CHAPTER I

A CONCEPTUAL FRAMEWORK FOR TESTING COMMUNICATIONS PROTOCOLS

1. Introduction

Recent advances in the telecommunications industry provide a wide range of enhanced services by enabling the interconnection of heterogeneous systems. Such services, including electronic mail, file transfer, sharing distributed resources, and network management, increase the complexity of the interfaces that are embodied in the communications protocols. A *protocol* defines a set of rules for message exchange and message syntax to achieve communications among those systems. In order to achieve a reliable environment for communicating systems where various manufacturers' equipment is required to interoperate, the implementations of protocols must be tested to ensure that they conform to their respective standards. This concept is especially emphasized by the international standards bodies, such as the International Organization for Standardization (ISO) and the International Telegraph and Telephone Consultative Committee (CCITT), since the development of the Open Systems Interconnection (OSI) – Basic Reference Model (OSI-RM) [1], [2], and related protocol standards [3].

However, the specifications of communications protocols are written in less than fully precise language (e.g., English). This often leads to different interpretations, and therefore, different implementations that may not interoperate with each other. Furthermore, the diversity of the implementations is increased by a wide range of implementation options defined in protocol specifications to meet user service requirements.

As a consequence of the complexity of communications protocols, ambiguity of the specifications and the variety of options allowed, protocol testing has become a challenging task and an integral part of communications systems design. The importance and complexity of protocol conformance testing have been recognized by researchers in the telecommunications industry and academia, resulting in a significant body of publications. A considerable number of papers in this field present the theoretical aspects of conformance testing, such as the generation of test sequences, coverage provided by the tests, and modeling of protocol specifications. A second body of literature addresses issues concerning conduct of testing, such as test methods, test system architectures, test suite definition and organization, test specification languages, and executable test systems. Although each group of papers represents a considerable amount of research and development, they often fail to complement each other, and fail to give a complete perspective. Due to the rapid growth of this field, and increased inter-dependence among information technology, data communications and distributed systems, both the theoretical and practical aspects of conformance testing need to be considered by those responsible for specification of communications protocols: product developers, test system developers, researchers, graduate and senior-level students.

Among the papers selected for your review in this chapter, Davidson's paper, *International Conformance Testing – Towards the Next Decade*, describes the relevance of the topic, and an evolving international infrastructure to develop conformance tests and test systems. Davidson's paper reflects the experience of over eight years in conformance testing: first, at the National Computing Centre in the U.K., and subsequently at the Corporation for Open Systems in the U.S.A. In Chapter II, the perspective of ISO is presented in *OSI Conformance Testing* by Rayner. The difference in their philosophies regarding development of test suites is noteworthy. Davidson proposes development of a test suite first, and only after gaining experience with it by actual testing, does he suggest standardizing it. Rayner, on the other hand, advocates standardization first and the specification of test suites in ISO's test language, named TTCN (Tree and Tabular Combined Notation).

In the early 1980's, research and development was initiated on methods to test the developing ISO's Open System Interconnection communications protocols within Europe. Initial collaboration was between Agence de l'Informatique (ADI) (in Paris, France), Gesellshaft fur Mathematik und Datenverarbeitung (GMD) (in Darmstadt, F.R.G), and the National Physical Laboratory (NPL) (in Teddington, Middlesex, U.K). Each research laboratory had its own industrial partners and focused on a different aspect of testing. ADI designed and implemented several tools including an X.25 protocol tester; GMD developed a language oriented analytic tool for passive monitoring and error detection for the Session Layer protocol used with CCITT's Teletex; NPL developed a Transport Service test system. Since the National Bureau of Standards (NBS)† in the U.S.A was developing a test system for ISO's Transport Layer Class 4 protocol, NBS was invited to participate. Each of the tools developed was based on different design architectures and philosophies. By 1984, other countries were invited to join the research effort. Much of the early research on OSI protocol testing is reported in the proceedings of the First through Fifth Workshops on Protocol Specification, Testing and Verification [4]-[8]. Additional research and development has been published in subsequent editions [9]-[15] and in a recently established workshop focused strictly on protocol testing [16]-[20].

Early work focused on a variety of test architectures, test languages, and demonstration of the viability of one method of testing or another. Parallel efforts focused on automated generation of test sequences from formal specifications and specification languages for protocols. Many of the various architectures and terminology coming out of the initial research efforts have been subsumed by the work initiated within ISO.

In 1983, ISO initiated work to develop standardized test methods for OSI protocols. CCITT subsequently joined the effort. By mutual agreement, CCITT will adopt the technical work of ISO, but will publish it as CCITT Recommendations. This has resulted in a focus and growth of interest in conformance evaluation methods which ultimately will result in a standardized test language and standardized test methods [21] for use in ISO and CCITT.

In the remainder of this chapter, we will introduce the underlying model of the communication protocols and services provided by them, and the methods to represent the protocol specifications. In Section 2, we illustrate the fundamental concepts in testing by giving an easy-to-follow example: testing a four-function calculator. We try to identify analogies between this easy-looking task and conformance testing of the communication protocols. In Section 3, we discuss the OSI-RM structure and define the difficulties involved in conformance testing. Formal tools to describe the protocol specifications are introduced in Section 4. Section 5 summarizes the chapter. Open research problems and suggestions for further reading are in Section 6.

2. A Challenge

Consider the following problem: you just bought a new four-function (add, subtract, multiply, and divide), 10-digit calculator, and your first task is to test the calculator to see if it works before you can apply it to real tasks! This is not an undertaking most of us would accept – but an instructive example which may be related to protocol testing. How would you proceed? The calculator's elements are already built-in; therefore, its internal components (integrated circuits, encoder/decoders, buffers, etc.) and algorithms that the manufacturer implemented to build the arithmetic-logic unit are not available! Thus, the calculator must be tested as a *black-box* (i.e., you may only stimulate the calculator by entering inputs via the keypad and observe its displayed outputs). This is a fundamental assumption made in conformance testing of communication protocols; the *implementation under test* (IUT) is tested as a black box.

The main advantage of the black-box approach is that the implementation details of a specification is invisible to the external tester allowing the manufacturers the freedom in their internal designs. The

† The National Institute of Standards and Technology (NIST) was formerly the National Bureau of Standards. We use the old name when referencing work predating the name change.

international standards organizations aim to conceal the implementation details by specifying the behavior of a protocol at its interface. Therefore, the black-box approach in conformance testing supports the standardization efforts, and vice versa. On the other hand, adopting the black-box approach also makes the task of conformance even more challenging. Hiding the implementation details from external testers, hence defining the conformance based on only the observable behavior of an implementation, is the primary disadvantage of the black-box approach. In Chapters II and III, we will discuss the black-box approach in testing communication protocol implementations in more detail.

Now, given the calculator and the fact that it must be tested as a black-box, the only knowledge that may be employed is understanding its inherent functions, and other characteristics by reading the calculator's published user's manual. We refer to this published manual as the *specification* of the calculator. The information obtained from the user's manual in our example is that there are four functions, and only 10 digits are available for display and computation.

The framework that we defined for you to test your calculator also defines what is testable by you, as the tester. As mentioned earlier, we defined your testing ability in such a way that you can only test the calculator in terms of externally controllable stimuli (i.e., the buttons on the keypad) and externally observable output (i.e., the display). Any other detail about the calculator is inside the black-box and, therefore, cannot be tested from your point of reference. For example, in this framework, you could not be able to test if pressing a digit button triggers the appropriate flip-flops since the flip-flops, and other circuitry are not within the framework. On the other hand, you would easily test if the digit button that you pressed is correctly displayed.

Having defined the testing approach as a black-box, you will immediately conclude that it is impossible to test all values that the calculator should compute (*exhaustive testing*). Therefore, a test suite must be designed with one or more fault models in mind. For example,

1) during addition, whether a carry properly propagates from one digit to another (fault model: failure to carry at any digit will result in wrong answers);

2) whether the calculator displays a negative sign if the amount accumulated is negative (fault model: the negative sign fails to illuminate when the value accumulated is negative due to a defective display segment or segment driver);

3) for all decimal digits and for all ten display elements, whether the proper display segments light up (fault model: one or more segments fails to illuminate due to either a defective decoder/encoder or defective display segments).

Each of the above aspects may be tested by designing a test, stimulating the calculator by entering the appropriate inputs via the keypad, and observing the outputs. The observed outputs should *conform* to those defined by the specification (i.e., the user's manual for the calculator).

Now, we should define tests for the above fault models. To test carry propagation, enter "999999999" + "1"; if the display contains "1000000000", the result is correct and carry propagated from the first to last digit. Each of the other examples may be tested in a similar manner. Surely, we can agree that the three examples are inadequate as a test suite. It is more difficult to agree on the answer for the following question: "What constitutes enough testing?" We will return to this topic later, and at least, suggest steps in the right direction – if not a complete answer. Note, in the three examples above, the questions posed imply a test purpose; i.e., the purpose of adding "999999999" to "1" was to check whether the add operation causes a carry to be conveyed from digit to digit. If it works correctly for this one example, a reasonable assumption is that it works just as well for any other combination of digits that cause a carry – but, this does not imply that the logic of the arithmetic element works for all combinations of digits.

Through this simple example, we introduced several important principles of testing:

1) Fault models must be identified;

2) Given fault models and knowledge of the functions of an entity (a specification of a calculator or a protocol), test purposes must be identified;

3) Test cases must be designed to satisfy the test purposes;

4) If test cases are properly designed, general conclusions can be made about the set of functions tested, and exhaustive testing need not be conducted.

Before pursuing this analogy further (a weak one at best), let us conclude by stating that, before testing can proceed, we need to know what we will test. Since this tutorial is about protocol testing, with a focus on OSI protocols and the work of ISO and CCITT, we direct your attention to some papers which provide a framework for protocol testing in the following sections.

3. Background

In this section we include a set of papers for your review, which introduce the fundamental concepts and terminology employed throughout this tutorial. *The OSI Reference Model* by Day and Zimmerman introduces the framework for development of OSI protocols. *Fundamentals of the Layer Service Definitions and Protocol Specifications* by Linington describes the relationship between service and protocol specifications which are essential ingredients of the OSI-RM. *Connections and Connectionless Data Transmission* by Chapin introduces the distinction between connection and connectionless modes of data transmission. *Towards an Objective Understanding of Conformance* by Rayner provides a background in understanding the problems associated with testing communications protocols – standardized protocols in particular. *Finite Descriptions of Communications Protocols* by Bochmann introduces finite state machines (FSMs) as a model for designing and specifying data communication protocols. This was a seminal concept where Bochmann, Danthine and Piatkowski were early proponents. *A Useful FSM Representation For Test Suite Design and Development,* by Kanungo et al. follows Bochmann's paper describing manual derivation of tests from state table descriptions of the X.25 Data-Link Layer protocol. A brief synopsis of each paper follows.

Day and Zimmerman describe the OSI-RM – a seven-layer framework for the specification of the OSI services and protocols. The OSI-RM partitions functions associated with providing data communications services into discrete services and sets of protocols which focus on well-defined aspects of the problems associated with data communications. The OSI-RM also defines a set of terms and concepts which has become well-established and generally accepted for the discussion of the problems associated with design, specification, validation and testing protocols.

As shown in Figure 1.1, the OSI-RM consists of seven hierarchical layers, each built on its predecessor layer. The number of layers implemented in a given system may vary depending on the function that the system is inteded to perform. At layer *N*, the purpose of an *(N)*-entity is to provide a set of services (called *(N)*-services to its upper layer entity (i.e., *(N+1)*-entity). Similarly, an *(N-1)*-entity provides *(N-1)*-services for an *(N)*-entity. Note that two *(N)*-entities are *peer entities.* These concepts are illustrated in Figure 1.2.

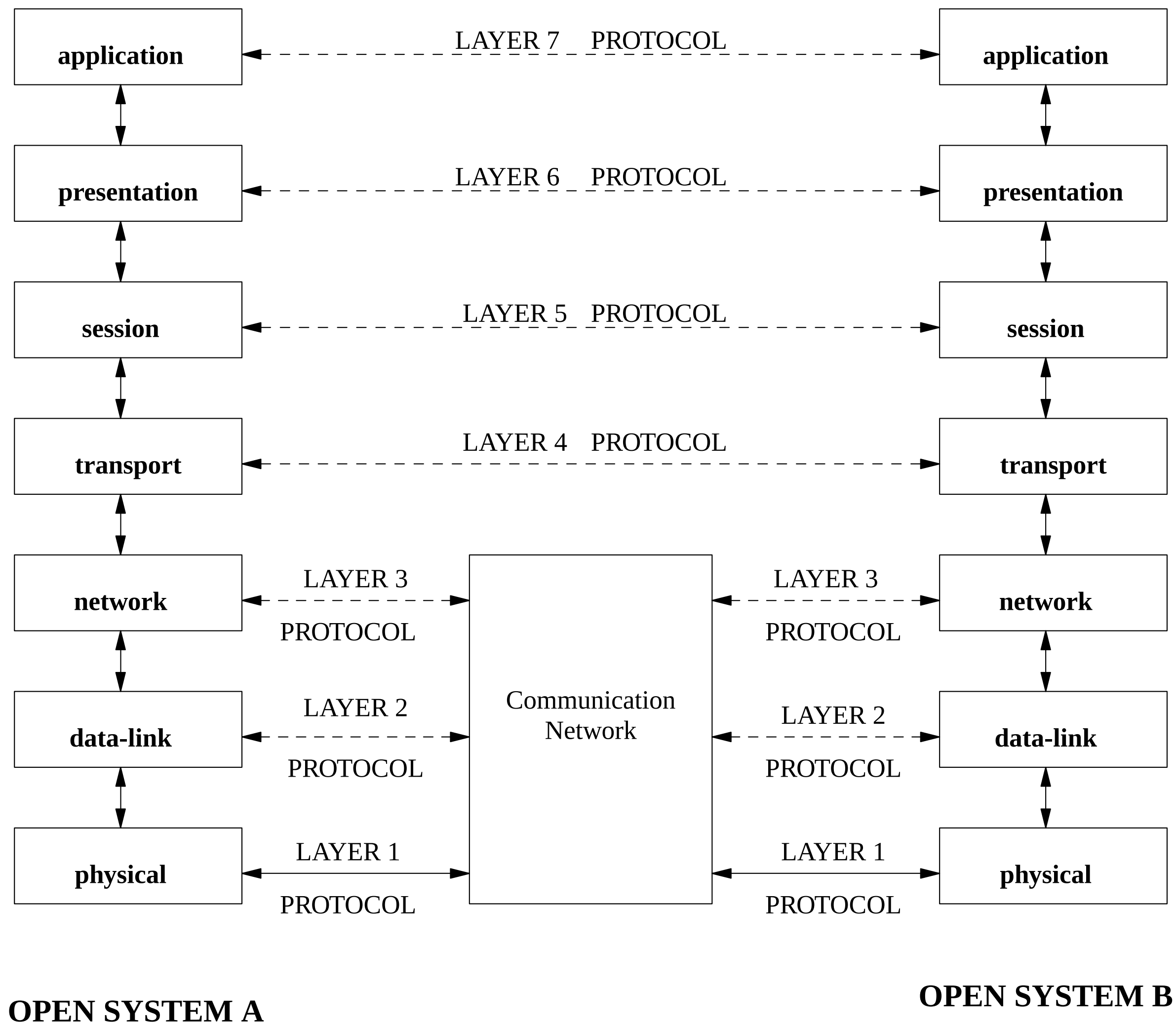

Figure 1.1. Open Systems Interconnection Reference Model

An *(N)*-entity provides *(N)*-services for its user entity by using and enhancing the services of the adjacent lower-layer entity. In this model, an *(N+1)*-entity receives services from an *(N)*-entity, but the implementation details and the services of the *(N-1)*-entity are completely isolated from the *(N+1)*-entity (Figure 1.2); for an *(N+1)*-entity, all lower-layer entities are referred to as *(N)*-service provider.

Peer *(N)*-entities communicate with each other by exchanging *(N)*-Protocol Data Units (*(N)*-PDUs). Conventions and rules for dynamic exchanges of *(N)*-PDUs between two *(N)*-entities constitute an *(N)*-protocol.

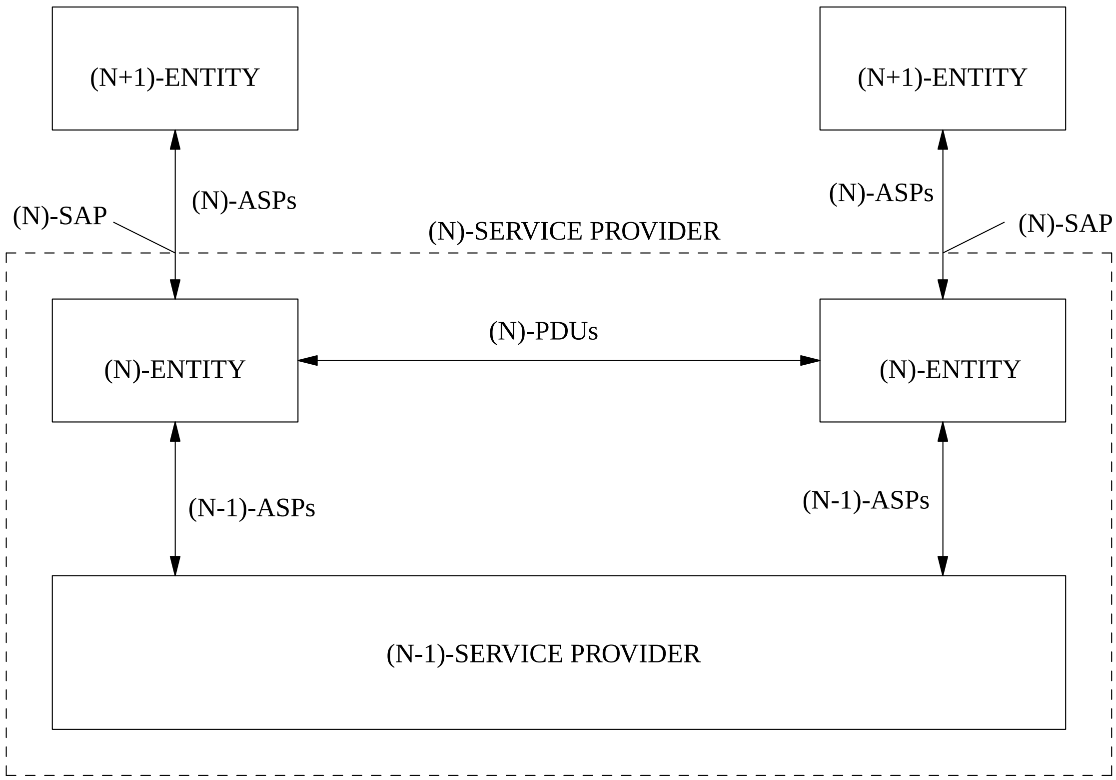

Figure 1.2. Definition of an (N)-Entity in the OSI-RM.

Figure 1.2 illustrates the interactions of an *(N)*-entity with its adjacent upper and lower entities that are defined at *(N)*- and *(N-1)*-service access points, respectively. An *(N)*-service access point (abbreviated as *(N)*-SAP) is a logical interface between *(N)*- and *(N+1)*-entities by which service requests and responses are made.

The OSI-RM spans from the physical media (layer 1) up to application protocols (layer 7). It accommodates a broad spectrum of media at the bottom (point-to point such as twisted pair, fiber, and coaxial cable) switched circuits, and broadcast medium (such as radio, satellite and coaxial buses). These may be integrated into local, metropolitan, and wide area networks (LAN, MAN, and WAN) by Data-Link and Network Layer protocols. Each layer masks the characteristics of lower layer components and leads to *composite services* employed by successively higher layers which are largely independent of the characteristics of lower layer components. A wine glass is an analogy commonly employed to describe the allocation of functions in the OSI-RM. There is a broad base of lower-layer, media-dependent technology and Data-Link protocols; these protocols are employed to realize an error-free, end-to-end, transport service via the Network and Transport Layer protocols (the stem of the wine glass), which collectively support the Session, Presentation and a wide variety of Application protocols (the bowl of the glass).

It is also important to understand what is *not* included in the OSI-RM. It does not define details of protocols or services. Details of services and protocols are defined in separate standards for each layer. The OSI-RM does not define programming interfaces to layer services, nor does it address issues local to an implementation of a protocol in a particular environment. At present, the OSI-RM does not address broadcast or multi-peer services and protocols. Note, however, that the OSI-RM is not a static entity. Since its adoption as a standard in 1983, OSI-RM has been the subject of questions raised by working groups within ISO and CCITT with a set of interpretations and answers published as "final answers to questions." Based on a five year maintenance cycle established by ISO for its standards, the OSI-RM is due to be reissued with revisions and extensions.

The OSI-RM serves as a conceptual schema within which broad classes of functions are defined, and one (or more) protocols may be realized to achieve the services of an individual layer. For example, protocols are tailored to the characteristics of the underlying physical media to support LANs employing carrier-sense multiple access/collision detection, token bus, and token ring control at the media access component of the Data-Link Layer.

In summary, the OSI-RM has defined a conceptual framework for discussion, design, specification, implementation and testing of protocols. Once service definitions have been completed between adjacent layers, these activities may proceed in parallel between layers, and sequentially within layers. At a later date, new protocols (and revisions to old ones) may be developed and incorporated into the suite of OSI protocols. One example is ISDN (Integrated Services Digital Network) protocols which are a set of specifications for the next generation of telephone switches to be deployed. ISDN offers the services of the Network Layer and below as part of the public switched telephone networks, but will integrate transmission of voice, data and digitized video signals using new technology.

The distinction between services and protocols is fundamental to the OSI-RM. Linington's paper titled *Fundamentals of the Layer Service Definitions and Protocol Specifications* presents those distinctions and their application to the SAPs. At an *(N)*-SAP, an *(N)*-entity interacts with an *(N+1)*-entity by exchanging *(N)*-Abstract Service Primitives (*(N)*-ASPs) (Figure 1.2). At the conceptual level, exchange of service primitives is an indivisible event. As shown in Figure 1.3, ervice primitives are categorized as follows:

1) *Request*: a primitive issued by an *(N)*-service user to invoke a particular function (e.g., connection establishment, data transfer, connection termination);

2) *Indication*: a primitive issued by the service provider (*(N)*-layer protocol entity) indicating that the peer *(N)*-service user has requested a specific service or function (e.g., connection establishment, data delivery, connection termination);

3) *Response*: a primitive issued by an *(N)*-service user confirming that a service function previously requested by its peer *(N)*-service user and signaled by a service indication, may proceed to completion (e.g., connection establishment);

4) *Confirm*: a service primitive issued by a service provider (*(N)*-layer protocol entity) indicating that services previously requested by the *(N)*-service user have been completed or established (e.g., connection establishment).

Typically, these category names are prefixed by a name denoting the functional aspects of the service (e.g., Connect Request, Connect Response, Connect Confirm, Data Request, Data Indication).

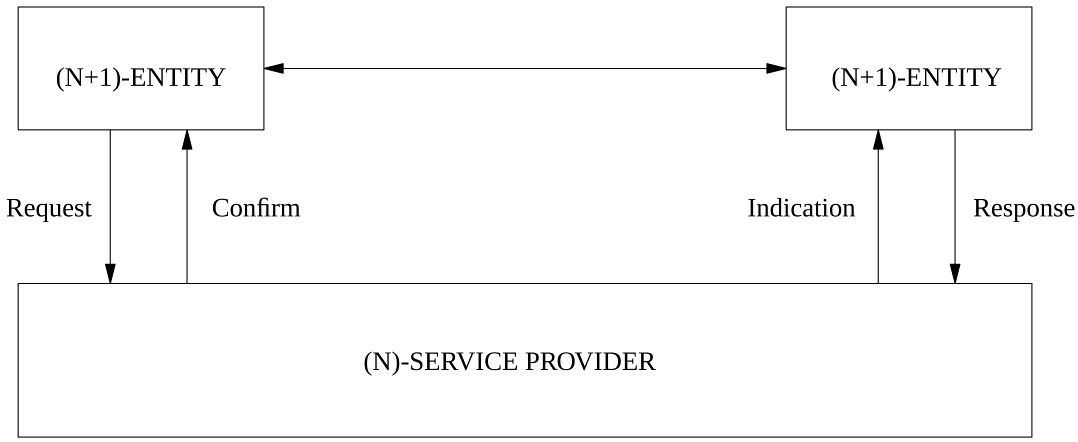

Figure 1.3. Exchange of Primitives Between (N+1)-Entities and an (N)-Service Provider.

Two common modes of service dialogue are:

1) *Confirmed*: dialogue which employs all four types of service primitives noted above, in the temporal relationship noted (1, 2, 3, 4); (Typical of confirmed services is connection establishment.)

2) *Unconfirmed*: dialogue which only employs the first two types of service primitives. Typical of this mode is data transfer over an established connection, or connection termination.

In both categories, the underlying protocol entities are responsible for exchange of PDUs to realize the service requests. Note, at this level of abstraction, reliable transfer of data is *not* assumed or assured; in fact, unreliable transfer is often assumed below the Transport Layer for some protocols.

Conceptually, protocol entities are event driven by service requests and responses from adjacent upper layer users and by data-indications from the *(N-1)*-service provider. As discussed earlier in this section, the *(N)*-layer protocol entities respond to these stimuli by exchanging PDUs via the *(N-1)*-service provider to communicate with each other and realize the *(N)*-service. They deliver information exchanged between peer *(N)*-entities to an adjacent upper layer via indication and confirm service primitives. Assumptions and rules governing the dynamic exchanges of messages between peer protocols entities are dependent on the functions assigned to the layer.

Historically, service interfaces have been made available in products at the Transport and Network Layers. Some products offer full service interfaces at the Application Layer; others do not! Users of a full OSI stack of protocols typically gain access through a command line interpreter or program offered by the product supplier. Service primitives and service access points need not be realized, as such, in an implementation of a protocol; i.e., they are conceptual entities which are *optional* in a product. Thus, their realization is outside the scope of standardization. However, their presence or absence in an implementation profoundly influences the assumptions that can be made when testing protocols. Absence of an exposed service interface presents a problem for those who wish to test protocols layer-by-layer. More will be stated regarding this issue later.

Chapin's paper *Connections and Connectionless Data Transmission* characterizes connection oriented and connectionless modes of data transmission. Connection mode transmission involves three entities: two peer entities and the underlying service for the duration of a *connection*. As such, it has the characteristic of pre-allocating or reserving resources for data transmission over a distinguishable lifetime (a connection). No data may be exchanged until a connection is established (which may involve negotiation). Additionally, queues are usually associated with each of the peer entities and the underlying service provider.

Within the OSI-RM, there is a relationship between the notions of a service access point (SAP) and a connection. If a SAP only services a single *(N+1)*-connection at one time, then the SAP may serve as the *connection end-point identifier*. However, if the *(N)*-layer protocol provides multiplexing of data from more than one *(N+1)*-entity for each *(N)*-layer connection, then a connection end-point identifier is associated with each *(N+1)*-connection, and the *(N)*-layer protocol must be able to bind each *(N+1)*-connection to a connection end-point identifier within the *(N)*-SAP; in other words, logically, a SAP is an addressable unit, and connection end-point identifiers allow distinction between PDUs destined for different users of the same SAP.

Connectionless service does not require a three-party agreement. *(N)*-entities use their underlying service independently, and simply address information to each other without prior pre-allocation or reservation of resources. The only prerequisite is that peer entities know the others' addresses (unless prior agreement is required for security, accounting, etc.). Each data unit exchanged is independent of any prior or future exchange of information. Typically, the exchanged units via an *(N-1)*-service are Data-Requests and Data-Indications. LAN environments are typical providers of connectionless services. While LANs are quite reliable, there is no guarantee of reliable transfer of data; for example, IEEE 802.3 makes a best effort at retransmission when a collision is detected, but data may be lost when the LAN is heavily loaded. Thus, upper layer protocols must recover from lost data (i.e., Transport Layer protocol, Class 4).

Connectionless mode of transmission may also be used to send information to multiple entities, simply by addressing the unit of information to more than one entity. For example, the IEEE 802 series of LANs include an address bit indicating broadcast mode.

In summary, the differences in mode of transmission influence the complexity of the protocol specification. Testing is influenced by: complexity of the protocol, reliability of data transmission, and queueing – particularly if testing is done over an underlying communications service.

Rayner's paper, *Towards an Objective Understanding of Conformance*, reflects experience in early attempts to test OSI protocols and problems associated with that experience. The paper lists a set of questions – born out of frustration – resulting from attempts to interpret requirements (explicitly stated or implied), non-requirements, options, conditional requirements (usually associated with related options), and prohibitive statements (definitions of what *should not* be implemented) found in standards. Note that some ambiguity in standards is due to failure to achieve a technical compromise which is satisfactory to those involved. Thus, either a natural language description is "crafted" (which masks the issue), or an option is added; the responsibility of resolving these ambiguities is left to the test suite designer and test laboratory.

Some of Rayner's comments focus on technical issues, and some on procurement and legal issues. Ambiguity in a protocol standard can ultimately lead to a court of law. For example, when a product fails to pass conformance tests, the supplier may sue the laboratory, or when two products fail to interwork (especially, when there is loss of life or property due to failure of a product), the customer may sue the suppliers, etc. Some concrete examples of technical issues illustrate the point of his frustration:

- Failure of standards to distinguish between static aspects of a protocol (e.g., an option to implement a feature or not) and its impact on dynamic aspects of the protocols behavior.
- The Transport Layer standard includes five classes:
  - Only Class 0 or Class 2 is mandatory – the other classes are optional in a product; nonetheless, CCITT mandates use of Class 0, when the application protocol is Message Handling Systems (MHS) – electronic mail.
  - During class negotiation within the connection establishment phase, one or more classes are proposed by the initiator of the connection. The responding Transport entity may respond affirmatively with any class equal to or lower than the highest class proposed, but the product initiating the connection may not support the class identified in the connect-confirm

PDU (e.g., Classes 1 or 3).

- Procurement standards typically define subsets (or supersets) of the functions mandated in an international standard. Recently, these have become known as *profiles* and are issued by users of products (including governments). Examples include:
  - Government OSI Profile (GOSIP) issued by the government of the U.S.A;
  - Manufacturing Automation Protocols (MAP) issued by an international group of manufactures (including General Motors Corp.);
  - Both of these profiles mandate Class 4 Transport (which is optional in the international standard).
- In Transport Class 4, there are three possible actions which may be taken in response to receipt of an erroneous PDU:
  - Do nothing and wait for retransmission (an unobservable action when testing);
  - Send an *error PDU* identifying the error;
  - Disconnect under certain circumstances (a reason code for disconnection is optional, and, if transmitted, the disconnect reason includes a code for "not specified");

  Also, any combination of the above three actions is allowed in a single product.
- The Session protocol is not clearly divided into subsets or classes; it is a collection of services or functions. Different application protocols may employ completely different subsets (e.g., the Message Handling and File Transfer, Access and Management protocols). The results are:
  - Functions are optional,
  - Products need only implement the minimal subset of functions required by an application.

The problem with a large number of options is that the combinatorial effects may preclude interoperation and make it extremely difficult to determine what is required to test in a particular product.

Given this situation, we can learn how to avoid many of the problems. Definition of PICS (Protocol Implementation Conformance Statement) and PIXIT (Protocol Implementation eXtra Information for Testing) proformas, and using formalism are the two major attempts to eliminate the difficulties in testing that Rayner points out. Let us discuss each effort briefly.

ISO has added two *required* components to its OSI standards which address a number of issues identified by Rayner.

1) A proforma called a PICS contains a set of questions which are to be answered by a product supplier, identifying features implemented (mandatory, optional, and conditional). A PICS may be statically analyzed to ascertain whether all mandatory features have been implemented, which optional features have been implemented, and whether conditional features (those depending on other options or features) have been implemented. The verdict of a conformance assessment is *fail*, without ever testing the product, if the answers indicate that requirements set forth in a standard are not met.

2) A proforma called a PIXIT contains a set of questions providing additional information required for testing. Examples of this class of information include values of timers, addresses of equipment or software components. Typically, this class of information is used to parameterize a test suite (bind in information which varies with each client's product).

CCITT has adopted ISO's requirements, and standards committees are actively preparing PICS for existing and new OSI protocols and PIXITs as part of standardized test suites for OSI protocols. The PICS and PIXIT proformas address some issues, but do not resolve other fundamental issues.

The second major attempt to solve problems associated with testing protocol implementations is using formalism in defining protocol specifications. Since natural languages (written and spoken) are inherently ambiguous, a specification written in a natural language must be crafted with care to define various protocol features. Unfortunately, for most of the existing specifications written in English, ambiguities still constitute major interpretation problems regarding implementation and testing of such protocols. Using tools such as state transition tables, directed graphs, and formal description techniques (FDTs) to specify protocols helps to eliminate under-specified protocol features. These tools, provided that they are used with care, define the temporal relationships for PDU exchanges more rigorously than natural language specifications. We discuss each of the formal representation tools in Section 4 of this chapter. However, one potential problem that the user of formal techniques should avoid is over-specifying a protocol which may lead to restriction or elimination of implementation of valid options (see Section 4.2 of this chapter).

A lesson can be learned from the semiconductor industry. The complexity of VLSI (Very Large Scale Integrated) circuits requires *built-in* facilities to test the circuits. Communication protocols are also complex – particularly in multi-layer configurations. Until protocols are designed to be *tested in products*, conformance assessment with a high degree of confidence in the observations and implied results will be difficult.

4. Modeling Communication Protocols

Formal methods for generation of conformance tests for communication protocols have been primarily based on the so-called *finite state machine (FSM)* model [22] of the protocol specifications. In their classic definition, FSMs are a class of theoretical automata which are often used to recognize strings in a language. In addition to modeling specifications of communication protocols, FSMs have been widely used as a model in compilers for the lexical analysis and parsing of programming languages.

An FSM is an automaton having a finite number of states which changes its state to another when an external stimulus is applied. The *state* of an FSM is defined as the stable condition in which the FSM rests until an external stimulus, called an *input*, is applied. An input causes the FSM to generate an *observable output* (which can be null corresponding to the protocol actions that are not externally observable) and to undergo a state transition from the current state to a new state (not necessarily different from the current state) where it stays until the next input occurs. The *initial state* of an FSM is the state that the FSM is in following the start up (e.g., after the power is turned on).

There is a set of input and output operations for every state of an FSM, called the *permissible input set I* and the *permissible output set O*, respectively. Note that, although the null output is allowed in the permissible output set of an FSM, a null input is usually not a member of the permissible input set since this implies that the FSM may change its state without an external stimulus. We assume that a permissible input can be generated and applied to an FSM implementation by an external source, such as a tester, at any given time.

An FSM is called *fully-specified* if, for every member of I, there exists a permissible output for every state of the FSM. In a fully-specified FSM, the permissible input of each state is equivalent; any permissible input is allowed at any state and generates a permissible output. An FSM is called *partially specified* if some of the inputs are not allowed in some of the states. In other words, in a partially specified FSM, no action is specified for some inputs in some states, including the generation of a null output. Most real-life protocols are partially specified.

An FSM is called *minimal* if the specification does not have two or more equivalent states. State s_i is equivalent to state s_j if the permissible input set of s_i is a subset of the permissible input set of s_j, and the corresponding outputs and the next states are identical.

An FSM is called *deterministic* if its output and the next state is a function of its current state and the input that is applied; otherwise the FSM is called *non-deterministic*.

Let us give a formal definition for an FSM as a tuple:

$$FSM = < S, I, O, T >$$

where:

S is a finite set of states;

I is a set of inputs (the input alphabet);

O is a set of outputs (the output alphabet);

T is a set of functions which maps the *current state* of the automaton into the *next state*:

$$T = \bigcup_j \; < s_j \, , i_j > - - \, t_j - - > < n, o_j > ;$$

where $s_j, n \in S$; $i_j \in I$; $o_j \in O$.

An action of the FSM is a mapping from the current state s_j with input i_j to the next state n, which produces output o_j. That is, for each pair of $< s_j \, , i_j >$, a transition $t_j \in T$ maps the *current state* s_j into the *next state* n, and as part of its action a_j, produces output o_j (which may be *null*). Note that in this model the input and output alphabets do not contain elements with parameters.

An extended FSM (EFSM) is an FSM where the next state and the output of the machine is a function of its current state, the input applied, a set of variables, and boolean expressions referencing the variables and input.

Using the definitions of *S, I,* and *O* from above, an EFSM may be represented as a tuple:

$$EFSM = < S, I, O, V, P, B, T >$$

where:

V is a set of variables;

P is the set of parameters associated with the set of input PDUs. Consequently, we restrict the interpretation of the input alphabet to the identifier of a PDU which we will denote as i_j; we denote the parameters of a PDU as $p_{j,k}$. Therefore, the input alphabet of an EFSM is the union of these sets:

$I = \bigcup_j \; i_j \; \bigcup_k \; p_{j,k}$ (the input alphabet consists of PDUs: a PDU identifier and the parameters of the PDU);

$M = V \; \bigcup \; P$ (i.e., the variables and parameters of PDUs serve as memory);

B is a set of Boolean expressions which may make reference to memory M (V or P);

T is the actions of an EFSM which consist of the set of spontaneous transitions (*ST*) and input transitions (*IT*):

$$T = ST \bigcup IT$$

where:

ST is a set of *spontaneous transitions* which may make reference to variables in memory (m_j) in a Boolean expression (b_j):

$$ST = \bigcup_j \; < s_j \, , b_j \, , m_j > - - \, t_j - - > < n, o_j >$$

where $s_j, n \in S$; $b_j \in B$; $m_j \in (V_k$ for $k = 1 \cdots z)$; $o_j \in O$.

IT is the set of input transitions which may make reference to inputs (i_j) and their parameters ($p_{j,k}$):

$$IT = \bigcup_j \; < s_j \, , b_j \, , i_j > - - \, t_j - - > < n \, , o_j > ;$$

where $s_j, n \in S$; $b_j \in B$; $i_j \, , (p_{j,k} \in P$, for $k = 0 \cdots z) \in I$; $o_j \in O$.

In the EFSM representation above, *ST* is a set of *spontaneous transitions* which may make reference to variables in memory m_j and requires no inputs; and *IT* is a set of *input transitions* which may make reference to parameters of inputs (PDUs) p_j and variables in memory m_j.

Note that spontaneous transitions can potentially make a specification non-deterministic. For example, expiry of a timer may trigger a spontaneous transition. If the cause (i.e., the event, or the value of the variable) that triggers the timeout cannot be controlled externally, the timeout may happen unexpectedly during testing. Under such circumstances, an implementation may never be tested for certain features. A similar argument is also valid for input transitions when the values of variables or input parameters cannot be controlled by an external tester. In Chapter III, we will discuss this issue in detail.

In the remainder of this book, the discussions will be primarily based on FSMs. However, the distinctions will be noted when the behavior of an FSM and an EFSM differs.

4.1. Representing FSMs

The most common way of representing FSMs is in a tabular form called the *state transition tables*. The rows of the table correspond to the states of the FSM and the columns to the permissible inputs. The first column and the top row of the table define the set of states and the permissible inputs, respectively. The intersection of a row and a column corresponding to state s_j and input i_k defines the output and the next state of the FSM. If an FSM is partially specified, there are empty entries in the state transition table representing the undefined actions for some inputs in some of the states. Figure 1.4 gives the state transition table of an example FSM with seven states, s_0 through s_6. This example is called *PhoneDTE* which resembles to Network Layer interface of a telecommunication system. In reality, such a system will have two interfaces, one for the user of the system and another for the communication network. Therefore, the inputs and outputs of PhoneDTE are exchanged with more than one entity. In Figure 1.4, the inputs with the prefix of *U* are received from the entity called *user*, and ones with the prefix of *N* are received from the entity called *network*; the same convention is also used for outputs. We will use the PhoneDTE specification throughout this chapter and in Chapter III.

inputs / **states**	**U.ConReq**	**N.SetupAck**	**U.Digit**	**N.Alert**	**N.Con**	**N.Disc**	**U.ClearReq**	**N.Prog**
s_0	N.Setup (s_1)	-	-	-	-	-	-	-
s_1	-	U.DialTone (s_2)	-	-	-	-	N.Disc (s_0)	-
s_2	-	-	U.Info ToneOff (s_3)	-	-	-	N.Disc (s_0)	U.Announce (s_6)
s_3	-	-	N.Info (s_3)	U.RingBack (s_4)	-	-	N.Disc (s_0)	U.Announce (s_6)
s_4	-	-	-	-	U.ConAck (s_5)	U.Stop RingBack (s_6)	N.Disc (s_0)	-
s_5	-	-	-	-	-	U.StopCon (s_6)	N.Disc (s_0)	-
s_6	-	-	-	-	-	-	N.Null (s_0)	-

Figure 1.4. An example FSM, called PhoneDTE, specified by a state transition table.

The permissible input set of the FSM is given at the top row of the table, and the states are represented in the left-most column. The output and the next state of the FSM are identified by the table entry at the intersection of corresponding row and column. For example, if the current state of the FSM is s_0 and the input called *U.ConReq* is applied, the FSM generates an output called *N.Setup* and moves to state s_1 (shown in parenthesis in Figure 1.4). Note that the FSM of Figure 1.4 is partially defined, since some of the inputs are not allowed in some states; for example, input *U.ConReq* is only allowed in state s_0. The inputs that are not allowed are represented as "-" in the corresponding entries of Figure 1.4.

Another commonly-used tool for representing FSMs is a directed graph $G = (V, E)$, which is equivalent to a state transition table. The set $V = \{ v_1 , \cdots , v_n \}$ of vertices corresponds to the specified set of states $\{s_1 , \ldots , s_n\}$ of the FSM. In G, a directed edge from vertex v_i to v_j with a label of $input_k$ / $output_l$ corresponds to a state transition in the FSM from state s_i to state s_j by applying the input called $input_k$ and observing the output called $output_l$. We will represent such an edge by using the following notation:

$$(v_i , v_j ; input_k / output_l)$$

Figure 1.5 gives the graph representation of the FSM defined in Figure 1.4.

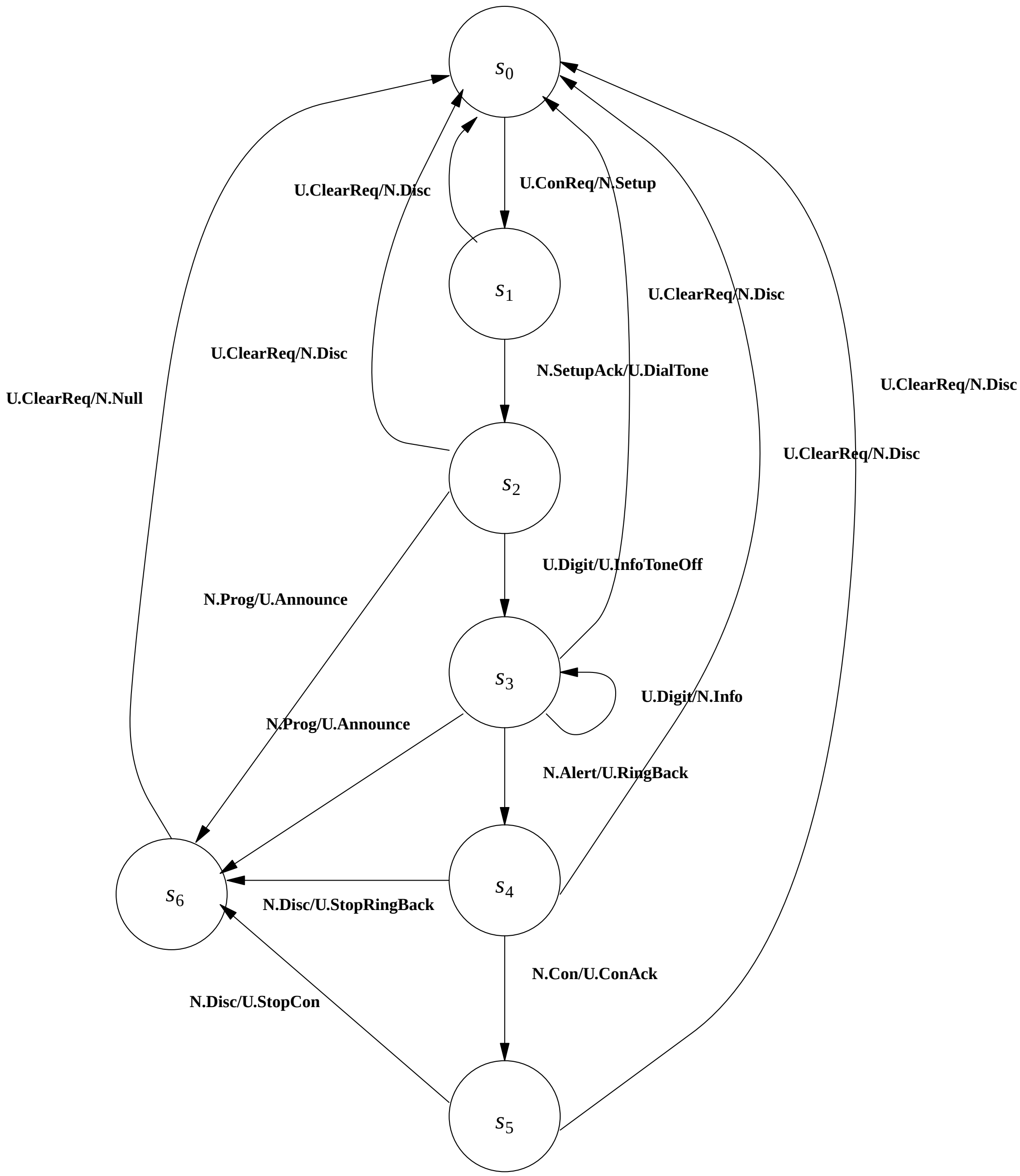

Figure 1.5. Directed graph representation of PhoneDTE example given in Figure 1.4.

In Figure 1.5., the directed edge from s_1 to s_2 with the label of *N.SetupAck/U.DialTone* is represented in this notation as

$$(s_1 , s_2 ; N.SetupAck / U.DialTone)$$

An edge in G that starts and ends at the same vertex is called a *self-loop*. In Figure 1.5, state s_3 has a self-loop labeled as $U.Digit/N.Info$:

$$(s_3 , s_3 ; U.Digit / N.Info)$$

A *tour* in G is a non-null sequence of consecutive edges that starts and ends at the same vertex. Note that, in a tour of G, an edge may appear more than once.

In a directed graph representing a deterministic FSM, there cannot be two edges $(v_i , v_j ; input_k / output_l)$ and $(v_i , v_m ; input_p / output_r)$ defined for the same vertex v_i where $input_k = input_p$ (i.e., two edges directed away from v_i with the same input). Such two edges, regardless of the outputs $output_l$ and $output_r$, would imply that the next state of the FSM is not a function of the current state and the input applied, and, therefore, the FSM is non-deterministic.

A directed graph is *strongly-connected* if, for any pair of distinct vertices v_i and v_j, there exists a non-null sequence of edges from v_i to v_j. A strongly-connected directed graph corresponds to an FSM specification that is free of livelocks and deadlocks. Livelock is a situation where no progress towards useful work can be made, such as an infinite-loop. Deadlock corresponds to a situation where the specification cannot make any progress at all; for example, waiting for an input which will never be sent is a deadlock.

Bochmann's paper *Finite State Description of Communicating Protocols* discusses several important topics related to rendezvous mode of communications, refinement, and verification (called validation in the paper) of protocols specified by FSM models. The highlights of the paper include:

- an overview of modeling protocols using FSMs, identifying both the advantages and the limitations of the FSM model (in terms of validation),
- application of the service model to decompose or *refine* the problem of specifying a protocol,
- several definitions of important principles in protocol design and protocol validation,
- dynamic behavioral analysis of a protocol entity including:
  - a simple example known as the "alternating bit protocol,"
  - the relative synchronization between pairs of peer entities (or "adjunct states" as Bochmann calls them),
  - the call setup procedures of the X.25 protocol. (In a later paper, Sarikaya and Bochmann [23] report their experience in generating test sequences from the same underlying model, which is discussed in Chapter III.)

The term "direct coupling" used in the paper describes the synchronous state changes in a pair of system components: one component produces an output while the other accepts it as an input. This synchronous input/output (I/O) exchange is commonly called *rendezvous* and is associated with non-buffered I/O. While this mechanism is assumed by Bochmann in most of the paper, synchronous I/O is not a necessary component of the FSM model; other authors may assume queued (or asynchronous) I/O. However, rendezvous is a more primitive I/O mode, because a queued I/O scheme maybe derived from non-buffered I/O exchanges by inserting a queuing process between a pair of communicating entities. Bochmann also discusses the notion of *spontaneous* transitions (i.e., transitions realized without an external input). As discussed previously, spontaneous transitions used in specifications may easily result in features that cannot be tested.

Bochmann illustrates the concept of a layered communications design which was treated in Day and Zimmerman's paper on the OSI-RM. Layering as a design principle predates the OSI-RM. (For example, Digital's DECNET and IBM's SNA were layered architectures which predate the OSI-RM;

thus, layered architectures were an established practice.) Section 4.3 of Bochmann's paper shows how the service between users (source and sink, in Figure 1) may be modeled by an FSM. The service FSM is extracted from the protocol FSM, which is backward from generally accepted design principles. Nonetheless, the paper illustrates that an FSM is an appropriate model for service descriptions. A similar analysis should be done between entities in adjacent layers, prior to the protocol specification.

The paper also discusses the issue of protocol validation, which is beyond the scope of this tutorial. However, the definitions of deadlock, livelock (e.g., infinite-loops), reachability (analysis), self synchronization, and characterization of the operation of an FSM by action sequences should be noted. These concepts are essential to understanding predicaments that are the results of design flaws and some of the principles underlying protocol testing. In Section 6 of this chapter we include a brief discussion of this topic.

Bochmann does not speak about fully-specified FSMs. Recall that, in a fully-specified FSM, there is an action associated with *every state for every possible input* (the cartesian product of states and inputs). Inputs which have no semantics in a particular state are usually either discarded, or precipitate an abrupt termination of the protocol. Because a specification is partially-specified, it does not mean that it is specified wrong! This is a design choice, although some authors do not agree with this position. Quite often, protocol specifications are partially specified because the authors intend to specify only valid behaviors, or it is not possible (or meaningful) to generate every input at every state. In ISO terminology, syntactically valid PDUs which arrive in a sequence that was *not* intended for are called *inopportune PDUs*. A partial specification *may be exploited* by test system designers to detect inopportune PDUs. These issues are discussed further in Chapters II and III.

The paper by Bochmann closes with an analysis of the call setup procedure of the 1976 version of the X.25 Network Layer Protocol. Two unexpected results (potential infinite-loops in the specification) are also reported.

A Useful FSM Representation For Test Suite Design and Development, by Kanungo et al. follows Bochmann's paper quite naturally. The authors describe the derivation of test sequences from a tabular description of the X.25 Data-Link protocol. A feature of the paper is that it illustrates expression of test cases in ISO's test language using the tabular form of Tree and Tabular Combined Notation (TTCN). Since test derivation is the topic of Chapter III, we do not pursue the topic any further at this point.

4.2. Formal Description Techniques

ISO and CCITT have defined formal description languages, known as formal description techniques (FDTs), to formally define communication protocol specifications. These FDTs, called SDL [24], Estelle [25] and LOTOS [26], are intended to eliminate ambiguities in the specifications, hence, promoting interoperability of implementations.

SDL and Estelle are based on EFSMs; LOTOS, which is a process algebra, is a successor of Milner's *Calculus of Communicating Systems*[50] and Hoare's *Communicating Sequential Processors* [27]. The definitions, characteristics and comparison of these formal languages are beyond the scope of this tutorial. Ideally, these formal description languages are functionally equivalent, since they are defined to specify the same entity: a communication protocol. In this section, for the sake of completeness, we include the formal description of the example protocol of Figure 1.4 in Estelle (Figure 1.6), in SDL (Figure 1.7) and LOTOS (Figure 1.8).

We should note that protocol designers using FDTs must keep in mind the important trade-off between precisely-specified protocols and over-specified protocols. It is much easier to test an implementation based on a precise specification. However, the precision of a specification should not limit manufacturers' freedom in product implementation – only a manufacturer can make the engineering trade-offs. The delicate balance between testability and implementation freedom remains as a challenge for protocol designers.

Figure 1.6. Estelle Description of PhoneDTE.

```
Specification PhoneDTE  Systemprocess;

 channel DTE2User (User, DTE);
  by User:
    {Signals/Interactions sent to the DTE}
    ConReq;   {off hook}
    ClearReq; {on  hook}
    Digit;    {Address digits}
  by DTE:
    {Signals/Interactions sent to the User}
    Announce; {Busy or out of service}
    DialTone; {Dial tone is on}
    InfoToneOff; {Dial tone is off}
    RingBack; {Remote party ringing}
    StopRingBack;{RingBack is off}
    ConAck;   {connection is established}
    StopCon; {connection is disconnected}
 channel DTE2Network (DTE, Net);
  by DTE:
    {Signals/Interactions sent to the Network}
    Info;
    Setup;
  by Net:
    {Signals/Interactions sent to the DTE}
    Alert;
    Con;
    Prog;
    SetupAck;
  by DTE, Net: {Sent by both}
    Disc; module PhoneConnection process;
 ip
  {declaration of interaction points of the module}
   UserInt: DTE2User(DTE) individual queue;
   NetInt: DTE2Net (DTE) individual queue;
   end;  {of module header}

{body of the module which defines the actions of the module}
 body Connection for PhoneConnection;

{Declaration of the control states of the module}
 state s0, s1, s2, s3, s4, s5, s6
 init  {initialize the control state}
  to s0
        begin  {no other action -- output is null}
        end;
 trans {transition declarations of the module}
 from s0 {current state}
  to s1 {next state}
   when UserInt.ConReq  {input at user interface}
       begin   output  NetInt.Setup
       end;
 from s1, s2, s3, s4, s5 {in any of these states,
                     take a common action}
  to s0
   when UserInt.ClearReq
       begin  output  NetInt.Disc
       end;
 from s1
  to s2
   when NetInt.SetupAck
       begin  output UserInt.DialTone
       end;
 from s2
  to s3
   when UserInt.Digit
       begin  output UserInt.InfoToneOff
       end;
 from s2, s3  {in both states, take the same action}
  to s6
   when  NetInt.Prog
       begin  output UserInt.Announce
       end;
 from s3
  to s3
   when UserInt.Digit
       begin  output  NetInt.Info
       end;
  to s4
   when NetInt.Alert
       begin  output UserInt.RingBack
       end;
 from s4
  to s5
   when NetInt.Con
       begin  output UserInt.ConAck
       end;
  to s6
   when NetInt.Disc
       begin  output UserInt.StopRingBack
       end;
 from s5
  to s6
   when NetInt.Disc
       begin  output UserInt.StopCon
       end;
 from s6
  to s0
   when UserInt.ClearReq
       begin  {no other action -- null output}
       end;
 end;  {of module body} end. {of specification}
```

Figure 1.7. SDL Description of PhoneDTE.

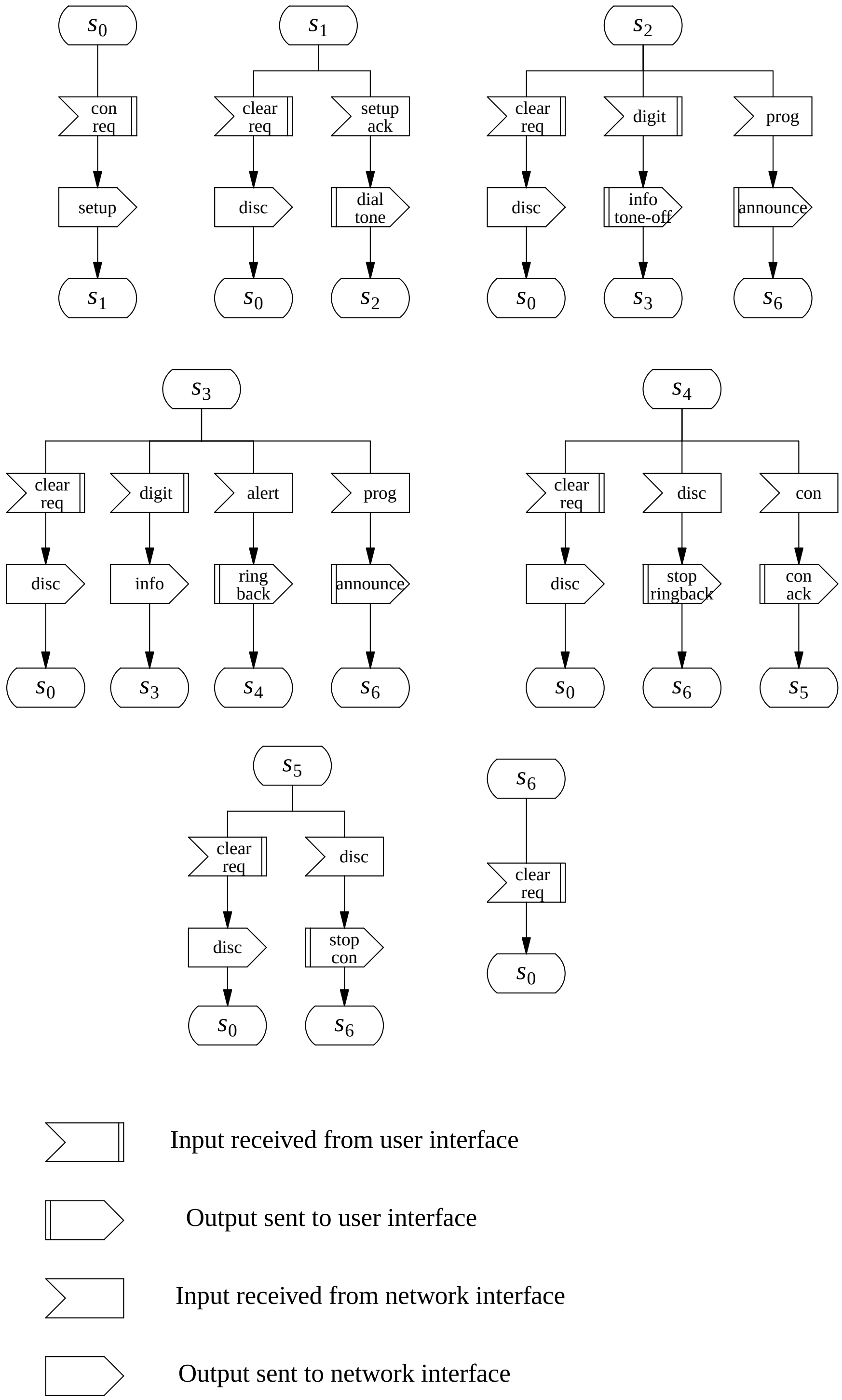

Figure 1.8. LOTOS Description of PhoneDTE.

```
(*
The user and network interfaces are
represented by two gates called
UserInt and NetInt, respectively
*)
specification [UserInt,NetInt] : noexit

type local_state is
        sorts local_state
endtype
type ConReq_type is
        sorts ConReq_type
endtype
type ClearReq_type is
        sorts ClearReq_type
endtype
type SetupAck_type is
        sorts SetupAck_type
endtype
type Digit_type is
        sorts Digit_type
endtype
type Prog_type is
        sorts Prog_type
endtype
type Alert_type is
        sorts Alert_type
endtype
type Disc_type is
        sorts Disc_type
endtype
type Con_type is
        sorts Con_type
endtype
behavior PhoneDTE [UserInt,NetInt]
 DTE [UserInt,NetInt] (s0 )
 where
  process DTE [UserInt,NetInt]
        (s: local_state) : noexit :=
        [s = s0] -> UserInt?ConReq:ConReq_type;
          NetInt!Setup; DTE[UserInt,NetInt] (s1)
  []
        [s = s1] ->
        (
          UserInt?ClearReq:ClearReq_type;
          NetInt!Disc; DTE[UserInt,NetInt] (s0)
        []
          NetInt?SetupAck:SetupAck_type;
          UserInt!DialTone; DTE[UserInt,NetInt] (s2)
        )
  []
        [s = s2] ->
        (
          UserInt?ClearReq:ClearReq_type;
          NetInt!Disc; DTE[UserInt,NetInt] (s0)
        []
          UserInt?Digit:Digit_type;
          UserInt!InfoToneOff; DTE[UserInt,NetInt] (s3)
        []
          NetInt?Prog:Prog_type;
          UserInt!Announce; DTE[UserInt,NetInt] (s6)
        )
  []
        [s = s3] ->
        (
```

```
            UserInt?ClearReq:ClearReq_type;
            NetInt!Disc; DTE[UserInt,NetInt] (s0)
          []
            UserInt?Digit:Digit_type;
            NetInt!Info; DTE[UserInt,NetInt] (s3)
          []
            NetInt?Prog:Prog_type;
            UserInt!Announce; DTE[UserInt,NetInt] (s6)
          []
            NetInt?Alert:Alert_type;
            UserInt!RingBack; DTE[UserInt,NetInt] (s4)
          )
  []
          [s = s4] ->
          (
            UserInt?ClearReq:ClearReq_type;
            NetInt!Disc; DTE[UserInt,NetInt] (s0)
          []
            NetInt?Disc:Disc_type;
            UserInt!StopRingBack; DTE[UserInt,NetInt] (s6)
          []
            NetInt?Con:Con_type;
            UserInt!ConAck; DTE[UserInt,NetInt] (s5)
          )
  []
          [s = s5] ->
          (
            UserInt?ClearReq:ClearReq_type;
            NetInt!Disc; DTE[UserInt,NetInt] (s0)
          []
            NetInt?Disc:Disc_type;
            UserInt!StopCon; DTE[UserInt,NetInt] (s6)
          )
  []
          [s = s6] ->
            UserInt?ClearReq:ClearReq_type;
            DTE[UserInt,NetInt] (s0)

  endproc (*end of process DTE*)
endproc (*end of behavior*)
endspec
```

5. Summary

In this chapter, we introduce the reader to the OSI Reference Model which is a fundamental framework underlying standardized communication protocols. Each layer of the Reference Model is defined such that it is independent of the implementation details of the other layers in the system. Definitions and relationships between protocol specifications and service descriptions are discussed. Two data transmission modes, namely, connection oriented and connectionless, are defined. The issues related to testing protocol implementations are briefly outlined. Finally, we introduce the formal tools to represent protocol specifications: FSMs and EFSMs and their representation as state transition tables and directed graphs. We also give an example for each FDT and leave the topic as an advance study for the interested readers.

Below are the full-papers that we include at the end of this chapter (in the order of appearance):

- I. C. Davidson, *"International Conformance Testing – Towards the Next Decade,"* **Proc. Second International Workshop on Protocol Test Systems**, J. de Meer, L. Mackert and W. Effelsberg (eds.), North-Holland, Oct. 1989, pp. 3-15.
- J.D. Day and H. Zimmermann, *"The OSI Reference Model,"* **Proc. of the IEEE**, No. 71, 1983, pp. 1334-1340.
- P.F. Linington, *"Fundamentals of the Layer Service Definitions and Protocol Specifications,"* **Proc. of the IEEE**, No. 71, 1983, pp. 1341-1345.
- A.L. Chapin, *"Connections and Connectionless Data Transmission,"* **Proc. of the IEEE**, Vol. 71, No. 12, 1983, pp. 1365-1371.
- D. Rayner, *"Towards an Objective Understanding of Conformance,"* **Proc. Protocol Specification, Testing, and Verification III**, H. Rudin and C.H. West (eds.), North-Holland, 1983.
- G.v. Bochmann, *"Finite Descriptions of Communications Protocols,* **Computer Networks**, Vol. 2, October 1978, pp. F3.1-F3.8.
- B. Kanungo, L. Lamont, R.L. Probert, and H. Ural, *"A Useful FSM Representation for Test Suite Design and Development,"* **Proc. Protocol Specification, Testing, and Verification VI**, B. Sarikaya and G.v. Bochmann (eds.), North-Holland, 1986, pp. 163-176.

6. Open Problems and Further Reading

The past decade has been a period of intense growth and development in data communications protocols. This is particularly true with respect to international standards with the development of the OSI-RM and the related standardized protocols. The topics related to the research and development in methodologies for specification and verification of communications protocols are beyond the scope of this tutorial. The reader is referred to the proceedings of symposia and workshops given in references [4]-[20].

In this chapter, we discussed the tools to represent protocol specifications with the emphasis on FSMs and EFSMs. For those interested in further reading on the topics of FSM models for protocol specification, we suggest the following papers by Bochmann et al. [28],[29], and Linn [30]. Also, we do not address Petri Nets as a tool to represent protocol specifications, since it can be shown that Petri Nets are equivalent to FSMs and EFSMs for protocol modeling. The interested reader can find an example of the use of Petri Nets for specifying protocols in Danthine's paper [31].

The theoretical framework of FSMs and process algebra has led to the development of FDTs. As Rayner notes, FDTs by themselves do not solve all problems. In fact, they present a new set of problems. Potential over-specification is just one. Also, any OSI protocol may be specified in one or more of the FDTs since we have Estelle, LOTOS, and SDL. It is beyond our current abilities to compare the specifications of the same protocol written in different FDTs and determine if they define the same dynamic behavior.

As discussed briefly in Section 4.2, there are many challenges to be overcome by the researchers and practitioners who apply the FDTs. Once a protocol is specified by using an FDT, the first burden is to prove (or demonstrate) that the specification is free of livelocks, deadlocks, and logical errors. This problem is known as *verification* of a specification (sometimes referred to as *validation*). The main issue in protocol verification is the so-called state explosion problem which arises for FDTs using multiple communicating processes to specify a protocol. The overall behavior of the protocol may be obtained by composing these processes into a global process and using reachability analysis techniques for the composed process. Typically, such a calculation and the size of the composed process is too large to analyze. For various issues involved in verification see the papers by Rudin [32], West [33], Holzmann [34], Lam and Shankar [35], Zafiropulo [36], Sabnani et al. [37], Vuong et al. [38], Maxemchuk [39] Pehrson [40], Gouda [41], Miller and Lundy [42], and Lam and Shankar [43].

Another issue that protocol designers face (regardless of whether they use FDTs or not) is the use of spontaneous transitions (as defined in Section 4 of this chapter) which make protocols easy to specify and embody a wide range of implementation options, but, at the same time, extremely difficult to test. We will discuss this issue in Chapter III when we introduce test generation methodologies.

Introductory material on FDTs include Estelle by Budkowski and Dembinski [44], Linn [45] and Tenney [46], LOTOS by Bolognesi and Brinksma [47], predecessors of LOTOS called Communicating Sequential Processes by Hoare [27] and Milner [50], and SDL by Dickson and Chatzal [48], and Saracco and Tilanus [49].

8. References for Chapter I

[1] *Information Processing Systems – Open Systems Interconnection – Basic Reference Model*, **International Organization for Standardization**, ISO 7498, 1974.

[2] A.S. Tanenbaum, *Computer Networks,* Prentice Hall, 1981.

[3] *Open Systems Interconnection (ISO) – Standard Architecture Protocols (Special Issue)*, **Proc. of the IEEE**, Vol. 71, No. 12, December 1983.

[4] *Protocol Testing – Towards Proof*, D. Rayner and R.W.S. Hale (eds.), Vol. 1-2, **INWG/NPL Workshop**, National Physical Laboratory, Teddington, Middlesex, TW11 OWL, U.K, 1981.

[5] **Proc. Protocol Specification, Testing, and Verification II**, C. Sunshine (ed.), North-Holland, 1982.

[6] **Proc. Protocol Specification, Testing, and Verification III**, H. Rudin and C.H. West (eds.), North-Holland, 1983.

[7] **Proc. Protocol Specification, Testing, and Verification IV**, Y. Yemini, R. Strom and S. Yemini (eds.), North-Holland, 1984.

[8] **Proc. Protocol Specification, Testing, and Verification V**, M. Diaz (ed.), North-Holland, 1985.

[9] **Proc. Protocol Specification, Testing, and Verification VI**, B. Sarikaya and G.v. Bochmann (eds.), North-Holland, 1986.

[10] **Proc. Protocol Specification, Testing, and Verification VII**, H. Rudin and C.H. West (eds.), North-Holland, 1987.

[11] **Proc. Protocol Specification, Testing, and Verification VIII**, K. Sabnani and S. Aggarwal (eds.), North-Holland, 1988.

[12] **Proc. Protocol Specification, Testing, and Verification IX**, E. Brinksma, G. Scollo, and C. Vissers (eds.), North-Holland, 1989.

[13] **Proc. Protocol Specification, Testing, and Verification X**, L. Logrippo, R. L. Probert, and H. Ural (eds.), North-Holland, 1990.

[14] **Proc. Protocol Specification, Testing, and Verification XI**, B. Jonsson, J. Parrow, and B. Pehrson (eds.), North-Holland, 1991.

[15] **Proc. Protocol Specification, Testing, and Verification XII**, R. J. Linn and M. U. Uyar (eds.), North-Holland, 1992.

[16] **Proc. First International Workshop on Protocol Test Systems**, S.T. Chanson and S.T, Vuong (eds.), North-Holland, 1990.

[17] **Proc. Second International Workshop on Protocol Test Systems** J. de Meer, W. Effelsberg, and L. Mackert (eds.), North-Holland, 1989.

[18] **Proc. Third International Workshop on Protocol Test Systems** I. Davidson and D. M. Litwack (eds.), North-Holland, 1990.

[19] **Proc. Fourth International Workshop on Protocol Test Systems** J. Kroon, R. J. Heijink, and E. Brinksma (eds.), North-Holland, 1991.

[20] **Proc. Fifth International Workshop on Protocol Test Systems** G. V. Bochmann, A. Das, and R. Dssouli (eds.), North-Holland, 1992.

[21] *Information Processing Systems – OSI Conformance Testing Methodology and Framework*, **International Organization for Standardization**, ISO/IEC JTC 1, IS 9646, 1991.

[22] Z. Kohavi, " *Switching and Finite Automata Theory*," McGraw-Hill, 1978.

[23] B. Sarikaya and G.v. Bochmann, "*Some Experience with Test Sequence Generation for Protocols*," **Proc. Protocol Specification, Testing and Verification II**, C. Sunshine (ed.), North-Holland, 1982, pp.555-567.

[24] *CCITT Recommendations Z.101-Z.104*, (Blue Book Series), *SDL*, **CCITT**, 1988.

[25] *Estelle: A Formal Description Technique Based on an Extended State Transition Model*, **International Organization for Standardization**, IS 9074, 1988.

[26] *Information Processing Systems – Open Systems Interconnection – LOTOS – A Formal Description Technique Based on Temporal Ordering of Observed Behavior*, **International Organization for Standardization**, IS 8807, 1988.

[27] C.A.R. Hoare, "*Communicating Sequential Processes*," Prentice-Hall, Englewood, New Jersey, 1985, pp. 111-117.

[28] G.v. Bochmann, "*A General Transition Model for Protocols and Communication Services*," **IEEE Transactions on Communications**, Special Issue on Computer Network Architectures and Protocols, Vol. COM-28, No. 4, Apr. 1980, pp. 643-650.

[29] G.v. Bochmann and C.A. Sunshine, "*Formal Methods in Communication Protocol Design*," **IEEE Transactions on Communications**, Special Issue on Computer Network Architectures and Protocols, Vol. COM-28, No. 4, ,April 1980, pp. 624-631.

[30] R.J. Linn, "*Conformance Evaluation Methodology and Protocol Testing*," **IEEE Jour. on Selected Areas in Communications**, Vol. 7, No. 7, Sept. 1989, pp. 1143-1158.

[31] A.A.S. Danthine, "*Protocol Representations with Finite-State Machine Models,"* **IEEE Transactions on Communications**, Special Issue on Computer Network Architectures and Protocols, Vol. COM-28, No. 4, April 1980. pp. 632-643,

[32] H. Rudin, "*An Informal Overview of Formal Protocol Specification*," **IEEE Communications Magazine**, Vol. 23, No. 3, March 1985, pp. 46-52.

[33] C.H. West, "*An Automated Technique of Communications Protocol Validation,"*, **IEEE Trans. on Communications**, Vol. COM-26, No. 8, Aug. 1978, pp. 1271-1275.

[34] G.J. Holzmann, "*An Improved Protocol Reachability Analysis Technique,*" **Software, Practice and Experience**, June 1987, pp.683-696.

[35] S.L. Lam and A.U. Shankar, "*Protocol Verification via Projections,*" **IEEE Trans. on Software Engineering**, Vol. SE-10, No. 4, July 1984, pp.325-342.

[36] P. Zafiropulo et al., "*Towards Analyzing and Synthesizing Protocols,*" **IEEE Transactions on Communications**, Special Issue on Computer Network Architectures and Protocols, Vol. COM-28, No. 4, April 1980, pp. 624-631.

[37] K.K. Sabnani, A.M. Lapone and M.U. Uyar, "*An Algorithmic Procedure for Checking Safety Properties of Protocols*," **IEEE Transactions on Communications**, Vol. COM-27, No. 9, Sept. 1989, pp. 940-948.

[38] S.T. Vuong, D.D. Hui and D.D. Cowan, "*Valira – A Tool for Protocol Verification via Reachability Analysis*," **Proc. Protocol Specification, Testing and Verification VI**, B. Sarikaya and G.v. Bochmann (eds.), North-Holland, 1986, pp. 35-42.

[39] N.F. Maxemchuk and K.K. Sabnani, "*Probabilistic Verification of Communications Protocols*," **Proc. Protocol Specification, Testing and Verification VII**, H. Rudin and C.H. West (eds.), North-Holland, 1987, pp. 307-320.

[40] B. Pehrson, "*Abstraction by Structural Reduction*," **Proc. Protocol Specification, Testing and Verification III**, H. Rudin and C.H. West (eds.), North-Holland, 1983.

[41] M.G. Gouda, "*Closed Covers: To Verify Progress for Communicating Finite State Machines*," **IEEE Transactions on Software Engineering**, Vol. SE-10, No. 6, Nov. 1984, pp. 846-855.

[42] R.E. Miller and G.M. Lundy, "*An Approach to Modeling Communication Protocols Using Finite State Machines and Shared Variables*," **Proc. IEEE Global Communications Conf.**, Dec. 1986, pp. 3.8.1-3.8.5.

[43] S.S. Lam and U. Shankar, "*Protocol Verification via Projections*," **IEEE Transactions on Software Engineering**, Vol. SE-10, No. 4, July 1984, pp. 474-491.

[44] S. Budkowski and P. Dembinski, "*An Introduction to Estelle: A Specification Language for Distributed Systems*," **Computer Networks and ISDN Systems**, Vol. 14, pp. 3-23, 1987.

[45] R.J. Linn, "*The Features and Facilities of Estelle*," National Institute of Standards and Technology, National Computer Systems Laboratory, Maryland, U.S.A., ICST/SNA 87/6, Nov. 1988.

[46] R. L. Tenney, "*A Tutorial Introduction to Estelle*," Tech. Report 88-1, University of Massachusetts, Boston, Massachusetts, June 1988, pp. 02125-3393.

[47] T. Bolognesi and E. Brinksma, "*Introduction to the ISO Specification Language LOTOS*," **Computer Networks and ISDN Systems,** Vol. 14, 1987.

[48] G. Dickson and P.E. Chatzal, "*Status of CCITT Description Techniques and Application to Protocol Specification*," **Proc. of the IEEE**, Vol. 71, No. 12, Dec. 1983, pp. 1346-1355.

[49] R. Saracco and P.A.J. Tilanus, "*CCITT SDL: Overview of the Language and its Applications*," **Computer Networks and ISDN Systems**, Vol. 13, No. 1, 1987, pp. 65-74.

[50] R. Milner, "*A Calculus of Communicating Systems*," **Lecture Notes in Computer Science 92**, Springer Verlag, New York, 1980.