the protocol. This criterion is based upon information concerning user traffic characteristics or usage requirements. Such information should be identified whenever possible since it may motivate major protocol entity design decisions. In the case of a transport protocol, for example, traffic may be known to be bursty, or connections may be known to have long lifetimes; users may consider reliability paramount or may require minimum delay. If anything is known about the traffic characteristics of the environment in which the protocol will operate, this information should be included in the description of the services. Based upon this information, performance trade-off guidance can be formulated. For a transport protocol, such guidance might state that high reliability or low delay is the overriding consideration in the performance of the protocol. For a file transfer protocol, performance trade-off guidance might specify whether the protocol is to be optimized for large or small files.

3.3 UPPER LAYER SERVICE/INTERFACE SPECIFICATIONS

The purpose of the Upper Layer Service/Interface specification is to formally define the services provided to the users of the protocol (i.e., the layer N+1 entities). These services are realized via interactions between the users of the protocol and the protocol. Hence services are specified in terms of the interface(s) through which the services are accessed. The services are defined as an abstract machine which defines responses by the layer N protocol to service requests by the users of the layer N protocol. These responses and requests are termed "interaction primitives". Interaction primitives are grouped into two classes: service request primitives and service response primitives. Service request primitives refer to interactions initiated by users of the protocol. Service response primitives refer to interactions initiated by the protocol itself and may be the result of events occurring at lower layers.

An interaction primitive defines the content of information units exchanged between protocol layers. Peer exchanges between protocol entities require interoperation of disjoint implementations, but interfaces are generally a localized concern. Hence while the information exchanged between layers must be specified, it is undesirable to specify the exact formats for these exchanges.

The extended state machine model for services provided to upper layer users defines the behavior of the entire "service machine" from the perspective of the upper layer protocols or users of the protocol. Since the services represent the global properties of the composition of the layer N protocol and the lower layer protocols, it is impossible to provide a deterministic specification of the services. The potential for failure of a protocol entity (i.e., the processor(s) on which the entity is implemented) or for a partitioning of the communications medium, for example, must be taken into account. Thus a reliable transport protocol cannot have the property that all data sent was delivered. Instead, the service provided by the transport protocol might be to deliver data to the destination user or to inform the source user that delivery was not confirmed or both. The abstract machine specification of the interface must therefore provide for such nondeterminism.

Interface specifications must also define allowed sequences of interactions. In a connection-oriented transport protocol, for example, the action taken upon commands to open and close connections are probably sequence dependent. Section 4 of this document describes an extended state machine specification technique which provides for both non-deterministic and sequence dependent actions.

The Upper Layer Service/Interface Specification section is divided into two major subsections. The first subsection defines the interaction primitives and the second subsection defines the interface machine. Section 5 of this document describes the format requirements for these subsections.

Sequencing requirements for upper layer protocols or users may be implied by the abstract service machine. Events may be acceptable only within certain states as defined by the state vector. This restriction on acceptability of events forces events to be properly sequenced with respect to the abstract service machine.

3.4 SERVICES REQUIRED FROM THE LOWER LAYER

The purpose of this section is to provide a general description of the services which this protocol requires from lower layer protocols. This section is analogous in form to the Services Provided to the Upper Layer section. As shown in Figure 3, the lower layer service requirements are provided by the composite of all the protocol layers below the protocol to be specified. The lower layer service requirements for a protocol should be equivalent to, or a subset of, the service specification for the layer directly below. Just as the Upper Layer Service section should focus on the value added by the protocol, this section should focus on the characteristics of the lower layers which must either be compensated for or are relied upon. In a reliable transport protocol specification, for example, this section might indicate that messages may be lost, reordered, or duplicated by the lower layers.

3.5 LOWER LAYER SERVICE/INTERFACE SPECIFICATION

This section specifies the services required from lower layer protocols in a more formal manner than the description of lower layer service requirements. In addition, it defines the interface through which these lower layer services are accessed. This section is analogous to the upper layer service/interface

```
  | LAYER N+1 |           | LAYER N+1 |
  |  ENTITY   |           |  ENTITY   |
  -------------           -------------
    |       ^               |       ^
    |       |               |       |
    V       |               V       |
------------------------------------------
|   |       ^               |       ^    |
|   |       |               |       |    |
|   V       |               V       |    |
|  -----------            -----------    |
|  | LAYER N |            | LAYER N |    |
|  | ENTITY  |            | ENTITY  |    |
|  -----------            -----------    |
|     |     ^                |     ^     |
|     |     |                |     |     |
|     V     |                V     |     |
|    ------------------------------      |
|    |    COMPOSITE SERVICE OF    |<------ SERVICES REQUIRED
|    |     ALL LOWER LAYERS       |      | FROM LOWER LAYER
|    ------------------------------      |
|                                        |
------------------------------------------
```

Figure 3. Services Required by Layer N Protocol

specification and thus is divided into two major subsections. The first subsection defines the interaction primitives and interface parameters for the interface between the protocol mechanism and the lower layer protocol; and the second section defines the abstract machine model for the services required from the lower layers. The format for the lower layer service/interface specification is the same as the format used in describing the upper layer service/interface specification and is described in Section 5 of this document.

3.6 PROTOCOL ENTITY SPECIFICATION

This section defines the internal operation of an instance of the protocol. It is divided into three major subsections. The first subsection is the protocol entity overview. The purpose of this section is to provide a prose overview of the mechanisms used in the protocol, provide an informal mapping into the services as defined in the service specification, and provide a place for documentation of the intent of the protocol designers. The second major subsection defines message formats. These are the formats for peer protocol entity exchanges. The messages exchanged between peer entities must be defined precisely in order to allow independent entities built by different implementors to interoperate. The third subsection defines the extended state machine model for the protocol entity and hence its precise operation.

The Protocol Entity Specification defines the requirements for implementation of the protocol. The entity is defined utilizing the extended state machine specification technique described in Section 4 of this document. The events for the protocol entity may be interaction primitives from the layer above or below, or signals from the execution environment. Actions in the entity state machine specification may change internal protocol mechanism data structures, generate interactions to adjacent protocol layers via execution environment requests, or request a service of the execution environment (e.g. setting a timer). Implementation dependent actions are difined in separate "action procedures" which may present various implementation approaches and policies. Any services which must be provided by the execution environment must appear in the execution environment requirements section.

3.7 EXECUTION ENVIRONMENT REQUIREMENTS

This part of a specification document defines the minimum set of execution environment services required by the protocol for proper operation. These requirements are not intended to impose a particular operating system on protocol implementations, but are instead intended to indicate the basic services that the operating system must provide in order to support the protocol. The execution environment requirements are derived from the protocol mechanism specification where they appear as generalized "system calls". The execution environment requirements section provides a complete list of these "system calls" as well as a brief description of the services that they are to provide. Examples of services which might be required are timers and interprocess communication mechanisms.

4. EXTENDED STATE MACHINE MODELLING TECHNIQUE

Finite state machines have been used quite successfully in specifying and modelling of simple protocols[4, 6]. When applied to more sophisticated and complex protocols, however, the finite state machine approach has been plagued with the "state explosion" problem[3, 14]. This refers to the phenomenon in which the state and event spaces become too large to be easily managed. For example, the introduction of sequence numbers as a protocol mechanism for marking data results in a different state for each possible sequence number value expected. In addition the sequence numbering of incoming data results in an event space large enough to accommodate all possible sequence numbers on incoming data.

In order to mitigate this state explosion problem, various techniques have been used. These techniques generally involve a multidimensional state space, commonly referred to as the introduction of state variables[3, 14, 15]. In order to handle the large number of states and events a method must be used to group classes of states and events together rather than listing, for every state in a large state space, every possible event which can occur. Such a method allows the state machine to be organized in a more understandable way and hence improves analyzability.

The extended state machine modelling technique defined in these guidelines introduces several extensions to a pure finite state machine model. These extensions include the specification of a state vector rather than an enumeration of all possible states and the categorization of events into event classes (this is analogous to the state vector and is referred to as the event vector). An additional method intended to help in the analysis of large state machines is the introduction of a hierarchy of state machines[12]. This approach allows the encapsulation of separable portions of the original state machine into modularized subordinate state machines which are invoked from the top level state machine. The technique is particularly useful for partitioning protocol functionality into phases of operation, where each phase of the operation may be represented by a completely separate, subordinate state machine. A transport protocol, for example, may be broken into a connection establishment phase, a connection maintenance phase, and a connection termination phase with separate subordinate state machines for each phase. This technique can greatly simplify the analysis of a protocol by organizing the state machine into more manageable pieces.

4.1 BASIC TECHNIQUE FOR EXTENDED STATE MACHINE SPECIFICATIONS

The representation of extended state machines is similar for both service/interface and mechanism specifications. As a result of their non-deterministic nature, service/interface specifications require some extensions and changes to the basic format. These differences are defined in Section 4.3. Other than those differences, the basic technique defined in this section applies to both types of specifications. The format requirements for the state machine specifications are described in Section 5 of this document.

Extended state machine specifications consist of six components: 1) Machine Instantiation Identifier, 2) State Diagram, 3) State Vector, 4) Data Structures, 5) Event List, and 6) Events and Actions. Each of these items is briefly discussed in the subparagraphs which follow.

4.1.1 Machine Instantiation Identifier

The machine instantiation identifier defines the information necessary to bind a particular event to a particular state machine instance. This information is included in the interface parameters which accompany all incoming events. For example, in the case of the Internet Protocol one state machine instance exists for each datagram. A datagram is identified by source and destination address, and the protocol and ID fields. These identifiers therefore specify a particular instance of an IP state machine. For a protocol which is connection oriented, like TCP, one machine instance could exist for each connection, and the machine instantiation identifier would be equivalent to the set of information necessary to uniquely name a connection.

For some protocols, the information used to bind incoming events to a particular state machine instance may be different for certain states or groups of states. In the case of TCP, for example, a connection (or State Machine Instance) is usually named by a "Socket Pair", but at certain times in a connection lifetime the connection may be named by only one socket ("Unspecified Passive Open"), or by a shorthand name (the "Local Connection Name"). In order to allow for these state machine instance "pseudonyms", multiple versions of the state machine instantiation identifier may be specified for an extended state machine. If more than one machine instatiation identifier is specified, each one must be accompanied by the following:

1. A list of all states (or state classes) for which the identifier is valid.

2. A list of all interaction primitives which may use the identifier.

It should be noted that incoming events which name a state machine by pseudonym are considered to be illegal if they cannot be bound to a state machine instance in an appropriate state to be named by that pseudonym.

4.1.2 State Diagram

State diagrams provide a pictorial summary of the protocol state machine. State diagrams are included only as an aid to the reader. The state diagram depicts all of the major states and the possible transitions between them. Although state diagrams should be included wherever appropriate to improve the understandability of the full state machine description, the full description of the protocol state machine takes precedence in defining protocol operation.

4.1.3 State Vector

The state vector defines, as a set of variables, all information which is to be retained across more than one event. In general, a state vector consists of one "main scalar element", which is commonly referred to as the "state", and a set of additional "state variables". For a transport protocol a state vector might consist of a "main scalar variable" like:

    State Name = (OPEN, CLOSED, CLOSE_PENDING . . .)

and "state variables" like:

    Send_sequence_number = INTEGER range 0...256

An initial value for each vector element must be provided in order to define the initial state of the abstract machine.

4.1.4 Data Structures

Naturally, all data structures used within the state machine specification must be defined. These data structures are the state vector information, and information to be retained which has been received across an interface, or is to be sent across an interface. The format for the data structures is Ada syntax[11]. Data items may be untyped if the typing imposes unnecessary implementation restrictions. In general, data types should be limited to those found in other programming languages such as Pascal. All data items should be accompanied by a comment in Ada format. The Ada format for comments is "--" followed by the comment text. Example:

    Sequence_number : INTEGER -- Next Sequence Number Expected

In addition to the predefined Ada types, the data types "queue" and "stack" may also be used for representing more sophisticated data structures, without creating unnecessary implementation restrictions.

4.1.5 Event List

This section contains a list of all of the "events" which may occur relative to the extended state machine. These "events" are actually groups or classes of events which will be refined further in the state machine actions defined in the events and actions section of the specification. While any event from the event list may occur in any state, events which result in no action within a particular state are not listed.

In general an event class will be defined by the primitive name in the interaction primitive, and the further refinement of the event will be based on the values of the additional associated parameters such as address and length. For example a "SEND" service request from a layer N+1 protocol entity could be considered an "event" for the layer N protocol, and be further refined by examination of the parameters associated with the "SEND" interaction primitive.

4.1.6 Events and Actions

This section defines the actions to be taken upon receipt of "events". These actions are dependent upon the event, and the state of the protocol at the time that the event occurs.

4.1.6.1 State/Event Correspondence

For every state, it is necessary to define the actions to be taken in response to any acceptable event. For a very simple state machine this could be done by simply listing every state, and for each state listing all possible events and associated actions which could occur while in that state, as illustrated in Figure 4. For more complicated state machines, however, this simple listing would become extremely large. In order to help keep the events and actions somewhat more manageable the correspondence between states and events is refined gradually by grouping classes of states and events together. State classes are groups of states defined by a value for one element of the state vector, usually the "main scalar element". Event classes are groups of events usually defined by a primitive name. The event class is made up of a set of events which have the specified primitive name and whose associated parameters fall within the defined range. These state and event classes are further refined by the use of enabling predicates, or logical statements. The technique for handling the events and actions for extended state machines is shown in Figure 5. A specific example of this "grouping" technique is illustrated in Figures 6 and 7. Figure 6 shows an ungrouped event list for the CONNECTION_OPEN state. In Figure 7 the events have been grouped into event classes. Further refinement of the event classes into atomic events is done in the "actions" section for the event class.

4.1.6.2 Actions

The "Actions" for these state and event classes may further refine the state/event correspondence, change internal data structures (e.g. the state vector), or generate interactions to interfacing entities. The syntax for specifying actions is drawn from several Ada constructs, and an optional table representation.

Briefly, state/event correspondence is accomplished through the use of Ada "IF" and "CASE" statements. Changes to internal data structures are represented by Ada assignment statements. Generation of interactions to interfacing entities is accomplished by the invocation of an execution environment service request. All such request requirements must be defined in the execution environment section of the protocol specification.

```
State 1:

     Event 1:  Actions for Event 1
     Event 2:  Actions for Event 2
     Event 3:  Actions for Event 3
        .
        .
        .
     Event N:  Actions for Event N

State 2:

     Event 1:  Actions for Event 1
     Event 2:  Actions for Event 2
     Event 3:  Actions for Event 3
        .
        .
        .
     Event N:  Actions for Event N

  .
  .
  .
State K:

     Event 1:  Actions for Event 1
     Event 2:  Actions for Event 2
     Event 3:  Actions for Event 3
        .
        .
        .
     Event N:  Actions for Event N
```

Note: States and Events are individually enumerated with no grouping.

Figure 4. Events and Actions for a Simple State Machine

```
+--------------------------------------------+
|                                            |
|          State Class 1                     |
|               Event Class 1                |
|                    Actions:                |
|               Event Class 2                |
|                    Actions:                |
|               .                            |
|               .                            |
|               .                            |
|               Event Class N                |
|                    Actions:                |
|                                            |
|          State Class 2                     |
|               Event Class 1                |
|                    Actions:                |
|               Event Class 2                |
|                    Actions:                |
|               Event Class 3                |
|                    Actions:                |
|               .                            |
|               .                            |
|               .                            |
|               Event Class M                |
|                    Actions:                |
|          .                                 |
|          .                                 |
|          .                                 |
|          State Class K                     |
|               Event Class 1                |
|                    Actions:                |
|               .                            |
|               .                            |
|               .                            |
|               Event Class J                |
|                                            |
+--------------------------------------------+
```

Events and Actions are grouped by State Class. Any Event Class not explicitly listed for a particular State or State Class is assumed to result in <u>no</u> action. Actions may further refine the state/event correspondence by the use of enabling predicates.

Figure 5. Events and Actions for an Extended State Machine

```
+---------------------------------------------------+
|                                                   |
|       State = CONNECTION_OPEN                     |
|                                                   |
|                                                   |
|       Events              Actions                 |
|                                                   |
|       open_cmd            discard                 |
|       close_cmd           close connection        |
|       data, seq = 0       deliver                 |
|       data, seq = 1       deliver                 |
|            .                 .                    |
|            .                 .                    |
|            .                 .                    |
|       data, seq = 7       deliver                 |
|       data, seq = 8       discard                 |
|       data, seq = 9       discard                 |
|            .                 .                    |
|            .                 .                    |
|            .                 .                    |
|       data, seq = 127     discard                 |
|                                                   |
+---------------------------------------------------+
```

Figure 6. Events Listed Without Grouping

```
+---------------------------------------------------+
|                                                   |
|       State = CONNECTION_OPEN                     |
|                                                   |
|                                                   |
|       Event Classes     Actions                   |
|                                                   |
|       open_cmd          discard                   |
|       close_cmd         close connection          |
|       data              if seq < 7 deliver        |
|                         else discard              |
|                                                   |
+---------------------------------------------------+
```

Figure 7. Events Listed in Classes

4.1.6.3 Table vs. Pseudo-Code Representation of Actions

The basic state machine approach described above combines two distinct types of "event processing" under the heading of actions. The first type is the decision functions necessary to refine the state/event correspondence based upon the state vector and event parameters. The second type of event processing is the modification of state variables (i.e., the state transition) and the generation of externally visible items (i.e., interactions with interfacing entities). This second type of event processing could be defined using a program-like or pseudo-code format with action procedures defined separately in a manner similar to program subroutines.

Unfortunately, such monolithic specifications of event processing can become quite large and difficult to analyze. As a result, the events and actions section of the extended state machine may be organized into a table format for the refinement of state/event correspondence. In these decision tables, much of the refinement is separated from the actions.

This approach can help organize large numbers of possible combinations of conditions into a more manageable form. Thus it provides a framework for systematically checking such qualities as consistency (whether two or more sets of conditions can be satisfied simultaneously yet imply different actions), completeness (whether every possible legitimate combination of conditions has been covered), and redundancy (whether a set of conditions implying a set of actions is given more than once). This framework not only aids in the development and specification of more correct and complete protocol mechanisms, but also provides a sequence-independent definition of condition checks which may be directly restructured to achieve implementation goals. Decision tables can be kept compact through the use of "don't care" entries to represent decision outcomes that do not affect the action procedure selected. In addition, the sharing of commonly used refinement functions and action procedures among tables reduces the amount of detailed pseudo-code needed. If refinement of the state/event correspondence is not necessary for a particular event class, a null table may be used.

The columns of the table represent decision functions and their outcomes. The rows represent combinations of decision function outcomes. At the right side of the table are names of action procedures to be invoked for the set of conditions represented by the row. The decision functions and procedures follow the tables. Examples of such tables, decision functions, and action procedures appear in Sections 6.3.6.1, 6.3.6.2, and 6.3.6.3 of the IP[1]. Decision functions may contain If and Case statements. Procedures may contain If, Case, and Assignment statements, as well as execution environment service requests (generalized system calls).

4.1.6.4 Implementation Dependent Actions

In some cases the actions to be taken in response to a particular event may vary depending on the specific implementation approach. TCP retransmission policy options are an example of this phenomenon. Naturally, any of the possible actions could be taken without impacting the correct operation of the protocol. Thus from a pure specification perspective, further definition is unnecessary. From an implementation perspective; however, the choice of policy may dramatically impact the protocol's efficiency. Therefore, these implementation dependent actions should be separated into special action procedures which discuss possible implementation strategies and provide trade-off guidance including discussion of the performance impact of various implementation approaches.

4.2 SUBORDINATE STATE MACHINES

Despite the use of state vectors and event vectors, the number of identified states may still become quite large. In order to enhance the analyzability of the state machine, the extended state machine model may be partitioned into a hierarchy of machines. In such a hierarchy, sequences of events and actions may be grouped together and specified in a separate subordinate state machine. This hierarchical representation technique provides for the logical partitioning of a state machine into more manageable pieces. Naturally, the hierarchical representation can be mapped into a non-hierarchical equivalent. An example of hierarchical and non-hierarchical representations for the same state machine is depicted in Figure 8.

The logical grouping or encapsulation of event sequences can be used to simplify the analysis of the state machine. In analyzing the main state machine, the encapsulated set of event sequences may be viewed as a single state transition, thus reducing the complexity of the main machine. Subordinate machines are specified as independent entities and thus may be analyzed separately. In addition, subordinate machines may be invoked from several places within a main machine and can therefore be used to "factor out" common elements of the state machine in order to limit duplication.

A possible application for this hierarchical modularization is separation of protocol "phases of operation" into separate subordinate machines. In the case of a connection-oriented transport protocol, for example, it might be desirable to separate connection establishment and connection termination sequences into separate subordinate state machines. In this way it is easier to separate the various sequences of events and actions which result in connection establishment from both connection maintenance and termination issues. Furthermore, the same sequence (i.e., the same subordinate state machine) may terminate the connection from multiple states.

4.2.1 Invocation of Subordinate Machines

Subordinate state machines may be invoked by the use of the "INVOKE" keyword in the "Actions" section of an extended state machine specification. The syntax is:

```
INVOKE Name (State Vector Elements, Data Structures)
```

Where:

INVOKE - Keyword, in all capitals, signifies invocation of the named subordinate machine.

Name - Name of subordinate machine to be invoked.

State Vector Elements - State vector elements of calling machine which the subordinate machine will have read/write access to. The subordinate machine may NOT access the "main scalar element" of the state vector for the main machine.

Data Structures - Data structures which the subordinate machine will have read/write access to.

NON-HIERARCHICAL STATE MACHINE HIERARCHICAL EQUIVALENT

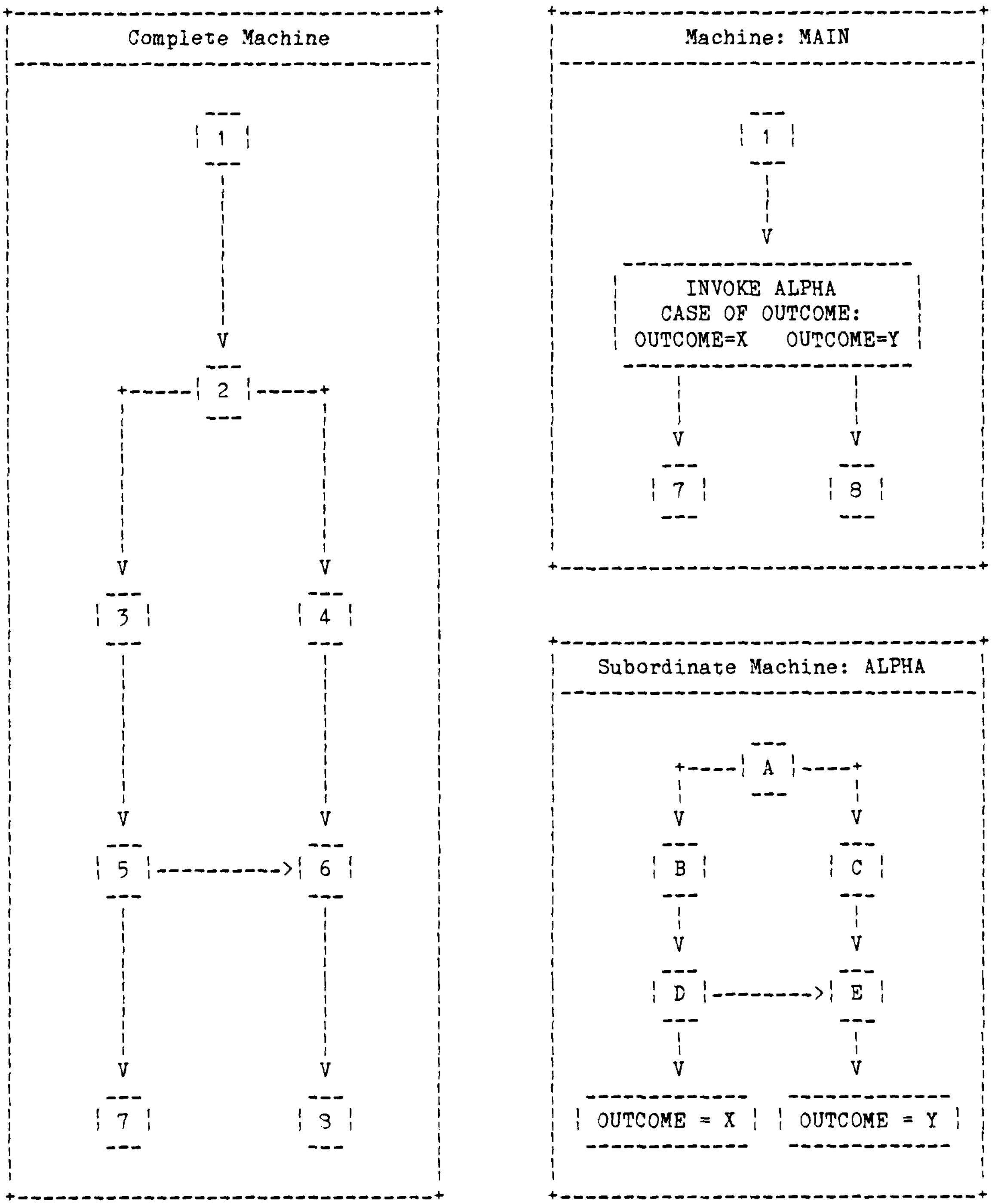


Figure 8. Hierarchical vs. Non-Hierarchical Representation

Immediately following the "INVOKE" statement is a "CASE OF OUTCOME" statement followed by a set of possible outcomes which may be returned by the subordinate machine. The actions to be taken by the calling machine for each outcome are specified directly following each possible outcome listed. The syntax for invocation of a subordinate machine is shown in Figure 9. Since the subordinate machine is actually part of the state machine, the calling machine can be thought of as disabled until an outcome is returned by the subordinate machine. Furthermore, the event list for the main machine serves as the event list for all subordinate machines invoked by that "main machine".

```
+------------------------------------------------------+
|                                                      |
| Invoke Name (State Vector Elements, Data Structures) |
|                                                      |
| Case of Outcome:                                     |
|                                                      |
|      Outcome 1 : Actions for Outcome 1               |
|      Outcome 2 : Actions for Outcome 2               |
|          .                                           |
|          .                                           |
|          .                                           |
|      Outcome N : Actions for Outcome N               |
|                                                      |
+------------------------------------------------------+
```

Note: The Invoke statement is analogous to a subroutine call. The Case of Outcome statement processes the returned value (i.e., the outcome).

Figure 9. Reference to Subordinate State Machine in Main State Machine

The same subordinate machine may be invoked in more than one place within a "main machine". Invocation of a subordinate machine should directly precede the setting of the "main scalar element" of the state vector. This main scalar element may be changed only once in processing a particular event, and setting it must be the last action listed.

The use of hierarchical machines is not limited to two levels. Invocation and specification of further levels of subordinate machines may be done using the same syntax and semantics as are used for the two level hierarchy.

4.2.2 Specification of Subordinate Machines

The specifications for subordinate machines are basically the same as for "main" machines. The following additional information is to be included:

1. State Vector
   The relationship between this subordinate machine's state vector and the "main" state vector must be specified including elements which are accessed from the "main" state vector. Initial values must be specified for all internal state vector elements thus defining the "initial state" for the subordinate machine.

2. Data Structures
   External data structures accessed must be referenced in the "data structures" section.

3. Additional Actions
   The additional action "FINAL OUTCOME = outcome" is used to return a final outcome to the calling machine.

4.3 EXTENSIONS FOR SERVICE/INTERFACE SPECIFICATION

In order to apply the extended state machine technique to the specification of protocol services, the technique must be enhanced with two minor extensions. The technique must first be extended to provide for the specification of non-deterministic actions inherent in protocol services. In addition, the ability to partition state vectors into logically separate parts is useful for modelling of the "delay" aspects of the protocol service. These extensions are further discussed in the sub-sections which follow.

It should also be noted that service/interface specifications do not have "event processing" in the same sense as protocol entity specifications. The actions are creation of response primitives with particular values for each of the identified parameters (see Section 3 or 5 of the IP[1]). Both the table format for decision functions and subordinate state machines may still prove useful, however, in simplifying service/interface specifications, since more complex, higher level protocols may require delineation of numerous command (i.e. service primitive) sequences.

4.3.1 Non-Deterministic Actions

Section 3.3 of this document identified the need for non-determinism in the specification of protocol services. The state machine which defines the actions of the layer N protocol entity can be deterministic since it is not directly affected by the actions of entities at other protocol layers, but instead deals with the local interaction involving the entities at the layers above and below. In the case of the layer N upper layer service machine, however, the composite actions of all lower layer entities affect the behavior of the machine. Since the lower layer behavior characteristics may be non-deterministic (e.g. data may be lost or corrupted), the state machine for the services may be non-deterministic. This distinction between the state machine for the protocol entity and the state machine for protocol services with respect to non-determinism is depicted in Figure 10.

As a result of the non-deterministic nature of protocol services, an additional construct for the specification of non-deterministic actions is provided in the extended state machine model. Instead of a single set of actions being shown in response to a single event, several such action sets may be specified separated by the keyword OR.

4.3.2 Partitioned State Vectors

In order to specify services which involve delay, it may be advantageous to partition the state vector into several logically separate parts. An example of a protocol service which lends itself to this approach is the data transfer phase of a transport protocol connection. The transfer of data between sender and receiver is non-instantaneous, and subject to non-uniform delay. By partitioning the state vector into a sender portion and a receiver portion and defining the delay characteristics of the communication between the two state [illegible], the communication delay may be easily included in the service [illegible]. This "split state vector" specification allows delay to be [illegible] ensuring the basic data transfer mechanism since the transfer [illegible] independently of delay.

[illegible] vectors may be specified by defining each state [illegible] segment, defining the access rules for each part, and defining the communication between the parts. The state vector segments are defined in the same way as a complete state vector. Communication may be defined to occur at [illegible] intervals, at intervals defined by a probability density, or at random time intervals. An example of the use of partitioned state vectors may be found in the specification of the data transfer service for TCP[2].

```
  PROTOCOL ENTITY MACHINE                  SERVICE MACHINE
      (DETERMINISTIC)                    (NON-DETERMINISTIC)
                                                  .
                                      SITE A      .       SITE B
      ---------------                             .
      |  LAYER N+1  |                             .
      |   ENTITY    |                ---------    .    ---------
      ---------------               |LAYER N+1|   .   |LAYER N+1|
          |      ^                  | ENTITY  |   .   | ENTITY  |
          |      |                   ---------    .    ---------
          |      |                    |    ^      .     |    ^
          V      |                    |    |      .     |    |
   ---------------------           ---|----|------------|----|---
   |                   |           |  |    | LAYER N SERVICE |   |
   |     LAYER N       |           |--|----|------------|----|---|
   |      ENTITY       |           |  |    |      .     |    |   |
   |                   |           |  V    |      .     V    |   |
   |                   |           |   -------    .   -------    |
   ---------------------           |  |LAYER N|   .  |LAYER N|   |
          |      ^                 |  | ENTITY|   .  | ENTITY|   |
          |      |                 |   -------    .   -------    |
          |      |                 |  |    ^      .     |    ^   |
          V      |                 |  V    |            V    |   |
      ---------------              |  ------------------------   |
      |  LAYER N-1  |              |  | *  COMPOSITE SERVICE |   |
      |   ENTITY    |              |  |  OF ALL LOWER LAYERS |   |
      ---------------              |  ------------------------   |
                                   |                             |
                                   -------------------------------
```

* NOTE: Reliance on the composite service provided by all lower protocol layers results in possible non-determinism in the layer N service state machine.

Figure 10. Motivation for Non-Deterministic Extension for Specification of Protocol Service State Machines

5. PROTOCOL SPECIFICATION FORMAT

The paragraphs which follow describe the format to be used for each section of a protocol specification. This description is divided into three major portions. First, the format for a protocol specification is presented, omitting the details of the extended state machine portions. The format for extended state machines is presented next, including descriptions of the formats for both simple (non-hierarchical) and hierarchical state machines. The third section defines a set of required correspondences between the various sections of a protocol specification. These correspondence requirements are provided to help insure that protocol specifications are internally consistent.

5.1 BASIC STRUCTURE

The overall structure for a protocol specification is shown in Figure 11: "Generic Table of Contents for Protocol Specifications". The subsections which follow parallel the table of contents and describe the format for each section. The format guidelines presented below recommend additional structure, beyond what is shown in Figure 11. This additional structure is not required for conformance with these guidelines, but should be followed if possible.

5.1.1 Overview

The format for the "Overview" is English prose. The "Overview" may optionally be broken into three subsections: "Architectural Context", "Highlights of Services and Mechanisms" and "Scenarios".

5.1.2 Services Provided to Upper Layer

The format for this informal description of the services provided to the upper layer is English prose. In general, it is recommended that each service be presented in a separate subsection, which should describe the service to be provided and discuss any performance criteria associated with that service.

5.1.3 Upper Layer Service/Interface Specification

As shown in Figure 11, the "Upper Layer Service/Interface" section is divided into two major subsections: "Interaction Primitives" and "Extended State Machine". The formats for these subsections are presented in the subparagraphs which follow.

1. OVERVIEW

2. SERVICES PROVIDED TO UPPER LAYER

3. UPPER LAYER SERVICE/INTERFACE SPECIFICATIONS

    3.1 INTERACTION PRIMITIVES
        3.1.1 Service Request Primitives
        3.1.2 Service Response Primitives

    3.2 EXTENDED STATE MACHINE SPECIFICATION OF SERVICES PROVIDED TO UPPER LAYER

4. SERVICES REQUIRED FROM LOWER LAYER

5. LOWER LAYER SERVICE/INTERFACE SPECIFICATIONS

    5.1 INTERACTION PRIMITIVES
        5.1.1 Service Request Primitives
        5.1.2 Service Response Primitives

    5.2 EXTENDED STATE MACHINE SPECIFICATION OF SERVICES REQUIRED FROM LOWER LAYER

6. PROTOCOL ENTITY SPECIFICATION

    6.1 OVERVIEW OF PROTOCOL MECHANISMS

    6.2 MESSAGE FORMATS FOR PEER EXCHANGES

    6.3 EXTENDED STATE MACHINE REPRESENTATION OF PROTOCOL ENTITY

7. EXECUTION ENVIRONMENT REQUIREMENTS

8. GLOSSARY

9. BIBLIOGRAPHY

Figure 11. Generic Table of Contents for Protocol Specifications

5.1.3.1 <u>Interaction Primitives</u>

The interaction primitives define the content of the messages exchanged between the upper layer and the protocol entity. As shown in Figure 11, these primitive name definitions are further divided into "Service Request Primitives" and "Service Response Primitives". The format for both service requests and responses is a list of primitive names and parameters, with one subsection for each primitive name and the parameters associated with it. Comments should be included to explain the purpose of each primitive name and each associated parameter.

The following outline depicts the basic format for the specification of the interaction primitives:

3.1 INTERACTION PRIMITIVES

3.1.1 Service Request Primitives

3.1.1.1 Primitive Name 1
Parameters Associated With Primitive Name 1
3.1.1.2 Primitive Name 2
Parameters Associated With Primitive Name 2

.
.
.

3.1.1.N Primitive Name N
Parameters Associated With Primitive Name N

3.1.2 Service Response Primitives

3.1.2.1 Primitive Name 1
Parameters Associated With Primitive Name 1
3.1.2.2 Primitive Name 2
Parameters Associated With Primitive Name 2

.
.
.

3.1.2.N Primitive Name N
Parameters Associated With Primitive Name N

Section 3.1 of the IP specification [1] contains an example "Interaction Primitives" section.

5.1.3.2 <u>Extended State Machine Specification of Services Provided to Upper Layer</u>

The extended state machine for the services provided to the upper layer may be specified either as a simple (non-hierarchical) or hierarchical extended state machine. The formats for both types of extended state machine specification are defined in Section 5.2 of this document.

5.1.4 Services Required From Lower Layer

The format for this informal description of the services required from the lower layer is English prose. In general, it is recommended that each service be presented in a separate subsection, which should describe a service requirement and discuss any performance criteria associated with that service. In describing these service requirements emphasis should be placed on discussing the limitations of these service requirements with respect to the value added by the layer N protocol.

5.1.5 Lower Layer Service/Interface Specification

As shown in Figure 11, the "Lower Layer Service/Interface" section is divided into two major subsections: "Interaction Primitives" and "Extended State Machine". The formats for these subsections are presented in the subparagraphs which follow.

5.1.5.1 Interaction Primitives

The interaction primitives define the content of the messages exchanged between the lower layer and the protocol entity. As shown in Figure 11, these primitive name definitions are further divided into "Service Request Primitives" and "Service Response Primitives". The format for both service requests and responses is a list of primitive names and parameters, with one subsection for each primitive name and the parameters associated with it. Comments should be included to explain the purpose of each primitive name and each associated parameter.

The following outline depicts the basic format for the specification of the interaction primitives:

5.1 INTERACTION PRIMITIVES

5.1.1 Service Request Primitives

5.1.1.1 Primitive Name 1
Parameters Associated With Primitive Type 1
5.1.1.2 Primitive Name 2
Parameters Associated With Primitive Name 2

.
.
.

5.1.1.N Primitive Name N
Parameters Associated With Primitive Name N

5.1.2 Service Response Primitives

5.1.2.1 Primitive Name 1
Parameters Associated With Primitive Name 1
5.1.2.2 Primitive Name 2
Parameters Associated With Primitive Name 2

.
.
.

5.1.2.N Primitive Name N
Parameters Associated With Primitive Name N

Section 5.1 of the IP specification [1] contains an example "Interaction Primitives" section for a "Lower Layer Service/Interface Specification".

5.1.5.2 Extended State Machine Specification of Services Required from Lower Layer

The extended state machine for the services required from the lower layer may be specified as either a simple (non-hierarchical) or hierarchical extended state machine. The formats for both types of extended state machine specification are defined in Section 5.2 of this document.

5.1.6 Protocol Entity Specification

As shown in Figure 11, the "Protocol Entity Specification" is divided into three subsections: "Overview of Protocol Mechanisms", "Message Formats for Peer Exchanges", and "Extended State Machine Representation of Protocol Entity". The formats for each of these subsections are defined in the subparagraphs which follow.

5.1.6.1 Overview of Protocol Mechanisms

The format for this "Overview" is English prose. Each mechanism should be presented informally in a separate subsection, with emphasis on the correspondence between the mechanism and the service requirements which motivate it.

5.1.6.2 Message Formats for Peer Exchanges

The specification of the message formats for exchanges with peer protocol entities consists of:

1. A diagram of the format or formats for peer exchanges.

2. Descriptions of each header field.

Each field should be described in a separate subsection. Field descriptions should include the following items:

a. Field Name
b. Abbreviation (if any)
c. Field Length
d. Units (if appropriate)
e. Default Value (if any)
f. A comment explaining the purpose of the field.

5.1.6.3 Extended State Machine Representation of the Protocol Entity

The extended state machine for the protocol entity may be specified as either a simple (non-hierarchical) or hierarchical extended state machine. The formats for both types of extended state machine specification are defined in Section 5.2 of this document.

5.1.7 Execution Environment Requirements

The execution environment requirements are specified as a set of system calls with descriptions of the service requirements for each call. Each call should be described in a separate subsection. These system call names are derived from the calls used for requesting these services in the protocol entity specification (Section 6 of the protocol specification).

5.1.8 Glossary

The glossary consists of a set of terms in alphabetical order with their definitions.

5.1.9 Bibliography

The bibliography should be produced in conformance with some generally accepted bibliographic format.

5.2 EXTENDED STATE MACHINE FORMATS

Extended state machines may be either simple (non-hierarchical) or hierarchical. The format for simple machines is depicted in Figure 12, and the format for hierarchical machines appears in Figure 13. In Figures 12 and 13, the "X.Y" in the section numbers represents the section number for the entire extended state machine specification. These section numbers are as follows:

Section 3.2 -- Upper Layer Service State Machine.

Section 5.2 -- Lower Layer Service State Machine.

Section 6.3 -- Protocol Entity State Machine.

```
+------------------------------------------------------------+
|                                                            |
|        *X.Y.1.  Machine Instantiation Identifier           |
|                                                            |
|         X.Y.2.  State Diagram(S)                           |
|                                                            |
|         X.Y.3.  State Vector                               |
|                                                            |
|         X.Y.4.  Data Structures                            |
|                                                            |
|         X.Y.5.  Event List                                 |
|                                                            |
|         X.Y.6.  Events and Actions                         |
|                                                            |
+------------------------------------------------------------+
```

*Where X.Y. is the section number for the extended state machine

Figure 12. Generic Table of Contents for Non-Hierarchical Extended State Machine Specifications

*X.Y.1. TOP LEVEL MACHINE SPECIFICATION

X.Y.1.1. Machine Instantiation Identifier
X.Y.1.2. State Diagram(s)
X.Y.1.3. State Vector
X.Y.1.4. Data Structures
X.Y.1.5. Event List
X.Y.1.6. Events and Actions

X.Y.2. SUBORDINATE MACHINE 1

X.Y.2.1. Machine Instantiation Identifier
X.Y.2.2. State Diagram(s)
X.Y.2.3. State Vector
X.Y.2.4. Data Structures
X.Y.2.5. Event List
X.Y.2.6. Events and Actions

X.Y.3. SUBORDINATE MACHINE 2

X.Y.3.1. Machine Instantiation Identifier
X.Y.3.2. State Diagram(s)
X.Y.3.3. State Vector
X.Y.3.4. Data Structures
X.Y.3.5. Event List
X.Y.3.6. Events and Actions

.
.
.

X.Y.N. SUBORDINATE MACHINE N-1

X.Y.N.1. Machine Instantiation Identifier
X.Y.N.2. State Diagram(s)
X.Y.N.3. State Vector
X.Y.N.4. Data Structures
X.Y.N.5. Event List
X.Y.N.6. Events and Actions

*Where X.Y. is the section number for the extended state machine

Figure 13. Generic Table of Contents for Hierarchical Extended State Machine Specifications

5.2.1 Format for Simple (Non-hierarchical) Extended State Machines

The formats for each of the sections shown in Figure 12 are described in the subparagraphs which follow.

5.2.1.1 Machine Instantiation Identifier

The machine instantiation identifier is specified as a set of item names which are to be used to bind an incoming event to the correct state machine. If multiple machine instantiation identifiers are used, each identifier must be accompanied by a list of all states (or state classes) for which the identifier is valid, and a list of all interaction primitives which may use the identifier.

5.2.1.2 State Diagrams

State diagrams may be included as a pictorial aid and should be in the form of a set of nodes and a set of directed arcs between the nodes. Nodes represent state classes, and arcs represent event classes and transitions. An example state diagram appears in Section 6.3.2 of the IP specification [1].

5.2.1.3 State Vector

The state vector is specified as a set of item names with comments. Each item name represents an element of the state vector. The comments explain the purpose of each item or state vector element. Possible values or value ranges should be included in the definitions of state vector elements if possible.

If a partitioned state vector approach is used, each portion of the state vector must be defined separately. If two or more parts of a partitioned state vector are identical, they may be defined only once. See TCP[2] page 21 for an example of a partitioned state vector definition.

5.2.1.4 Data Structures

The data structures for the state machine are specified in Ada. Data structures may be partially or completely untyped where specific formats or data types are implementation-dependent. It should be noted that Ada is used as a basis for the data structure syntax, but it is strongly recommended that only the subset of Ada data types available in other structured languages such as Pascal be used. In addition to the pre-defined Ada types, the data types "queue" and "stack" may also be used. Data structures for the state vector and interaction primitives should include references to the subsections in which they are initially defined.

5.2.1.5 Event List

The event list consists of a list of all of the interaction primitives which may be received as input to the state machine. The list contains the names of the interaction primitives as well as brief comments describing the purpose of each one. These names must correspond to the primitive names defined in the "Interaction Primitives" and "Data Structures" sections of the protocol specification.

5.2.1.6 Events and Actions

The events and actions section of the extended state machine specification may be organized in either a decision table or pseudo-code format. Whichever approach is chosen should be used throughout the events and actions section of the state machine. The subparagraphs which follow define the formats to be used for both the table and pseudo-code approaches, and provide some general guidance for choosing the approach which is best suited to the state machine to be specified.

5.2.1.6.1 Decision Tables vs. Pseudo-Code

Usage of decision tables rather than pseudo-code representation is largely a matter of design style. Certain general guidelines can be stated, however. In state machines where every event requires little or no refinement, pseudo-code is adequate. However, in more complex machines where most events require non-trivial refinements and can result in a number of different actions, decision tables are recommended.

5.2.1.6.2 Format for Decision Table Specification of Events and Actions

The decision table events and actions specification is divided into the following three sections:

1. Decision Tables

2. Decision Functions

3. Decision Table Action Procedures

These three sections, presented in the above order, constitute a decision table type specification of state machine events and actions. The format for each of these sections is described in the subparagraphs which follow. An example of an "Events and Actions" section which is organized in the decision table format appears in Section 6.3.6 of the IP specification [1].

5.2.1.6.2.1 Decision Tables

Decision tables are organized by State. One table exists for each event within a state.

The columns of the decision table correspond to "Decision Function" outcomes, and the rows specify "Action Procedures" which are to be invoked based on the combination of outcomes which appear in the row. The "Decision Function" names appear at the tops of the columns, and the "Action Procedure" names appear at the right sides of the appropriate rows.

If no decisions are required for the processing of an event within a particular state, a null table should be used which specifies the "Action Procedure" to be invoked.

5.2.1.6.2.2 Decision Functions

Each decision function is specified in a separate subsection of the "Decision Functions" section. Decision functions specify the following items:

1. Data Structure Elements Examined

2. Return Values (a complete list)

3. The Algorithm

5.2.1.6.2.3 Decision Table Action Procedures

Decision table actions are specified in the form of action procedures. Each action procedure is specified in a separate subsection of the "Decision Table Action Procedures" section. Action procedures specify the following items:

1. Data Structure Elements Examined

2. Data Structure Elements Modified

3. The Procedure

Action procedures which contain implementation dependent actions should be described in english, with appropriate discussion of various implementation choices and trade-off guidence.

5.2.1.6.3 Format for Pseudo-Code Specification of Events and Actions

The pseudo-code actions are organized by state and event. Each State should begin on a new page, with a separate pseudo-code procedure provided for each Event within the State.

5.2.2 Format for Hierarchical Extended State Machines

The format for hierarchical extended state machines is depicted in Figure 13: "Generic Table of Contents for Hierarchical Extended State Machine Specifications". The format requirements for the "Top Level Machine Specification" are the same as those defined for non-hierarchical extended state machines in Section 5.2.1 of this document.

The following sections of the top level machine are used by all of its subordinate machines and need be referenced only in specifying subordinate machines:

1. Machine Instantiation Identifier

2. Event List

The formats for the subsections of a subordinate machine specification are described in the subparagraphs which follow.

5.2.2.1 Machine Instantiation Identifier

The machine instantiation identifier for the top level machine should be referenced.

5.2.2.2 State Diagrams

The format for state diagrams in subordinate machines is identical to the format for state diagrams in non-hierarchical machines described in Section 5.2.1.2 of this document.

5.2.2.3 State Vector

The state vector for subordinate machines is specified in two parts. First, the state vector elements of the top level machine which are to be used by the subordinate machine are listed, and then the state vector elements which are internal to the subordinate machine are specified.

The internal state vector is specified as a set of item names with comments. Each item name represents an element of the state vector. The comments explain the purpose of each item or state vector element. Possible values or value ranges should be included in the definitions of state vector elements if possible.

5.2.2.4 Data Structures

The only data structure which must be specified for subordinate state machines is the state vector. All other data structures are identical to those defined in the top level machine and should be referenced. The state vector data structure is specified in Ada with optional typing as defined in Section 5.2.1.4 of this document.

5.2.2.5 Event List

The Event List for the top level machine should be referenced.

5.2.2.6 Events and Actions

The events and actions specification for subordinate state machines may be in either a decision table or pseudo-code format. Guidance for selection of one of these approaches, and the format to be used for each approach, are provided in Section 5.2.1.6 of this document.

5.3 CORRESPONDENCE REQUIREMENTS

This report has presented a structure for protocol specifications in which the protocol is described from several perspectives. This section describes a set of correspondence requirements, or consistency checks, designed to help insure that the specifications from these various perspectives are consistent. Correspondence requirements are provided for the following sections:

1. Upper Layer Service/Interface Specification

2. Lower Layer Service/Interface Specification

3. Protocol Entity Specification

4. Execution Environment Requirements

The other sections of the protocol specification are less formal in nature, and therefore need no formal correspondence requirements. The correspondence requirements for the more formal sections listed above are described in the subparagraphs which follow.

5.3.1 Upper Layer Service/Interface Specification

The requirements for correspondence between information in the Upper Layer Service/Interface and information contained in other sections of the protocol specification are described in the subparagraphs which follow.

5.3.1.1 Interaction Primitives

1. The Primitive Names must correspond to the names used in specifying the Interaction Primitive Data Structures for the Upper Layer Service/Interface and Protocol Entity extended state machine specifications.

2. The Parameter Names which are to be used to bind incoming events to the appropriate state machine must match the names used in the Machine Instantiation Identifiers for the Upper Layer Service/Interface, and for the Protocol Entity extended state machine specifications.

5.3.1.1.1 Service Request Primitives

The Primitive Names must correspond to the names used in specifying the Event Lists for the Upper Layer Service/Interface, and for the Protocol Entity extended state machine specifications.

5.3.1.2 Extended State Machine Specification of Services Provided to Upper Layer

5.3.1.2.1 Machine Instantiation Identifier

The Machine Instantiation Identifier must be included in the parameter list for every event which may be accepted by the Upper Layer Service state machine. The names used for these parameters must correspond to the names used in the specification of these interaction primitives in the Service Request Primitives for the Upper Layer Service/Interface Specification.

5.3.1.2.2 <u>State Vector</u>

The names used in defining the state vector elements must be the same as the names used in the declaration of the data structure for the state vector.

5.3.1.2.3 <u>Data Structures</u>

The Data Structures subsection includes Ada data structures (optional typing) for the state vector, and all of the interaction primitives defined in the Upper Layer Service/Interface Specification. The names used in defining the data structures must match the names used for the Primitive Names, Parameters, and State Vector Elements in the Upper Layer Service/Interface Specification.

5.3.1.2.4 <u>Event List</u>

The Event List for the Upper Layer Service/Interface state machine consists of the aggregate of the Primitive Names defined in the Service Request Primitives for the Upper Layer Service/Interface Specification.

5.3.2 <u>Lower Layer Service/Interface Specification</u>

The requirements for correspondence between information in the Lower Layer Service/Interface and information contained in other sections of the protocol specification are described in the subparagraphs which follow.

5.3.2.1 <u>Interaction Primitives</u>

1. The Primitive Names must correspond to the names used in specifying the Interaction Primitive Data Structures for the Lower Layer Service/Interface and Protocol Entity extended state machine specifications.

2. The Parameter names which are to be used to bind incoming events to the appropriate state machine must match the names used in the Machine Instantiation Identifiers for the Lower Layer Service/Interface, and for the Protocol Entity extended state machine specifications.

5.3.2.1.1 <u>Service Request Primitives</u>

The Primitive Names must correspond to the names used in specifying the Event List for the Lower Layer Service/Interface extended state machine.

5.3.2.1.2 <u>Service Response Primitives</u>

The Primitive Names must correspond to the names used in specifying the Event List for the Protocol Entity extended state machine.

5.3.2.2 Extended State Machine Specification of Services Required from Lower Layer

5.3.2.2.1 Machine Instantiation Identifier

The Machine Instantiation Identifier must be included in the parameter list for every event which may be accepted by the Lower Layer Service state machine. The names used for these parameters must correspond to the names used in the specification of these interaction primitives in the Service Request Primitives of the Lower Layer Service/Interface Specification.

5.3.2.2.2 State Vector

The names used in defining the state vector elements must be the same as the names used in the declaration of the data structure for the state vector.

5.3.2.2.3 Data Structures

The Data Structures subsection includes Ada data structures (optional typing) for the state vector, and all of the interaction primitives defined in the Lower Layer Service/Interface Specification. The names used in defining the data structures must match the names used for the Primitive Names, Parameters, and State Vector Elements in the Lower Layer Service/Interface Specification.

5.3.2.2.4 Event List

The Event List for the Lower Layer Service/Interface state machine consists of the aggregate of the primitive names defined in the Service Request Primitives for the Lower Layer Service/Interface Specification.

5.3.3 Protocol Entity Specification

The requirements for correspondence between information in the extended state machine specification of the protocol entity, and information contained in other sections of the protocol specification are described in the subparagraphs which follow.

5.3.3.1 Machine Instantiation Identifier

The Machine Instantiation Identifier must be included in the parameter list for every event which may be accepted by the protocol entity. The names used for these parameters must correspond to the names used in the specification of these interaction primitives in the Service Request Primitives of the Upper Layer Service/Interface Specification, and the Service Response Primitives of the Lower Layer Service/Interface Specification.

5.3.3.2 State Vector

The names used in defining the state vector elements must be the same as the names used in the declaration of the data structure for the state vector.

5.3.3.3 Data Structures

The Data Structures subsection includes Ada data structures (optional typing) for the state vector for the protocol entity, and all of the interaction primitives defined in the upper and lower layer service/interface machines. The data structures for these interaction primitives should correspond exactly to the interaction primitive data structures defined in the service/interface state machines.

5.3.3.4 Event List

The Event List for the protocol entity consists of the aggregate of the primitive names defined in the Service Request Primitives for the Upper Layer Service/Interface Specification and the Service Response Primitives for the Lower Layer Service/Interface Specification.

5.3.4 Execution Environment Requirements

The names of the system calls described in the Execution Environment Requirements section should match the names used for requesting these services in the Events and Actions section of the Protocol Entity Extended State Machine Specification.

6. BIBLIOGRAPHY

[1] Bernstein, M., "Proposed DoD Internet Protocol Standard," DCEC Protocols Standardization Program, System Development Corporation TM-7038/205/01, December 1981.

[2] Bernstein, M., "Proposed DoD Transmission Control Protocol Standard," DCEC Protocols Standardization Program, System Development Corporation, TM-7038/207/01, December 1981.

[3] Bochmann, G., "A General Transition Model for Protocols and Communication Services," IEEE Transactions on Communications, April 1980.

[4] Bochmann, G., "Finite State Description of Communication Protocols," Computer Networks, October 1978.

[5] Bochmann, G. and C. Sunshine, "Formal Methods in Communication Protocol Design," IEEE Transactions on Communications, April 1980.

[6] Danthine, A., "Protocol Representation with Finite-State Models," IEEE Transactions on Communications, April 1980.

[7] "Data Processing Open Systems Interconnection--Basic Reference Model," Draft Version of ISO/TC97/SC 16 N 537 Revised, November 1980.

[8] "DoD Standard Internet Protocol," Defense Advanced Research Projects Agency, January 1980.

[9] "DoD Standard Transmission Control Protocol," Defense Advanced Research Projects Agency, January 1980.

[10] "Preliminary Architecture Report," DCEC Protocols Standardization Program, System Development Corporation TM-7038/200/00, February 1981.

[11] "Reference Manual for the Ada Programming Language," U.S. Government Printing Office, 1981.

[12] Shotting, K., "On the Formal Specification of Computer Communication Protocols," master's thesis.

[13] Simon, G., "Protocol Specification Report," DCEC Protocols Standardization Program, System Development Corporation TM-7038/204/00, July 1981.

[14] Sunshine, C., "Formal Modeling of Communication Protocols," Working Paper, University of Southern California Information Sciences Institute, December 1980.

[15] Tenney, R., "Specification Technique," in Formal Description Techniques for Network Protocols, Report No. ICST/HLNP80-3, National Bureau of Standards, June 1980.

LMEL

8