ELSEVIER

Information and Software Technology 41 (1999) 729–743

INFORMATION AND SOFTWARE TECHNOLOGY

Communications software design for testability: specification transformations and testability measures

R. Dssouli*, K. Karoui, K. Saleh[1], O. Cherkaoui[2]

Département d'informatique et de recherche opérationnelle, Faculté des arts et des sciences, Université de Montréal, C.P. 6128, Succursale A, Montréal, Québec, Canada H3C 3J7

Abstract

To deal with the increased complexity related to the testing of communications software, we propose the integration and application of finite state machine based specification transformations and testability measures early in the communications software development process. Based on this integration, the testability of a given design is estimated and appropriate specification transformations are defined and applied iteratively to enhance the testability of the product implementation.

Keywords: Communications software; Design for testability; Finite state machine; Specification transformation; Testability measure; Testing

1. Introduction

Communications software play an important role in the successful operation of large and distributed computer systems applications. A wide range of safety-critical applications relies on the high reliability and availability of such software. Within the communications software engineering process, testing is a critical activity in which the quality and functions of the software implementation are checked against the software specification. Testing communications software and distributed systems in general is becoming an increasingly complex activity requiring lots of effort and time [1–3].

Despite enormous efforts devoted to developing testing techniques, serious problems with respect to the generation and application of tests remain to be solved [4]. Although improved testing methods had undoubtedly helped alleviating 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 requires that a software specification to contain mechanisms to make the implementation easier to test, thereby reducing development costs and increasing software reliability and maintainability.

Design for testability (DFT) is the process of introducing some features into a protocol specification to facilitate and enhance the testing process [5–7]. For a better understanding of communication software testability issues, we adopt the following general definition: "A software possesses the testability property if it includes facilities allowing the easy application of testing methods, and the detection and isolation of existing faults". Unfortunately, most existing protocols have been designed and documented without consideration for the testing requirements [6]. We argue that design for testability should start at the earliest possible phase of the protocol life cycle, i.e. at the specification level.

DFT may only be able to improve the effectiveness of the testing process under certain constraints. In particular, all basic elements of a finite state machine (FSM) based 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, for example, a "read-state" message, then new outputs, one per state, are added into the protocol. In this case, the problem of constructing an easily testable FSM becomes trivial. However, such solution seems expensive, since a protocol entity should support more messages than originally required.

In practice, this has not been accepted as a universal solution. To improve the protocol's testability, we can rely only on combinations of states and events for which the protocol behavior is not defined. In this paper, the problem of DFT at the specification level can be formulated

* Corresponding author. Tel.: + 1-514-343-6111; fax: + 1-514-343-5834.

E-mail address: dssouli@iro.umontreal.ca (R. Dssouli)

[1] On one-year sabbatical leave from the Department of Electrical and Computer Engineering, Kuwait University.

[2] Departement d'informatique, Université du Québec à Montréal.

0950-5849/99/$ - see front matter
PII: S0950-5849(99)00033-6

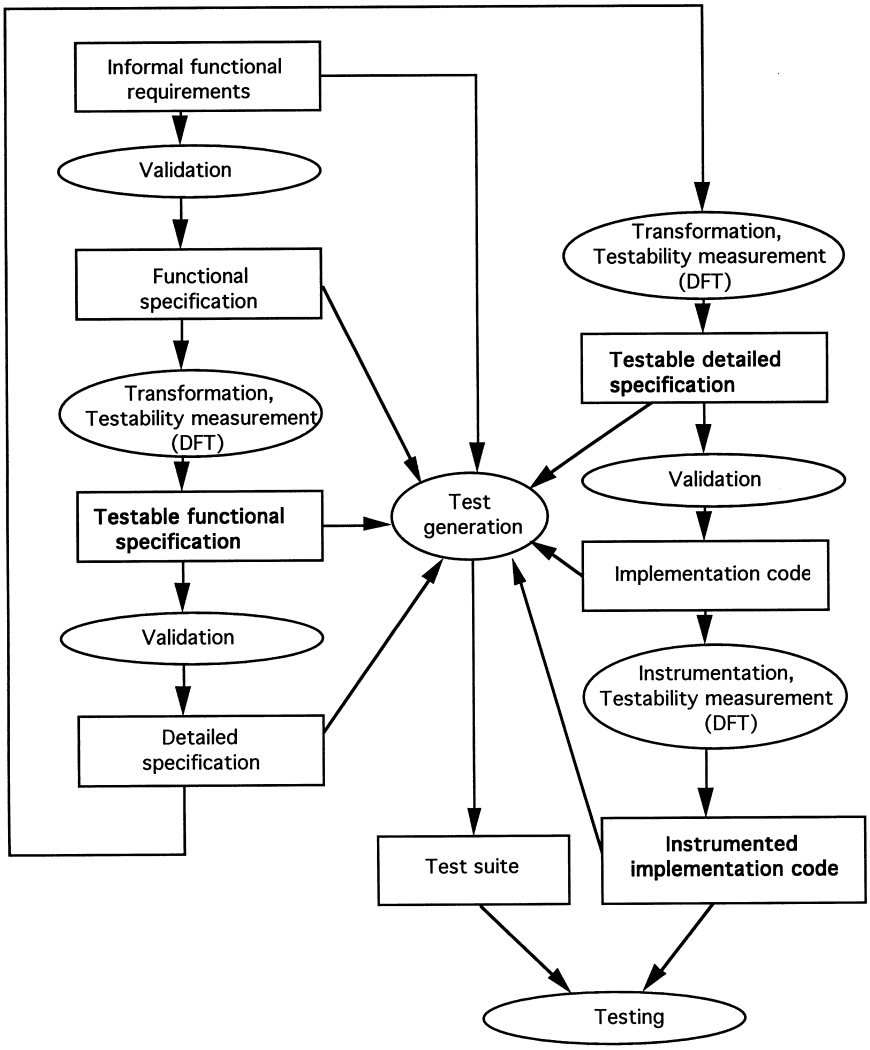

Fig. 1. Testability-oriented communication software development cycle.

as follows: given a protocol entity specification modeled by a finite state machine, we must apply one or more transformations to obtain a specification is more testable than the given one.

The set of properties that characterize testable software is not yet well-defined, but software designers have an intuitive idea as to what constitutes a testable software. Observability and controllability of software are widely acknowledged as two important attributes that influence software testability [8]. Although these attributes can be expressed in many different ways, it is often easier to describe them in terms of various criteria [5,9].

DFT aims at reducing the difficulties encountered in the testing process. Specifically, we list the following difficulties:

1. Selection of the test suite with minimal length and maximal coverage of faults: a test set may be infinite, in which case, total test coverage cannot be achieved unless a fault domain is restricted in advance.
2. Application of a test suite to the implementation under test: most test suites are derived from protocol specifications, and therefore are called abstract test suites. The efforts spent in adapting these test suites for specific implementations can diminish the advantage of using them.
3. Identification of the design schemes that lead to parts which are not-testable or difficult to test.
4. Analysis and interpretation of test results (traces): the problem of finding a matching trace in the specification requires the identification of each input and output in both the expected and the observed traces.
5. 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 further complicate the fault isolation problem.

In this paper, we address DFT at the specification level. We start by defining a generic framework for communications software design for testability integrated within the

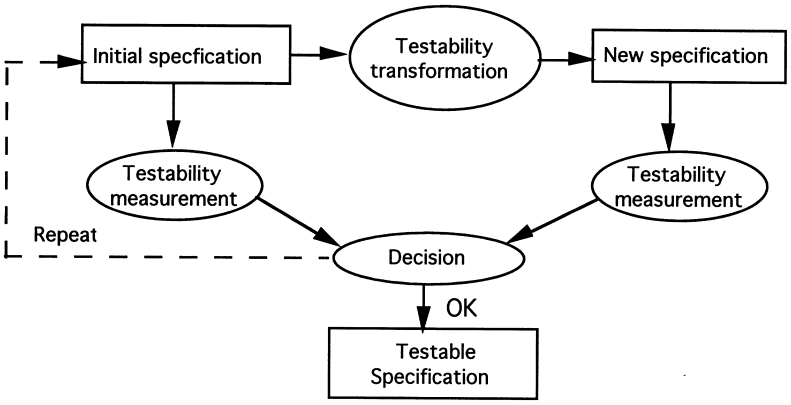

Fig. 2. Testability enhancement based on transformation and measurement.

protocol development life cycle activities. Then, we identify and define various testability measures, such as the controllability, fuzziness, state characterization and abstraction degrees, which together define a unified testability measure. Finally, we identify and define some transformations that may enhance the unified testability measure of a given specification. These transformations are related to three testability-enhancing factors, namely, specifiedness, minimality and determinacy of the specification.

The rest of the paper is organized as follows. In Section 2, we introduce a protocol life cycle testability framework and a testability-oriented classification and specification transformations based on three testability factors: specifiedness, minimality and determinacy. In Section 3, we present some basic notions and definitions of the FSM model for protocol specifications. In Section 4, we propose a new measure based on factors influencing the length of a test suite. In Section 5, we introduce and define specification-based transformations contributing to the enhancement of the testability of protocol implementations. Finally, in Section 6, we conclude the paper and discuss future research work in this area.

2. A testability framework

In this section, we propose a testability-oriented communications software development cycle based on an iterative process consisting of specification transformations and testability measurements. Finally, we present a testability-oriented specification classification based on the determinacy, specifiedness and minimality of the specifications.

2.1. A life cycle approach

We consider design for testability as an iterative process of modifying specifications and measuring the testability of the newly obtained specification. These transformations can be partially automated and integrated in the traditional communications software development cycle as shown in Fig. 1. The ultimate goal of these transformations is to produce a more testable specification/implementation [5].

Fig. 2 illustrates the use of certain testability measures that provide an indication on the quality of the end product. 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. These factors include [5,6]: (i) the formal specification language; (ii) the complexity of the specification; (iii) specification non-determinism due to concurrency; (iv) the degree of freedom allowed in the specification (i.e. completeness and unambiguity); (v) the level of abstraction that can be used for testing; (vi) the test architecture; and finally, (vii) the test strategy.

Our proposed framework seems simple to apply when testability problems are well identified and for which solutions exist. However, in general, we address below the following issues: (i) the selection of the appropriate specification transformations; (ii) the use of meaningful testability measures; and (iii) the termination of the transformation process.

Unfortunately, most existing protocols have been designed and documented without consideration for testing requirements [6]. Therefore, it is not realistic to expect that we could alter basic communication functions to improve testability of future protocol implementations. Thus, DFT 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) mode, 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 by finding a minimal augmentation of the specification such that all states become distinguishable at a minimal cost. This augmentation can be based on specifying “don’t care” transitions that do not generate additional end-to-end exchanges [10].

Earlier, Freedman [11], Voas et al. [12] and Petrenko et

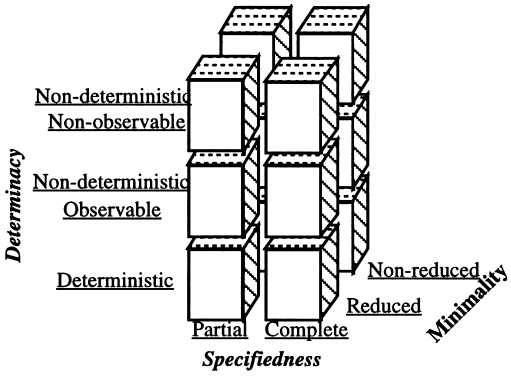

Fig. 3. Testability-oriented classification of FSM based specifications.

al. [7] proposed various testability measures focusing on a particular aspect of testability. However, they cannot be applied at each step of the communication software development process. A model for measuring software testability, for the purpose of comparison, would help designers in overcoming many difficulties. Ideally, this model should reflect all aspects of 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 aspect of testability [9].

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

In general, the selected transformations depend on the properties of the initial specification. This selection can be seen as a problem of specification classification. In the following section, we present a classification based on three properties: determinacy, minimality, and specifiedness.

2.2. Testability-oriented classification and specification transformations

The identification of testability problems of an FSM can be viewed as a classification problem for FSMs [13]. It is well known that non-deterministic FSMs are usually less testable than deterministic ones. Within non-deterministic FSMs, non-observable machines are less testable than the observable ones. The major factor that affects the testability in this case is the controllability aspect. However, non-reduced FSMs are less testable than reduced FSM (minimality) and the major factor of testability affected in this case is the observability aspect. Completely specified machines tend to be more testable than partially specified ones. A partial machine has fewer transitions than a complete one, however, the identification of its states may require longer input test sequences.

Moreover, designers may want to modify a specification by applying a set of transformations involving one or several of the following elements; input messages, output messages, transitions and output functions. Those transformations move the specification from one class to a more testable class. Certain transformations are well known, and their objectives are described in the literature. For example, a minimization algorithm can be viewed as a transformation method bringing an FSM from the class of non-reduced FSMs to the class of reduced FSMs. The addition of "read-state" message is another transformation. Yevtushenko et al. [10] give 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 Fig. 2. Depending on our pre-defined testability objectives, some heuristics can be developed to find minimal modifications of an FSM that meet the defined objectives.

The choice of a specific transformation is related to the freedom that the designer might have. It also depends on the reached step in the development process. In the case where a specification is a standard, and there already exists an implementation, then transformations are restricted to those that do not generate end-to-end exchanges (i.e. additional PDUs). 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.

We propose a three-dimensional classification of specifications, based on three testability properties: determinism, minimality and specifiedness (Fig. 3). This classification encompasses the most testable class (complete, reduced and deterministic specification) and the least testable class (partial, non-reduced, non-deterministic and non-observable specification).

3. Specification model: notions and definitions

In this section, we introduce some definitions and notions that will be used throughout this paper. Some of those definitions are presented in Ref. [14]:

Definition 1. A Finite State Machine (FSM) M is defined by a 5-tuple (Q, I, O, D_M, h), where: (i) Q is a set of n states ($n >= 1$), and $q_0 \in Q$ is the initial state of M; (ii) $I = \{i_1, i_2, \ldots, i_p\}$ ($|I| = p$) is a finite set of inputs of M; (iii) $O = \{o_1, o_2, \ldots, o_r\}$ ($|O| = r$) is a finite set of outputs, (iv) D_M is a set of defined transitions of M, (v), h is the behavior function: $h : D_M \rightarrow P(Q \times O)$ where, $P(Q \times O)$ is the powerset of $Q \times O$.

Definition 2. h can be extended to I_M^* (all input sequences accepted by M), $h : Q \times I_M^* \rightarrow P(Q \times O^*)$.

Definition 3. $h^1(q,i) = \{q' | \exists o \in O : (q', q) \in h(q,i)\}$ and $h^2(s,x) = \{o | \exists q' \in Q : (q', o) \in h(q,i)\}$, are called transition and output function, respectively.

Definition 4. $M' = (Q, I', O, h', D'_M)$ is a *projection* of $M = (Q, I, O, H, D_M)$ on $I' \subseteq I$ if $D'_M \subseteq D_M$ and $h' \subseteq h$, M' is a unary projection of M if M' is a projection of M and $|I'| = 1$, $M_1, M_2, \ldots, M_p$ are the unary projections of M on $i_1, i_2, \ldots, i_p$, respectively.

Definition 5. M is a *completely specified* machine if $D_M = Q \times I$, M is a *partially specified* machine if $(Q \times I) - D_M \neq \varnothing$.

Definition 6. A "don't care" transition is an undefined transition that can be assigned to any state and any output.

Definition 7. $i_1 \ldots i_k \in I_M^*$ is an *acceptable input sequence* for the state q if there exist k states $q_1, \ldots, q_k$ from Q: $h^1(q, i_1) = q_1$ and $h^1(q_j, i_{j+1}) = q_{j+1}$, $q = 1, \ldots, k-1$. I_q^* denotes a set of all input sequences acceptable for the state q.

Definition 8. A sequence is said to be *homogeneous* if it is a sequence of the same symbol, i.e. $i^r = ii^{r-1}$, $r \geq 1$.

Definition 9. q_i and q_j are *equivalent* if and only if $I_i^* = I_j^*$ and $h^2(q_i, \omega) = h^2(q_j, \omega) \forall \omega I^* \in M$. Two machines M_1 and M_2 are equivalent if and only if their initial states are equivalent.

Definition 10. q_i and q_j of FSM M are *distinguishable* if $\exists \alpha \in I_i^* \cap I_j^* : h^2(q_i, \alpha) \neq h^2(q_j, \alpha)$. If all the states of M are distinguishable then M is *reduced*.

Definition 11. q_i and q_j are *compatible* if and only if $\forall \omega \in I_i^* \cap I_j^*, h^2(q_i, \omega) = h^2(q_j, \omega)$ and $h^1(q_i, \omega)$ and $h^1(q_j, \omega)$ are comparable.

Definition 12. State q_i of M_1 is said to *cover*, or contain, state q_i of M_2 if and only if every input sequence applicable to q_j is also applicable to q_i, and its application to both M_1 and M_2 when they are initially in q_i and q_j, respectively, results in identical output sequences whenever the outputs of M_2 are specified. A machine M_1 covers machine M_2 if and only if, for every state q_j in M_2, there is a corresponding state q_i in M_1: q_i covers q_j.

Definition 13. A machine M is *deterministic* if $|h(q,i)| = 1 \forall (q,i) \in D_M$.

Definition 14. A machine M is *completely connected M* if for every state q in Q, there is one transfer sequence $v \in V$ such that $q \in h^1(q_1, v)$.

4. Testability measure

The length of the complete test suite can be used as a good testability evaluation of FSM-based models [7]; the length depends on the properties of the reference specification. For an FSM M with n states, d defined transitions, and an implementation M' with $m \geq n$ states, a complete test suite can be expressed as [15]:

$$\mathrm{TS} = V \cdot I^{fm-n+1} \cap I_M^*) \cdot W \tag{1}$$

where $(V \cdot I^{fm-n+1} \cap I_M^*) \cdot W)$ is the concatenation of the sets $V, I^{fm-n+1} \cap I_M^*$ and W, f is the fuzziness degree, I the set of inputs, V is the reachability set, $I^{fm-n+1} \cap I_M^*$ is the set of all input sequences acceptable by M having a length up to $(fm - n + 1)$, and W is the characterization set.

In order to estimate the specification testability, Petrenko [7] used the upper bound of the complete test suite length for completely specified and deterministic FSM: $\mathrm{Length(TS)} \leq mn^2 p^{m-n+1}$. The following testability estimation was proposed: $T = |O| / mn^2 I^{m-n+1}$. This estimation cannot be applied to partially non-deterministic FSM (PNFSM). For this FSM class, the length of a complete test suite is not estimated yet. The testability is then not computable unless the complete test suite is generated. For these reasons, we propose a less restrictive new testability measure.

4.1. A new testability measure

As expressed in Eq. (1), the complete test suite combines different testability elements contributing to the length of the complete test suite. We propose an individual evaluation of each of those elements that are collected in a unique vector that we call the testability vector:

$$\mathrm{TV}(M) = \langle C(M), f(M), W(M), AB(M) \rangle \tag{2}$$

In the following, we evaluate each element of TV(M) and focus only on observable and initially connected machines $((\forall q \in Q, \forall i \in I, \exists! o \in O : o = h^2(q,i))$ and $(\forall q \in Q, \exists \alpha I \in_M^* : q \in h^1(q_o, \alpha)))$.

4.1.1. The controllability degree (C(M))

Let V be the reachability set of a completely connected FSM M. V constitutes a skeleton of test suites and guarantees that every transition from every state will be tested. The machines may possess several reachability sets. The effort required to control all the machine's states depends on the chosen reachability set. To characterize the controllability of a given machine we use a minimal size reachability set. The size of V depends on two factors: the length $L(v)$ of the transfer sequences v required to force the machine to reach a certain state and the number of possible output sequences obtained by applying v. We integrate the above factors into a single formula based on the notion of the weight of the reachability set constructed for the observable form of the machine [19]: $\omega(V) = \sum_{v \in V} L(v) |h^2(q_1, v)|$.

In the best case, for every state, except for the initial state, there exists just a single input that takes the machine to this state, then $\omega(V) = n - 1$. For the worst case, the maximal length of a transfer sequence may reach $n - 1$, and the output produced by such sequence may be equal to $A^{|v|}_{|v|;|O|} = |v|\cdot|O|!/||v|\cdot(|O| - 1)|!$ (the arrangement of the length $|v|$ of output symbols). Thus, $\omega(V)$ has the following lower and upper bounds: $\omega_1(V) = n - 1 \leq \omega(V) \leq \omega_u(V) = \sum_{p=1}^{n-1} p\cdot A^p_{p\cdot|O|}$.

Let $c_1(V) = 1/\omega_u(V) c_u(V) = 1/\omega_1(V)$, and $c(V) = 1/\omega_{\min}(V)$, where $\omega_{\min}(V)$ is the minimal value of the weight among all its possible reachability sets. We define the controllability $C(M)$ of M as:

$$C(M) = \frac{c(V) - c_1(V)}{c_u(V) - c_1(V)}, \text{ and } 0 \leq C(M) \leq 1. \quad (3)$$

From the previous equations, we can define the controllability of an initially connected deterministic FSMs as:

$$c_u(V) = 1/n - 1, \qquad c_1(V) = 2/n(n-1), \qquad \textit{and}$$

$$C(M) = [n(n-1)c(V) - 2]/(n-2).$$

4.1.2. The fuzziness degree (f(M))

Given an arbitrary observable FSM M, we compute a partition Π of the set Q into subsets $\Pi = \{Q_1, \ldots, Q_k)$ such that every subset includes only pairwise distinguishable states. The number of subsets in the minimal partition $\Pi_{\min}$ is called the fuzziness of M [10]. This factor is represented by f in Eq. (1) and indicates the difficulties to distinguish states. We propose to evaluate the fuzziness degree as follows:

$$f(M) = (n - \min(k))/(n-1) \text{ with } 0 \leq f(M) \leq 1. \quad (4)$$

In the worst case, all states are not distinguishable $k = n$, thus $f(M) = 0$. In the best possible case, all states are distinguishable $k = 1$, thus $f(M) = 1$.

4.1.3. The state characterization degree (W(M))

An FSM usually possesses several characterization sets. For each subset of states $Q_i(|Q_i| > 1)$ including distinguishable states, we can find a set of input sequences W_i that uniquely distinguishes the states of Q_i. Let W be the union of the sets W_i, the number $|W|$ of sequences in W and their total length $L(W)$ characterize the effort required to identify distinguishable states of M. We integrate $L(W)$ and $|W|$ into a single formula based on the notion of the weight $\omega|W|$ of the characterization set [19]: $\omega(W) = |W|\cdot L|W|$.

In the best possible case, there exists just a single input that distinguishes all the states of the machine, then $\omega|W| = 1$. In the worst case, for reduced partial machine, each pair of states is distinguishable by a unique sequence w_i:

$$|W| = C_n^2 = n(n-1)/2, \text{ and } L(W) = \sum_{i=1}^{n(n-1)/2} L(w_i)$$

$$= \sum_{i=1}^{n(n-1)/2} \frac{i(i-1)}{2}.$$

For non-reduced machines, w_i cannot always exist, thus: $\omega(W) = L(W) = L(w_i) = +\infty$.

Then the distinguishability $W(M)$ of a 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:

$$0 \leq W(M) = 1/\omega_{\min}(W) \leq 1. \quad (5)$$

4.1.4. The abstraction degree (AB(M))

The specification and the implementation can be considered as different representations of the same behavior. The only difference is their belonging to different levels of abstraction. In general, an implementation (m states) has more states than a specification (n states). We propose the following estimation of the abstraction:

$$\text{AB}(M) = n/m \textit{ and } 0 < AB(M) \leq 1. \quad (6)$$

4.2. A unified testability measure

Each of the previous individual measures is bounded by 0 and 1. This means that $\text{TV}(M) = \langle 1, 1, 1, 1, 1\rangle$ is the best case, and its testability vector is called BTV (Best Testability Vector). We propose derive a unified testability measure by computing and distance between TV(M) and BTV. We call this measure the UTM(M) (Unified Testability Measure), and we define it as follows:

$$\text{UTM}(M) = 1 - \frac{\sqrt{(1 - C(M))^2 + (1 - f(M))^2 + (1 - W(M))^2 + (1 - \text{AB}(M))^2}}{2}. \quad (7)$$

The most interesting thing about the UTM is that, it does not require any strong hypothesis such as completeness or determinacy.

5. Testability-directed specification transformations

In the DFT activities, designers might want to modify and transform a specification. In the FSMs model, the designer can act on the states, the inputs, the outputs or the transitions. The transformations may involve one or several of these elements. Each transformation attempts to change the testability class of an initial specification and produces a specification included in a better testability class. A class represents a set of all the specifications having the same properties (Fig. 2). There are three directions for obtaining testable transformations (see Section 2): determinism,

specifiedness and minimality. Each direction is divided into two or more classes.

The choice of a specific transformation depends on the freedom that the designer might have. We suppose that a designer can complete the specification by adding some unspecified transitions. This operation transforms the initial FSM M to M' covering M. Each additional transition extends the language accepted by M and adds at least one data flow path to M. The number N of paths added is difficult to evaluate and can be infinite. Instead of N, we can compute G, the number of the independent paths added as defined by McCabe [18]. We need to compute, for each transition added, the number of independent paths V_j added. We consider a submachine M_{qj} of M, $M_{qj} = (Q_j, I_j, O_j, D_{M_{qj}}, h_j)$ is constructed by considering only the states and the transitions reachable from q_j. q_j is the end state of the transition to add, q_j is the initial state of M_{qj}:

$Q_j = \{q \in Q : \exists \omega \in I_M^* \text{ and } h^1(q_j, \omega) = q\}$, $I_j \subset I, O_j \subset O, h_j : Q_j \times I_{Mqi}^* \rightarrow P(Q_j \times O_j^*)$ and $D_{M_{qj}} \subset D_{M_{qj}}$. We can then use the formula defined in Ref. [18] to compute $V_j = e_j - n_j + p_j$, where e_j is the number of defined transitions in M_{qj} (excluding self loops), n_j the number of states in M_{qj}, and p_j the number of connected component in M_{qj}. We suppose that from any state q (q' or q_j) of M (M' or M_{qj}) we can find a path leading to the initial state q_0 (q'_o or q_j). From all the V_j we can derive the following formula for G:

$$G = \sum_{j}^{\text{number of transitions added}} V_j. \tag{8}$$

5.1. Specifiedness transformation

In some applications, it is known that some input sequences are not important or will never be applied. This may be the case, for example, if the input sequences are obtained from the output of another machine. The fact that a machine will never receive certain input sequences often means that some of the transitions will never be traversed. Then in a physical realization of the machine it does not matter where those ("don't" care) transitions lead, or what output values are assigned to them. The designer is free to arrange these transitions to provide the best realization. Therefore, there are many possibilities to assign the "don't care" transitions. Each of them influences directly the length of the complete test suite.

In the following, we propose two specifiedness transformations: S-trans and H-trans. They are based on the assignment of the unspecified transitions and particularly the "don't care" transitions. Usually, when a specification contains "don't care" transitions, means that the specification allows any further behavior starting from a given state under an undefined input.

5.1.1. Self loop transformation (S-trans)

An implementation is generally completely specified if it accepts all the inputs independently of its state. A designer can simplify the implementation task by completing the specification. The most used technique consists of assigning a self-loop to all the unspecified transitions. The outputs of the new transitions are chosen such that we obtain a better characterization set W (better $W(M)$). This transformation is called S-trans, it completes the specification, but increases the number of transitions and may increase the testing cost. In the Appendix, we propose an algorithm for S-trans.

Fig. 4, illustrates the use of S-trans. It starts from a specification (Fig. 4(a)) where the extraction of the test suite (with w-method) is impossible (Table 1). Then, we generate a more testable specification (Fig. 4(b)). There is a set of unspecified transitions in the initial specification: A, B, D, and E. S-trans completes them such that we enhance $W(M)$:

$h^1(A, a) = A, \quad h^2(A, a) = 1, \quad h^1(A, b) = A, h^2(A, b) = 0;$

$h^1(A, d) = A, \quad h^2(A, d) = 1;$

$h^1(B, d) = B, \quad h^2(B, d) = 1;$

$h^1(D, a) = D, \quad h^2(D, a) = 1;$

$h^1(D, d) = D, \quad h^2(D, d) = 1;$

$h^1(E, b) = E, \quad h^2(E, b) = 0.$

In this case, the new generated specification becomes more testable: The UTM (formula (8)) passes form 0.34 to 0.46. The factors characterizing the new generated FSM are summarized in the Table 1. All states become distinguishable by $W = \{bba\}$, the reachability set is $V = \{\varepsilon, ca, cc, ccb, c\}$ and the test suite: TS = {*abba*, *caabba*, *ccabba*, *ccbabba*, *cabba*, *bbba*, *cabbba*, *ccbbba*, *ccbbbba*, *cbbba*, *cbba*, *cacbba*, *cccbba*, *ccbcbba*, *cabba*, *dbba*, *cadbba*, *ccdbba*, *ccbdbba*, *cdbba*} and its length is $L(\text{TS}) = 112$.

From the tester point of view, S-trans is not always profitable. It can increase the cost of testing by increasing the number of transitions to test. However, this transformation preserves the initial specification structure, the number of states and the controllability degree of the machine.

5.1.2. Homogeneous transformation (H-trans)

In a previous work [10], we have proposed to complete an FSM by assigning its "don't care" transitions. This technique tries to distinguish the maximal number of non-completely specified states by an homogeneous sequence of inputs (Proposition 1). This strategy gives good results whenever an homogeneous sequence exists (Proposition 2). It ensures that $|W| = 1$ and leads to a short test suite [10].

Proposition 1. Let M_j be a unary projection of an FSM M on i_j (see Section 3), and M_j is a partial FSM. For every non-specified state q of M_j, we can find an homogeneous sequence that distinguishes q from all the other states of M.

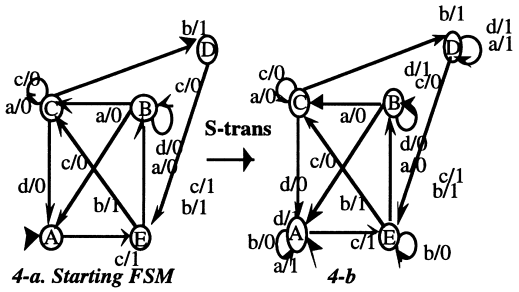

Fig. 4. Example of S-transformations application.

Proof. The proof of this theorem can be found in Ref. [10]. □

The transformation proposed in Ref. [10] tries to complete a partial FSM M by adding the "don't care" transitions to one of its unary projections $M_1, M_2, \ldots, M_p$. It is always possible to assign the don't care transitions such that we make the non-completely specified states distinguishable by a unique homogeneous sequence. This transformation does not guarantee that all states of the completed machine are distinguishable by a unique homogeneous sequence. In order to insure that, we introduce a new condition for the existence of a unique homogeneous distinguishing sequence. This condition states that M must have at least one reduced unary projections (see Proposition 2).

Proposition 2. Let $\mathrm{Pr} = \{M_1, M_2, \ldots, M_p\}$ be the set of unary projections of an FSM M on $i_1, i_2, \ldots, i_q$, respectively. M possesses an homogeneous distinguishing sequence if and only if there exists at least a $M_j \in \mathrm{Pr}$ such that M_j is reduced (no equivalent states).

Proof. An FSM is reduced if none of its states accept the same set of input/output sequences.

Suppose that none of the elements of Pr is reduced, this means that for every $M_j \in \mathrm{Pr}$ there exists at least two states that responds in the same manner for all the sequences of input sequences i_j^*, and cannot be distinguished by an homogeneous sequence i_j^*. □

In order to consider this condition and obtain a machine that has a behavior close to the initial one, we have extended the algorithm proposed in Ref. [10], and called it H-trans. In this section, we give some basic steps of H-trans. The complete algorithm can be found in the Appendix:

- if the partial FSM M admits some unary reduced projections, we complete the unary projection producing the shorter distinguishing homogeneous sequence ($W(M)$) (formula (6));

  * if there is a set of projections that have an homogeneous distinguishing sequence with the same length, we choose one projection such that the behavior of the obtained FSM M' is close to M (the smallest G as in formula (8));

  — if there is still a set of projections that have the same $W(M)$ and G, we choose the one that ameliorate the controllability of the machine M ($C(M)$);
  # if there is still a set of projections inducing the same $W(M)$ and $C(M)$ and G (formulas (3), (6) and (8)), we choose to complete the projection that has the minimum number of non-specified transitions,

  # otherwise, we randomly choose a projection.

- else, H-trans is not applicable.

To generate a completely specified specification, we can extend H-trans by adding self loops to what is left undefined after the application of H-trans. We call this new version of H-trans *$\mathscr{H}$-trans*.

If we apply *$\mathscr{H}$-trans* to the example of Fig. 4(a). We notice that the two possible reduced unary projections are M_2 and M_4 (projections of M on b and d, respectively). However M_1 and M_3 (projections of M on a and c, respectively) are not reduced. M_2 and M_4 have the same $W(M)$, and we choose to complete M_2 because it has a better G. The states A and E are completed: $h^1(A,b) = C, h^2(A,b) = 0$ and $h^1(E,b) = A, h^2(E,b) = 0$ (Fig. 5), then we complete the specification by adding the self-loops:

$$h^1(A,a) = A, \quad h^2(A,a) = 1;$$

$$h^1(A,d) = A, \quad h^2(A,d) = 1;$$

$$h^1(B,d) = B, \quad h^2(B,d) = 1;$$

$$h^1(D,a) = D, \quad h^2(D,a) = 1;$$

$$h^1(D,d) = D, \quad h^2(D,d) = 1.$$

This transformation increases the number of defined transition (from 13 to 20). However, it improves the reachability set $V = \{\varepsilon, ca, b, bc, c\}$, and produces a unique homogeneous distinguishing sequence $W = \{bbb\}$ and a test suite TS = {*abbb*, *caabbb*, *babbb*, *bbabbb*, *cabbbb*, *bbbbbb*, *cbbbb*, *cacbbb*, *bcbbb*, *bbcbbb*, *ccbbb*, *dbbb*, *cadbbb*, *bdbbb*, *bbdbbb*, *cdbbb*} with a length $L(\mathrm{TS}) = 86$ and

Table 1
Application of the testability transformations

Initial spec.	S-trans	H-trans	C-trans	K-trans	
Figure no.	Fig. 3(a)	Fig. 3(b)	Fig. 3(c)	Fig. 9	Fig. 13(c)
No. of transitions	13	20	20	12	12
No. of states	5	5	5	3	3
$C(M)$	1/6	1/6	4/9	0	1
$f(M)$ 3/4	1	1	1	1	
$W(M)$	0	1/3	1/3	1/2	1/2
AB(M)	1	1	1	1	1
L(TS)	Hard	112	86	41	38
UTM(M)	0.34	0.46	0.57	0.44	0.75

UTM = 0.612 (formula (8)). H-trans does not change the number of states and the initial structure of the specification remains the same. In Table 1, we represent the testability vector of this new specification and we compare it with the initial one.

5.2. Minimality transformations

Minimality transformations eliminate redundant behavior of the FSMs. In the case of completely specified FSMs, these transformations merge equivalent states and lead to a unique equivalent FSM. Minimization algorithms are well known and easy to implement [14]. However, for partial FSMs (PFSMs), these algorithms are more difficult to develop. We can merge states having exactly the same behavior but not compatible states (having the same behavior on common specified inputs). There are some algorithms that merge compatible states by transforming them into equivalent states. This is done by adding to each of the compatible states the common non-specified behavior. Those algorithms lead to a non-unique reduced FSM, and sometimes to non-deterministic ones. The behavior of the new generated FSM is an extension of the initial FSM.

The problem of minimality transformation algorithms for PFSMs amounts to the choice of the best reduced FSM. For this purpose, we propose two general criteria:

- the behavior of the new generated FSM must be as close as possible to the initial FSM and
- the new generated FSM must have a better testability.

5.2.1. Choice of the reduced machine

To reduce an incompletely specified machine M [14], we determine all the possible pairs of compatible states, then we gather them in subsets C_i. Each C_i contains the maximum pairwise compatible states. The set $C_{\max} = \{C_j, 1 \le j \le k\}$ is called the set of maximal compatibles. From $C_{\max}$, we can construct a set P of partitions P_q on Q where:

$$P_q = \{P_{q,i} : P_{q,i} \subseteq C_j \text{ and } \cap P_{q,i} = \varnothing \text{ and } \cup P_{q,i} = Q \text{ and } i1 \neq i2 \Rightarrow \phi j P_{q,i1} \cup P_{q,i2} ! C_j\} \quad (9)$$

All the partitions P_q obtained by the above formula can be represented by a machine whose states are compound states

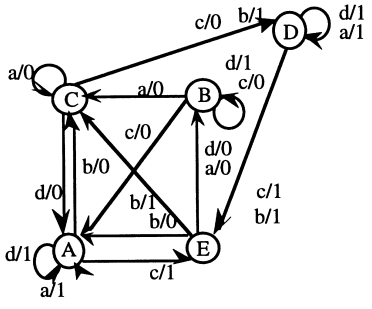

Fig. 5. $\mathscr{H}$-*trans* transformation of the specification in Fig. 4(a).

constructed from all the states of the subsets $P_{q,i}$ and each transition $q_1 - a|b \rightarrow q_2$ of the initial machine is replaced by $P_{q,i} - a|b \rightarrow P_{q,j}$ for $q_1 \in P_{q,i}$ and $q_2 \in P_{q,j}$.

For the specification in Fig. 6, we found that the following pairs of states are compatible: (*AB*), (*AC*), (*AD*), (*BC*), (*BD*), (*BE*), (*CD*), (*CF*) and (*EF*), thus $C_{\max} = \{(ABCD), (BE), (CF), (EF)\}$. There are two choices for merging state *F* for example: with the state *C* or *E*. Each of these two possibilities leads to a machine that is a minimization of *M*. The partitions $P_1 = \{(ABCD), (EF)\}$ or $P_2 = \{(AD), (BE), (CF)\}$ are partitions over *Q* and can be represented by the machines M_1 and M_2 (Figs. 7 and 8). The behaviors of M_1 or M_2 cover *M*, and both M_1 and M_2 are minimal machines.

The minimization of a partially specified machine can produce a non-deterministic machine. For example, M_1 is not deterministic (Fig. 7). To choose one machine from the set of the produced minimal machines and to avoid the problem of non-determinism, we propose the following guidelines:

Let *P* be the set of partitions representing the minimal machines of *M*:

- We extract from *P* the subset of the partitions representing deterministic machines. We eliminate non-deterministic machines because they are hard to test.
- We extract from this set a subset of machines whose behaviors are close to the initial one. The choice of those partitions can be done by comparing the behavior of the machines (the smallest *G*; formula (8)).
- We choose the machine having a better testability vector elements (specially *W*(*M*) and *C*(*M*): formulas (3) and (6)).
- If there is still a set of selected machines, we can take the machines with the minimum number of states.

Finally, if we can not isolate a unique machine, we can choose randomly, from the remainder set, a machine.

If we apply those guidelines on the example in Fig. 6, with the first guideline, we eliminate M_1 because it is a non-deterministic FSM. In the following section we present an example of minimality transformations.

5.2.2. Compatible transformation (C-trans)

C-trans transforms an initial specification by completing and merging some of its compatible states. This can be done by making the compatible states equivalent. C-trans completes the undefined behavior of each compatible state by adding to it the specified behavior of the other compatible states. Also, C-trans replaces some states (compatible) by states which cover them. Whenever a state q_i cover a state q_j from the same FSM, then q_i can be used to do the job of both states, and q_j can be removed. C-trans uses the previous guidelines to choose the reduced machine. Finally, C-trans minimizes the new FSM by merging the new equivalent states.

The obtained machine can be extended by completing the remaining unspecified transitions by assigning them self

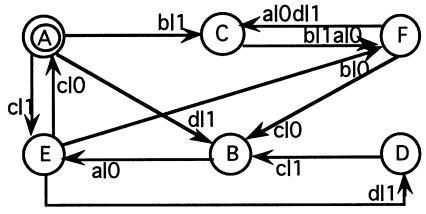

Fig. 6. Example of machines containing compatible states.

loops. This new version of C-trans is called C-trans. If we apply C-trans: Initial FSM—C-trans → New FSM, the new FSM is completely specified and reduced. It contains less states, but covers all the behavior of the initial one. If we test all the transitions of the new generated FSM we are sure to test all the behavior of the initial FSM.

In the Appendix, we propose an algorithm for C-trans that completes partial specifications by adding iteratively each non-defined input. This algorithm is based on the guidelines presented earlier.

If we apply C-trans to the example of Fig. 4(a), we notice that *A* and *D* are compatible, *B* and *C* are compatible, *B* and *E* are compatible. We merge these compatible states by adding the following transitions: *A* and *D* are merged in one state *AD* by adding $h^1(A,b) = E, h^2(A,b) = 1$, *B* and *C* are merged in one state *BC* by adding $h^1(B,d) = A, h^2(B,d) = 0$, *B* and *E* are merged in one state *BE* by adding $h^1(E,b) = A, h^2(E,b) = 1, h^1(B,d) = B, h^2(B,d) = 0$. $C_{\max} = \{(AD), (BC), (BE)\}$, and there are two choices to merge state *B*: with the state *C* or *E*. This leads to two machines that are minimizations of *M*. The partitions $P_1 = \{(AD), (BC), (E)\}$ or $P_2 = \{(AD), (BE), (C)\}$ are partitions over *Q* and can be represented by a machine M_1 or M_2 (formula (9)). If we apply the previous guidelines, we keep M_1 because M_2 is non-deterministic. We complete M_1 by adding self loops to states *AD*, and *E* remains unspecified: $h^1(AD,a) = AD, h^2(AD,a) = 1$; $h^1(AD,d) = AD, h^2(AD,d) = 1, h^1(E,b) = E, h^2(E,b) = 0$. The resulting machine is illustrated in Fig. 9. By adding those transitions, we derive the new reachability set $V = \{\varepsilon, b, bc\}$, the characterization set $W = \{bb\}$, the test suite TS = {*abb*, *babb*, *bcabb*, *bbbb*, *bcbbb*, *cbb*, *bccbb*, *dbb*, *bdbb*, *bcdbb*}, $L(\text{TS}) = 41$ and UTM = 0.499. The testability vector is described in Table 1. This transformation may change the initial structure of the specification.

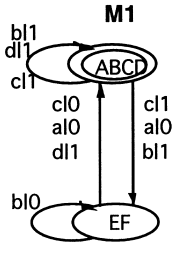

Fig. 7. Machine M_1, minimization of *M*.

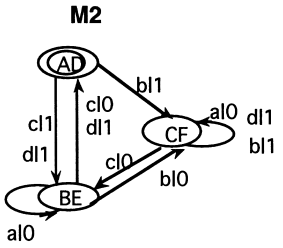

Fig. 8. Machine M_2, minimization of *M*.

5.3. Determinacy transformations

Non-deterministic protocols are difficult to test, two executions of the same system may produce different, but nevertheless valid sequence of events. Therefore it is much more difficult to detect errors, and to test all possible scenarios. Generally, even if the specification is non-deterministic, the non-determinism is removed in the implementation. There are two classes of non-determinism: observable and non-observable. It is easier to test an observable non-deterministic FSM than a non-observable non-deterministic one. The transformation from the non-observable to observable non-deterministic FSM is well known [16] (see Fig. 10). However, this transformation does not eliminate non-determinism. Nondeterminism has many origins [17], and in general, is due to the lack of information. For this class, we can remove the non-determinism by refining a specification, i.e. by adding some specific details to the specification. The refinement can be done either by adding states or transitions.

5.3.1. State refinement

A state in a specification is a label which is an abstraction of specific values for all local and global variables and the contents of all message channels. The state of a specification can be refined by considering (in its description) the values of one or more specific global or local variables. This refinement splits the state to a set of its instantiations, each of them is characterized by the same label as for the initial state and the value of the variables considered. In the case of a non-deterministic state, such refinement can break the non-determinism by assigning each branch of the non-deterministic transitions to an instantiation of the state. This transformation, increases exponentially the number of states. Fig. 11, illustrates this situation by representing

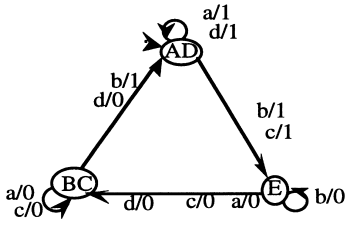

Fig. 9. C-trans transformation of the specification in Fig. 4(a).

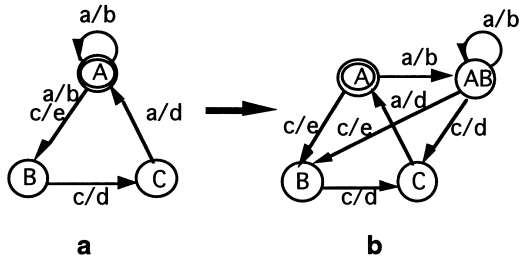

Fig. 10. Example of observability transformation.

two copies of the same protocol: INRES Initiator. Fig. 11(a) represents the Initiator without specifying the number of transmissions of CR-PDU. This specification is non-deterministic: from state Wait, when the initiator receives a time-out it can either send an IDISind primitive and goes to the Disconnected state or CR and remains in the same state. Fig. 11(b) is a state refinement of the INRES Initiator (Fig. 11(a)), it removes the non-determinism of the state Wait. In this case, when the Initiator did not receive the CC, it can send another CR before sending an IDISind. The introduction of the variable describing the number of possible retransmission of the CR, splits the Wait state into two states: Wait_1 and Wait_2. This refinement eliminates the non-determinism at this state.

5.3.2. Transition refinement

A transition is defined by its initial state, an input/output label, and the final state. In the case of non-determinism, a unique input event can lead from an initial state to more than one final state. In some cases, we can remove this non-determinism by refining the transition. The refinement consists of assigning some specific parameters to the input events. This operation splits the transition to a set of transitions, each is characterized by the value of the parameters considered. Fig. 11(b), illustrates the situation where there is only one type of acknowledgement the AK-PDU (without parameters). This is a non-deterministic situation, because the AK does not inform the initiator if the data has been received correctly. The initiator can be confused and consider either that it is a positive acknowledgement and moves to the Connected state to transmit the next data or negative acknowledgement and remains in the Sending state to retransmit the same data. We can refine this input event (AK) by assigning to it a Boolean parameter $r(r, \neg r)$ indicating if the acknowledgement is positive or negative. This refinement eliminates the non-determinism in the state Sending (Fig. 12).

5.3.3. Deterministic transformation (D-trans)

Determinacy transformations are composed of two sub-transformations:

- transformation of the non-observable non-deterministic FSM to an observable non-deterministic FSM;
- the transformation from a non-deterministic FSM to a deterministic FSM.

The first transformation is well known [16] and can be completely automated. However, the transformation from non-deterministic to deterministic FSM, cannot be done automatically. We saw that some kinds of non-determinism can be removed by refining the states or the transitions of the specification. Those refinements can be considered as manual transformations, they need the designer's intervention to introduce more specific information. If we have the choice of refining either the states or transitions, we must refine transitions since this refinement does not increase the complexity of the specifications.

5.4. Combined transformations (K-trans)

Two or more testable transformations can be combined in order to produce better results. We propose a new testable transformation that is a combination of the previously presented ones (D-trans, C-trans, H-trans and S-trans). We call this transformation K-trans. It starts from a partial non-reduced FSM M_0, the applies D-trans yielding M_1, then applies C-trans on M_1 yielding M_2,

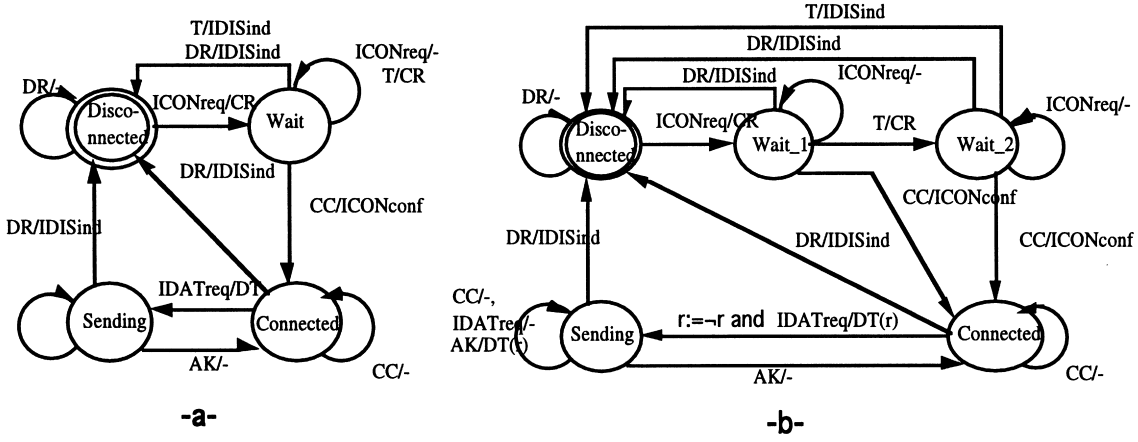

Fig. 11. The INRES Initiator and its state refinement.

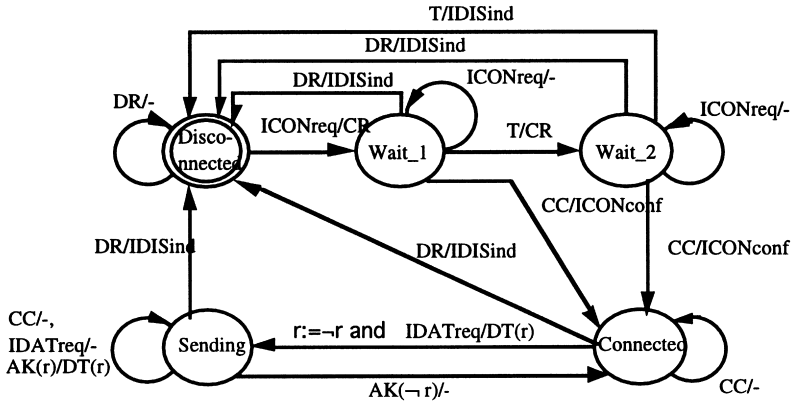

Fig. 12. Transition refinement of the INRES Initiator specification.

then applies H-trans on M_2 yielding M_3, and finally applies S-trans on M_3 yielding M_4. We call this combined transformation K-trans (M'_o—K-trans $\rightarrow$ M_3). K-trans is a path of transformations leading to a better testable class of the specification. If one or more of the component of K-trans is not applicable, we continue with the next transformation. A more detailed algorithm of K-trans is given in the Appendix.

To illustrate K-trans, we have applied it to the specification of the Fig. 4(a). The specification we obtained is shown in Fig. 13(c). We start from an initial specification which is hard to test (Table 1):

- Since the specification is deterministic, D-trans is not applicable. We start by C-trans (without the self loops). Fig. 13(a) illustrates this transformation (Section 5.2.2).
- After C-trans, we apply H-trans (without adding the self loops). There are three unspecified transitions ($h^1(AD,a), h^1(AD,d)$, and $h^1(E,b)$). We have the choice to complete either the projections on d or on b that are minimal. We eliminate the projection on the input a because it is not reduced (Section 5.1.2): BC and E are equivalent. We choose the projection on d, because it enhances the reachability set of the machine. Based on Ref. [10], we add the transition $h^1(AD,d) = BC, h^2(AD,d) = 1$. This follows us to obtain a machine (Fig. 13(b)) with a unique homogeneous sequence dd.
- Finally, we apply S-trans to complete the specification. We add self loops to the last unspecified transitions: $h^1(AD,a) = AD$ and $h^2(AD,a) = 1; h^1(E,b) = E$and $h^2(E,b) = 0$.

By applying K-trans we generate a new specification (Fig. 13(c)) that has the reachability set $V\{\varepsilon, b, d\}$, the characterization set $W = \{dd\}$, and the test suite TS = {*add*, *badd*, *dadd*, *bbdd*, *dddd*, *cdd*, *bcdd*, *dcdd*, *bddd*, *dddd*} whose length is $L(\mathrm{TS}) = 38$ and UTM = 0.687.

To compare the results of K-trans with S-trans, H-trans and C-trans, we present in Table 1 the results of the application of all of these transformations on the FSM in Fig. 4(a).

It is possible that one of the transformations composing K-trans does not produce a better testability (testability activity), in this case we backtrack to the previous version of the specification and apply on it the next transformation. The activity of testability validates the transformation by evaluating the testability improvement, and the convenience of the modifications relatively to the initial specification. An algorithm describing this new phase is shown in the Appendix.

If C-trans is applicable, it means that the new produced specification has less distinct states and less transitions. This makes all the remaining phases easier to perform. The validation, implementation and testing activities are improved since the new generated specification has less states.

6. Conclusions and future work

In this paper, we have proposed a testability enhancement

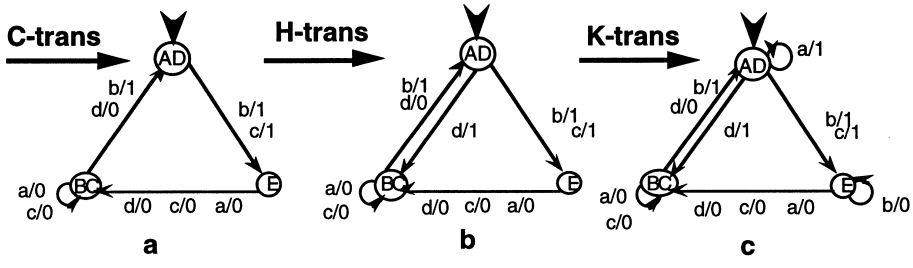

Fig. 13. K-trans tranformation.

activity to be incorporated in the protocol development life cycle. This activity is based on the application of a set of transformations to increase the specification testability. The proposed formal transformations are: D-trans, S-trans, H-trans, C-trans and K-trans. Ideally, each transformation produces a more testable specification. The testability enhancement is verified by computing two new testability measures called testability vector (TV) and unified testability measure (UTM). The measures will confirm whether the transformation is beneficial, i.e the change does not disturb the protocol functionality, and increases the specification testability. The proposed measures are based on the evaluation of some factors that influence the length of the complete test suite.

For future research, we will address transformations that take into account other factors influencing the testability of FSM-based protocol specifications. We will also adapt our formal transformations to the Extended Finite State Machine (EFSM) model for protocol specifications written in FDTs such as SDL, Estelle, and LOTOS.

Acknowledgements

The authors would like to thank J. Drissi for his comments on a draft of this paper. This work was supported by NSERC individual GRANT.

Appendix

A.1. S-trans algorithm

Begin
DC ← set of the non-specified transitions;
I_d ← set of inputs labeling the non-specified transitions in the original machine;
  For each $i \in I_d$
    Do
  $P_{all} \leftarrow \{\text{out: } \forall q_j h^2(q_j, i) = \text{out}\}$; $P_S \leftarrow \{\text{out: } h^1(q_j,i) = q_j$ and $h^2(q_j,i) = \text{out}\}$;
For each $(q,i) \in$ DC
    Do
    $P \leftarrow \{\text{out: } h^1(q_j,i) = \text{q and } h^2(q_j,i) = \text{out}\}$;
    $h^1(q,i) = q$;
    if ($\exists$o $\notin P_{all}$) Then $h^2(q,i) =$ o; $h^1(q,i) = q$
    Else If ($\exists$o $\notin P_S$) Then $h^2(q,i) =$ o
    Else If ($\exists$o $\notin P$) Then $h^2(q,i) =$ o
      Else $h^2(q,i) =$ null
      Endif
    Endif
    Endif
    next q
  EndDo
    next i
  EndDo
End.

B.1. C-trans algorithm

Begin
S_0 ← the initial specification containing compatible states; Extract (C_i); /*C_i ← the set of maximum pairwise compatible states in Ref. [14] */
Extract (C_{max}); /*
$C_{max} = \{C_i\}$*/
G ← number of independent paths of S_0
N ← number of states of S_0;
$W \leftarrow 0$; $V \leftarrow 0$;
$P \leftarrow \{P_q\}$
  /* Partitions from C_{max} that covers all the states of Q */
For each P_q /*From C_{max} we choose all partitions P_q verifying formula (1) */
  Do
  M_q ← Machine from P_q:
  If det(M_q) /*M_q is deterministic */
    Then
    $M \leftarrow M_q$; $N_q \leftarrow$ number of states of M_q;
    If $N_q \leq N$ Then $N \leftarrow N_q$ Endif;
    $G_q \leftarrow$ number of independent paths of M_q;
    If $G_q \leq \Gamma$ Then $G \leftarrow G_q$ Endif;
    $W_q \leftarrow W(M_q)$; /* the state characterization degree */
    If $W \leq W_q$ Then $W \leftarrow W_q$ Endif;
    $V_q \leftarrow V(M_q)$; /*the controlability degree */
    If $V \leq V_q$ Then $V \leftarrow V_q$ Endif
  Endif;
next P_q
EndDo
$M \leftarrow$ Best_min(M, G, W, V, N)/* Use of the guidelines to choose the best minimal machines*/
If $|M| > 1$
  Then $M_{min} \leftarrow$ Rand_choice(M) /* It remains more than one machine, we choose randomly */
Endif
End.

C.1. H-trans algorithm

Begin
maxdisting ← ‘‘; seqdisting ← ’’;
DC ← set of the don’t care transitions;
I_d ← set of inputs labeling “don’t care” transitions in the original machine;
For each $i \in I_d$
Do
    $Q_i \leftarrow$ The set of states that accepts the input i;
    $Q_i' \leftarrow$ The set of states that accepts the sequence i*;
    $M_i \leftarrow (Q,\{i\}, O, h_i, D_{M_i})$ /*unary projection of M on i */
    If M_i not reduced /*equivalent states see proposition 2 */
    Then next i;
    Else

For each $(q, i) \in$ DC
Do
$A \leftarrow$ The set of states from which we can reach q;
$M_i' \leftarrow (Q_i' \cup A, \{i\}, O, h_i', D_{M_i'})$;
assign the don't care transition relatively to the four following cases;
disting $\leftarrow$ calcdisting (M_i', i) /*distinguishing sequence of M_i' */;
If |disting| > |maxdisting| Then maxdisting $\leftarrow$ disting Endif;
next q
EndDo;
If |maxdisting| < |seqdisting|
Then seqdisting $\leftarrow$ maxdisting; $M' \leftarrow$ complete(M, M_i')
Endif
next i;
Endif
EndDo
If seqdisting $\neq$ '' Then return (M', seqdisting)
Else look for the second best case
Endif
End.

Case 1 (q, i) is a "don't care" transition. All paths terminate in s. M_i has no cycle: We determine the longest path $q_1 \rightarrow q_2 \rightarrow \cdots \rightarrow qk \rightarrow q$ and define a transition in state q on input i: $h^1(q, i) = q_1$ and $h^2(q, i) = o$, where output o is such that sequence $h^2(q_1, i),\ldots,h^2(q_k, i)o$ cannot be represented as any of its proper prefixes repeated several times. If $|O| > 1$ then it is always possible to find such output. Assigning the transition (q,i), we obtain a completely specified FSM.

Case 2 (q,i) is a "don't care" transition. M_i has cycles, but its paths terminate in s: In this case, we take an arbitrary cycle $(q_1 \rightarrow q_2 \rightarrow \cdots \rightarrow q_k)$ and define a transition from state q to the state q_1, i.e. $h^1(q,i) = q_1$, with the output $h^2(q, i) = o$ such that $o \neq h^2(s_k, x)$. By assigning the transition (q, i), we again obtain a new completely specified FSM.

Case 3 M_i has cycling paths, but none of them is dominant: In this case, the starting state of each cycling path is equivalent to another state of this path. To assign the "don't care" transition (q, i) we choose an arbitrary path $(q_1 \rightarrow \cdots \rightarrow q_i + 1 \rightarrow \cdots \rightarrow q_k)$ which ends with the cycle $(q_i + 1 \rightarrow \cdots \rightarrow q_k)$, where $i \geq 1$. Since state q_1 is equivalent to a certain state of the path, it is also equivalent to a state q_j of this cycle, $i + 1 \leq j \leq k$. Then we assign $h^1(q, i) = q_j$ with the output $h^2(q, i) = o$ such that $o \neq h^2(q_{j-1}, i)$. If $j = 1$ then $o \neq h^2(q_k, i)$.

Case 4 M_i has a dominant cycling path: In this case, there exists a cycling path $P = (q_1 \rightarrow \cdots \rightarrow q_{i+1} \rightarrow \cdots \rightarrow q_k)$, $i \geq 1$ such that s_1 is not equivalent to any other state of P, moreover, in accordance with Proposition 1, only another starting state of a cycling path might be equivalent to q_1. We transform the "don't care" transition (q, i) into a defined one in the following way. $h^1(q, i) = q_1$ and $h^2(q, i)$ is assigned any $o \in O$. Similarly to the cases considered above, we claim that state q is now distinguishable from any other state of augmented FSM.

D.1. K-trans algorithm

Begin
$S_0 \leftarrow$ First_spec; $S1 \leftarrow$ D-trans($S0$);
If det($S1$) Then $S0 \leftarrow S1$; $T0 \leftarrow$ Testability ($S0$) Endif; /* if $S1$ is determinist */
$S1 \leftarrow$ C-trans($S0$); $T1 \leftarrow$ Testability ($S1$);
If $T1 > T0$ Then $S0 \leftarrow S1$; $T0 \leftarrow T1$ Endif; /* if testability of $S1$ is better than $S0$ */
$S1 \leftarrow$ H-trans($S0$); $T1 \leftarrow$ Testability($S1$);
If $T1 > T0$ Then $S0 \leftarrow S1$; $T0 \leftarrow T1$ /* if testability of $S1$ is better than $S0$ */
Else $S1 \leftarrow$ S-trans($S0$)
Endif
End

References

[1] L. Cacciari, O. Rafiq, Controllability and observability in distributed testing, Information and Software Technology 41 (1999) 767–780.

[2] R. Dssouli, K. Saleh, El M. Aboulhamid, A. En-Nouaary, C. Bourhfir, Test development for communication protocols: towards automation in computer networks, in: A. Cavalli (Ed.), The International Journal of Computer and Telecommunications Networks, Advanced topics on SDL and MSC, Elsevier, Amsterdam, 31(17) 1999, pp. 1835–1872.

[3] T. Walter, J. Grabowski, A framework for the specification of test cases for real-time distributed systems. Information and Software Technology 41 (1999) 781–798.

[4] R. Lai, Towards more industrially relevant academic researches on testing of communicating systems, Journal of Systems and Software (1999) (in press).

[5] R. Dssouli, R. Fournier, Communication software testability, in: I. Davidson, W. Litwack (Eds.), IFIP Transactions, Protocol Testing Systems III, Proceedings of IFIP TC6 Third International Workshop on Protocol Test Systems, North-Holland, Amsterdam, 1991, pp. 45–55.

[6] S.T. Vuong, A.A.F. Loureiro, S.T. Chanson, A framework for the design for testability of communication protocols, in: O. Rafiq (Ed.), IFIP Transactions, Protocol Test Systems VI, Proceedings of IFIP TC6 Fifth International Workshop on Protocol Test Systems, North-Holland, Amsterdam, 1994, pp. 89–108.

[7] A. Petrenko, R. Dssouli, H. Konig, On evaluation of testability of protocol structures, in: O. Rafiq (Ed.), IFIP Transactions, Protocol Test Systems VI, Proceedings of IFIP TC6 Fifth International Workshop on Protocol Test Systems, North-Holland, Amsterdam, 1994, pp. 111–123.

[8] K. Saleh, Testability-directed service definitions and their synthesis, in: Proceedings of the Eleventh IEEE International Phoenix Conference on Computers and Communications (IPCCC-92), March 1992, pp. 674–678.

[9] K. Karoui, R. Dssouli, O. Cherkaoui, 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.
[10] N. Yevtushenko, A. Petrenko, R. Dssouli, K. Karoui, S. Prokopenko, On the design for testability of communication protocols, in: IWPTS'95, France.
[11] R.S. Freedman, Testability of software components, IEEE Transactions on Software Engineering SE-17 (6) (1991) 553–564.
[12] J. Voas, L. Morrell, K. Miller, Predicting where faults can hide from testing, IEEE Software, March 1991 41–48.
[13] A. Petrenko, G.v. Bochmann, R. Dssouli, Conformance relations and test derivation, in: O. Rafiq (Ed.), IFIP Transactions, Protocol Test Systems VI, Proceedings of IFIP TC6 Fifth International Workshop on Protocol Test Systems, North-Holland, Amsterdam, 1994, pp. 157–178.
[14] Z. Kohavi, Switching and Finite Automata Theory, McGraw-Hill, New York, 1970.
[15] G.v. Bochmann, A. Petrenko, M. Yao, Fault coverage of tests based on finite state models, in: Proceedings of IFIP TC6 Seventh International Workshop on Protocol Test Systems, Japan, 1994.
[16] G. Luo, A. Petrenko, G.v. Bochmann, Selecting test sequences for partially-specified non-deterministic finite state machines, in: Proceedings of IFIP TC6 Seventh International Workshop on Protocol Test Systems, Japan, 1994.
[17] S.T. Vuong, A framework for the design for testability of communication protocols, in: Pau IWPTS 1991.
[18] T.J. McCabe, C.W. Butler, Design complexity measurement and testing, Communication of the ACM 32 (12) (1989) 1415–1424.
[19] R. Dssouli, K. Karoui, A. Petrenko, O. Rafiq, Towards design for testability of communication software, IFIP Intl workshop on protocol test systems (IWPTS '95), Chapman and Hall, 1995, pp. 237–251.

Further reading

M. Abramovici, M.A. Breuer, A.D. Friedman, Digital Systems Testing and Testable Design, Computer Science Press, Oxford, 1990.
T.S. Chow, Testing software design modeled by finite-state machines, IEEE Transactions on Software Engineering SE-4 (3) (1978) 178–187.
S. Fujiwara, G.v. Bochmann, Testing non-deterministic state machines with fault coverage, in: J. Kroon, R.J. Heijink, E. Brinksma (Eds.), IFIP Transactions, Protocol Test Systems IV, Proceedings of IFIP TC6 Fourth International Workshop on Protocol Test Systems, North-Holland, Amsterdam, 1992, pp. 267–280.
J.E. Hopcroft, J.D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, Reading, MA, 1979 p. 418.
G. Luo, G.v. Bochmann, A. Petrenko, Test selection based on communicating non-deterministic finite state machines using a generalized Wp-method, IEEE Transactions on Software Engineering SE-20 (2) (1994) 149–162.
D. Lee, M. Yannakakis, Testing finite-state machines: state identification and verification, IEEE Transactions on Computers 43 (3) (1994) 306–320.
D. Morris, B. Tamm (Eds.), Concise Encyclopedia of Software Engineering Pergamon Press, Oxford, 1993.
A. Petrenko, G.V. Bochmann, On fault coverage of protocol tests, Computer Networks and ISDN Systems on Protocol Testing, 1995 (submitted for publication).
A. Petrenko, T. Higashino, T. Kaji, Handling redundant and additional states in protocol testing, in: IWPTS'95, France, 1995.
A. Petrenko, N. Yevtushenko, R. Dssouli, Testing strategies for communicating FSMs, in: IWPTS'94, Japan, 1994.