Towards testable communication software

(Extended Abstract)

R. Dssouli¹, K. Karoui¹, A. Petrenko¹, and O. Rafiq²

1 - Université de Montréal, Dept. IRO. Pav. André Aisenstadt, bureau 2225, C.P. 6128, succursale Centre-Ville, Montréal (Québec) H3C 3J7, CANADA. Tel. (514) 343-7599; FAX. 343-5834 {dssouli, karoui, petrenko}@iro.umontreal.ca

2 - Université de Pau, Lab. TASC-Informatique, Av. de l'université, 64000, Pau, France. Tel. (+33) 59923199; FAX. (+33) 59808374, rafiq@univ-pau.fr

Abstract

Design For Testability (DFT) is understood as the process of introducing some features into a communication software that facilitate the testing process of protocol implementations. DFT at the implementation level deals with a particular realization on a given platform, whereas DFT at the specification level affects all possible implementations regardless of the implementation process. In this paper, we propose in the first part, a generic framework of design for testability where DFT activity is integrated in the communication software development process. A generic model for testability transformation based on modification of a given specification and testability measurement is explained. In the second part, we address one particular problem of DFT, the problem of analyzing the testability of a given module specified as a finite state machine. Transformations in this paper are not the main objective, but they are used to illustrate how testability can be enhanced by finding an augmentation of the given protocol behavior such that a newly obtained specification is more testable than the original one. A measure of testability of a protocol entity is assumed to be based on the shortest length of a test suite needed to achieve guaranteed coverage of certain faults. We consider the FSM model of a protocol machine and propose a classification method based on the formalized notions of controllability and observability.

Keywords

Testing, testability, observability, controllability, testability measures, transformation

1 INTRODUCTION

As computer systems continue to play an ever-increasing role in high-risk and critical applications as well as in more mundane activities, ensuring the reliability and maintainability of such systems is a major concern. Failures may result in loss of human life or economic disasters, although in most cases the result is less drastic. Though improvements in the development technologies can dramatically improve reliability and maintainability, testing and verification still remain important aspects in the development process.

Testing is now widely accepted as an integral part of the software development cycle. Despite enormous efforts devoted to developing testing techniques, serious problems with respect to the generation and application of tests still persist. Although better testing methods will undoubtedly help alleviate some of these problems, it is unlikely that these will lead to effective and economical testing of arbitrary software. Issues related to testing can no longer be considered in isolation from the specification and implementation of software products. Effective and economical testing will require that a specification contain mechanisms to make software more easily tested, thereby reducing development costs and increasing reliability and maintainability. Moreover, considering these issues at an early stage of the development process may in fact help reduce the chances of introducing an error in the first place.

In general terms, testability of a protocol entity means that it has some features that will facilitate the testing process [DsFo91], [VLC93], [PDK93]. Design for testability (DFT) is understood as the process of introducing these features into the protocol entity. DFT at the implementation level deals with a *particular* realization on a given platform, whereas DFT at the specification level affects *all* possible implementations regardless of the implementation process. Unfortunately, most existing protocols have been designed and documented without testing requirements in mind [VLC93]. We argue that design for testability should start at the earliest possible phase of the protocol life cycle, i.e. at the specification level. However, it is not realistic to expect that we could alter basic communication functions to improve testability of future protocol implementations. Thus, the design for testability may only attempt to improve the effectiveness of the testing process under certain constraints. In particular, all basic elements of a protocol specification, such as defined sets of messages and states, should normally be maintained. If the protocol designer is free to add an additional input which might be, for example, a so-called "read-state" message, then new outputs, one per state, are added into the protocol. In this case, the problem of constructing an easily testable machine becomes trivial. However, such a solution seems expensive, since a protocol entity should support more messages than originally required, so in practice, it has not been accepted as a universal solution. To improve the protocol's testability, one could rely only on combinations of states and events for which the protocol behavior is not defined. Assuming that the behavior in such situations can be arbitrary defined, the problem of DFT at the specification level can be formulated as follows. We must find a minimal transformation of the given protocol behavior such that a newly obtained specification is more testable than the original one.

The paper is organized in two parts. In the first part, we address the problem of DFT in a general framework. In the second part, we address one particular problem of DFT, the problem of analyzing the testability of a given module specified as a finite state machine. Transformations in this paper are not the main objective, but they are used to illustrate how testability can be enhanced by finding an augmentation of the given protocol behavior such that a newly obtained specification is more testable than the original one. A measure of testability of a protocol entity is assumed to be based on the shortest length of a test suite needed to achieve guaranteed coverage of certain faults. We consider the FSM model of a protocol machine and propose also a classification method based on the formalized notions of controllability and observability.

2 A TESTABILITY FRAMEWORK

2.1 Communication software testability

Design For Testability (DFT) is understood as the process of introducing some features into a protocol specification or implementation that facilitate the testing process. DFT applied at the implementation level deals with a particular realization on a given platform and it is often called "instrumentation", whereas DFT at the specification level affects all possible implementations regardless of the implementation process.

For a better understanding of communication software testability issues, we give the following rather general definition:

"A software has the property of testability if it includes facilities allowing the easy application of testing methods, the detection or isolation of existing faults".

Observability and controllability of software are widely acknowledged as two important attributes that influence software testability. Although these attributes can be expressed in many different ways, it is often easier to describe them in terms of various criteria [DsFo91], [KDC94].

The set of properties that characterize testable software is not yet well-defined, but software designers have an intuitive idea as to what constitutes testable software. Therefore, this concept corresponds to several desires developed to confront difficulties perceived daily in testing. These difficulties are summarized as follows:

- The selection of the best test suite with minimal length and maximal coverage of faults may be difficult. A test set may be infinite, in which case, total test coverage cannot be achieved unless a fault domain is restricted in advance. This problem is not always due to the test selection techniques in use. The software design or test architecture can also be the cause for the undetected faults during the test campaign.
- The application of a test suite to the implementation under test may be difficult since most test suites are derived from protocol specifications. Such test suites are called abstract test suites. The advantage of abstract test suites is their reusability for testing different implementations of the same specification. The efforts spent in adapting these test suites for specific implementations can diminish the advantage of using such suites.
- It is not easy to identify the design schemes that lead to parts which are untestable or difficult to test.
- The analysis and interpretation of test results (traces) is another difficulty frequently encountered in testing. The problem of finding a matching trace in the specification necessitates the identification of each input and output in both the expected and the observed traces.
- Another important issue in the software development process is the fault diagnostics. When a test case fails, it may be difficult to determine the cause of the failure. The presence of multiple faults in the implementation can make the fault isolation problem even more complex.

2.2 A generic framework

Design for testability can be seen as an iterative process of modifying specifications and/or implementations and measuring the testability of the newly obtained specification. These transformations can be partially automated and integrated in the traditional software development cycle as described in Figure 1. The ultimate goal of these transformations is to produce a more testable specification/ implementation [Dsso91].

Figure 1 Communication software development cycle

Figure 2 illustrates how the testability of a specification can be improved based on certain testability measures which give an indication on the quality of the end product. The methodology seems simple to apply when testability problems are well identified and for which solutions exist. However for the general case, the Figure 2 raises the following questions:

1. What are the appropriate transformations the protocol designer could apply to the given specification?

2. What are the proper testability measures that the protocol designer should use?
3. When will the process of transformation terminate?

Figure 2 Testability enhancement based on transformation

Generally speaking, a testability transformation should be guided by the identification of what is not easy to test. Designers should try to determine factors that degrade the testability of a particular specification. The list of these factors includes [Dsso91], [VLC93]:

- the formal description language,
- the complexity of the specification,
- nondeterminism due to concurrency,
- the degree of freedom allowed in the specification,
- the level of abstraction that can be used for testing,
- the test architecture,
- the test strategy.

Unfortunately, most existing protocols have been designed and documented without testing requirements in mind [VLC93]. Therefore, it is not realistic to expect that we could alter basic communication functions to improve testability of future protocol implementations. Thus, the design for testability may only attempt to improve the effectiveness of the testing process under certain constraints. It is still possible to automate the procedure of testability transformation for some factors. In particular, consider the partially specified finite state machine (PFSM) model, and a testing strategy based on state identification. If the PFSM has no distinguishing sequence, then the machine is hard to test, and the cost of testing is high. In this case, a systematic approach can be applied: find a minimal augmentation of a specification such that all states becomes distinguishable at a minimal cost. This augmentation can be based on specifying "don't care" transitions that do not generate additional end-to-end exchange; this approach is elaborated in the accompanying paper [YPD95].

For question 2, [Free91], [VMM91] and [PDK93] proposed various testability measures. These measures focus on a particular aspect of testability, and cannot be applied at each step in communication software development process. A model which permits the measurement of testability of software, for the purposes of comparison, would help designers in overcoming many difficulties. Ideally, this model should reflect all aspects of what we call "easily

testable" software. However, different testing strategies require different measures to support the management of the design and testing processes in communication software development. The testability measure might be seen rather as a vector in which each element is a measure of a particular factor or aspect of testability [KDC94].

For the last question, the termination of the process of transformation is related to the satisfaction of testability criteria. The desired degree of testability that leads to a confidence in the design should be used as a threshold.

2.3 Instrumentation of implementations

As we said already, the DFT activity can be applied at different steps in the software development cycle. DFT applied to the detailed specification will affect a particular implementation. It offers a way to define various test interfaces depending on designers', and testers' specific needs.

Communication software systems are usually composed of many communicating modules. Each module can also have its own internal structure (Figure 3). Such a multilevel structure may be used as additional information for grey-box testing [PYD94].

Figure 3 Multilevel structure of communication software

To enhance the testability of such modular systems, it is desirable to make their internal structure visible for testers. This can be done by instrumenting an implementation with internal interaction points accessible by testers. These are points of observation (PO) in addition to external points of control and observation (PCO).

There are several ways of systematic instrumentation such as the addition of primitives, selective broadcasting and the direct call to trace procedures. The instrumentation based on

the use of formal description techniques can be often done easily especially when it uses normal constructs of the formal languages.

Addition of primitives

This consists of increasing the functionality of the interaction point by adding a specific primitive whose role is to take the trace of any event happening at this interaction point. This trace may be immediately analyzed by an observer or stocked in memory for later use.

Selective broadcasting

Here each event is selectively broadcasted to its original destination and to the observer. However this method changes substantially the semantics of an interaction point in the usual specification languages (Estelle, LOTOS, SDL).

Direct call to trace procedure

It consists of adding calls to a trace procedure within the code. The choice of the locations where these calls are to be added may correspond to the interaction points of the structure of the software to be implemented, or to a fixed location.

The observation of an implementation may be granular where the degree of granularity depends on what we can control/observe and on what we wish to control/observe. This granularity is based on the inclusion of test points and points of observation. Notice that these points are not necessarily interaction points. We consider the following levels of granularity of instrumentation: structure based instrumentation, transition based instrumentation and state based instrumentation.

Structure based instrumentation

This is the most appropriate level of observation for communication software since it requires an approach based on modular decomposition and communication between modules. It allows the observation and/or control of all specified interaction points (internal or external). It may be associated with a multilevel structure where each module may have its own structure.

Transition based instrumentation

Some software do not allow the application of the modular approach. However if a piece of software is based on transitions (automata), it is possible to include points of observation at each transition thus permitting the recording of their traces. We could also have PCO's since each transition may be executed separately.

State based instrumentation

This type of instrumentation is based on the so-called "read-state" message. If the protocol designer is free to add an additional input ("read-state" message), then new outputs, one per state, are added into the protocol. In this case, the problem of constructing an easily testable implementation becomes trivial. However, such a solution seems expensive, since a protocol entity should support more messages than originally required, so in practice, it has not been accepted as a universal solution.

To conclude our general discussion on testability issues, we notice that formal methods for DFT of protocols have not yet been explored. Note that the existing formal methods and techniques for improving testability which have been developed mainly in the hardware area, see, for example, [ShLe94], [ABF90], [Jose78], cannot be applied in this domain, since they either rely on the structure of an implementation, or they change the set of states, or the sets of input/output events, in a way that is similar to the adding of a read-state message in protocol engineering.

3 DESIGN FOR TESTABILITY FOR THE FSM MODEL

The finite state models in general, and the FSM model in particular, have been extensively used in conformance testing of communication protocols [PBD93], [BoPe94] as well as in hardware and software testing. Testing based on the finite state models has the advantages that the concept of full or complete fault coverage of tests can be formally addressed, and FSM parameters which influence the testability of FSM-based implementations can be identified.

3.1 Basic definitions

An FSM M is a 6-tuple (S, X, Y, h, s_1, D_S) , where S is a set of states with s_1 as the initial state; X - a finite set of input symbols; Y - a finite set of output symbols; D_S - a specification domain which is a subset of $S \times X$; h - a behavior function $h: D_S \rightarrow \mathcal{P}(S \times Y)$, and $\mathcal{P}(S \times Y)$ is the powerset of $S \times Y$ [PBD93]. According to the semantics of the input/output behavior all transitions are labeled by a pair of input and output, however, the null (input or output) event can be additionally introduced to implicitly model situations involving spontaneous transitions, for example timeouts [PBD93]. We say that there is a transition from state s to state s' labeled by the pair x/y if $(s', y) \in h(s, x)$, in this case, $s' \in h^1(s, x)$ and $y \in h^2(s, x)$, where h^1 is the first projection (the transfer function) of the behavior function, and h^2 is the second projection (the output function) of the behavior function. The set of states reached after an input/output sequence α/β is accepted by the initial state of M is defined by $h^1\beta(s_1, \alpha)$.

The above definition of the FSM model includes as special cases various classes of FSMs studied in the literature.

Depending on the specification domain, an NFSM can be partially or completely specified. If $D_S = S \times X$ then M is completely specified or *complete*, otherwise it is *partial*.

An FSM is said to be *observable* if starting from any given state the machine can reach only one state in response to any input/output sequence accepted for the given state. In a *deterministic* machine, an output sequence is not required to determine a unique state reached when an input sequence is accepted by the given state. All deterministic machines are observable, however, an observable machine can still be *nondeterministic*. The equivalent transformation of nonobservable machines into observable forms is possible [Star72], [LBP94]. Similar to the case of automata determinization [HoU179], a nonobservable machine with n states may have up to 2^n states in its observable form. An example of this transformation is given in Section 3.3.

Two states are said to be *distinguishable* if there exists an input sequence accepted by both states which produces different sets of output sequences when it is applied to each of them, otherwise these states are *compatible* states. Compatible states with the same set of acceptable input sequences are called *equivalent*.. Opposed to the compatible states, the equivalent states can always be merged without affecting the specified behavior of the machine. Thus, we exclude the class of FSMs with equivalent states from our discussion. The FSM is *reduced* if all its states are pairwise distinguishable, otherwise it is *nonreduced*. A deterministic, reduced complete FSM is usually referred to as a *minimal* machine, since every such machine possesses a unique reduced form.

Figure 4 Classification of FSMs

Figure 4 shows various classes of FSMs. Intuitively clear, that the testability of machines from different classes varies. To assess the testability of a given FSM specification one could first find the class to which the FSM belongs. The class indicates the testability issues typical for all machines of this class, as we discuss in the following section. A more precise testability analysis requires the FSM testability parameters such as controllability and observability, formally defined later in the paper.

3.2 Testability and transformation of FSMs

Within the generic testability framework instantiated with the FSM model we first address the following general questions:

- Q1 - How can one identify the testability problems in a given FSM specification?
- Q2 - How can one estimate the degree of testability of a given FSM?
- Q3 - What are the transformations that can be applied by the protocol designer?
- Q4 - What freedom has the designer in choosing a specific transformation to apply?

Q1 - The identification of testability problems of an FSM can be viewed as a classification problem of FSMs [PBD93], see Figure 4. It is well known that nondeterministic FSMs are usually less testable than deterministic machines. Within nondeterministic FSMs nonobservable machines are less testable than the observable ones. The major factor that affects the testability in this case is the controllability. Nonreduced FSMs are less testable than reduced FSM and the major factor of testability that is important in this case is the observability, more specifically the fuzziness of the machine. Completely specified machines tend to be more testable than partially specified ones. Given two machines defined over the same sets of states and inputs, a partial machine has fewer transitions than a complete one, however, identification of its states may require longer sequences than that for the latter.

Q2 - A particular type of the given machine gives an idea on the FSM testability, however, to assess it in a more accurate way we have to first instantiate the definitions of controllability and observability for the FSM model. We address this issue later in the paper.

Q3 - Designers might have the possibility to modify an FSM by acting on the following elements: states, inputs, outputs, transition and output functions. Transformations may involve one or several elements. Certain transformations are well known, and their objectives are described in the literature. A minimization algorithm brings an FSM from the class of nonreduced FSMs to the class of reduced FSMs can be viewed as a transformation method. The addition of "read-state" message is another transformation. [YPD95] gives a transformation method for a partial FSM which increases the degree of its specifiedness.

The designer should categorize properly the specification, clearly define testability objectives in terms of which class of FSMs he/she would like to obtain after the termination of the transformation process given in Figure 2. Depending on the testability objectives that one can define, some heuristics can be developed in order to find minimal modifications of an FSM that meet the defined objectives.

Q4 - The choice of a specific transformation is related to the freedom that the designer might have. It depends also on the reached step in the development process. In the case where a specification is a standard, and there exist already its implementations, then transformations are restricted to those that do not generate end-to-end exchange (problem of symmetric protocols), in terms of additional PDU's. In the case where a lower layer can absorb all additional PDUs or a peer entity is designed with the ability to ignore them, then a more fundamental transformation might be allowed.

3.3 Controllability of FSMs

Let V be a state cover of M , i.e. for every state s in S there is at least one transfer sequence $v \in V$ such that $s \in h^1(s_1, x)$. If the machine M is not initially connected then there exists at least one state that cannot be reached from the initial state. In this case, we may assume that this state has an infinite transfer sequence. Note that FSMs with unreachable states usually require lengthy test suites [PHK95]. A state cover constitutes a skeleton of test suites with guaranteed or full fault coverage in the sense that every transition from every state specified in the machine should be tested by such a test suite. The machines may possess several state covers, therefore the amount of efforts required to bring the machine to all possible states depends on the state cover chosen for testing. To get an objective characterization of the controllability of the given machine we should rely on a state cover with the minimal amount of these efforts.

In fact, this amount depends on a number of factors. First, the length $L(v)$ of a transfer sequence v is important when we are required to force the machine to reach a certain state. Next, if the machine is nondeterministic then not only one, but several states can be reached when the transfer sequence is applied to the initial state. If, moreover, the FSM is not observable then even observing the output sequence produced by the transfer sequence does not help much in determining the final state of the machine. In fact, only states of its observable form are controllable. These are sometimes called multi-states of the original nonobservable machine [FuBo91]. Finally, the machine may produce in response to the transfer sequence not just a single output sequence but a number of them. It implies that we have to repeatedly apply the same transfer sequence until a desirable output sequence is produced indicating that a proper (multi-) state is reached (cf. the so-called complete testing assumption in [LBP94]). We integrate the above factors into a single formula based on the notion of the weight of a state cover constructed for the observable form of the machine.

Given state cover V of an observable machine, the *weight* $\alpha(V)$ of V is defined as follows:

$$\alpha(V) = \sum_{v \in V} L(v) |h^2(s_1, v)|$$

Then the *controllability* $C(M)$ of the given machine M is defined as $1/\omega_{\min}(V)$, where $\omega_{\min}(V)$ is the minimal value of the weight among all its possible state covers of its observable form.

Example

Figure 5 (a) shows an example of a nonobservable FSM. Its equivalent observable form is given in Figure 5 (b). Compared to the original machine, there is a new state introduced, viz. the combination of states 2 and 3. The set $V = \{\epsilon, a, ab\}$ is a minimal state cover. $\omega_{\min}(V) = 0 + 1(2) + 2(3) = 8$. Thus, $C(M) = 1/8$.

Figure 5 A nonobservable FSM and its observable form.

Assume that the behavior specified by the given nonobservable FSM can be restricted such that it becomes completely deterministic. This can be achieved if two outgoing transitions labeled with the output x are removed. The modified machine has the same minimal state cover $\{\epsilon, a, ab\}$. However, the controllability has now increased to $1/3$.

Machines not initially connected are the least controllable as there exists a transfer sequence of the infinite length. The controllability of an initially connected deterministic FSM is simply determined by the overall length $L(V_{\min})$ of its minimal state cover $V_{\min}$. In the best possible case, for every state except for the initial state, there exists just a single input which takes the machine from the initial state into this state. Then $C(M) = 1/n-1$, where n is the number of states in M . In the worst case, the maximal length of a transfer sequence may reach $n-1$, thus the total length of a state cover does not exceed $n(n-1)/2$. Thus, the controllability of initially connected deterministic FSMs has the following lower and upper bounds:

$$2/n(n-1) \leq C(M) \leq 1/n-1.$$

Next we switch our discussion to the notion of observability of FSMs.

3.4 Observability of FSMs

Testing with complete fault coverage usually requires the equivalence between states to be verified [Vasi73]. At the same time, black-box testing implies that each state has to be identified. If a given machine is not observable then we can never identify the state reached by the machine after it has accepted an input/output sequence. This is why all currently available test derivation methods for FSMs deal only with observable forms whose states are multi-states of the original nondeterministic machine. Certain observable machines are "more observable" than the others if their states require less test events for identification, thus the definition of the observability of observable FSMs should be based on the notion of states' distinguishability. As discussed above, an arbitrary FSM may be either reduced or not. Opposed to reduced observable machines, the nonreduced ones usually require complete test

suites of an exponential length, since compatible states can be implemented as a single state. The influence of compatible states on the observability of the given FSM can be characterized in the following way.

Given an arbitrary observable FSM M , we compute a partition Π of the set S of its states into subsets $\Pi = \{S_1, \dots, S_f\}$ such that every subset includes only pairwise distinguishable states. The number $f(M)$ of subsets in the minimal partition $\Pi_{\min}$ is called the *fuzziness* of M [YePe89], [LPB94].

If M happens to be complete and minimal, or partial and reduced, then $f(M) = 1$. The larger the number of pairs of compatible states, the higher the fuzziness usually is. For a "fully unreduced" machine $f(M) = n$.

Example

Consider a nonreduced FSM shown in Figure 6.

Figure 6 A nonreduced FSM

States 1 and 4 are compatible states. The set of states has the following minimal partition on subsets pairwise distinguishable states: $\Pi_{\min} = \{1, 2, 3, 5; 4\}$. Thus, the fuzziness of the machine is two.

The fuzziness characterizes the influence of compatible states on the observability of the given machine, however, distinguishable states require a special consideration.

Assume that the given machine M is reduced, i.e. it has no compatible states. In this case, all states can be uniquely identified (observed) using a so-called *characterization set* W of the given FSM which contains input sequences to tell each pair of states apart [Vasi73], [Chow78], [LeYa94]. The number $|W|$ of sequences in W and their total length $L(W)$ characterize the amount of efforts required to identify states of M . The machine usually possesses several characterization sets. The *weight* $\omega(W)$ of the given W set can be determined as follows:

$$\omega(W) = |W| L(W).$$

Then the *distinguishability* $D(M)$ of the given reduced observable machine M is defined as $1/\omega_{\min}(W)$, where $\omega_{\min}(W)$ is the minimal value of the weight among all its possible characterization sets.

The distinguishability of the reduced machine completely characterizes its observability. The most observable machines are FSMs which possess a distinguishing sequence of the length one, e.g. the "read-state" input. Thus, $D(M) \leq 1$. There exists also the least upper bounds for the W set of complete reduced machines. In particular, as shown in [TyBa75], $L(W) \leq n(n-1)/2$. In this case, $|W| = n$. Then $D(M) \geq 2/n^2(n-1)$ for complete reduced machines.

Reduced partial machines tend to be less observable than complete machines since parameters of W sets have higher tight bounds. The total length of sequences of a characterization set W has the least upper bound of $[(n^2-n+1)^2-1]/8$ for reduced partial FSMs

[PeBo95]. Then $D(M) \geq 8/n[(n^2-n+1)^2-1]$ for this class of machines.

The observability of a nonreduced partial machine M depends on both parameters, its fuzziness $f(M)$ and distinguishability $D(M)$. The definition of a characterization set is extended to cover nonreduced machines [Petr91]. In particular, such a set should contain separating sequences for every pair of distinguishable states. Then the distinguishability of a nonreduced machine can be evaluated in a similar manner. The observability of such machine is characterized by both, the distinguishability and fuzziness parameters.

Example

Consider again the FSM in Figure 6. As shown above, its fuzziness is two. If now we find the minimal characterization set, such as $W = \{aa, bb\}$ then its distinguishability can be evaluated: $D(M) = 1/\omega_{\min}(W) = 1/8$. Now we compare these parameters of the given machine with those of its reduced form. It is obtained by merging compatible states 1 and 4. The fuzziness of the reduced machine is just one. The set $W' = \{aa, b\}$ is the minimal characterization set, the distinguishability of the reduced form becomes now $1/6$. A test suite that is complete for the given nonreduced machine (Figure 6) in the class of all FSMs with up to four states derived in [Petr91] has forty test events. The length of a corresponding test suite for the reduced form is only 25. Thus, transforming a nonreduced machine into its reduced form may improve its observability, and hence its testability.

4 CONCLUSION

Design for testability integrated in the communication software life cycle attempts to reduce the development costs and at the same time to improve the quality of a final product. To achieve these objectives, formal methods for testability evaluation and enhancement should be developed within an appropriate framework.

In this paper, we have proposed in the first part, a generic framework of design for testability where DFT activity is integrated in the communication software development process. A generic model for testability transformation based on modification of a given specification and testability measurement has been explained. In the second part, we have discussed the notion of testability in terms of controllability and observability. We have considered the FSM model of a protocol machine and proposed a classification method based on the formalized notions of controllability and observability. We have addressed one particular problem of DFT, the problem of analyzing the testability of a given module specified as a finite state machine. Transformations in this paper are not the main objective, but they are used to illustrate how testability can be enhanced by finding an augmentation of the given protocol behavior such that a newly obtained specification is more testable than the original one. A measure of testability of a protocol entity is assumed to be based on the shortest length of a test suite needed to achieve guaranteed coverage of certain faults.

Future work will address testability evaluation and enhancement of protocol specifications modeled by extended FSMs and written in FDTs such as SDL, Estelle, and LOTOS.

5 REFERENCES

- [ABF90] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design*, Computer Science Press, Oxford, England, 1990.
- [BoPe94] G. v. Bochmann and A. Petrenko, "Protocol Testing: Review of Methods and Relevance for Software Testing", ISSTA'94, ACM International Symposium on Software Testing and Analysis, Seattle, U.S.A., 1994, pp. 109-124.
- [BPY94] G. v. Bochmann, A. Petrenko, and M. Yao, "Fault Coverage of Tests Based on

- Finite State Models", the Proceedings of IFIP TC6 Seventh International Workshop on Protocol Test Systems, 1994, Japan.
- [Chow78] T. S. Chow, "Testing Software Design Modeled by Finite-State Machines", IEEE Transactions on Software Engineering, Vol. SE-4, No.3, 1978, pp.178-187.
- [DsFo91] R. Dssouli and R. Fournier, "Communication Software Testability", IFIP Transactions, Protocol Testing Systems III (the Proceedings of IFIP TC6 Third International Workshop on Protocol Test Systems), Ed. by I. Davidson and W. Litwack, North Holland, 1991, pp.45-55.
- [Free91] R. S. Freedman, "Testability of Software Components", IEEE Transactions on Software Engineering, Vol. SE-17, No. 6, June 1991.
- [FuBo91] S. Fujiwara and G. v. Bochmann, "Testing Non-deterministic State Machines with Fault Coverage", IFIP Transactions, Protocol Test Systems, IV (the Proceedings of IFIP TC6 Fourth International Workshop on Protocol Test Systems, 1991), Ed. by Jan Kroon, Rudolf J. Heijink and Ed Brinksma, 1992, North-Holland, pp. 267-280.
- [Jose78] J. Joseph, "On Easily Diagnosable Sequential Machines", IEEE Transactions on Computers, Vol. C-27, February, 1978, pp.159-162.
- [HoUl79] J. E. Hopcroft, J. D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley Publishing Company, Inc., 1979, 418p.
- [KDC94] K Karoui, R. Dssouli, O. Cherkaoui et A. Khoumsi, "Estimation de la testabilité d'un logiciel modélisé par les relations", publication #921, Département d'Informatique et de Recherche Opérationnelle (DIRO), Université de Montréal.
- [LBP94] G. Luo, G. v. Bochmann, A. Petrenko, "Test Selection based on Communicating Nondeterministic Finite State Machines using a Generalized Wp-Method", IEEE Transactions on Software Engineering, Vol. SE-20, No. 2, 1994, pp.149-162.
- [LPB94] G. Luo, A. Petrenko, and G. v. Bochmann, "Selecting Test Sequences for Partially-Specified Nondeterministic Finite State Machines", the Proceedings of IFIP TC6 Seventh International Workshop on Protocol Test Systems, 1994, Japan.
- [LeYa94] D. Lee and M. Yannakakis, "Testing Finite-State Machines: State Identification and Verification", IEEE Trans. on Computers, Vol. 43, No. 3, 1994, pp. 306-320.
- [Petr91] A. Petrenko, "Checking Experiments with Protocol Machines", IFIP Transactions, Protocol Test Systems, IV (the Proceedings of IFIP TC6 Fourth International Workshop on Protocol Test Systems, 1991), Ed. by Jan Kroon, Rudolf J. Heijink and Ed Brinksma, 1992, North-Holland, pp. 83-94.
- [PBD93] A. Petrenko, G. v. Bochmann, and R. Dssouli, "Conformance Relations and Test Derivation", IFIP Transactions, Protocol Test Systems, VI, (the Proceedings of IFIP TC6 Fifth International Workshop on Protocol Test Systems, 1993), Ed. by O. Rafiq, 1994, North-Holland, pp.157-178.
- [PDK93] A. Petrenko, R. Dssouli, and H. König, "On Evaluation of Testability of Protocol Structures", IFIP Transactions, Protocol Test Systems, VI, (the Proceedings of IFIP TC6 Fifth International Workshop on Protocol Test Systems, 1993), Ed. by O. Rafiq, 1994, North-Holland, pp.111-123.
- [PeBo95] A. Petrenko, G. v. Bochmann, "On Fault Coverage of Protocol Tests", submitted to Computer Networks and ISDN Systems on Protocol Testing, 1995.
- [PHK95] A. Petrenko, T. Higashino, and T. Kaji, "Handling Redundant and Additional States in Protocol Testing", IWPTS'95, France.
- [PYD94] A. Petrenko, N. Yevtushenko, R. Dssouli, "Testing Strategies for Communicating FSMs", IWPTS'94, Japan, 1994.
- [Star72] P. H. Starke, Abstract Automata, North-Holland/American Elsevier, 1972, 419p.
- [ShLe94] M. L. Sheu and C. L. Lee, "Simplifying Sequential Circuit Test Generation", IEEE Design and Test of Computers, Fall 1994, pp. 28-38.
- [TyBa75] T. Tylaska and J. D. Bargainer, "An Improved Bound for Checking Experiments that Use Simple Input-Output and Characterizing Sequences", IEEE Transactions on Computers, Vol. C-24, No.6, 1975, pp. 670-673.
- [Vasi73] M. P. Vasilevski, "Failure Diagnosis of Automata", Cybernetics, Plenum Publishing Corporation, New York, No.4, 1973, pp.653-665.

- [VLC93] S. T. Vuong, A. A. F. Loureiro, and S. T. Chanson, "A Framework for the Design for Testability of Communication Protocols", IFIP Transactions, Protocol Test Systems, VI, (the Proceedings of IFIP TC6 Fifth International Workshop on Protocol Test Systems, 1993), Ed. by O. Rafiq, 1994, North-Holland, pp.89-108.
- [VMM91] J. Voas, L. Morrell and K. Miller, "Predicting Where Faults Can Hide From Testing", IEEE Software, March 1991.
- [YePe89] N. Yevtushenko and A. Petrenko, "Fault-Detection Capability of Multiple Experiments", Automatic Control and Computer Sciences, Allerton Press, Inc., New York, Vol.23, No.3, 1989, pp.7-11.
- [YPD95] N. Yevtushenko, A. Petrenko, R. Dssouli, K. Karoui, and S. Prokopenko, "On the Design for Testability of Communication Protocols", IWPTS'95, France.

Acknowledgments

This work was partly supported by the HP-NSERC-CITI Industrial Research Chair on Communication Protocols at Université de Montréal, the NSERC Strategic and Individual Research Grant.

6 BIOGRAPHY

Rachida Dssouli received the Doctorat d'université degree in computer science from the Université Paul-Sabatier of Toulouse, France, in 1981, and the Ph.D. degree in computer science in 1987, from the Université de Montréal, Canada. She is currently an Associate professor at the University of Montréal. Her research area is software engineering and her research interests include requirements engineering, software evolution, protocol specification, testing and testability.

Kamel Karoui is a Ph.D. student of the Université de Montréal, Canada.

Alexandre Petrenko received the Dipl. degree in electrical and computer engineering from Riga Polytechnic Institute in 1970 and the Ph.D. in computer science from the Institute of Electronics and Computer Science, Riga, USSR, in 1974. Since 1992, he has been with the Université de Montréal, Canada. His current research interests include communication software engineering, protocol engineering, conformance testing, and testability.

Omar Rafiq received the Doctor ès-Sciences degree (1983) in computer science from the University of Bordeaux-I where he was an assistant professor from 1974 to 1978. He spent 10 years in banking, research and industry before joining the University of Pau in 1987 as professor. He is currently a full professor at the same university. His research interests include computer networks, protocol engineering and distributed processing.