Protocol Testing:
Review of Methods and Relevance for Software Testing

G. v. Bochmann and Alexandre Petrenko

Publication # 923

Département d'informatique et de recherche opérationnelle

Université de Montréal

Juin 1994

Protocol Testing: Review of Methods and Relevance for Software Testing

Gregor v. Bochmann and Alexandre Petrenko

Département d'informatique et de recherche opérationnelle, Université de Montréal,
C.P. 6128, Succ. Centre-Ville, Montréal, Canada H3C 3J7

Abstract

Communication protocols are the rules that govern the communication between the different components within a distributed computer system. Since protocols are implemented in software and/or hardware, the question arises whether the existing hardware and software testing methods would be adequate for the testing of communication protocols. The purpose of this paper is to explain in which way the problem of testing protocol implementations is different from the usual problem of software testing. We review the major results in the area of protocol testing and discuss in which way these methods may also be relevant in the more general context of software testing.

1. Introduction

During the last ten years, much research results have been obtained in the area of protocol conformance testing in view of obtaining better and more systematic methods for the testing of protocol implementations against their specifications. At the same time, the methods for testing computer hardware and software have also evolved. Since protocols are implemented in software and/or hardware, the question may arise whether the existing hardware and software testing methods would be adequate for the testing of communication protocols.

The purpose of this paper is to explain in which way the problem of testing protocol implementations is different from the usual problem of software testing. We will review the major results in the area of protocol testing and discuss in which way these methods may also be relevant in the more general context of software testing.

Protocol testing is mainly based on the black-box approach. Therefore the nature of the protocol specification has a strong influence on protocol testing. In fact, the methods for the development of test cases are largely dependent on the specification formalism. Most communication protocols have a reactive nature; therefore specification languages for reactive systems are favored which precisely define the temporal ordering of interactions. It is therefore understandable that the finite state machine (FSM) model is often used for defining protocol specifications. For this reason, most work on protocol testing has been based on FSM models.

In Section 1, we discuss the main characteristics of communication protocols and different formal description techniques which have been used in the protocol engineering cycle.

In Section 2, we review the main results in the area of test suite development based on specifications given in the form of deterministic or nondeterministic FSM models. Emphasis is put on methods that are related to a particular conformance relation which the implementation under test should satisfy in respect to the specification. Also, the test suite should be complete in respect to a given fault model, which is defined in terms of the set of faulty implementations that the test suite should be able to detect.

We then give in Section 3 an overview of some testing issues related to test suite development for specification written in LOTOS, or other specification formalisms based on rendezvous communication.

In Section 4, we discuss the main approaches to the testing in respect to specifications written in an extended FSM formalism, such as the languages SDL and Estelle. In this context, there are many relations with software testing methods, because the extensions of the FSM model are usually defined with programming language concepts.

In Section 5, we discuss the impact of the test architecture which, in the case of protocol testing, usually involves several access points, partial observation and synchronization problems between the different parts of the test system. These issues have a strong impact on testability. Finally, we review the practice of protocol conformance testing and state our conclusions.

1.1. The nature of protocol testing

Communication protocols are the rules that govern the communication between the different components within a distributed computer system. In order to organize the complexity of these rules, they are usually partitioned into a hierarchical structure of protocol layers, as exemplified by the seven layers of the standardized OSI Reference Model [IS7498].

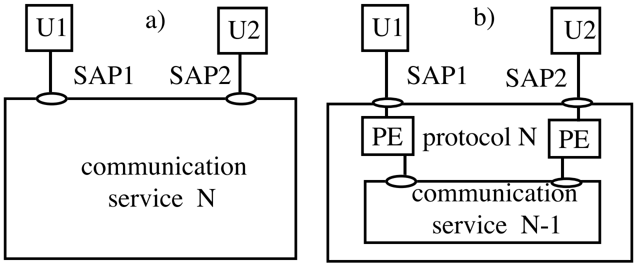

Figure 1: The architecture of a communication system.

Figure 1(a) shows a communication system from the point of view of two users. The users interact with the communication service through interactions, called *service primitives*, exchanged at so-called *service access points* (SAP). The definition of the behavior of the box which extends between the two users is called the service specification. It defines the local rules of interactions at a given service access point, as well as the so-called end-to-end properties which relate the interactions at the different access points and represent the communication properties, such as end-to-end connection establishment or reliable data transfer.

Figure 1(b) shows a more detailed view of the service box showing two so-called *protocol entities* (PE) which communicate through an underlying, more simple communication service. The definition of the required behavior for a protocol entity is called the *protocol specification*, and involves the interactions at the upper and lower service access points. In addition, the protocol specification usually identifies different types of so-called *protocol data units* (PDU, or messages) which are coded and exchanged between the protocol entities through the underlying medium.

The discipline of protocol engineering [Boch93], [Boch90] deals with the methods for developing communication protocol specifications and implementations, including the activities of validation. Protocol implementations are tested against the protocol specification in order to assure compatibility with other implementations of the same protocol. This is called *protocol conformance testing*. In addition, implementations are usually also tested for other properties, not specified by the protocol, which may include implementation-specific choices, robustness to user errors and performance properties. We call this type of testing *implementation assessment*.

For certain protocol standards, the standardization committees have also defined standardized conformance test suites, which usually consist of a large number of test cases. The successful execution of these test cases should provide a reasonable assurance that the tested implementations follows all the rules defined by the protocol.

One of the main characteristics of protocol testing is the fact that conformance testing is black-box testing. This implies that a precise reference specification must be provided, which is the basis for the derivation of the test cases and the analysis of the test results. (This specification is in fact the protocol specification, which defines the required properties of any implemented protocol entity).

1.2. Specification languages for communications protocols

As they develop, protocols must be described for many purposes. Early descriptions provide a reference for cooperation among designers of different parts of a protocol system. The design must be checked for logical correctness. Then the protocol must be implemented, and if the protocol is in wide use, many different implementations may have to be checked for compliance with a standard. Although narrative descriptions and informal walk-throughs are invaluable elements of this process, painful experience has shown that by themselves they are inadequate.

The informal techniques traditionally used to design and implement communication protocols have been largely successful, but have also yielded a disturbing number of errors or unexpected and undesirable behavior in most protocols. The use of a specification written in natural language gives the illusion of being easily understood, but leads to lengthy and informal specifications which often contain ambiguities and are difficult to check for completeness and correctness. The arguments for the use of formal specification methods in the general context of software engineering apply also to protocols.

In this context, many different formal description techniques have been proposed for the protocol engineering cycle, including finite state machines (FSM), Petri nets, formal grammars, high-level programming languages, process algebras, abstract data types, and temporal logic. The simpler models, such as FSM, Petri nets and formal grammars, were often extended by the addition of data parameters and attributes in order to naturally deal with certain properties of the protocols, such as sequence numbering and addressing [Boch90].

With the beginning work on the standardization for Open Systems Interconnection (OSI), special working groups on "Formal Description Techniques" (FDT) were established within ISO and CCITT in the early eighties with the purpose of studying the possibility of using formal specifications for the definition of the OSI protocols and services. Their work led to the proposal of three languages, Estelle, LOTOS and SDL, which are further discussed below (for a tutorial introduction and further references, see [Budk87], [Bolo87] and [Beli89], respectively). These languages are called <u>formal</u> description techniques, since care has been taken to define not only a formal syntax for the language, but also a formal semantics which defines the meaning, in a formal manner, of any valid specification. This is in contrast to most programming languages which have a formally defined syntax (for instance in BNF), but an informally defined semantics. The formal semantics is essential for the construction of tools which are helpful for the validation of specifications or the development of implementations.

While SDL had been developed within CCITT since the seventies for the description of switching systems, Estelle and LOTOS were developed within ISO for the specification of communication protocols and services. However, all these languages have potentially a much broader scope of applications. However, their effective use in the OSI area, so far, has been relatively slow. This may be partly explained by the competition between these three languages, which each have certain advantages, and by the difficulty many people have in learning a new language.

In Estelle, a specification module is modeled by an extended FSM. The extensions are related to interaction parameters and additional state variables, and involve type definitions, expressions and statements of the Pascal programming language. In addition, certain "Estelle statements" cover aspects related to the creation of the overall system structure consisting in general of a hierarchy of module instances. Communication between modules takes place through the interaction points of the modules which have been interconnected by the parent module. Communication is asynchronous, that is, an output message is stored in an input queue of the receiving module before it is processed.

SDL, which has the longest history, is also based on an extended FSM model. For the data extensions, it uses the concept of abstract data types with the addition of a notation of program variables and data structures, similar to what is included in Estelle. However, the notation is not related to Pascal but to CHILL, the programming language recommended by CCITT. The communication is asynchronous and the destination process of an output message can be identified by various means, including process identifiers or the names of channels or routes. Recently, the language has been extended to include certain features for object-oriented specifications. In contrast to the other FDT's, SDL was developed, right from the beginning, with an orientation towards a graphical representation. The language includes graphical elements for the FSM aspects of a process and the overall structure of a specification. The data aspects are only represented in the usual linear, program-like form. In addition, a completely program-like form is also defined, called SDL-PR, which is mainly used for the exchange of specifications between different SDL support systems.

LOTOS is based on an algebraic calculus for communicating systems (CCS [Miln80]) which includes the concept of finite state machines plus parallel processes which communicate through a rendezvous mechanism which allows the specification of rendezvous between two or more processes. Asynchronous communication can be modeled by introducing queues explicitly as data types. The interactions are associated with gates which can be passed as parameters to other processes participating in the interactions. These gates play a role similar to the interaction points in Estelle. The data aspects are covered by an algebraic notation for abstract data types, called ACT ONE, which is quite powerful, but would benefit from the introduction of certain abbreviated notations for the description of common data structures. A graphical representation for LOTOS has also been defined.

In addition to the formal description techniques discussed above, the OSI standardization committees also use certain semi-formal languages (which have no formally defined semantics). In particular, a language called TTCN [Sari92] is used for describing conformance test cases, and the ASN.1 notation is used for describing the data structures of the protocol data units (messages) exchanged by the OSI application layer protocols [Neuf92]. This notation is associated with a coding scheme which defines the format in which these data units are exchanged over the communication medium. All the other languages mentioned above do not address this problem.

In the context of application layer protocols, in particular for Open Distributed Processing and distributed systems management, certain forms of object-oriented specifications are being used which are based on extensions of ASN.1. Also the formal specification language Z has been proposed, which is based on set theory and predicate calculus, and was originally developed for software specifications.

2. Testing based on FSM specifications

The development of testing methods based on FSM specifications has initially been driven by problems arising from functional testing of sequential circuits. The appearance of communication protocols has given to this theory a new boost. Currently, this model is also attracting much attention in relation with testing of object-oriented software systems [Hoff93], [Turn92]. Moreover, recent results in binary decision diagram (BDD) technology have increased our ability to deal with the large number of states [Brya86]. Traditional work in this field has relied on the model of completely specified deterministic finite state machines. Recently, more complex FSM models have also been studied.

2.1. Basic definitions: FSM models and conformance relations

A nondeterministic finite state machine (NFSM) is an initialized nondeterministic Mealy machine which can be formally defined as follows [PBD93]. A *nondeterministic finite state machine* A is a 6-tuple (S, X, Y, h, s_0, D_A), where S is a set of n states with s_0 as the initial state; X is a finite set of input symbols; Y is a finite set of output symbols; D_A is a specification domain which is a subset of $S \times X$; and h is a behavior function $h: D_A \to P(S \times Y)$, where $P(S \times Y)$ is the powerset of $S \times Y$. The behavior functions defines the possible transitions of the machine. For each present state s_i and input symbol x, each pair $(s_ij\ y)$ in the result of the function represents a possible transition leading to the next state s_j and producing the output y. This is also written as a transition of the form s_i-x/y->sj.

The machine A becomes *deterministic* (FSM) when $|h(s,x)| = 1$ for all $(s,x) \in D_A$. In a deterministic machine, instead of the behavior function, we use two functions: the next state function δ, and the output function λ.

In the general case, let $\alpha = x_1x_2...x_k \in X^*$, α is called an *acceptable input sequence for state* $s_i \in S$, if there exist k states $s_{i1}, s_{i2}, ..., s_{ik} \in S$ and an output sequence $\gamma = y_1y_2...y_k \in Y^*$ such that there is a sequence of transitions s_i-x_1/y_1->s_{i1}-x_2/y_2->s_{i2} -> ... -> s_{ik-1}-x_k/y_k->s_{ik} in the machine. We use X_i^* to denote the set of all the acceptable input sequences for the state s_i and X_A^* for the state s_0, i.e. for A. We extend the behavior function to the set X_A^* of all acceptable input words (sequences) containing the empty word e, i.e., $h: S \times X_A^* \rightarrow P(S \times Y^*)$ [PBD93].

An NFSM A is said to be *completely specified*, if $D_A = S \times X$. If $D_A \subset S \times X$ then A is considered *partially specified.* An NFSM will be also referred to as a complete or a partial machine.

The *equivalence* relation between two states s_i and s_j in the NFSM A holds if $X_i^* = X_j^*$ and $\forall \alpha \in X_i^*$ $(h^2(s_i, \alpha) = h^2(s_j, \alpha))$, otherwise, they are nonequivalent, here h^2 is the second projection of the behavior function which corresponds to the output function of the deterministic FSM [Star72]. The machine is said to be *reduced* if all its states are pairwise nonequivalent. Two NFSMs A and B are said to be *equivalent* if their initial states are equivalent; intuitively, this means that the observable behaviors of the two machines are identical.

Given a NFSM $A = (S, X, Y, h, s_0, D_A)$, A is said to be *initially connected* if $\forall s \in S$ $\exists \alpha \in X_A^*$ $(s \in h^1(s_0, \alpha))$, where h^1 is the first projection of the behavior function which corresponds to the next function of the deterministic FSM [Star72]. Every NFSM is equivalent to an initially connected one.

The complete NFSM $B = (T,X,Y,H,t_0)$ is said to be *quasi-equivalent to A* if for all acceptable input sequences $\alpha \in X_A^*$ $(H^2(t_0, \alpha) = h^2(s_0, \alpha))$. Intuitively, this means that for the input sequences for which the behavior of A is defined, the behavior of B is identical.

Given the NFSM $A = (S, X, Y, h, s_0, D_A)$, and complete NFSM B, B is said to be a *reduction* of A, written $B \leq A$, if $\forall \alpha \in X_A^*$ $(H^2(t_0, \alpha) \subseteq h^2(s_0, \alpha))$. Intuitively, this means that for the input sequences for which the behavior of A is defined, the output sequences of B are included in those defined for A.

The equivalence, quasi-equivalence, and reduction relations between NFSMs are widely used as the conformance relations between implementations and their specifications in protocol test derivation based on the finite state machines.

2.2. Interfaces, fault models and complete test suites

In protocol engineering, it is customary to distinguish (at least) two levels of abstraction for the description of module interfaces: For the protocol and service specifications, the interactions at a SAP are described at an abstract level in terms of atomic events which normally correspond to a request by the user or an indication received by the user from the protocol entity. For each implementation of the protocol, an implementation of the abstract interface is provided either in software (e.g. procedure calls, operating systems calls) or hardware. The nature of the interface implementation is not defined by the protocol specification. Only the abstract properties of interactions, their parameters and ordering constraints defined by the protocol specification must be reflected by the implementation.

The test cases developed for the conformance testing of a given protocol should be applicable to all implementations of that protocol, irrespective of the interface conventions used by the implementation. Therefore these test cases are based on the abstract interface definition of the specification. In order to execute such a test case with a given implementation under test (IUT), these abstract interactions must be realized in terms of the concrete interface adopted by the protocol implementation. For instance, the upper tester (UT) accessing the upper SAP of the IUT (see Figure 2) must be adapted to the particular SAP interface adopted by the IUT.

We conclude that in case of black-box testing based on the specification of the IUT, the test suite must be developed based on the abstract interfaces defined within the specification, and it must be assumed that the (abstract) properties of these interfaces are satisfied by the concrete realization of these interfaces in the implementation under test; we call this the *correct interface assumption*. (How the validity of this assumption can be tested is outside the scope of this paper).

In the context of FSM-based testing, it is assumed that the interface is event-driven, and moreover, that all events on the interface are alternately controlled by the environment and the IUT. In particular, there exist two channels, one is to convey "inputs" of the environment and the second is to receive "outputs" from the IUT. Each abstract input yields exactly one output symbol, and the next input symbol can be sent to the IUT only after the IUT has moved into a stable state and produced an output symbol in response to the previous input. An output reaction of the IUT to any input can be in the form of a meaningful output (as explicitly defined by the specification), an "error" or a "null" output (which means "no output", and which will be detected by waiting a specified time-out period). All these outputs can be observed and are treated as abstract output symbols.

Based on such an interface, the reference behavior is assumed to be specified as an NFSM. The correct interface assumption implies that any IUT can also be modeled as an NFSM. In particular, no IUT can "refuse" to execute any input in any of its states or produce an output without any input. Any IUT is thus a **completely** specified NFSM whose input alphabet at least includes the one of the specification NFSM, even if the specification is a partial machine.

In order to introduce the notion of test coverage, it is necessary to determine which faults of an implementation should be covered. One way to introduce a fault model is by defining a set of implementations with erroneous behavior that should be detected by the test suite. If this set is finite, a finite test suite exists that covers all faults; if it is not finite, there may be no finite test suite that covers all faults. In order to define a fault model, we take the mutation approach, where a *mutant* of a specification *A* is any implementation that satisfies the correct interface assumption. In the case of FSM-based conformance testing, the *mutants* of *A* are the class of all completely specified NFSMs with the same input alphabet. A fault model $\Im$ is a subset of this class.

The concept of conformance relation is needed to distinguish mutants which could exhibit an erroneous behavior [PBD93]. It is usually introduced when both the IUT and its corresponding specification are represented with the same formalism. In the case of the FSM model, the equivalence, quasi-equivalence, and reduction relation are natural candidates for conformance relation. The given specification NFSM *A* and a conformance relation ***conf*** partition the set of mutants $\Im$ into the subsets of conforming implementations $C(A) = \{I \mid I \in \Im \ \&\ I\ \textbf{\textit{conf}}\ A\}$ and non-conforming implementations $N(A)$ for the given specification *A*, i.e., $N(A) = \{I \mid I \in \Im \ \&\ I\ \textit{not}\textbf{\textit{conf}}\ A\}$. Based on this partition, the notion of a complete test suite for a given NFSM *A* with respect to a given conformance relation and fault model can be formally introduced as follows.

A test suite $E \subseteq X_A^*$ is said to be *complete* for the specification NFSM *A* and the fault model $\Im$ w.r.t. the reduction relation if for any NFSM $B = (T, X, Y, H, t_0)$ in $\Im$ which is not a reduction of A, there exist an $\alpha \in E$ and a $\gamma \in H^2(t_0,\alpha)$ such that $\gamma \notin h^2(s_0,\alpha)$. Similarly, complete test suites are defined for the equivalence and quasi-equivalence relations. A complete test suite guarantees detection of all mutants in the fault model that do not conform to the specification.

There are different ways to characterize a mutant, or the set of mutants within a given fault model. The traditional mutation approach defines a set of *mutation types*, each representing a specific type of structural fault, that is, a modification which may be introduced into a specification in order to obtain a different behavior. In the so-called *single fault model*, the mutants considered are those obtained from the specification by applying a single mutation of one of the specified types. In the case of the so-called *multiple fault model*, the mutants are obtained by one or several such mutations. In the case of FSM specifications, the mutation types considered are usually *output faults* (the output of a transition is wrong) or *transfer faults* (the next state of a transition is wrong). In the area of software testing, various types of mutations may be considered, such as sequencing faults (e.g. missing alternative path, improper nesting of loops, WHILE instead of REPEAT, wrong logic expression to control a conditional or looping statement); arithmetic and manipulative errors, such as ignoring overflow, wrong operator (e.g. GT instead of GE or + instead of *); calling of the wrong function; wrong specification of data type and/or wrong format of data representation; wrong initial value, or wrong number of data elements; or a reference to an undefined variable or the wrong variable.

This "fault-based" identification of mutants seems most natural in the context of diagnostic testing where one wants not only detect erroneous behavior, but also determine how the implementation could be changed in order to make it conform to the specification (e.g. identify a faulty component to be replaced, or identify the part of the program, or the part of the FSM state table that should be corrected). It seems that it works well in the context of deterministic and completely defined FSM specifications, however, in the context of nondeterministic systems, there are many examples where a few mutations lead to a mutant that conforms to the specification; the correspondence between mutations and erroneous behavior is therefore not straightforward. Therefore one uses often a "wholistic" identification of mutants, simply characterizing the erroneous behavior of the mutant by an appropriate model without comparing its structure with the structure of the (correct) specification.

2.3. Test derivation from completely specified deterministic FSMs

The simplest fault models are defined for specifications which can be represented by completely specified deterministic FSMs. In this class of machines, the equivalence relation serves as conformance relation, and the mutants are completely specified deterministic machines.

When applied to such a machine, the mutation technique yields a certain number of mutant FSMs. A particular mutant FSM can be generated from the specification FSM by changing the next state and/or the output of a number of selected transitions between states. This mutant machine represents a certain combination of output and transfer faults. The number of all possible mutants grows exponentially when the number of states and inputs increase, therefore, some heuristics, such as information about the implementation, severity of faults, test hypotheses should guide this process. Based on the set of mutants considered, it is possible to find in one way or another a set of tests which distinguish (kill) all these mutants from the specification FSM.

There are different methods for defining the set of mutants in the fault model, such as the following. In each case, corresponding methods of deriving a complete test suite have been developed.

(1) Limiting the number of states: The fault model consists of the universe $\Im_m$ of all complete deterministic FSMs with a number of states less or equal to *m*, and which have the input alphabet of the specification FSM *A*, where $m \geq n$, the number of states in *A*. Guessing the bound *m* is an intuitive process based on the knowledge of a specification, the class of implementations and their interior structure which have to be tested for conformance. The value of *m* can be also limited due to economic considerations. The larger the bound is chosen, the larger the complete test suite is, as it is illustrated by the existing upper bounds for the complexity of tests [PBD93].

The methods for deriving complete test suites from complete deterministic machines are all based on transition checking approach [Henn64]. Each method, however, requires a distinct type of a characterization set [Vasi73] (a W-set) used for state identification. The UIOv-method [Vuon89], for instance, constructs the W-set only from so-called unique input/output sequences. The methods [Yevt90a], [Petr91], [Petr92] are based on "harmonized" state identifiers eliminating the verification phase that is necessary for the W- and Wp- methods [Chow78], [Fuji91]. The methods in [Vasi73], [Chow78], [Fuji91], [Yevt90a] allow the IUT's to have more states than in the specification (i.e., $m \geq n$); and the others [Vuon89], [Petr91], [Petr92] do not (i.e., $m=n$). Which of these methods can produce a shorter complete test suite for a given FSM and $m=n$, is still an open question. All the mentioned methods assume that a reliable reset is available in the IUT's; note that there is another group of methods which guarantee full fault coverage without this assumption (see, e.g., [Henn64], and the UIOG-method in [Yao93]). They can be applied for reduced, deterministic, strongly connected FSM's. However, these methods usually yield much longer complete test suites than the former. Earlier reviews of the test derivation methods especially of those that do not guarantee full fault coverage, can be found elsewhere (see, e.g., [Ural91], [Sidh89]).

The complexity of a corresponding complete test suite seems to be the price for assuming such a broad fault model. A more restricted fault model requires, however, a justification for each particular application.

(2) Output faults only: These faults are relatively easily to catch. Any test suite based a transition tour of A (i.e. traversing each transition of A at least once) is already complete for this fault model [Nait81]. This corresponds to the "all branches" criteria used in software testing.

(3) Fault functions: grouped faults which can be represented by a so-called fault function [Petr92]. Let $A = (S, X, Y, \delta, \lambda, s_0)$ be a reduced initially connected FSM. It is assumed that all possible mutants are deterministic reductions of an appropriate NFSM $F(A) = (S', X, Y, H, s_0)$ constructed from the specification FSM A in such a way that A is a deterministic submachine of $FA)$. The behavior function H describes all output and transfer faults for each transition and is called a *fault function* for A. If $F(A)$ is weakly initialized, i.e. it has more that one initial state, then initialization faults are also included in this fault model. A fault function is a general and flexible formalism to model different faults. In fact, if $H(s,x) = S' \times Y$ for all $(s,x) \in S' \times X$, and $|S'|=m$, then all machines of the set $\Im_m$ are deterministic submachines of $F(A)$. Output faults are easy expressed by a specific form of the fault function as well: $H(s,x) = \{[\delta(s,x),y] \mid y \in Y\}$ for all $(s,x) \in S \times X$, $S'=S$. A natural question on how a fault function can be chosen for the given FSM arises. We believe, this function is, first of all, a tool to formalize hypotheses of a test engineer about the defects in a given IUT, but their formulation is a process which is rather intuitive and largely based on experience. He may have certain ideas on which part of the protocol (e.g., transitions) is more likely to be incorrectly implemented, which errors are most critical or frequently made by implementators of particular class of protocols, which part does require testing and so on. Particular forms of the fault function are constructed in an obvious way in cases when, e.g., the reference FSM describes the behavior of a composite system and some component machines are assumed to be error-free. Such cases are typical for grey-box testing and encountered when a protocol is represented by a composition of several FSMs or by an EFSM. A so-called FF-method for generating a test suite which is complete for all faults defined by any given fault function, is proposed in [Petr92]. It tunes a test suite to user-defined classes of faults. Similar to the above considered methods, it is also based on transition identification. However, next states of transitions are identified in certain subsets of states determined with the help of the given fault function.

Concluding this section, we note, that to our knowledge, there have been no systematic studies to determine whether the assumptions on a particular fault model are realistic. The assumption of no additional states seems to be justified in the case that a direct implementation of a pure FSM specification is realized by systematic translation; expected faults could be wrong output or transfer for specific transitions, but the number of states of the implementation would normally be the same as for the specification. However, if additional parameters are present (extended FSM model) the implementation of these parameters may possibly lead to more states in the implementation.

2.4. Partially specified FSMs

Most of the existing protocols are not completely specified, in the sense that in real communication systems, not all the sequences of interactions are feasible. The model of a partial FSM adequately expresses this feature of protocols. The semantics of partial machines needs, however, further clarification. A specific feature of a partial FSM is that it has "undefined" transitions called also "non-core" or "don't care". There are different conventions that may be used to give a formal meaning to "undefined" (we note that in the context of labeled transition systems, see Section 3, there is a different notation, since an "undefined" means "blocking", no transition possible):

(1) "Implicitly defined" transitions. With this convention, a partial FSM represents a completely specified FSM by adopting some "*completeness assumption*". Such an assumption is based on the fact that all implementations can be represented by complete machines which never refuse any input. It states that all "don't care" transitions in the specification are substituted by looping transitions with the null outputs (Convention 1a), or with transitions to an error (terminal) state with an "error" output (Convention 1b). In SDL specifications, for instance, unexpected or inopportune events are ignored, so Convention 1a is applied. The convention used during the process of implementation is assumed to be known for testing purposes. The given partial FSM is substituted by a quasi-equivalent completely specified FSM, and the problem of test derivation from a partial machine is thus avoided.

(2) "Undefined by default" transitions. This convention means that a partial FSM is interpreted as a set of complete FSMs taking into account that the result of a "don't care" transition might be any state and any output. This interpretation is suitable for conformance testing of the given implementation when information about the convention used by an implementator is absent.

In contrast to the first convention, all possible options are left up to the implementor of the partial machine. A conforming implementation is, in fact, a complete FSM that implements any of these options [Petr91]. The quasi-equivalence conformance relation exactly captures this interpretation. It is also called *weak conformance* [Sidh89].

(3) "Forbidden" transitions. In some cases, the behavior of a system is not completely specified because its environment will never execute certain input sequences. Invalid entries in state tables used in the OSI protocol standards exemplify this situation. The set of acceptable input sequences X_A^* of an FSM A represents all "admissible" sequences, the remaining ones are "forbidden", they define "forbidden" transitions in the given partial FSM. This convention is similar to the one of "undefined by default" transitions, the difference is that in this case, a tester cannot submit certain inputs to certain states of the given partial FSM. An example is a lower service provider that cannot deliver any messages to certain states of a tested protocol entity and an upper protocol entity that cannot deliver an unexpected service primitive to any state of the IUT. In all these situations we have to treat "don't care" transitions as "forbidden" and to avoid them in test derivation. In contrast, transitions that are undefined by default can be tested to determine the reaction of the implementation for these "unexpected" inputs; this is called "robustness testing" or testing for *strong conformance* [Sidh89].

The conclusion is that to derive an abstract test suite from even a completed specification of an embedded IUT we might have to come back to a partially specified machine. Therefore, even though some specification languages based on the FSM model impose complete specifications by a build-in mechanism of implicitly defined transitions, there is a need for deriving conformance tests directly from partial machines. Note that grey-box testing, in general, requires the model of a partial FSM [Petr94].

Constructing a complete test suite for a machine from this class is complicated by the fact that minimality of a given FSM cannot anymore be taken for granted, unlike in the case of complete machines. If the machine is not reduced then some of its states cannot be distinguished neither on the specification level nor on the implementation level. The traditional transition identification approach is no more applicable to these machines. A more general approach based on the idea of counting distinguishable states traversed by a test sequence gives methods [Yevt90b], [Petr91], [Yevt91], [Luo94] for test derivation which can treat partial as well as complete deterministic machines.

2.5. Test derivation from nondeterministic FSM specifications

All the three major specification languages for protocols, LOTOS, ESTELLE, and SDL support the description of nondeterminism. So a machine abstracted from the formal description of a protocol might also be nondeterministic. Generally speaking, a nondeterministic FSM model may

be used to represent different situations, such as the following:
- a protocol entity with inherent nondeterminism;
- a set of deterministic protocol entities considered as options of a given protocol;
- a deterministic IUT embedded in a given system in such a way that a tester cannot directly observe (see [Petr94] for more detail);
- nondeterminism due to concurrency.

For nondeterministic protocol implementations, a given test may give rise to different observations of the output reaction. It is therefore not enough to execute a test once, the same test should be repeatedly executed until all possible observations are obtained. If the nondeterminism within the IUT cannot be influenced from the testing environment, it may be difficult, in general, to determine after how many executions of the given test all possible observations have been obtained. In order to determine the verdict for a given test, it is therefore necessary to assume the so-called *complete testing assumption* [Luo94] which supposes that all possible observations have been obtained (see also the "all weather conditions" assumption for LTS's in [Miln80]). Note that so-called intermittent faults can also be modeled as an NFSM, however, it is not clear how the complete testing assumption can be satisfied. Their detection is not guaranteed.

The equivalence, quasi-equivalence, and reduction relations defined for NFSMs seem suitable to serve as conformance relations in protocol testing. We note once more that for complete NFSMs, the quasi-equivalence relation coincides with the equivalence relation, and the choice has to be made between the two relations. If, however, an implementation submitted for conformance testing is known to exhibit a deterministic behavior only, then the reduction relation has to be taken as a conformance relation between a nondeterministic specification and its deterministic implementation.

Test derivation from such machines is currently an active research area. Several methods have already been reported, some of them are guided by certain heuristics, the others, like [Yevt91], [Petr93], [Petr91], [Luo94], are based on fault models and provide complete test suite w.r.t. the chosen conformance relation. Similar to deterministic partial machines, these methods exploit the idea of counting distinguishable (in the sense of the chosen relation) states traversed by a test sequence, to ensure the required relation between the specification and implementation. Different conformance relations yield usually different complete test suites for the same specification machine, since states which are not (quasi-) equivalent may satisfy the reduction relation.

2.6. Length of complete test suites

The expected length of a complete test suite for an arbitrary given (partially specified and nondeterministic) FSM w.r.t. the chosen conformance relation is usually well within the upper bound of the corresponding test derivation method. In the worst case, it does not exceed the value

$|X|^{nm}$, where X is an input alphabet, n is the number of states in the given machine, and m is the upper bound on the number of states in possible implementations [Petr93]. The complete deterministic FSMs have better upper bounds, see for instance [Vasi73], [Yann91], [PBD93].

However, the actual complexity of the test suite for FSMs modeling protocols seems much less these upper bounds. The complexity depends on properties of the characterization set on which the test derivation method is based. Even though their construction is shown to be PSPACE-complete [Yann91], several empirical studies show that protocol machines tend to have rather short state identification sequences which result in test suites of reasonable size. We give some examples in Section 6.

2.7. Fault coverage analysis

As with other software testing, the evaluation of fault coverage for a given test suite is an important issue in testing based on the FSM model and has been studied in the context of traditional hardware testing and, in recent years, in the context of protocols. Methods that have been proposed are essentially variations of the mutation analysis technique. For instance, instead of using the exhaustive mutation analysis approach, some researchers have introduced a number of classes of mutant machines [Dahb88], [Dubu91], [Sidh89] and [Mott93]. The mutant machines in a class will contain a certain number of faults resulting from the changes of next states and/or outputs of some transitions. For each class, a limited number of mutant machines are randomly generated which are then executed against the given test suite. Apparently, the fault coverage evaluated in such a way is an estimation of the real fault coverage of the test suite and its accuracy relies on the total number of mutant machines randomly generated and executed.

Recently, a different procedure has been developed which, without the need of explicitly generating and then executing a (usually large) number of mutant machines, can decide if the given test suite provides complete fault coverage [YPB94]. When the test suite does not provide full fault coverage, the proposed approach can derive from the test suite, by analyzing it against the specification machine, an incorrect implementation machine which can pass the test suite and therefore allows an additional test case to be generated to distinguish this particular implementation machine from the specification. As such, full fault coverage can be achieved by repeatedly applying this procedure. However, this approach does not provide a numeric measure for a test suite which does not have full fault coverage. Consequently, it is impossible to use this approach to compare the fault coverage of two test suites, if none of them provides full fault coverage. A metric-based approach [Yao94] to the evaluation of fault coverage of a test suite in respect to a given specification machine can be used for this purpose. This approach avoids the necessity of explicit generation and execution of mutant machines representing possible implementations of the given specification machine. It provides a numeric measure for a test suite no matter whether the test suite has complete fault coverage (i.e., 100%) or not. The experiments show [Yao94] that this method gives a good approximation of the real fault coverage of the given test suite.

3. Testing based on specifications in the form of labeled transition systems

Labeled transitions systems (LTS's) are in some sense a more general specification model than FSM's, since interactions of a specified subsystem with its environment are usually considered as rendezvous interactions making no distinction between input and output (see for instance the specification formalism CCS [Miln80] or CSP [Hoar85]). In the case of the LOTOS language, more than two processes may participate in a given rendezvous interaction. For an interaction to occur, it is necessary that all participating processes have a specified transition that leads from their present state to a next state involving the interaction in question. If such a transition is not defined for the present state, we say that the interaction is blocked for the process.

An LTS may be considered as a partially specified FSM without outputs where the interactions of the LTS correspond to the inputs of the FSM, and the meaning of an "undefined" transition is the blocking behavior, as explained above. In addition, LTS's usually allow for internal events that are not observable from the environment, denoted τ (or i, in the case of LOTOS). Nondeterminism may be introduced by internal events or by several transitions, for the same interaction, leading to different states.

Different kinds of conformance relations may be considered for LTS's. If only the traces of observed interactions are of interest (like the traces of inputs and outputs as considered for FSM's), one may consider the trace equivalence and trace inclusion relations, which correspond to the equivalence and reduction relations defined for FSM's.

In the case of nondeterministic systems, the blocking behavior cannot be deduced from the possible traces. Therefore it becomes important to consider explicitly the blocking behavior. This is usually done by considering, for each possible trace, a set of associated refusal sets; a *refusal set* for trace t is a set of interactions that may be all refused (i.e. blocked) by the system after executing the trace t. The refusal behavior of an implementation may be tested by a testing environment that executes, in rendezvous with the IUT, the trace t and then offers all the interactions in the refusal set. If an execution of this test leads to deadlock after the trace t, it is shown that the IUT includes the refusal set in question. Note that in general, because of the nondeterministic nature of the specification (and possibly the implementation) each test must be executed several times, as in the case of testing nondeterministic FSMs, since after a given trace t, the IUT may be in different states, each corresponding to a different refusal set. Based on this testing method, also called *failure semantics*, the conformance relations *test equivalence*, *reduction*, *extension* and "*conformance*" may be defined [Tret91].

Many other conformance relations may be considered (see for instance [Miln80], [Miln81], [Glab90]). Depending on the properties of a tester and the interface between the tester and the IUT, different kinds of experiments on processes can be defined, giving rise to an abundance of semantics for the relations between processes. In the *failure trace semantics*, for instance, a tester can not only observe deadlocks but also continue testing afterwards by offering another set of interactions, which is not allowed for the failure semantics. In several simulation-related semantics, the tester is, at any time during a run of the IUT, capable of making arbitrary many copies of the IUT in its current state and observe them independently. This assumption is, in fact, much stronger than the reliable reset assumption in the realm of FSM's. It seems to us that test hypotheses deeply influence the choice of a conformance relation among this hierarchy of the preorder relations and the associated equivalence relations. We refer to work of R.J. van Glabbek [Glab93], [Glab90], where 155 notions of observability have been reviewed. The relations have been mainly used for verification purposes in process algebra and concurrent systems.

To our knowledge, current research in testing implementations against a given reference LTS has been concentrated on relations based on trace and failure trace semantics. In recent years, much work on the derivation of conformance tests from a given LOTOS, or corresponding LTS specification, has been done in the protocol engineering community [Alde90], [Brin87], [Drir93], [FuBo91], [Lang89], [Ledu91], [Pitt90], [Weze90], [Brin89], [Tret91]. Most of this work has addressed the difficult problem of dealing with nondeterministic specifications.

It seems that the theories of test derivation from FSM and LTS specifications have been developed almost independently. Recently, it has been shown [PBD93] that it is possible to transfer the problem of deriving a conformance test suite for an LTS to the realm of the input/output FSM model, where the test derivation theory has been elaborated for several decades and a number of useful results have been obtained already. The idea is to define, for a given LTS specification and given conformance relation, a corresponding FSM specification such that the application of the FSM test suite (based on trace conformance) is equivalent to the verification of the given conformance relation in respect to the LTS specification. Unfortunately, the number of transitions in the corresponding FSM may be much larger than in the original LTS. Further work is required to determine whether this approach can lead to practical results.

4. Testing of extended FSM models

The FSM model is often too restrictive for defining all aspects of a protocol or any other kind of specification. Therefore an extended FSM model is often used. With this model, the specification of a system module, represented as an extended FSM, includes additional state variables (in addition to the FSM state variable) and the input and output interactions include parameters. A transition is in general characterized, in addition to the FSM characterization of present and next state and input and output interactions, by a so-called enabling predicate and a transition action. The enabling predicate must be satisfied for the transition to be possible. It depends on the present module state (FSM and additional state variables) and the effective input parameters. The action defines the output produced, including the effective output parameters, and possibly some updates of the additional state variables.

The notation used for the description of the transition predicates and actions are usually borrowed from some high-level programming language. Therefore, the issues of the systematic testing of an implementation based on a given extended FSM specification are closely related to the issues of software testing. If certain limiting assumptions are made about the form of the predicates and actions, the analysis of the behavior of the specification and systematic test selection remains decidable [Higa94], but in general, in particular when the actions may include loops, the question of deciding which input parameters should be used in order to force the selection of a particular transition becomes undecidable, like the question of deciding the executability of a given branch of the program in software testing.

Several researchers have proposed to use dataflow analysis for the systematic selection of test cases for extended FSM specifications. The dataflow analysis involves the input and output parameters and the additional state variables. By selecting appropriate test cases involving appropriate transitions, it is possible to satisfy the testing criteria that have been developed for software testing, such as "all definition-use pairs", etc. [Fran93]. A number of methods are based on the construction of control and data flow graphs [Sari87], [Ural87], [Ural91], [Mill92], [Ural93]. By analyzing the graphs, some guidelines are provided for choosing test sequences based on traditional software testing criteria. In order to clearly separate the control and data flow aspects of a specification, the transformation into a *normal form* specification has been proposed [Sari87]. In such a specification, all control flow aspects are represented by the underlying FSM model, whereas the actions can be represented by straight-line code without branching statements nor loops.

Clearly, the selection of the transition to be executed cannot be done in an arbitrary manner, since the FSM part of the specification prescribes certain rules about the possible sequencing of the state transitions. At the same time, it is desirable to cover the faults of the FSM part of the specification, using one of the test selection methods developed for pure FSM specification discussed in Section 2. This interplay between the control part of the specification (represented by the pure FSM model) and the data part (represented by the transition predicates and actions) makes these test selection methods interesting.

The test selection methods for extended FSM models usually ignore the problem of selecting appropriate input parameter values in order to assure that the desired transition predicates become enabled. The selection of appropriate input parameter values is also important for the other reason that a dataflow fault may not be revealed for a given combination of input parameter values if the resulting output parameter values are (by chance) not affected by the dataflow fault. Therefore the method of repeating tests with different input parameter values, selecting for each parameter the extreme and some intermediate values, well-known in software testing, is also proposed to be used for protocol testing [IS9646].

An EFSM can be viewed as a compressed notation of an FSM. It is possible to unfold it into a pure FSM by expanding the values of the parameters [Favr86], [Cast86], [Vuon89], [Faro90], [Chun90]. Their domains have to be reduced in order to avoid state and input explosion problem. In this case, the FSM-based methods cover not only the control part of a specification but also its data part.

The mutant-killing technique has also been applied to derive test sequences from an EFSM specification of a protocol. A limited number of mutant EFSM's are first constructed based on the given EFSM, and then by comparing the behavior of two machines a sequence which can distinguish them is found, see for instance [Guo91], [Wang93]. The BDD technology [Brya86] and implicit state enumeration methods can be used to save space for constructing distinguishing sequences, see, e.g. [Cho91].

A similar problem is encountered in the area of hardware design verification. To verify the design one has finally to compare the two EFSM's and to try to distinguish them, one represents the project, the second one - the actual design.

The weak point of this technique (and possibly all the EFSM-based methods) is the exhaustive search for suitable paths. Fortunately, most of the protocols do not contain complicated computation and reveal the content of their variables within several transitions, moreover, they tend to have simple predicates.

5. Test architectures and testability

A typical protocol entity has two service access points (SAP) to communicate with adjacent protocol layers, as shown in Figure 1(b). Their existence is a distinctive feature of communication software, it gives rise to a variety of architectures for protocol testing. Several system architectures for conformance testing have been identified in the context of OSI standardization [Rayn87]. These architectures can also be used for implementation assessment. The international standard [IS9646] distinguishes four basic types of a test architecture: local, distributed, coordinated and remote. Local test architecture corresponds to traditional software testing, here both SAP's of the implementation under test (IUT) are accessed by a single tester. In this architecture, the SAP's can be viewed as parts of an interface between the tester and the protocol implementation since the IUT and the tester reside within the same computer system. The other (external) architectures assume that the lower SAP is accessed via an underlying communication service by the so-called "lower tester" which is implemented in a separate test system computer. The so-called distributed and coordinated test methods require along with the lower tester the "upper tester" which has to be interfaced with the IUT on its upper SAP. Figure 2 shows the distributed architecture, where the IUT and a test user, called "upper tester", reside in a computer that is connected through a network to a remote test system computer.

The difference between the two methods is that the coordinated test architecture relies on a special test coordination protocol which supports coordination or synchronization of the actions of both testers. A separate communication channel is usually introduced, sometimes in the form of a terminal connection to the remote test system, which allows the operator at the system under test to coordinate the activities of the test system with the operation of the system under test. Various test coordination protocols, sometimes using a separate channel, have also been developed for automatically coordinating the actions of the upper and lower testers.

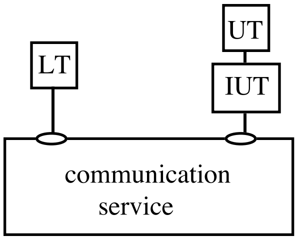

Figure 2: Distributed test method.

The remote test architecture excludes the upper SAP from the testing process since in a number of practical situations, the IUT is embedded into a complex system in such a way that the upper tester simply cannot be included into the system. It is important to note that complete testing of a protocol implementation implies the observation of the interactions at the upper <u>and</u> lower interfaces. Nevertheless, in many cases the remote test architecture is used which does not have a special upper tester and therefore does not check the communication service provided to the user. The remote and distributed test architectures have the advantage that the test system resides in a separate computer and can be accessed over distance from a variety of systems under test. In the context of OSI, certain test centers provide public conformance testing services which are accessed over public data networks.

The distributed test architecture presents some difficulty to test execution concerning the synchronization between the upper and lower testers, since they reside in different computers and communicate only indirectly through the IUT. Since the synchronization becomes an important issue in the cases when the upper and lower testers do not reside in the same computer, several FSM-methods [Sari84], [Boyd91], [Luo93] attempt to produce so-called synchronizable test suites which do not need an external coordination of the two testers at the penalty of an increased length and worse fault coverage. Some protocols are intrinsically nonsynchronizable, hence the distributed test architecture without coordination between testers cannot guarantee complete fault coverage with any given test suite.

Each of the external test methods comes in three variant forms: single-layer, multi-layer or embedded. Single-layer methods are designed for testing a single-layer protocol implementation without reference to the layers above it, but assuming that the protocol below has already been tested. Multi-layer methods are designed for testing a component that implements several protocol layers as a whole. Embedded test methods tune tests to a single protocol assuming that the protocols above and below this layer have already been tested. The variety of test architectures is suitable for different testing situations. It is obvious that these architectures have different impacts on testability, i.e. controllability and observability of the IUT.

As may be expected, testability of a protocol implementation usually deteriorates when it is being tested through an environment, e.g. other protocol layers. The use of additional points of control and observation (PCO) could compensate such a deterioration, moreover, these PCO's could significantly decrease the length of a required test suite for an isolated protocol implementation. PCO's for testability are one of the techniques considered in the design for testability of communication protocols, which is an active research area [Dsso90], [Dsso91], [Vuon93]. However, a third party (conformance) testing usually does not rely on such special features since most existing protocols have been specified and implemented without conformance testing in mind.

External and embedded test architectures entail serious testing problems which are best formalized in the context of grey-box testing. Opposed to the black-box approaches, grey-box testing attempts to profit from the internal knowledge of the system accessed by testers. The major issue is that a model, e.g. an LTS or an FSM, used to represent the behavior of a given protocol entity alone, does not characterize its behavior in such a test architecture. Some actions are no longer directly controlled or observed, certain faults are not easy to activate by and/or to propagate to a tester, i.e. they even may go undetected. At the same time, testability in this case has its limits in the sense that there are implementations that would not conform to the isolated protocol specification, but when considered in the given architecture, become conforming. This problem has been formally treated in terms of the FSM model in [Petr93], [Petr94] and the LTS model in [Drir93]. As shown in [Petr94], the behavior of a component under test even if it is deterministic and completely specified, when it can be externally controlled and observed, may be adequately represented by a partially specified and nondeterministic FSM within a particular context. Tests derived from such a testable representation ensure that all faults detectable within the given test architecture are revealed. Similar considerations are reported in the area of hardware, see for instance [Wata93].

6. The practice of protocol conformance testing

Testing of distributed systems is required at several levels, such as in-house validation during the design and implementation process, protocol conformance and implementation assessment testing, inter-operability testing between different implementations of the same protocol, and quality of service testing concerning specific performance questions. For conformance testing of standardized protocols, standardized conformance test suites have been developed which are intended to be applied in a standardized test architecture and environment. Most of these test suites have been developed using ad hoc techniques inspired by ideas of the existing formal methods.

Based on standardized test suites conformance testing services are offered by a number of protocol test centers throughout the world. According to data provided by the Open Systems Testing Consortium (OSTC), the test campaign for a protocol implementation may last from a few days to several weeks depending on the complexity of the protocol. For example, 7 days are required to perform conformance testing of an OSI session protocol implementation, according to the OSTC. Unfortunately, not much data is currently available on the statistics of real faults in tested protocol implementations. One of the possible reasons for this is that protocol implementators are as usually reluctant to disclose such information.

Protocol test derivation and execution requires specific support tools. Due to space limitations, we cannot discuss these issues here; the interested reader is referred to a recent review [Chan93]. According to some data, the cost of test suite production can be reduced by at least 40% by using such tools. Note that tools for automated test derivation quite often constitute a part of a protocol development environment based on one of the FDT's [Prob89], [Chan93].

Software tools based on formal methods for test derivation were used in a number of protocol case studies; we mention only a few. The Wp-method [Fuji91], for example, has been applied in [Liu93] to derive test sequences for a control subset of the XTP protocol. This subset is modeled by an FSM with 10 states. Altogether, 458 test cases were generated, each of lengths is shorter than 10 inputs. Another case study [Shev91] dealt with the OSI Session protocol, which can be represented by a state table with 29 control states; 28 context variables, among them 9 are integers and the others Boolean; and 71 predicates. Based on this state table, a number of FSMs for different functional blocks of the protocol have been constructed by unfolding and decomposing the table. Domains of input parameters and integers were first reduced in such a way that all the predicates can be executed at least once. Finally, the method [Petr91] was applied to each of these

FSMs and a test suite of approximately 11700 test cases was generated. The total number of input test events is about 67000. The test suite provides complete fault coverage in terms of the constructed FSM models. The ISDN protocol LAPD was used in a case study at AT&T [Sher90]. An FSM model with 9 states was constructed for the protocol and a test suite of 1500 input test events was formally generated by applying the UIO-method [Aho88]. Later on, a document containing the standardized test suite for ISDN LAPD became available, which consists of more than 1200 pages containing tables of test cases specified in TTCN. These applications and others [Sidh89] have demonstrated that formal methods for conformance test generation are quite effective for rather complex communication protocols. However, the data portion of protocols remains the most difficult problem for the existing formal methods.

7. Conclusions

A protocol entity is a reactive system; therefore the temporal ordering of the interactions, with the user and with the remote peer entity, is an important aspect of each protocol specification. For this reason, FSM models are often used for protocol specifications. However, in order to capture the real requirements of a protocol specification, the model of a partially specified, partly nondeterministic FSM must be used. In addition to this "aspect", there are usually other aspects which are more easily specified in terms of data parameters and operations; in the specification of protocol standards, these aspects are usually described in English; however, they may be formalized using an extended FSM formalism. Several so-called formal description techniques (FDT's) have been developed for the specification of communication protocols and services (SDL, LOTOS, and Estelle) which include facilities for the control and data aspects of system specifications. These specification requirements are similar to other kinds of reactive systems.

Another specification aspect specific to communication protocols is the coding of the protocol data units which are exchanged between the peer entities. Usually, ad hoc specification methods are used for this purpose. However, in the case of most application layer protocols, generic rules for coding data structures are used for this purpose; some rules of this type are defined by the ASN.1 notation used for OSI application layer protocols.

Protocol conformance testing uses a black-box approach based on the protocol specification. Most methods for the development of a test suite from a given specification are based on a fault model of possible erroneous implementations that should be detected by the test suite. Guarantees for the coverage of the test suite are therefore dependent on the assumption of the fault model. These fault models depend on the nature of the specification. Corresponding to the different aspects of a protocol specification, different fault models relate to the control part (FSM), data part (software related fault models) and coding (specific fault model depending on the coding scheme, not much work has been done in this area). The authors do not know of any systematic study showing how realistic these fault models are.

The methods for test suite development are not only specific to the fault model and specification paradigm used for the protocol specification, but depend also on the conformance relation that should hold between the implementation under test and the specification. In the case of partial and nondeterministic specifications, different conformance relations may be considered. The test methods based on deterministic, completely specified FSM models are best investigated, however, most protocol specifications do not satisfy the underlying assumptions. Methods for partial and nondeterministic systems have been developed recently, including work on testing based on LTS specifications (e.g. written in LOTOS) which are based on rendezvous interactions.

For the testing based on extended FSM models, two approaches can be adopted: (1) unfolding the specification into the form of a pure FSM model (with the danger of state explosion), or (2) separating the control and data aspects of the specification, and using FSM and software testing methods for these two aspects, respectively.

Testing of distributed systems is required at several levels, such as in-house validation during the design and implementation process, protocol conformance and implementation assessment testing, inter-operability testing between different implementations of the same protocol, and quality of service testing concerning specific performance questions. For conformance testing of standardized protocols, standardized conformance test suites have been developed which are intended to be applied in a standardized test architecture and environment. Most of these test suites have been developed using ad hoc techniques inspired by the formal methods.

The complexity of real protocol specifications are such that the systematic test development methods described in Sections 2 through 4 could provide good benefit. Containing typically between 10 and 20 control states and a few data variables, the complexity of the specification can be handled by the automated tools, but is usually too big to be processed "by hand". The length of the obtained test suites are usually much shorter than those predicted by the "upper bound" formulas which make worst case assumptions on the specification.

It seems that the methods developed for protocol testing have a much larger applicability. In the area of software testing, application domains such as reactive systems, control systems, possibly designed in an object-oriented framework, could make use of these methods. They can also be applied to the functional testing of sequential circuits.

Areas for future research include design for testability and test architectures; systematic test suite development for nondeterministic and partial specifications; the integration of the FSM fault model for control flow and the software fault model suitable for the extensions of the FSM model within the context of extended FSM models, including test derivation based on specifications written in languages such as SDL, Estelle or LOTOS; test development in the case of object-oriented specialization hierarchies; and testing of real-time aspects.

It is difficult to describe in a short paper all directions of research related to protocol testing. Further details can be found in the proceedings of IFIP-sponsored conference series, such as International Symposiums on Protocol Specification, Testing and Verification (PSTV), International Workshops on Protocol Test Systems (IWPTS), International Conferences on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE).

Acknowledgments: This research was supported by the IDACOM-NSERC-CWARC Industrial Research Chair on Communication Protocols at the University of Montreal. The authors wish to thank Tom Ostrand for a list of questions which stimulated the preparation of this article.

References

[Aho88] A. V. Aho et al., "An Optimization Technique for Protocol Conformance Test Generation based on UIO Sequences and Rural Chinese Postman Tours", PSTV'88.

[Alde90] R. Alderden. "COOPER, the Compositional Construction of a Canonical Tester", FORTE'90.

[Beli89] F. Belina and D. Hogrefe, "The CCITT-Specification and Description Language SDL", Computer Networks and ISDN Systems, Vol. 16, 1989.

[Boch90] G. v. Bochmann, "Protocol specification for OSI", Comp. Networks and ISDN Systems 18, April, 1990.

[Boch91] G. v. Bochmann, et al. "Fault Models in Testing", IWPTS'91.

[Boch93] G. v. Bochmann, "Protocol Engineering", contribution to Concise Encyclopedia of Software Engineering, Derrick Morris and Boris Tamm eds., Pergamon Press, 1993.

[Bolo87] T. Bolognesi and E. Brinksma, "Introduction to the ISO Specification Language Lotos", Computer Networks and ISDN Systems, vol. 14, No. 1, 1987.

[Boyd91] S. Boyd, H. Ural, "The Synchronization Problem in Protocol Testing and its Complexity", Inf. Proc. Letters, Vol.40, No. 8, 1991.

[Brin87] E. Brinksma, "On the Existence of Canonical Testers", Memorandum INF-87-5, Univ. of Twente, 1987.

[Brin89] E. Brinksma, et al, "A Formal Approach to Conformance Testing", IWPTS'89.

[Brin91] E. Brinksma, et al, "A Framework for Test Selection", IWPTS'91.

[Brya86] R. E. Bryant, "Graph-based Algorimths for Boolean Function Manipulation", IEEE Trans., vol. C-35, No.12, 1986.

[Budk87] S. Budkowski and P. Dembinski, "An Introduction to Estelle: a Specification Language for Distributed Systems", Computer Networks and ISDN Systems, vol. 14, No. 1, 1987.

[Cast86] R. Castanet et al, " Methods and Semi-automatic Tools for Preparing Distributed Testing", PSTV'86.

[Chan93] S. T. Chanson, A. A. F. Loureiro, S. T. Vuong, "On Tools Supporting the Use of Formal Description Techniques in Protocol Development", Computer Networks and ISDN Systems, 25, 1993.

[Chen93] K.-T. Cheng, A. S. Krishnakumar, "Automatic Functional Test Generation using the Extended Finite State Machine Model", Proc. of DAC'93.

[Cho91] H. Cho, G. Hachel, F. Somenzi, "Fast Sequential ATPG Based on Implicit State Enumeration", Intl. Test Conference, 1991.

[Chow78] T. S. Chow, “Test Design Modeled by Finite-State Machines”, IEEE Trans. SE-4, 3, 1978.

[Chun90] W. Chun and P. Amer, "Test Case Generation for Protocols Specified in ESTELLE", FORTE'90.

[Dahb88] A. Dahbura and K. Sabnani, “Experience in Estimating Fault Coverage of a Protocol Test”, INFOCOM’88.

[Drir93] K. Drira, et al., "Testability of a Communicating System through an Environment", LNCS, 668, 1993.

[Dsso90] R. Dssouli, R. Fournier, G. v. Bochmann, "Conformance Testing and FIFO Queues", FORTE'90.

[Dsso91] R. Dssouli and R. Fournier, "Communication Software Testability", IWPTS'91.

[Dubu91] M. Dubuc, R. Dssouli and G.v. Bochmann, “TESTL: A Tool for Incremental Test Suite Design Based on Finite State Model”, IWPTS'91.

[Faro90] A. Faro and A. Petrenko, "Sequence Generation from EFSMs for Protocol Testing", COMNET'90.

[Favr86] J. -P. Favreau, R. J. Linn, "Automatic Generation of Test Scenario Skeletons from Protocol Specifications Written in ESTELLE", ISPTSV'86.

[Fran93] P. G. Frankl, S. N. Weiss, "An Experimental Comparison of the Effectiveness of Branch Testing and Data Flow Testing", IEEE Trans., vol SE-19, No.8, 1993

[FuBo91] S. Fujiwara, G. v. Bochmann, "Testing Non-deterministic State Machines with Fault Coverage", IWPTS'91.

[Fuji91] S. Fujiwara, G. v. Bochmann, F. Khendek, M. Amalou, A. Ghedamsi, "Test Selection Based on Finite State Models", IEEE Trans., SE-17, No.6, 1991.

[Glab90] R. J. v. Glabbeek, "The Linear Time-Branching Time Spectrum", LNCS, 458, 1990.

[Glab93] R. J. v. Glabbeek, "The linear Time-Branching Time Spectrum II", LNCS, 715, 1993.

[Guo91] F. Guo, R. Probert, "E-MPT Protocol Testing: Preliminary Experimental Results", PSTV'91.

[Henn64] F. C. Hennie, "Fault Detecting Experiments for Sequential Circuits", IEEE 5th Ann. Symp. on Switching Circuits Theory and Logical Design, 1964.

[Higa94] T. Higashino, G. v. Bochmann, "Automatic Analysis and Test Derivation for a Restricted Class of LOTOS Expressions with Data Parameters", IEEE Tran., SE-20, No. 1, 1994.

[Hoar 85] C. A. R. Hoare, Communicating Sequential Processes, Prentice Hall, 1985.

[Hoff93], D. Hoffman and P. Strooper, "A Case Study in Class Testing", in CASCON'93.

[IS9646] OSI Conformance Testing Methodology and Framework.

[IS7498] Reference Model for OSI.

[Lang89] R. Langerak, "A Testing Theory for LOTOS Using Deadlock Detection", PSTV'89.

[Ledu91] G. Leduc, "Conformance Relation, Associated Equivalence, and New Canonical Tester in LOTOS", PSTV'91.

[Liu93] F. Liu, "Test Generation Based on the FSM Model with Timers and Counters,", M. Sc. Theses, Université de Montréal, DIRO, 1993.

[Luo93] G. Luo, R. Dssouli, G. v. Bochmann, P. Venkataram, A. Ghedamsi, "Generating Synchronizable Test Sequences based on Finite State Machines with Distributed Ports", IWPTS'1993.

[Luo94] G. Luo, A. Petrenko, G. v. Bochmann, "Test Selection based on Communicating Nondeterministic Finite State Machines using a Generalized Wp-Method", IEEE Trans., Vol. SE-20, No. 2, 1994.

[Miln80] R. Milner, A Calculus of Communicating Systems, LNCS, 92, 1980.

[Mill92] R. E. Miller and S. Paul, "Generating Conformance Test Sequences for Combined Control and Data Flow of Communicatiion Protocols", PSTV'92.

[Moor56] E. F. Moore, "Gedanken-Experiments on Sequential Machines", Automata Studies, Princeton University Press, Princeton, New Jersey, 1956.

[Mott93] H. Motteler, A. Chung and D. Sidhu, "Fault Coverage of UIO-based Methods for Protocol Testing", IWPTS'1993.

[Neuf92] G. Neufeld and S. Vuong, "An overview of ASN.1", Computer Networks and ISDN Systems, 23, 1992.

[Petr91] A. Petrenko, "Checking Experiments with Protocol Machines", IWPTS'1991.

[Petr92] A. Petrenko, N. Yevtushenko, "Test Suite Generation for a FSM with a Given Type of Implementation Errors", ISPTSV'92.

[PBD93] A. Petrenko, G. v. Bochmann, R. Dssouli, "Conformance Relations and Test Derivation", IWPTS'1993.

[Petr93] A. Petrenko, N. Yevtushenko, A. Lebedev, A. Das, "Nondeterministic State Machines in

Protocol Conformance Testing", IWPTS'1993.

[Petr94] A. Petrenko, N. Yevtushenko, R. Dssouli, "Grey-Box FSM-based Testing Strategies", Department Publication 911, Université de Montréal, 1994, 22p.

[Prob89] R. L. Probert, H. Ural, M. W. A. Hornbeek, "A Comprehensive Software Environment for Developing Standardized Conformance Test Suites", Computer Networks and ISDN Systems, 18, 1989/1990.

[Rayn87] D. Rayner, "OSI Conformance Testing", Computer Networks and ISDN Systems, 14, 1987.

[Sari84] B. Sarikaya, G. v. Bochmann, "Synchronization and Specification Issues in Protocol Testing", IEEE Trans., vol. COM-32, No.4, 1984.

[Sari87] B. Sarikaya, G. v. Bochmann, E. Cerny, "A Test Design Methodology for Protocol Testing", IEEE Trans., vol. SE-13, No.5, 1987.

[Sari92] B. Sarikaya, A. Wiles, "Standard Conformance Test Specification Language TTCN", Computer Standards & Interfaces, Vol.14, No.2, 1992.

[Sher90] M. H. Sherif, M. U. Uyar, "Protocol Modeling for Conformance Testing: Case Study for the ISDN LAPD Protocol", AT&T Technical Journal, January 1990.

[Shev91] V. Shevelkov, "Development of Methods and Tools for Session Interconnection Provision in Open Systems Networks", Ph.D. Theses, Riga, 1991.

[Sidh89] D. P. Sidhu and T. K. Leung, “Formal Methods for Protocol Testing: A Detailed Study”, IEEE Trans. SE-15, 4, 1989.

[Star72] P. H. Starke, Abstract Automata, North-Holland/American Elsevier, 1972, 419p.

[Tret91] J. Tretmans, P. Kars, E. Brinskma, "Protocol Conformance Testing: A Formal Perspective on ISO IS-9646", IWPTS'91.

[Turn92] C.D. Turner and D.J. Robson, "The Testing of Object-Oriented Programs", TR-13/92, Univ. of Durham, 1992.

[Ural87] H. Ural, "Test Sequence Selection based on Static Data Flow Analysis", Computer Communications, Vol. 10, No. 5, 1987.

[Ural91] H. Ural, "Formal Methods for Test Sequence Generation", Computer Comm., Vol. 15, No. 5, 1992.

[Ural93] H. Ural, A. Williams, "Test Generation by Explosing Control and Data Dependencies within System Specifications in SDL", FORTE'93.

[Vasi73] M. P. Vasilevski, "Failure Diagnosis of Automata", Cybernetics, Plenum Publ. Corporation, N.Y., No. 4, 1973.

[Vuon89] S. T. Vuong, W. L. Chan, and M. R. Ito, "The UIOv-method for Protocol Test Sequence Generation", IWPTS'89.

[Vuon93] S. Vuong, A. A. F. Loureiro, S. T. Chanson, "A Framework for the Design of

Testability of Communication Protocols", IWPTS'93.

[Wang93] C.-J. Wang and M. T. Liu, "Generating Test Cases for EFSM with Given Fault Model", INFOCOM'93.

[Wata93] Y. Watanabe, R. Brayton, "The Maximum Set of Permissable Behaviors for FSM Networks", CAD'93.

[Weze90] C. D. Wezeman, "The CO-OP Method for Compositional Derivation of Conformance Testers", PSTV'90.

[Yann91] M. Yannakakis, "Testing Finite State Machines", Proc. of the 23d Annual ACM Symposium on Theory of Computing, 1991.

[Yao94] M. Yao, A. Petrenko, G. v. Bochmann, " A Structural Analysis Approach to Evaluating Fault Coverage of Software Testing in Respect to the FSM Model", Dep. Publ. #920, Universite de Montreal, 1994.

[Yevt90a] N. Yevtushenko, A. Petrenko, "Synthesis of Test Experiments in Some Classes of Automata", Automatic Control and Computer Sciences, Allerton Press, Inc., N.Y., Vol.24, No.4, 1990.

[Yevt90b] N. Yevtushenko, A. Petrenko, "Method of Constructing a Test Experiment for an Arbitrary Deterministic Automaton", Automatic Control and Computer Sciences, Allerton Press, Inc., N.Y., Vol.24, No.5, 1990.

[Yevt91] N. Yevtushenko, A. Lebedev, A. Petrenko, "On the Checking Experiments with Nondeterministic Automata", Automatic Control and Computer Sciences, Allerton Press, Inc., N.Y., Vol.25, No.6, 1991.

[YPB93] M. Yao, A. Petrenko and G.v. Bochmann, "Conformance Testing of Protocol Machines without Reset", PSTV'93.

[YPB94] M. Yao, A. Petrenko and G.v. Bochmann, “Fault Coverage Analysis in Respect to an FSM Specification”, INFOCOM’94.