CHAPTER II
ARCHITECTURAL ASPECTS OF CONFORMANCE TESTING

1. Introduction

Having introduced the main concepts of communication protocols based on the standardized Basic Reference Model for Open Systems Interconnection (OSI-RM) [1] in Chapter I, we now discuss the test systems and the issues that test system designers and test laboratories face. The set of papers that we include in this chapter outlines the test methodology and framework of the International Organization for Standardization (ISO), and gives examples of real test systems built in the industry and research organizations. In fairness, we must point out that most of the researchers of the 1980’s based their theories on the results of their predecessors. Thus, the citations of other authors in the selected papers contain a wealth of theory and background information.

ISO defines the basic concepts related to conformance testing of OSI protocol implementations. Standardization efforts, which resulted in a 5-part standard [2], identify the issues regarding the definition and implementation of test suites in a test laboratory. The subjects that are covered in this standard include general concepts of conformance testing (Part 1), abstract test suite specification (Part 2), Tree and Tabular Combined Notation (TTCN) (Part 3), test realization (Part 4), and requirements on test laboratories and clients for conformance assessment process (Part 5). In Section 2 of this chapter, we highlight the fundamental definitions of the conformance testing standard including test methods, test suite organization, and test notation. We also discuss the minimum requirements of a test system architecture to implement a chosen test method.

Section 3 of this chapter discusses four papers. Conformance Testing for OSI Protocols by Linn introduces ISO’s work and provides a context for the following two papers which focus on different aspects of ISO’s work: OSI Conformance Testing by Rayner, and TTCN: Status and Assessment by Doubet. Various examples for TTCN is presented by Probert and Monkewich in TTCN: The International Notation for Specifying Tests of Communications Systems.

The papers included in Section 3 do not address the testing aspects of PDU encoding/decoding, but several of them make reference to ASN.1 (Abstract Syntax Notation One) [3][4], a notation to formally define the syntax of messages (PDUs). ASN.1 is a powerful technique to represent the encoding/decoding of the PDUs; however, it also adds another dimension to the complexity of protocol testing. In Section 4, we introduce the work of CCITT and ISO on ASN.1. The paper by Neufeld and Vuong titled An Overview of ASN.1 examines various issues related to ANSI.1 in more detail.

Section 5 presents the architectures of real test systems that are built to assess conformance of OSI and ISDN protocols. MHTS/X.400 – Testing Message Handling Systems by Austermuehl gives a perspective on upper-layer testing and the role of ASN.1 in describing the syntax of messages exchanged. Testing to Assure Interworking of Implementations of ISO/OSI Protocols by Linn describes mid-layer testing for Transport Layer protocols. Nightingale’s Application of the ISO Distributed Single Layer Testing Method to the Connectionless Network Protocol, describes testing of a connectionless protocol. The last paper in this cluster, Protocol Testing for the ISDN D-Channel Network Layer Protocol by Rathgeb et al., focuses on lower layer protocols. Note that the papers included in Section 5 give illustrative examples of test systems for connection and connectionless mode protocols, ranging from the Physical Layer to the Application Layer of the OSI-RM. By no means, does this set of papers represent a complete list of real test systems, nor does it represent or imply any endorsement by the authors, the National Institute of Standards and Technology (NIST), or AT&T Bell Laboratories.

In Section 6, the paper titled Progress in Conformance Testing Services? by Dwyer summarizes eight years of experience in protocol testing conducted by the National Computing Centre in the U.K. and reports some of the lessons learned and not learned (also see an earlier paper that he coauthored with Pavel and Dwyer [5]). Section 6 also includes a discussion about the opportunities missed during the design of the Transport Layer protocol (Class 4) and CLNP test systems at the National Bureau of
Standards (NBS), U.S.A. We hope that the readers can learn from the experiences described.

Section 7 presents issues regarding synchronization of the components of a distributed test environment. Synchronization in protocol conformance testing deals with coordinating the actions of various entities interacting with an implementation under test. Synchronization issues affect test cases, elements of a test system and design of protocol implementations. In Section 7, we discuss *Synchronization and Specification Issues in Protocol Testing* by Sarikaya and Bochmann, which gives definitions of synchronization concepts. We also discuss the approach taken by standards bodies towards synchronization issues and the challenges for test suite designers.

Sections 8 summarizes this chapter. Section 9 presents open research problems and references for further reading.

2. Background

In this section we briefly describe the basic concepts that are defined in the ISO standard on conformance testing titled *OSI Conformance Testing Framework and Methodology* [2]. For further information, the reader is referred to [8].

Recall from Chapter I that an (N)-entity interacts with adjacent upper and lower entities by (N)- and (N-1)-ASPs, respectively. These ASPs define the externally controllable and observable behavior of an (N)-entity. Also recall the black-box approach to protocol conformance testing considers only the externally visible behavior of an (N)-entity implementation. Therefore, when applying the terminology defined by ISO to black-box testing approach, conformance of an (N)-entity to its specification is determined by applying the ASPs that are defined as inputs and observing the ASPs that are defined as outputs at the (N)- and (N-1)-SAPs.

In a multi-layer product, where the (N)-entity is installed between the (N-1)- and (N+1)-entities, some of the ASPs may become inaccessible due to design restrictions. In such cases, some of the observability and/or controllability of an (N)-entity is lost. In an (N)-entity implementation, the points where the exchange of (N)-PDUs, (N)- and (N-1)-ASPs can be controlled and observed are called points of control and observation (PCOs).

The conformance testing methods that are defined by ISO are based on the availability of the PCOs in a given (N)-entity implementation under test (IUT) [2]. The system in which the IUT resides is referred to as the system under test (SUT). Note that the IUT is a component of the SUT.

We caution the reader about the use of word "test method" since the term has two distinct meanings in two different contexts. The papers in this chapter use "test method" to refer to design configurations of test systems (e.g., distributed test method), whereas, in Chapter III, a "test method" refers to a test generation technique (e.g., transition tour method).

At this point we also make a subtle, but important, distinction between a test method and test system architecture. The test methods that ISO defines are at least one level of abstraction higher than the architecture of a test system. In order to clarify the distinctions between a test method and test architecture, we outline the elements of a test architecture later in this section.

Part 2 of ISO’s conformance testing standard defines abstract test methods for test suite specifications. These methods are independent of the implementation details of a specific testbed and are defined based on the availability of ASPs at PCO. In ISO test methods, the lower tester is responsible for the control and observation of the (N-1)-ASPs at the PCO either below the IUT or at a remote site. The lower tester is the peer entity of the IUT. The upper tester controls and observes the (N)-ASPs at the upper service boundary of the IUT. For single-layer testing, each method is defined as follows (see Linn’s paper *Conformance Testing for OSI Protocols* included in this chapter for an in-depth study of these test methods):

*Local Testing Method*: The PCOs are defined at the upper and lower service boundaries of the IUT (Figure 2.1.a) assuming that (N)-PDUs, (N)- and (N-1)-ASPs can be controlled and observed by the upper and lower testers.
Distributed Testing Method: The IUT does not have a PCO at the lower service boundary, therefore, (N)-ASPs and (N)-PDUs are controlled and observed indirectly at the remote end of the (N-1)-Service Provider (Figure 2.1.b) allowing the upper and lower testers to reside in physically separate locations. The coordination between the upper and lower testers are provided by test coordination procedures.

Coordinated Testing Method: This method is an enhanced version of the distributed method. Here, the actions of the upper tester (i.e., control and observation of (N)-ASPs) are coordinated by a test management protocol. The test management protocol uses test management PDUs (TM-PDUs) for the coordination between the upper and lower testers (Figure 2.1.c). Note that, even though a PCO appears in Figure 2.1.c, it is optional; as a choice made by the protocol implementor, an upper tester may be integrated as part of the IUT. Figure 2.1.c depicts the logical demarcation of functions.

Remote Testing Method: The IUT does not have a PCO at the upper service boundary (Figure 2.1.d). Therefore, no explicit test coordination procedures or upper tester are employed in this method. The coordination between the upper and lower testers is either manual (e.g., talking over the telephone), implicit in the PDUs initiated by the lower tester, or provided by actions taken by an upper layer entity to stimulate the IUT. The dotted lines in Figure 2.1.d, denoting the absence of an upper tester and test coordination procedures, are given for comparison purpose.

Ferry Method†: The upper tester is replaced with an entity, called ferry control entity [9] which acts like a logical loop-back between the upper and lower testers (Figure 2.1.e). The test coordination procedures, upper and lower testers reside at the same site. The ferry control entity routes the (N)-ASPs between the upper tester and the upper service boundary of the IUT.

The above testing methods, with the exception of local method, are also referred to as the external test methods. Test laboratories can employ one or more of the above methods in their testbeds depending on the availability of PCOs in an IUT. The standard also defines distributed, coordinated, remote and local test methods for an SUT composed of multiple layers or an embedded layer (single layer within a multi-layer IUT).

In terms of error detection capabilities, the local testing method is the most effective among these methods since both the upper and lower service boundaries of the IUT are assumed to be observable and controllable. However, the local testing method is usually restricted to in-house testing by a manufacturer since the upper and lower service boundaries are typically not exposed once an implementation is incorporated in a product.

The distributed and coordinated methods have less error detection capabilities compared to the local method since the (N-1)-ASPs are observed and controlled at a remote site of the IUT, as opposed to the site where IUT resides. In addition, the distributed method requires an exposed upper service boundary at the IUT. In the coordinated method, interchange of the TM-PDUs places additional requirements, such as out-of-band communications‡ services, on the test system designers. We will revisit the implementation issues related to TM-PDUs in Section 7 of this chapter when we discuss synchronization issues.

The ferry method is an external version of the local method where the problem of synchronization between the upper and lower testers is eliminated since they both reside at the same site. However, this method requires an exposed upper service boundary, and usually assumes out-of-band services for the ferry protocol – a so-called ferry control channel.

The remote testing method is the most restrictive of the above methods since the (N)-ASPs at the upper service boundary of the IUT are not controllable nor observable. However, the remote testing method is the easiest method to implement since it does not place any external requirements on an IUT. For this reason, the remote method is the most widely used method for conformance assessment.

† Note that recently standards organizations have deferred any further development or use of the ferry method; but, we include the method here for the sake of completeness.
‡ Out-of-band refers to a communication channel which is unaffected by the traffic generated during testing and is usually assumed to provide reliable data exchange.
Standards organizations have defined four categories of tests to assess the conformance of an IUT to the relevant standard specification:

Figure 2.1. Various test methods defined by ISO: a) Local Method, b) Distributed Method, c) Coordinated Method, d) Remote Method, e) Ferry Method.
- **Basic Interconnection Tests** provide limited testing to assure that the IUT can perform basic functions before thorough testing is started; for example, connection establishment and termination, and simple data transfer.

- **Capability Tests** assure that the IUT can provide the observable capabilities based on the static conformance requirements which are the requirements describing the options, ranges of values for parameters and timers, etc.

- **Behavior Tests** assess the dynamic conformance requirements of an IUT which define the observable behavior of a protocol (and options). Behavior tests include tests for handling normal, expedited or special data types, timer expiration, sequence numbering and window rotation mechanisms, error handling cases, inopportune PDUs (PDUs that are valid in another state, but not valid in the current state because no action defined for them), and other functional aspects of a protocol. Behavior tests constitute the major portion of conformance tests. The formal test generation methods that will be discussed in Chapter III are mainly applicable to behavior tests.

- **Conformance Resolution Tests** provide definite diagnostic answers to specific requirements, such as previously identified situations that may cause incorrect behavior of an IUT. For example, they provide a yes/no type of answer for whether a particular feature (e.g., reset) is implemented in an IUT.

- **Other Test Types:** In addition to conformance testing, the following types of tests may be performed on an IUT depending on the application: performance tests to measure parameters such as maximum throughput that can be obtained, and stress tests to examine the behavior of an IUT under heavy load conditions.

Note that the last two categories of tests, as opposed to the first three, are not likely to be standardized for specific protocols, but, they may be offered by test laboratories as optional services.

A **test event** is an atomic interaction between the IUT and an upper or lower tester (including expiration of a timer).

**Test steps** may consist of several test events and are the building blocks of an abstract conformance test suite where test cases are represented in a hierarchical structure of test steps. The key element is called a **test case** which is composed of a set of test steps whose objective is based on a narrowly defined test purpose. A test purpose describes the objective of the corresponding test case as specifically as possible (e.g., the expected behavior of an IUT upon receipt of a PDU in a certain state). **Test groups** consist of several test cases related by a logical grouping of functions (e.g., receipt of inopportune PDUs in a given protocol state).

In an abstract test case, there may be three major components (steps): **test preamble**, **test body**, and **test postamble**. A preamble defines the necessary events to bring the IUT into the desired starting state to achieve the purpose of the test case. A test body defines the test events that are needed to achieve the test purpose. Any PDU exchanges used to verify the new state of the IUT before the test case completion are also part of the test body. A test postamble is used to put the IUT into a stable state after a test body is run. In order to achieve efficient run time, one may execute the preambles and postambles conditionally (i.e., only if a previous test fails) by concatenating the test bodies in the manner of tour. This issue is described in detail when we discuss the applicability of formal test generation techniques to the standards framework in Chapter III.

An abstract test notation, called Tree and Tabular Combined Notation (TTCN) is defined in Part 3 of the conformance testing standard and is used to specify abstract test suites in a standard notation. In TTCN, the behavior and actions of an IUT during the execution of the test cases (i.e., sending and receiving various PDUs defined by the test cases) are described as a tree of possible behaviors. Research and development are in progress to generate machine-executable versions of the abstract test suites that are written in TTCN.

The options and capabilities that are implemented in an IUT have to be known by a test laboratory before a conformance test is conducted in order to select the applicable test cases and eliminate the irrelevant
ones. This information is described in a protocol implementation conformance statement (PICS) and is described in Part 4 of the standard. Additionally, a test laboratory requires a protocol implementation extra information for testing (PIXIT) in order to match the testing environment to the capabilities of an IUT. (See Section 3 in Chapter I for a discussion about the roles of PICS and PIXIT in conformance testing.)

Various issues related to the requirements on test laboratories and the assessment of the results of a conformance test are described in Part 5 of the standard.

Before closing this section, we return to a point that was mentioned earlier: differences between a test method (as ISO defines it) and a test system architecture. A test method is independent of implementation considerations while a test system architecture is not. A test system designer must consider the following essential items when specifying the system architecture while a test method does not (see Figure 2.1):

- The operator’s interface to the system must be defined such that it is easy to operate, flexible enough to run various tests individually or as a group, and powerful enough to guide the operator how to resume testing when a test fails.
- Flexibility and performance of a test system are directly influenced by the test suite design structure (hierarchical or flat), storage medium, notation employed, a choice between an interpretive environment versus compiled test cases, and parameterization of test cases (i.e., binding values into the test suite such as timers and addresses for the SUT).
- A **conformance test log** records the events that take place during conformance testing. The structure and information contained in the conformance test log must include the stimuli sent to the IUT and observed responses in order to support verdicts rendered in real time, or support interpretation of stimuli and responses and generation of verdicts in non-real time.
- Attachments to the SUT:
  - Local testing methods assume a direct attachment to an SUT at the exposed interfaces.
  - External methods assume that an underlying communications service is used to attach to the product. Thus, a set of components of the SUT provides the communications services necessary to conduct testing. Note that the definition of an IUT is restricted to the components of the SUT in which the IUT resides.
- At lower layers, the test system architecture must address potential data loss, corruption, and misordering due to inherent quality of protocol services to support the communications between the SUT and the test system.
- Test coordination procedures employed to synchronize the actions of the SUT and the distributed components of the test system significantly enhance or limit the capabilities of the system in terms of time and labor required to conduct tests. Test coordination procedures may include an active element of the test harness above the IUT (upper tester), and a test management protocol to direct the actions of this element and report observations made above the protocol layer(s) being tested. (Synchronization issues are discussed further in Section 7 of this chapter.)
- The design philosophy of the active testing elements must consider implementation of tests which require error injection and recovery. In other words, tester should be able to inject errors (such as duplicated or corrupted PDUs) to test the error recovery mechanism of an IUT; similarly, the test system should handle erroneous behavior of an IUT. Active testing elements can be designed as:
  - an emulator of the protocol, but not constrained by the rules of the protocol so that errors may be injected to test error recovery specified within the protocol (e.g., an encoder/decoder);
  - a test case interpreter which is the user of a reference implementation of the protocol and (optionally) an exception generator which may be either
+ below the reference implementation;
+ embedded within the reference implementation – a so-called augmented reference entity;
+ a hybrid combination of the above.

In addition to the above there are many issues beyond the scope of this tutorial. However, further in this chapter, Section 6 "Lessons Learned" gives examples of the interplay between some of these choices. Section 9 of this chapter identifies references with further information about test system implementations.

3. The OSI Methodology and Framework

We discuss Linn’s Conformance Testing for OSI Protocols and Rayner’s OSI Conformance Testing together. There is an obvious overlap between the two papers, but they are complementary due to the differences in their focus, depth and breadth. Doubet’s TTCN: Status and Update is less technical, but complements both. Doubet’s paper is the most recent and Rayner’s is the oldest. Thus, where they differ, Doubet’s paper should be considered the most authoritative. For example, TTCN is now Part 3 of the standard, and it was an Annex to Part 2 when Rayner wrote his paper; Linn reports that operational semantics are being developed for TTCN while Doubet states that they are complete for those elements not deferred to a developing addendum. It is also worthwhile to note how the standardization process occurs for those who are not familiar with it. A standard progresses from a working document to a Draft Proposed Standard (DP), a Draft International Standard (DIS), and finally to an International Standard (IS). Each step is accompanied by a letter ballot and editing meetings to resolve negative ballot comments before circulation for the next ballot. Thus, Doubet’s paper reflects the character of the process and the progression of TTCN from DP to DIS status.

Linn’s paper opens with a characterization of the problems confronting the test system designer and experts working on the problem in ISO. It summarizes the research in the early 1980’s and identifies several demonstrations of OSI protocols which were based on early experience on testing.

Rayner opens with an explanation of the meaning of conformance within ISO and the types of testing. The details reflect the philosophy embedded in ISO’s work and articulate the philosophy for structuring test suites into the four groups described earlier in this chapter: Basic Interconnection, Capability, Behavior, and Conformance Resolution Tests. Organization of test cases usually reflects a strategic hierarchy, arranged from simplest to most complex. Note the statement from Dywer’s paper: "... protocol experts who operate the [NCC] testing service can ascertain whether an implementation is ‘good’ or ‘bad’ within the first sixteen tests.” This experience supports the practice of organizing a test suite into a set which serves as a "foundation" for more sophisticated and complex tests; the most sophisticated tests focus on error recovery and are usually executed last. ISO calls the foundation upon which other tests depend the Basic Interconnection Tests.

In Linn’s paper, a brief synopsis of TTCN, ISO’s test notation, is presented. For a more concrete description of TTCN, we include TTCN: the international notation for specifying tests of communications systems by Probert and Monkewich.

TTCN goes several steps further than most test languages with respect to the organization of test suites. TTCN permits the components of a test case to be organized in a hierarchical fashion so that fragments of TTCN text (e.g., tables to declare variables, PDU’s, default values, and named groups of test steps) may be combined into a test case, test groups, and ultimately the test suite. Rayner gives the motivation behind this hierarchical structuring, but he does not describe TTCN. In TTCN, the test suite is a tree structure reflecting a tree of possible behaviors. A TTCN test suite may (but need not) be organized based on subtrees of behavior (named collections of test steps) which may be aggregated into test cases sharing a common text. Test groups reflect a strategic organization of test cases. At the highest level, the test suite defines an overall testing strategy. However, ISO does not mandate execution of test cases in any particular order; selection of tests and execution is in the domain of test laboratories.

The attach construct in TTCN supports this hierarchical structuring and Part 3 of the standard gives
recursive rules for the proper inclusion of text to compose the actual components of a test case. We believe this aspect of TTCN to be unique. In effect, it allows the collection of components of a test suite into a library of behaviors – some for the valid behavior while others for error recovery mechanisms in an IUT. This structuring has the advantage of unnecessary replication of text in the test suite; thus, hierarchical inclusion of text may reduce test suite development and maintenance efforts. However, structuring the test suite into a library of behaviors also has some disadvantages, including:

- The test suite designer (maintainer) must remember the actions taken and verdicts rendered by behavior subtrees when incorporating the text into different test cases. However, this task is not trivial since the behavior descriptions are typically complex.
- The readers of a test suite has to often "flip" pages forwards and backwards through a document to discern the actual behavior due references to text which is printed in tables elsewhere;
- Since verdicts may be rendered on almost any test step, it may be difficult to group test steps so that a verdict rendered in one fragment matches the intent of a test case where it is employed. For example, a PDU called disconnect may be the expected behavior in one test and yields a pass verdict. But, in another situation, a disconnect PDU may indicate premature termination of a connection; because a test purpose was not satisfied, an inconclusive verdict should be rendered in this case.
- If the specification of a protocol changes, the verdicts associated with a named set of test steps may no longer be correct. Thus, the composition of test cases from a hierarchy of named test steps presents the potential for some unique problems due to changes propagating throughout a set of test cases.

Linn describes a subset of ISO’s test methods in greater detail than Rayner who elaborates more on the various combinations; therefore, the presentations complement each other. In practice, the single-layer remote test method is one of the most commonly employed methods. ISO has a difficult task of trying to describe incremental single-layer embedded tests for the Session through FTAM layer protocols because of the number of protocols involved (FTAM, ACSE, Presentation and Session). The number of protocols involved, the number of implementation options allowed, and the possibility of errors in any of the four protocols make the task of structuring the test suite and isolating the source of errors extremely difficult. Those laboratories offering OSI Testing Services, such as the National Computing Centre (NCC) in the U.K. and Corporation for Open Systems (COS) in the U.S.A, have only recently announced products or realized significant progress on systems which test the Session protocol embedded under FTAM and MHS.

Rayner briefly introduces ISO’s multi-layer test methods (i.e., simultaneously testing several layer implementations in an SUT). Although Rayner enumerates the various schemes possible, there is very limited practical experience with any of them. This topic is currently an open research problem and is discussed further in Section 9 of this chapter.

Rayner describes real testers and the execution of test suites (both of which are out of the domain of ISO’s authority) and outlines a number of options available to the designer of a test system. Note that, in this description, Rayner uses the terms of test responder and test driver instead of the terms of upper and lower tester, respectively. Rayner devotes Section 10 of his paper to executable test suite derivation. Since a figure comparable to Figure 21 of the paper was controversial within ISO, we note:

1) the notion of generic test suites has been abandoned by ISO (see Doubet’s paper for the reasoning);

2) there are many paths from the box labeled as "Abstract Test Suite" to the box labeled as "Parameterized Executable Test Suite." Intervening paths and boxes are conceptual, and may or may not exist in a real environment/test system. The alternate paths simply identify alternate ways to achieve the same goal – an executable test suite.

Doubet’s paper identifies some present and planned features of TTCN, which were not discussed by either Linn or Rayner, and amplifies some of the ideas presented above. It mentions ASN.1 and its
application to testing. Since her paper was written, ASN.1 value notation has been incorporated into a TTCN test suite to describe PDUs and their components for the Network Layer testing of the ISDN D-channel. The significance of this fact is that ASN.1 has not yet been widely recognized as a powerful tool for the description of PDUs and complex data values in a test suite. In the case of ISDN testing, the underlying problem is that there are many variant forms of PDUs and no easy way to express the values of interest using other techniques. For this reason, we give a brief overview of ASN.1 in the next section.

4. An Overview of ASN.1

Thus far, the majority of discussion has been on test methods which focus on the dynamic aspects of testing, not on how the PDUs are encoded and decoded. Coding is considered a static aspect of conformance because the syntax and rules for encoding PDUs (messages) are defined and do not influence the temporal relationships of the PDUs. However, coding is an important factor of conformance evaluation since syntactically and erroneously encoded PDUs lead to fail verdicts.

With the introduction of X.409 [7] within CCITT and its subsequent adoption by ISO as ASN.1, a grammar-based notation for the description of the syntax of PDUs (messages) became an international standard. CCITT applied X.409 to the description of Message Handling Systems (MHS)[6] and ISO applied ASN.1 within Application and Presentation Layers. The standard for ASN.1 is in two parts [3][4]. The first part specifies the notation for defining data types and values of the types. The second part specifies the Basic Encoding Rules which define encoding of values as an octet string. ASN.1 is a nonprocedural language, although encoding and decoding procedures are implicit in the standard. ASN.1 is the first activity within the international standards bodies which treats messages exchanged between protocol entities as a language. A collection of type definitions in ASN.1 are called a module and is expressed in a set of grammar rules which employ an extended BNF (Bacus Naur Form) notation to define relationships between elements of a message (PDU). Also, the notation for expressing values for the types defined in a module has an underlying grammar. Thus, formal language-oriented theory and tools may be employed to process either the type definitions or the value notation, with the exception of ASN.1 macros. (An in-depth discussion of ASN.1 macros is outside the scope of this tutorial; see, for example, Gaudette [10] for a tutorial on ASN.1.)

The power of ASN.1 is derived from its ability to define a set of data types using grammar rules. A set of types defines an abstract syntax. Specifically, it defines the abstract structure of a set of values contained in a PDU; i.e., an abstract syntax defines both the structure and the domains of the values embedded within the structure of a PDU. But, it does not define the encoding of a PDU. Given an abstract syntax, the Basic Encoding Rules implicitly define a corresponding transfer syntax; i.e., a representation of any value as an octet string that is machine-independent, self-delimiting, and mechanically encodable and decodable. Such a transfer syntax is encoded as nested triples (tag, length and value) that form a linear representation of the tree structure of an abstract value. A notation for values is also defined by the standard (although implicitly). For simplicity, we call this a print value (because, among other things, they may be printed and readily interpreted by a person); the standard calls them "values in the defined notation." Print values are most often used to express examples of typical values for a given type in a non-encoded form (e.g., the components of an electronic mail address, or the header of a PDU). Also, the implementation of a protocol will have its own representation of values in some local data structure, but this is not subject to standardization. These local values are encoded into a PDU whose structure is defined by the abstract syntax, but after encoding the PDU is an octet string which conforms to the transfer syntax; when an encoded PDU is received, it is decoded and stored as a set of local values within the data structures defined by the implementor. Finally, to define a consistent set of terms, when a set of values corresponding to a abstract syntax are stored on a peripheral device (e.g., disk), we refer to them as a set of filed values.

Type definitions in ASN.1 are based on simple types that refer to well-known sets of values and structured types that construct new types from related groups of other types. The type definitions together represent a kind of grammar, which defines a language whose sentential forms are values. The type definitions for a typical protocol specify a language of values, i.e., a set of PDUs, that is enormously large and variable. Furthermore, the encoding rules allow a significant range of alternate encoding strategies for any particular value.
How can ASN.1 improve conformance testing? We know from language theory [11],[12] that grammars and grammar-based tools can be used to analyze any valid sentential form and can also be used to systematically generate some of the important sentential forms [13],[14]. Hand-written encoders and decoders, while valuable for many purposes, are less likely to handle all of the variants that are possible within the rules of the grammar. We also know that it is important to generate and test PDU decoders for obscure but feasible variants of encoded PDUs.

One way to create test cases is to use an ordinary text editor to compose a set of print values which correspond to ASN.1 types. In a second step, transform the test cases with a tool, which translates them from a human-readable form to an internal machine-dependent form, and save them on a disk (print value to filed value). This step insures that the test cases are syntactically correct and comply with all the relationships defined in the grammar (type definitions). In a third step, filed values can be encoded (transformed into transfer values) and either saved or presented to a decoder. This is essentially the technique described by Austermuehl in the next section. They create test cases with a syntax directed editor which is guided by a collection of ASN.1 type definitions. After creating print values of messages they either send them interactively, or save them on a disk. Reverse transformations are also possible and the encoded values may be decoded and displayed as print values. NIST has employed similar transformations to the testing of a gateway for electronic mail [15] and an FTAM system (see the first paper by Linn included at the end of this chapter), but without the aid of a syntax directed editor.

Note that, given an ASN.1 (X.409) module definition (a collection of type definitions), both semantically valid and invalid PDUs may be expressed in an ASN.1-based test notation, because syntactically valid PDUs are not constrained by the semantics of the protocol. Note that, here, syntax refers to the permissible format or structure of PDUs, not the contents of specific fields. Semantic constraints, on the other hand, define permissible values of fields. For example, values allowed by the syntax definition may be out of range in the protocol definition; therefore, it is possible to specify and transmit a semantically invalid PDU to test an error recovery mechanism.

Having introduced the positive aspects of using ASN.1 in conformance test systems, let us now consider its disadvantages. ASN.1 presents some unique issues with respect to coding problems for which currently there are no simple answers. We give the following example to illustrate the magnitude of the coding problem with ASN.1. The transfer syntax of an ASN.1 value is encoded as a triple: < tag, length, value >, where

- *tag* is an encoded ASN.1 type identifier;
- *length* is an octet string varying in length from one to 127 octets long and may be in three forms as indicated by the first octet:
  (1) indefinite length meaning a value is delimited by a marker,
  (2) short definite length meaning a one-octet length field prefixes the value and contains an octet count,
  (3) long definite length meaning a variable-length octet count prefixes the value;
- *value* is a nested triple of the same form (i.e., this structure is applied recursively to encode a tree of values).

Given *n* ASN.1 type definitions which do not refer to each other recursively, the number of possible combinations of length encodings of *just the length parts* for *n* values is at least \((n!) \times (2^{8 \times 126})\). To this large number, add the possibilities of:

- optional and default values which may be associated with type definitions;
- reordering of elements within certain types;
- recursive application of types;

II.10
- application-specific contexts which restrict the alphabets of values to specific subdomains.

It simply is not possible to completely test domains of this size. Thus, the old problem of selecting tests via some heuristics or human intuition remains.

We include An overview of ASN.1 by Neufeld and Vuong for a more detailed presentation of ASN.1.

We close this section with a note on fault models for ASN.1. If decoders are to be robust, then they must be relatively intelligent. Specifically, they must have access to data describing the syntax of the object that is to be decoded: for example, an invalid tag, a missing subtree that is required, or an invalid length renders a PDU undecodable. Our experience indicates that in circumstances when the key protocol control information (PCI) is corrupted or missing, the behavior of a protocol entity is unpredictable. Without information regarding the syntax of a PDU, there is no way for a decoder to even detect and report an error. Additionally, it is possible to receive an object too large to store locally; e.g., a 96-bit integer. Current standards say nothing about alternate forms of length encoding or size limits for many objects. Thus, decoders must be able to cope with various forms of length, and must at least report failure when an object is either undecodable or exceeds locally defined limits of storage. Such situations may easily cause protocol implementations crash.

5. Real-World Test Systems

In this section, we describe a selected set of tester systems that have been implemented and compare their characteristics to the principles of ISO. We note that reference to specific test systems and companies is for illustrative purposes only†.

Since some of the papers selected describe precursors to current testing products, we also indicate current status of each system (if it is known to the authors). Each real tester is described using the outline below which gives an explanation of the components before it is applied.

Overview of the protocol and/or paper followed by:

- Test Method: ISO’s method employed;
- Test Language: Basis for the test language;
- Status: Evolution of status (if known);
- Exception Generation Capability: Yes/No/Comments;
- Vary PDU Format: Yes/No/Comments;
- Vary PDU Encoding: Yes/No/Comments;
- Mode of operation: Manual or Semi-automatic via test scripts, and manual or automated interpretation of results.
- General Comments:

We selected the papers describing the systems that have been built to test various layers of the OSI-RM:

- Application Layer test system for Message Handling Systems (MHS – electronic mail),
- Transport Layer (Class 4) test system,
- Network Layer test system for the Protocol for Providing the Connectionless Network Service (CLNP),
- Network through Physical Layers test system for Integrated Services Digital Network (ISDN – the next generation of telephone switching systems).

Among the protocols that are mentioned above, MHS and CLNP are connectionless mode, and the others are connection oriented. (See Chapter I for the definitions of the classification of protocols.)

† Such identification does not constitute product endorsement by the National Institute of Standards and Technology, AT&T Bell Laboratories, or the authors.
5.1. Message Handling Systems (MHS), CCITT X.400 Series

*MH TS/X.400 – Testing Message Handling Systems* by Austermuehl describes a test system for CCITT’s Application Layer protocols for electronic mail [6]. To understand the test system, one must have insight to the components and functions of MHS. MHS is specified as a User Agent (UA) which is a supplier provided Application Layer entity (program). The UA acts on behalf of the user as an interface to the MHS services. The UA is not a finite state machine based protocol entity. Instead, it is defined as a set of ASPs (message: submission request, delivery indication, and delivery/nondelivery notification for messages submitted), plus a set of X.409 (ASN.1) syntax descriptions of an inter-personal message (body of the mail). This ASN.1 module is called P2.

Next in the MHS hierarchy is a component, is called a Message Transfer Agent (MTA), which accepts messages and wraps them in an envelope (another ASN.1 module known as P1). MTAs are also not based on a finite state machine model of a protocol entity. An MTA entity is responsible for interaction with the UA, and the submission, delivery and relaying of messages (i.e., MHS is a store-and-forward system where intermediate MTA entities receive messages and relay them on toward their final destination). The user may request an optional delivery notification in which an MTA indicates success or failure to deliver the message.

The third entity in the MHS hierarchy is called a Reliable Transfer Service (RTS) entity. From the user’s perspective, the UA/MTA entities operate in a connectionless mode. However, the RTS entity uses a subset of the Session Layer services in a well-defined manner to establish connections and transmit a message to the nearest neighboring MTA enroute to the final destination. Thus, the RTS simply maps the MTA service requests into sequences of Session Layer services. A few other concepts are used in the paper. Management domains are addressing domains and may be: (1) "public administrations" – so-called “administrative management domains” (ADMDs) which are the public switched networks employing the worldwide telephone numbering system as addresses; or (2) “private management domains” (PRMDs) which may be any private enterprise operating a network and able to assign addresses. PRMDs may be attached to an ADMD (a public data network) or another PRMD (private data network).

Since the MHS specification is not based on a finite state machine model, testing focuses on encoding/decoding issues and relaying (routing) messages. The active elements in the test system are based on an encoder/decoder design.

The characteristics of the test system described by Austermuehl are:

- Test Method: Single-layer incremental spanning UA, MTA, and RTS sublayers.
- Test Language: ASN.1 print values.
- Status: Precursor to a product offered by Danet GmbH.
- Exception Generation Capability: Limited – the ASN.1 syntax descriptions do not preclude the user from specifying invalid values (e.g., out of range) in a test case. However, reliable transmission of messages is assumed, therefore, exception generation is not a significant issue.
- Vary PDU Format: Yes, the system is parameterizable to allow for varying profiles. Thus, to some degree, this facility could be used to vary formats.
- Vary PDU Encoding: Unknown. As noted earlier, coding is a significant issue in testing systems whose syntax is based on ASN.1. However, it is also an open topic of research and development.
- Mode of operation: Both interactive and semi-automatic via stored test cases; interpretation of results is manual.
- Comments: A syntax directed editor (based on the ASN.1 descriptions of messages) is used to compose test cases. Thus, the test language is a set of print values composed interactively and either transmitted in real-time or saved on a disk. Messages received are transformed from their transfer syntax (encoded form) to print values for manual inspection and interpretation of results. The system described was developed during the early period of ISO’s efforts to standardize OSI.
test methods and does not reflect ISO’s current work. MHS testing is focused on the Protocol Control Information (PCI), coding of PDUs (the data domain of a protocol), and relaying information between intermediate MTA entities. Since MHS is not a state based specification, this system represents the current state-of-the-art for testing the data domain of a non-state based protocol. Specifically, intuition of the test designer guides the creation, selection and maintenance of the test suite.

Note also that CCITT has specified test cases in TTCN for MHS testing. Thus, the system described in this paper, does not reflect subsequent work done within CCITT which employs TTCN nor changes by Danet in its commercial product.

5.2. Class 4 Transport

Testing to Assure Interworking of OSI Protocols by Linn describes a test system for ISO’s Transport, Class 4, protocol (TP4). It is the product of early research and does not reflect the work of ISO. To appreciate the problems of testing TP4, one must realize that TP4 was designed to provide reliable, end-to-end, data transport services in the face of potential loss, duplication, corruption, and misordering of PDUs by the underlying Network service. Specifically, the protocol was designed to operate over connectionless LANs and subnetworks concatenated with the CLNP protocol which does not assure data delivery (see the next subsection). Thus, a necessary element in the test harness is an exception generator. Specifically, this is a means to create corrupted PDUs, and simulate lost, duplicated and misordered PDUs on demand because these types of errors occur infrequently and unpredictably in real-life networks.

The system employs a reference implementation of the TP4 protocol which was derived from a formal description of the protocol by translating the formal description, and employs an exception generator. The reference implementation and exception generator are under the control of a scenario interpreter which reads and interprets test cases. The "lower tester" functionality is composed of this trio: scenario interpreter, reference implementation and exception generation. Upper tester functionality is achieved using a second scenario interpreter residing above the IUT.

Characteristics of the test system are:

- Test Method: Distributed Single layer.
- Test Language: Based on Transport ASPs.
- Status: Precursor to testing products marketed by the NCC and COS.
- Exception Generation Capability: Yes, extensive.
- Vary PDU Format: No, except through editing capabilities of the exception generator.
- Vary PDU Encoding: No, except through editing capabilities of the exception generator.
- Mode of operation: Semi-automatic via test cases and manual interpretation of results.

- General Comments: A self-imposed design constraint was no "upper tester" (scenario interpreter) could be required of the supplier of an IUT – but was optional. This implies a remote test method by default with a distributed mode as an option. Additionally, it was assumed that the IUT was not reliable enough to convey test coordination procedures between the test lab and the client site. Thus, no test management protocol could be employed to synchronize the system. In retrospect, these constraints were unnecessary. At this point in time, it is generally accepted that products will provide an exposed Transport interface.

The kernel set of functions in the pair of scenario interpreters are the same. Both could execute nested test cases when the distributed mode of testing was employed. However, this lead to synchronization problems when the IUT failed a test. Thus, operation was semi-automated with operators manually selecting tests to be executed – after interpreting the results of the previous test with the aid of a tool to display the conformance log.

The NCC in the U.K. enhanced the system by replacing both scenario interpreters with their own
"test driver" and "responder" which employ a test management protocol and serve as integral components of the lower and upper testers. Thus, the lower tester components include the test driver, TP4 reference entity, and exception generator. The test responder realizes the upper tester functions. Test cases are exchanged in real-time by using a test management protocol; a test driver at the test center directs a slave test responder which resides above the IUT and interprets test directives sent to it by the test driver. This master/slave mode of operation significantly reduced synchronization problems, but interpretation of results is still a manual operation aided by tools which interpret the conformance log.

5.3. Connectionless Network Protocol (CLNP)

Application of the ISO Distributed Single Layer Testing Method to the Connectionless Network Protocol by Nightingale describes an early effort to apply the work of ISO. The CLNP is designed to concatenate subnetworks and relay (route) PDUs destined to end-systems located on different subnetworks. It operates in a connectionless mode, segmenting \((N+1)\)-PDUs that are larger than those carried in a subnetwork; i.e., breaks Transport PDUs into sequentially transmitted segments which fit with the maximum size allowed in a subnetwork. PDUs may be discarded if they are corrupted (checksum failure), transmission time (hop count) is exceeded, or a switching node runs out of buffers. Thus, error recovery by retransmission is a function of the TP4.

Characteristics of the test system include:

- Test Method: Coordinated Single layer.
- Test Language: Based on CLN-PDU syntax, and includes test directives to encode PDUs from named PDU templates stored on a disk, and direct actions of the Upper Tester.
- Status: Precursor to a product marketed by COS.
- Exception Generation Capability: Yes, any field can be corrupted.
- Vary PDU Format: Yes, named PDU templates with named PDU parameters/fields are employed to direct the encoding process.
- Vary PDU Encoding: Yes, but limited.
- Mode of operation: Semi-automatic via test cases and semi-automatic interpretation of results.
- General Comments: A unique feature is that default values for fields of PDUs can be associated with a PDU template. Thus, repetitious detail can be omitted and individual test cases focus on fields and parameters which are of specific interest within a test. Additionally, the template defines the sequence for encoding a PDU. Thus, varied formats of PDUs are easily expressed by changing templates. Design goals did not include embedded testing under TP4. As Nightingale notes in Section 5 of his paper, this precluded its use in several instances because the product suppliers did not provide exposed interfaces above their implementations of CLNP. Subsequently, COS integrated the TP4 and CLNP test systems into a distributed single-layer, embedded test system with incremental testing. Specifically, the CLNP was installed below the TP4 reference entity (or Exception Generator) in the test harness for TP4. Further discussion of the lessons learned from this experience is deferred to a later subsection.

5.4. Integrated Services Digital Network (ISDN)

ISDN is a set of specifications for the Network through Physical Layers of the next generation of telephone systems (currently being deployed). It integrates transmission of voice, data, and video. The basic services offered include two B-channels (so-called bearer channels) which provide a pair of 64 K-bit/sec. transparent channels to convey data, plus one 16 K-bit/sec. D-channel for signaling and control. The D-channel protocol has its roots in CCITT Recommendation X.25.

Protocol Testing for the ISDN D-Channel Network Layer by Rathgeb et al. presents a test system which is best described as a hybrid. The authors were aware of the work of ISO and express their opinions of the strengths and weaknesses of reference implementations and encoders/decoders (see Section 3.2 of the
paper). They chose to employ reference implementations for layers 3 through 1 (see Section 5.2 of the paper), plus "virtual users” which interpret test cases for each layer.

Characteristics of the test system are:

- Test Method: Distributed, Single-layer Embedded with incremental testing spanning layers 1 through 3.
- Test Language(s): Based on structure of PDUs for each layer.
- Status: Unknown.
- Exception Generation Capability: Implied by the authors’ stated need for robustness tests (which is outside the scope of protocol testing as defined by ISO).
- Vary PDU Format: Unknown.
- Vary PDU Encoding: Unknown.
- Mode of operation: Semi-automatic via test cases; and semi-automatic interpretation of results aided by verdicts embedded in test cases.
- General Comments: The system has some unique features including reference implementations for several layers; each may serve as an (N-I)-service provider. Thus, a test case need only focus on one layer at a time. But, focus of testing may be shifted between layers at will. This differs from ISO’s philosophy of defining monolithic test cases spanning several layers. However, it is not clear that a reference entity may reside above the layer under test and serve as a driver or supplier of data (as does COS’s embedded CLNP test system which is driven by a TP4 reference entity).

Note that more recent test suites for ISDN implementations are being specified in TTCN by an organization called the North American ISDN Users’ Forum. Also, because of the complexity and diversity of Network Layer PDUs, it has been proposed to describe test suites using ASN.1 print values for PDU descriptions — even though the syntax of the PDUs is not specified in ASN.1. Thus, this is another example where the complexity of the PDUs is such that a grammar based language for syntax description (ASN.1) is a significant asset in dealing with a large number of options and formats of PDUs.

The implications of these facts are obvious to those at least vaguely familiar with language theory. Machine-to-machine dialogue comprises a language in theory and practice. Compiler theorists recognized years ago that finite state machines were an appropriate model to define the temporal relationships between elements of a language; i.e., finite state machines are an excellent means to recognize the elements of a language. Furthermore, regular expressions and grammars are the tools of defining languages. Hence, the grammar based rules of an ASN.1 specification allows the expression of values (both print and encoded values) of the messages comprising the language and their recognition (decoding). These same powerful theoretical, but also practical, tools can be employed in testing. For those interested in this aspect, we include some useful references: [11]-[14],[16]-[19].

6. Experiences and Lessons Learned

In an unabashed and satirical style, Dwyer critiques the work of ISO and others with a broad brush in the paper entitled Progress in Conformance Testing? His paper complements the views presented by Davidson and the concerns expressed by Rayner (see the papers selected for Chapter I). His conclusions, which are based on experience, pragmatism, and discipline, reinforce the need for conformance testing. To say more would only dilute the messages he delivers.

At this point, we include several lessons which can be passed on from the early experiences at NBS. We share them so that others may benefit from our lack of vision when designing the TP4 and CLNP test systems described in the previous subsections.

Initially, an interactive system was employed at NBS for testing TP4 implementations. However, interactive testing of TP4 was too slow; we could not drive the test system fast enough to initiate specific situations required for testing. A new design included the scenario interpreter, reference implementation,
and exception generator/logging as three modules which provided the functions of a lower tester. Since we did not want to place the requirement of a Scenario Interpreter (upper tester) on the client, we allowed an operator to drive the IUT interactively. Therefore, we chose the solution of using Scenario Interpreters without a test management protocol (TMP). In other words, Scenario Interpreters were substitutes for "very fast operators." Exception generation (i.e., sending unexpected and/or invalid PDUs to an IUT) and logging (i.e., recording the PDU exchanges during conformance testing) were essential at the test center. They were incorporated as a single optional process. However, exception generation and logging should have been designed as separate processes since logging is not an option while exception generation is.

The second lesson from the TP4 test system is that a test management protocol was an essential feature to coordinate the test entities. (In practice, no client chose to operate in a manually-controlled remote mode of testing: it is too slow, and all clients implemented a scenario interpreter.) Multiplexing, which interleaves the streams of data requests from more than one transport protocol user, was addressed by just starting another test case. Actually, test cases could be nested. Thus, one test case could start several, one after the other (execution could be interleaved for multiplexing tests). However, this turned out to be a synchronization issue in itself. If unanticipated behavior occurred within one execution stream, the peer scenario interpreter continued without any knowledge of the problem its peer had encountered due to the absence of a test management protocol (TMP). Recall from Section 5.2 of this chapter that the NCC added a TMP to the system which solved a number of synchronization issues related to asynchronous execution of Scenario Interpreters. However, a few synchronization issues remained with the Exception Generator (a separate process running in parallel).

Another lesson is that single-layer test entities should be designed to be integrated into a multi-layer test entity. As indicated earlier, some CLNP implementations could not be tested with the system described earlier because of assumptions about an exposed interface at the TP4/CLNP boundary. COS integrated the TP4 and CLNP test systems. At NIST, this lesson was applied to the multi-layer FTAM test system. The FTAM test system also integrates the exception generator function into each layer (see Section 7.3 of the first paper by Linn selected for this chapter and Section IX of [8].) Specifically, a layer identifier is included in each test directive. Thus, it is easy to direct any entity in the stack to take an action which precipitates an error recovery mechanism in the IUT. This approach specifically avoids synchronization issues associated with an outboard exception generator executing as an independent process.

Finally, the FTAM test system runs in either a remote mode of testing, or a coordinated mode. In the latter case, test cases are exchanged by sending files, and reports are returned in the same manner; files used to exchange test cases and reports have reserved file names. All these design decisions were deliberate actions based on experience gained with the TP4 and CLNP test systems.

In an electronic mail gateway test system (MHS/SMTP) designed by NIST [15], we used the mail system to coordinate and control an upper tester. The TMP is embedded in the body of mail messages exchanged. Messages are parsed to extract the TMP which is coded as ASCII text. This "in-band" coordination is possible if the test case does not change the state of the IUT during a test. Specifically, in the case of electronic mail, each message is independent, so it is no problem. In the case of FTAM, files are exchanged before and after the test. However, in both cases the IUT must be capable of demonstrating minimal function first. In the case of FTAM, testing starts in a remote mode with the lower tester as the initiator (thus a master). Minimal function is demonstrated by transferring a file to the IUT, reading the file which was transferred, and comparing it to the original.

In summary, the architecture of a test system and a range of implementation choices either constrain or enhance the ability of its users to achieve testing goals. Poor architectural and implementation choices may preclude testing certain protocol functions – particularly when timing and synchronization are important factors. This aspect is particularly true when real-time constraints are vital; e.g., when testing error recovery mechanisms, timers, and multiplexing.

Compiled test cases offer speed of execution, but limited flexibility. Interpreted test cases offer flexibility, but execute slower; and interpreted interactive testing is the most flexible, but the slowest mode. However, the performance of an interpretive environment may be enhanced so that interpretation
is not a bottle-neck while maintaining a high degree of flexibility. In an interpretative environment, a test case may be read as a whole with lexical analysis and parsing done before execution of the test commences. Specifically, the input text is reduced to a stream of tokens to be interpreted before execution commences. Thus, overhead associated with interpretation is significantly reduced during execution. But, the test system operator can adjust and create new tests by tailoring an old test case as dictated by circumstances. In our judgment, this is an optimal compromise between speed of execution and flexibility.

7. Synchronization Issues

Synchronization is one of the key issues in test system design and implementation. When there is more than one entity, such as a lower and an upper testers, interacting with an IUT there is a potential problem that incorrect verdicts may result during conformance testing if the actions of these entities are not properly coordinated (i.e., synchronized). Let us consider a simple example where a distributed test method is implemented; i.e., a test suite controls both lower and upper testers. Suppose that, during a test, the lower tester exchanges $x$ number of PDUs with the IUT and the upper tester is not involved in these exchanges (i.e., neither sends or receives any ASPs to/from the IUT). After the $x$th PDU exchange is completed between the lower tester and the IUT, suppose the upper tester needs to send an ASP to the IUT. At this point, without a synchronization mechanism between the upper and lower testers, there is no way for the upper tester to know when to send the ASP to the IUT. In general, in an interaction between an entity $A$ and an IUT such that the second entity $B$ does not participate in this interaction, a synchronization problem occurs when entity $B$ needs to interact with the IUT. The only way to stimulate entity $B$ to take some action is for entity $A$ to send a message directing it to take an action (because entity $B$ has no information about what has transpired). ISO provides a means to synchronize the upper and lower testers using the coordinated test method.

Sarikaya and Bochmann discuss these synchronization issues in *Synchronization and Specification Issues in Protocol Testing*. The lesson to be learned is simple: synchronization issues must be considered as part of the architectural constraints and design of test systems. The authors formally describe what they call *intrinsically unsynchronizable* transitions or states (transient states) in the context of generating test sequences. They identify specific examples in the X.25 call setup procedures and identify procedures to eliminate these unsynchronizable situations from test sequences. The authors also give a brief summary of the various conformance test sequence generation techniques which are discussed in detail in Chapter III.

Synchronization problems may lead to inconclusive test verdicts, or to fail verdicts for correct implementations (and vice versa). ISO and CCITT address some of the synchronization issues in conformance testing by introducing test coordination procedures and management protocols in the distributed and coordinated test methods. Test management PDUs may be exchanged between the upper and lower testers either via a separate medium (called *out-of-band* services) or through the SUT (called *in-band* services); each alternative affects the design of test cases, test suite elements, test system architectures and the IUT.

Thus far, discussion of synchronization issues has focused on the upper and lower tester, and test coordination achieved by test management protocols (or lack thereof). However, there are other issues the test system-suite designer must consider.

First, overzealous efforts to achieve tight synchronization of upper and lower testers can reduce a full-duplex protocol to operating in a half-duplex mode (i.e., there are no simultaneous exchange of PDUs).

Second, there are issues related to the protocol itself. It is impossible to observe the actions of protocol entities if the protocol specification contains transient states. Specifically, this means the IUT takes some action with no stimuli from either the upper or lower tester. Examples include actions in response expiration of timers and spontaneous transitions (see Chapter I for the formal definitions). Additionally, receipt of either corrupted or inopportune PDUs may lead to transient states (i.e., an input arrives and is unrecognizable or in the wrong sequence). The IUT may discard the input with no state change or transition to an intermediate state and produce no output. In either case, the action may go unrecognized by the components of the test system. Or, the IUT may produce an output which is not specified in a test
case. In this event, the best outcome is that a fail verdict is produced, and upon investigation, the test is rerun. In the worst case, the test system may deadlock, and if no record of exchanged PDUs is kept, the situation precipitating deadlock often remains a mystery!

Other issues to consider include the following.

- Because the IUT and lower tester are separated physically (by distance) and consequently in time (by transmission delays), the tester may observe events in a temporal sequence that is unanticipated. We call this effect time skew. Time skew can lead to synchronization problems if misordered observations are unanticipated.

- Time skew can also lead to synchronization errors if control is maintained by tokens. If both the IUT and tester attempt to exchange the same token (carried in PDUs) and the PDUs pass each other traveling in opposite directions in the underlying medium, the IUT may behave in unanticipated ways [19],[20].

- The test system (test case) must be able to anticipate and accommodate any behavior by the IUT — including deadlock — and not fail (or deadlock itself). Timers in the test system can solve the deadlock problem by aborting the test when the timer expires.

- Mismatched processor speeds or timers can lead to unanticipated behaviors. (The test system usually must be as fast (or faster) than the system being tested.)

- Test suites should avoid tests involving transient states, at least during initial tests. Transient states may be involved in error recovery mechanisms; for example, retransmission of a PDU due to failure by the lower tester to acknowledge a PDU transmitted by the IUT (simulation of data loss).

8. Summary

In this chapter, we present the conformance testing methodologies and the framework as defined by the international standards bodies. Examples of real test systems are given, each of which is analyzed in terms of the standards framework. Although we cannot possibly cover all real test systems existing in industry and research institutions, we give an example for most OSI-Reference Model layers, and one for the connectionless mode protocols. ASN.1, which is a standard notation to formally define the syntax of PDUs, is analyzed for its benefits and disadvantages in protocol conformance testing. Finally, we share experiences, reported in the literature and those of the authors, hoping that the readers may learn from restrictive architectural design decisions and the lack of vision of others.

Among the definitions and the framework provided by the standards organizations for protocol conformance testing, there is no mention of how to generate conformance tests. The next chapter presents formal methodologies to generate conformance tests and identifies some open research problems.

Following is the list of full papers that we include for your review at the end of this chapter (in the order that they appear):


9. Open Problems and Further Reading:

Multi-layer testing is among several crucial subjects that need further investigation. In Section 7.2 of Conformance Testing for OSI Protocols, Linn reports some progress on multi-layer testing, and another paper [8] gives a complementary overview. ISO’s work on multi-layer test methods is not based on significant experience. Section 8.4 of Rayner’s paper entitled OSI Conformance Testing enumerates possibilities – but there is no known practical experience to back it up. Furthermore, for all practical purposes, the current definition of TTCN precludes specifying multi-layer tests except as a monolithic test case because the number of points of control and observation in a test case is limited to exactly one per lower tester (i.e., a multi-layer test case cannot be composed of a "stack of test cases" driving each other). Due to pressures from other OSI protocol Working Groups, initial steps have been taken to address the architectural requirements for multi-party, and multi-peer test methods. In these test methods several distributed, peer protocol entities are responsible for different aspects of an action (e.g., the Transaction Processing Protocol). Note that this is a step beyond end-to-end peer testing in a connection-oriented environment; it includes routing protocols in the Network Layer which use the connectionless mode of communications.
The current extensions proposed for the description of parallel execution of trees of behavior described in
TTCN require the definition of a framework for concurrent execution and synchronization of "multiple
lower testers." The extensions would allow parallel execution of test cases which could be combined
horizontally or vertically and synchronized by a "master element." Some of the participants in ISO
believe that each element in this matrix of test entities may have to "connect" or communicate with every
other element in which case synchronization will be a significant issue. At this point, it is premature to
report additional detail in this subject.

In Section 7 of this chapter, we presented some of the important problems that designers of test systems,
test cases and product suppliers need to consider. There are a few more items need to be added this list.
Protocols must be designed such that they are synchronizable; specifications must avoid the use of
unnecessary transient states which complicate synchronization and make generating and applying the test
sequences nearly impossible. Solutions to synchronization issues of conformance testing require
extensive further research.

There are additional issues related to test system implementations that we do not address in this chapter
including: implementing adaptive test cases (i.e., a different set of test cases for a different set of
implementation options of a protocol), resolving conflicts in case of inconclusive test verdicts, and
whether or not to entirely retest an SUT after a set of tests fails. We suggest the following set of
references as a starting point for the reader to obtain detailed information regarding the implementation of
test systems: [5][15][17].

10. References for Chapter II


Rules for Abstract Syntax Notation One (ASN.1), International Organization for

Rules for Abstract Syntax Notation One (ASN.1), International Organization for


