12

Relating conformance test coverage to formal specifications

Roland Groz, Olivier Charles, Josiane Renévot[1]
FRANCE TÉLÉCOM - CNET LAA/EIA/EVP
2 av. Pierre Marzin, F-22307 Lannion Cedex, France
groz@lannion.cnet.fr, charleso@lannion.cnet.fr

Abstract : This paper presents a practical approach useful for assessing the coverage of test suites which are generated automatically from Extended Finite State Machine specifications. It is based on transition coverage, a variant of branch coverage which seems to correspond to the needs of test designers for conformance testing of communication protocols. It presents a tool for implementing this approach, and discusses the facilities needed for a qualitative analysis of coverage.

Keywords : test coverage, conformance testing, protocol engineering, Estelle, TTCN

1 INTRODUCTION

Conformance testing has long been deemed a major issue of strategic importance for the acceptance and use of OSI and other products based on standards for open systems. ISO has established a methodology in its IS-9646 standard [ISO 94]. The central point in a conformance testing process lies in the availability of a test suite which must be closely related to the protocol (or system) specification. Designing such test suites is hard work. In fact, computer aided test generation based on Formal Description Techniques such as Estelle, Lotos or SDL is progressively becoming a reality, but existing tools are still limited [Groz 95].

The main goal of conformance testing for network products is to ensure that they will be able to interoperate according to a given protocol specification. To that end, the quality of the test suite used is crucial. Apart from the fact that this test suite should be manageable (in the context of a test laboratory), the main quality expected from such a suite is that it should test as exhaustively as possible all the requirements expressed in the specification. This is usually referred to as (having a good) *test coverage*. However, there are many interpretations of what this notion of coverage should be.

This paper presents a method and a tool to compute and, most importantly analyse (providing useful feedback to the test designer) a notion of test coverage which :

1. applies to conformance testing of communication protocols (as defined in IS-9646);
2. corresponds to what appears to be the primary expectation of our test laboratories;
3. capitalizes on our test generation environment called TVEDA, but could be extended to encompass other approaches.

1. Josiane Renévot is now with Prologue SA, F-91941 Les Ulis, France..

R. Gotzhein et al. (eds.), *Formal Description Techniques IX*

Contrary to the usual trend in research on test coverage (which is aimed at providing smart definitions of coverage), we have voluntarily adopted a rather weak notion of coverage (derived from the informal definition found in IS-9646), at least as a first step, because our first goal has been to match the common practice of conformance test designers. In fact, we followed an empirical approach similar to the approach we had used in test case generation [Phalippou 90]; this approach has proved quite successful. In order to provide a tool that can attract conformance testing experts to formal methods, we do not try to implement smart coverage measures; on the contrary, we try to provide the type of measure which they usually consider as 'good' coverage. We are not questioning how 'good' this actually is (in our own opinion, it is somehow deficient). We are just meekly accepting that test experts may have their own views and needs, and that one of our primary concern, as FDT experts, should be to placate those needs.

In the IS-9646 framework, testing is based on a test suite, wherein each test (called a test case) is associated to a requirement (from the - informal - specification). Usually, the test purpose (a key element of IS-9646 test suite design) can be seen as an instantiation of a requirement. As to coverage, the usual understanding, is that a requirement is 'covered' if there are corresponding tests in the test suite. Of course, there can be several tests associated to a single requirement, which leaves open the question of the 'degree' of coverage a single test can contribute.

In this paper, we give a simple formalization of this notion of coverage. Since we had to feed some formal specification into our tool, we could not trace test purposes back to informal requirements, and we had to rely on existing formal specifications, which, in our case, means Estelle or SDL. Therefore, instead of tracing a test purpose back to informal requirements, we based our correspondence on the EFSM transition notion. Consequently, this paper is built around the notion of what we call *transition coverage*.

Once we have instantiated this notion of coverage in our context, we describe how this can be built into a tool to provide useful feedback to protocol test experts in assessing test suites. In that view, we insist on the parameters and the Man-Machine Interface that have seemed necessary to make coverage computation meaningful and manageable.

The rest of this paper is organized as follows. Section 2 defines precisely the type of coverage which we are considering. Section 3 describes our tool for displaying coverage information useful for test designers. In particular, it discusses our choices of interactions with the user which appear to be at least as important as the actual definition of coverage considered. Section 4 compares our work with other approaches, and reports on an experiment where both our approach and a mutation-based tool were used on the ISDN LAPD protocol.

2 DEFINING TEST COVERAGE FOR CONFORMANCE TESTING

2.1 Basic expectations and types of coverage in conformance testing

As mentioned in the introduction, there can be many interpretations of coverage. In this paper, we are neither trying to compare them, nor even to provide any classification. However, in general, when we talk with the 'rank and file' test designers, there are two broad concepts of coverage that they are thinking of.

The most straightforward concept is what we shall call 'specification coverage', as illustrated in figure 1. The idea is that we are testing an implementation to see if it fulfils the requirements expressed in a specification. As a matter of fact, this is the main idea driving the usual test design approach (as in IS-9646): a test purpose is derived from one or more requirements. In turn, the corresponding test case is considered to cover (partially or totally) the requirement. Simple though it may seem, this notion raises many questions: what is the granularity of reference to the specification, how are multiple references handled, how do we define a partial coverage etc., let alone the fact that for many test generation approaches, no clear purpose can be as-

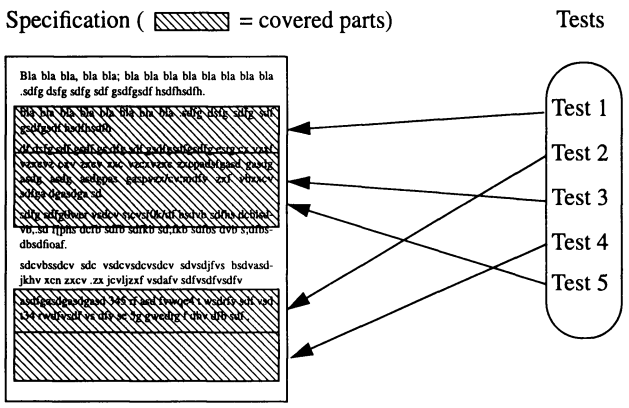

Figure 1 : Specification coverage

cribed to a given test ? Anyway, since this was felt as being a very natural notion for conformance testers, we decided that we should translate it into more formal terms, as described in the following sub-sections.

The second concept is that of fault coverage, which is directly related to the efficiency in eliciting errors in an implementation. This notion better corresponds to the traditional approach to testing as an error-finding activity [Myers 79]. Provided the test hypothesis on implementations (i.e. the set of implementations considered, or their models as in [Tretmans 93]) is not unduly restrictive, fault coverage is more demanding than specification coverage. Fault coverage is considered a key notion in research on test selection techniques. However, strange though it may seem, it does not appear to be the main focus in practical conformance testing. As a matter of fact, this is one item of divergence between the practical approach presented in IS-9646 and its theoretical counterpart FMCT [ISO 95].

2.2 Basing coverage on references to requirements

IS-9646 part 2 (section 10.3.5) relates 'adequate coverage' to the existence of 'at least one purpose related to each distinct conformance requirement' (or 'set of related conformance requirements' in the case of multi-party testing, which we are not dealing with here). Section 10.4 of this standard provides an example that clearly shows that *in IS-9646, coverage is more or less assumed to be based on relating tests to requirements, i.e. to the specification*, although the standard leaves open the precise (or formal) definition of it.

On the other hand, the intermediate FMCT document [ISO 95] (possibly to be completed in that respect in later versions) devotes only one paragraph to coverage, and that (8.5 in [ISO 95]) is entitled 'Fault coverage'. However, one can find in [Tretmans 93] a slightly broader view of coverage.

It is our feeling that the notion of fault coverage is better suited for validation tests, whereby a test entity tries to make its best to ensure that the product tested has a good quality. In the case of pure conformance testing, we are not interested in identifying faults (only errors are significant, not faults), and certainly not in removing them, but only in the fulfilment of requirements. This is because the output of a conformance test lab. is in the form of a report that states which requirements are not met by an implementation.

Consequently, we think that the 'adequate coverage' notion of IS-9646 which is a requirement-based approach is better represented by the specification coverage notion. For instance, if the

implementation of a specific requirement can be ascribed to a module which contains 10 (detected) faults, this would have 10 times more weight in a fault coverage measure than detection of a single fault. Whereas in a specification coverage approach, the weight would only depend on the fact that this single requirement was violated, and possibly on the importance of the requirement, which is what is expected from a test lab. report.

2.3 From requirements to EFSM-based specifications

Formal versions of conformance requirements in, say, logic-based requirement languages, are not currently available for protocols. Instead, we may have either Lotos, SDL or Estelle formal specifications. In our environment, all formal descriptions would be available in SDL or Estelle. Therefore, in this paper, we instantiate the specification coverage approach on an EFSM model.

This is rather natural for the kind of protocols we are considering (mainly lower-layers, up to layer 5 protocols, although this is also applicable to many application layer protocols which are described on a state basis). For such protocols (as for INRES and LAPD which are presented in figures below), test purposes would be written following very similar patterns :

'To ensure that IUT sends B upon receipt of A in state S'.

'To send A to IUT in state S'.

'To check that A is ignored in state S (the IUT remains in state S)'.

'Check whether the IUT will send an A when the upper layer issues a P primitive'.

or even more explicitly as in the case of INRES (see figure 3):

'Verify that the IUT sends B (resp. nothing) in response to A in state S. The IUT is expected to enter state U'.

In such phrases, a conformance requirement corresponds to a transition of an EFSM. In fact, since 'A' above is an event which is usually particularized with a constraint, this may correspond to a partial instantiation of the extended transition (i.e. a subset of the sets of transitions of the underlying FSM associated to the transition of the EFSM).

More formally, following [Phalippou 94], we shall define an EFSM on an underlying IOSM as a finite set of collections of transitions of the IOSM. Each collection is called an extended transition.

Definition 1 : an *input-output state machine* (IOSM) is a 4-tuple $M = \langle S, L, T, s_0 \rangle$ where:

1. S is a finite nonempty set of states;
2. L is a finite nonempty set of interactions;
3. $T \subset S \times ((\{?, !\} \times L) \cup \{\tau\}) \times S$ is the transition relation. Each element from T is a transition, from an origin state to a destination state. This transition is associated either to an observable action (input ?a or output !a), or to the internal action τ.
4. s_0 is the initial state of the IOSM.

As usual, we can define a path of an IOSM as a sequence of chainable transitions (the start state of all transitions in the sequence but the first is the end state of the preceding transition).

Definition 2 : let $M = \langle S_S, L, T, s_{0S} \rangle$ be an IOSM. An *extended transition* on M is a nonempty subset $t \subseteq T$. An *extended finite state machine* (EFSM) on M is a finite set of extended transitions on M: $X = \{t_i\}_{1 \leq i \leq n}$.

This definition of an EFSM has been explained in [Phalippou 94]. It focuses on the idea that an EFSM consists basically in structuring an underlying, huge, FSM (here IOSM distinguishing inputs and outputs for testing purposes). The underlying FSM corresponds to the reachability

graph of the intended EFSM. Note that this simple structuring notion generalizes the more traditional EFSM concept, because it allows factorizing into an extended transition even unrelated transitions. For instance, in Estelle, we can factorize transitions that share common input events and transition bodies with differing from-states, but we cannot factorize transitions that do not share the same input event (when clause).

The key advantage is that we can view a specification at different levels of abstractions. For instance, Estelle, and an expansion of an Estelle specification into its normal form [Sarikaya 85] are two levels of EFSM on the same FSM specification. The notion of coverage we are now defining can be computed on either level. This is an answer to the question of the 'granularity' that we raised earlier about the notion of 'specification coverage'. Our EFSM model takes into account various levels of reference. The 'grains' referenced correspond to the extended transitions of the EFSM considered.

As in [ISO 95], a test case will itself be modelled by an IOSM *tc*. To comply with IS-9646, we consider that our tester *tc* has three distinguished states: *pass*, *inconclusive* and *fail*. We will consider that those states are sink states (a finer approach to TTCN would distinguish between temporary and final verdicts, but we do not take that distinction into account here).

Test execution is modelled by the parallel composition of the tester *tc* with an IOSM model of the implementation. This parallel composition is defined as follows.

Definition 3 : the *parallel composition* of two IOSM $M_1 = \langle S_1, L_1, T_1, s_{01} \rangle$ and $M_2 = \langle S_2, L_2, T_2, s_{02} \rangle$ is an IOSM $M = \langle S, L, T, s_0 \rangle = M_1 \| M_2$ defined by :

1. $S = S_1 \times S_2$
2. $L = L_1 \cup L_2$
3. $s_0 = (s_{01}, s_{02})$
4. $(s_1, \tau, s'_1) \in T_1 \Rightarrow \forall s_2 \in S_2, ((s_1, s_2), \tau, (s'_1, s_2)) \in T$
   $(s_2, \tau, s'_2) \in T_2 \Rightarrow \forall s_1 \in S_1, ((s_1, s_2), \tau, (s_1, s'_2)) \in T$
   if $a \in L_1 \cap L_2$ then $(s_1, ?a, s'_1) \in T_1 \wedge (s_2, !a, s'_2) \in T_2 \Rightarrow ((s_1, s_2), \tau, (s'_1, s'_2)) \in T$ and
   $(s_1, !a, s'_1) \in T_1 \wedge (s_2, ?a, s'_2) \in T_2 \Rightarrow ((s_1, s_2), \tau, (s'_1, s'_2)) \in T$
   if $a \in L_1 - L_2$ then $(s_1, ?a, s'_1) \in T_1 \Rightarrow \forall s_2 \in S_2, ((s_1, s_2), ?a, (s'_1, s_2)) \in T$ and
   $(s_1, !a, s'_1) \in T_1 \Rightarrow \forall s_2 \in S_2, ((s_1, s_2), !a, (s'_1, s_2)) \in T$
   if $a \in L_2 - L_1$ then $(s_2, ?a, s'_2) \in T_2 \Rightarrow \forall s_1 \in S_1, ((s_1, s_2), ?a, (s_1, s'_2)) \in T$ and
   $(s_2, !a, s'_2) \in T_2 \Rightarrow \forall s_1 \in S_1, ((s_1, s_2), !a, (s_1, s'_2)) \in T$

In item 4 of this definition, we say that the transitions of M_1 and M_2appearing on the left-hand side of the implications are *exercised* by the corresponding transition(s) of M appearing on the right-hand side of the same implications. We say that a path of the composite IOSM $M_1 \| M_2$ *exercises* a transition t_x of a component machine iff (if and only if) this path contains a transition which exercises t_x.

A state of the composite IOSM $M_1 \| M_2$ is said to be a *pass-state* (resp. an *inconclusive-state*, a *fail-state*) iff its second component (remember states of $M_1 \| M_2$ are couples) is *pass* (resp. *inconclusive*, *fail*).

Now, since we are interested in specification coverage, our aim is to see how a test case covers a requirement, and we have seen that requirements are associated to extended transitions (although this is not necessarily a one-to-one correspondence, and there might be overlapping).

This leads us to base our definition of coverage on the parallel composition of the test case with the specification, and to see which transitions are exercised.

Definition 4 : let t be a transition of the IOSM S of the specification.
$tc\ covers_p\ t$ iff there is a path in $tc \| S$ from the initial state to a *pass-state* which exercises t.
$tc\ covers_i\ t$ iff there is a path in $tc \| S$ from the initial state to an *inconclusive-state* which exercises t.
Finally, $tc\ covers\ t$ iff either $tc\ covers_p\ t$ or $tc\ covers_i\ t$ (or both).

These notions are now naturally extended to extended transitions.

Definition 5 : let tr be an extended transition of an EFSM on the IOSM S of the specification.
$tc\ covers_p\ tr$ iff $(\exists t \in tr)\ (tc\ covers_p\ t)$
$tc\ covers_i\ tr$ iff $(\exists t \in tr)\ (tc\ covers_i\ t)$
Finally, $tc\ covers\ tr$ iff either $tc\ covers_p\ tr$ or $tc\ covers_i\ tr$ (or both).

Depending on whether we consider a test case in full, or whether we consider the test preamble as a separate test step, we could adjust definition 4 to consider only subpaths starting from the state of the tester reached after the preamble. In this case, 'initial state' would be interpreted as a state whose first component would be the state of the tester after preamble. This is what is actually done in the LAPD experiment presented below.

Based on these notions, we are now able to define a family of related coverage measures which we call the *transition coverage* approach.

We will use the following notations (where *card* means the cardinality of the following set). Given an EFSM X (this maybe tailored by the user as a subset of a complete EFSM specification, see section 3.3) and a set T of test cases (this may also be a subset of a complete test suite) we define.

$$Ntr = card\ X$$
$$Np = card\{tr \in X |\ (\exists tc \in T)\ (tc\ covers_p\ tr)\}$$
$$Ni = card\{tr \in X |\ \exists tc \in T (tc\ covers_i\ tr) \wedge \neg(\exists tc \in T (tc\ covers_p\ tr))\}$$
$$Nc = card\{tr \in X |\ (\exists tc \in T)\ (tc\ covers\ tr)\}$$
$$Nr(tr) = card\{tc \in T | tc\ covers\ tr\}$$

2.4 Transition coverage : a family of coverage definitions

In our tool, we have decided not to impose any coverage function on the user, but to let him (or her) combine freely the ingredients available. However, for convenience, we provide three predefined classes of measures. All three can be seen as a kind of weighted mean of transitions. Each class can in turn be parametrised by weight constants or functions. Currently, the user can choose between:

1. A transition is covered (weight 1) iff it is *referenced in at least one test*. The coverage rate in this case is: Nc/Ntr
2. The weight associated to each transition in the coverage depends on the verdict. We define two rates : a rate R_p associated to transitions which are pass-covered and another rate R_i for transitions which are not pass-covered, but only inconclusive-covered. Of course, $0 \le R_x \le 1$. In this case, the coverage is defined as: $(Np \times R_p + Ni \times R_i) / (Ntr)$. Typically, the value of R_p could be 1, and the value of R_i would be lower (if it is equal to 1, we get the same rate as computed above in item 1). On the other hand, if we would like to compute

separately 'pass coverage' and 'inconclusive coverage', we could use $R_p = 1, R_i = 0$ and $R_p = 0, R_i = 1$ respectively.

3. In the third predefined class, a transition is covered *depending on the number of references* to it (regardless of verdicts); so the weight associated to a transition is a monotonic rate function R of the number $Nr(tr)$ of references to it. In the current implementation, this function is described piecewise by its value on three intervals: $R(x)$ is 1 for $x \geq N$, where N is a constant, some user-defined arithmetic expression on x for $1 < x < N$, and a fixed value for $x = 1$; implicitly, the weight is 0 when $x = 0$. The coverage is computed as: $(\sum_{tr \in X} R(Nr(tr))) / (Ntr)$. In this case, the coverage can only be 100% if all transitions are referenced at least N times each in the test suite.

Other interesting variants were suggested by an anonymous referee : associating 'weight factors' to transitions (related to their importance), or associating labels to them and computing coverage for each set of transitions with a certain label (actually, this last variant can be computed by our tool since we can compute coverage on subsets of the specification).

2.5 Implementing transition coverage computation

In order to compute the $covers_x$ relations, we have considered two approaches. The first one is specific to test suites generated from FDT specifications: it consists in computing references to the covered transitions in the course of generating test cases. This is exactly what we have implemented in our test generation tool TVEDA. The other approach which we are considering next is to compute the relations by executing the test cases against the specification with an FDT simulator (in our case the SDL Geode tool); in that case, we can take into account test suites which have not be generated from the FDT, but we must first put them into a format suitable for driving the simulator (in our case, converting TTCN test cases into observers of Geode).

In fact, our test generation tool TVEDA automatically provides references to the specification within the TTCN suite in the form of hypertext links. Currently, only links pointing to Estelle transitions (the dynamic part of the behaviour) are available, although extensions to data types are considered, and an SDL version is planned. The links are found in test purposes (to the transition(s) targeted for the PASS verdicts) and on test event lines.

2.6 Comparison with other approaches to coverage

It should be noted that since extended transitions in an EFSM-based FDT are related to the structuring FSM (based on major states in Estelle), which can be seen as a control graph, what we call 'transition coverage' is a variant of 'branch coverage' at that level of control. However, as can be seen, this variant adapts the notion of branch coverage to the concepts of conformance testing for protocols. In particular, it is related to the control graph of the specification, not of the implementation. It also includes the notion of a verdict, and the implied nondeterminism, as well as a customizable level of abstraction (by choosing an adequate EFSM on the underlying IOSM).

Branch coverage is known to be a weak coverage criterion. As argued before, our choice was deliberate. The user of this notion should be aware that it suffers from the well-known limitations for test methods such as the Transition Tour method : it would cover output faults (wrong output in extended transition, i.e. for all corresponding transitions), but not all transfer faults or additional state faults etc.

Another drawback is that the criterion is based on extended transitions, which, in our case, would be derived from the syntactic structure of the Estelle or SDL specification. Definition 5

makes it clear that only one instance of an extended transition need be considered (although taking the number of references into account can be seen as a makeshift corrective). Therefore, the criterion is dependent on the specification style.

If we turn to other approaches that have been proposed for protocol testing, such as the metric-based approach of [Vuong 91], the fault-model based approach of [Bochmann 91] and its derivatives to include data-flow analysis [Caouette 93] or avoid mutation generation for actual computation of the coverage [Yao 94], the fault coverage definition of [Tretmans 93], or the approach based on identification as in [Zhu 94], they all provide finer criteria for coverage. It should be noted, though, that since they are often applied in practice only to the FSM-based control part of a protocol specification, they suffer from similar drawbacks regarding the impact of structuring the specification into extended transitions (typically, variable values and the parameters of inputs and outputs are disregarded, therefore only one transition instance is considered).

However, the main point which should be noted is that we have insisted on defining a notion of coverage wherein it does make sense to say that a specific test covers specific transitions. Most other approaches are global in the sense that the coverage can only be defined for a whole test suite and a whole specification. Being global is an advantage in the general case, because many theoretical test methods (especially the methods based on FSM: DS, UIO, Wp...) do not follow a requirement-based test generation, but produce a test suite which tests a specification as a whole. However, as argued before, this may not be informative enough for a test designer that follows the IS-9646 approach. This is why we concentrated on defining a notion of coverage that can be computed locally. This is emphasized in the design of our tool (see section 3).

Finally, our approach concentrates on transitions, because this is the main part of an EFSM specification and of most test suites for lower layers (up to layer 5). The other significant part of such test suites consists in tests related to the encoding of PDUs. However, this is usually not described in the current FDTs (Estelle, Lotos and SDL). Typically, our work should be extended to incorporate a notion of coverage based on e.g. ASN.1 descriptions.

3 TOOL DESCRIPTION

3.1 Background

Before this work, we have developed a family of test generation tools called TVEDA [Phalippou 90] [Phalippou 94] [Clatin 95]. Our approach to test generation has emphasized the need to suit the practical needs of conformance testers [Groz 95] rather than applying unrealistic full-coverage methods based on theoretical models such as FSM. TVEDA can generate test suites in TTCN (or Menuet, a proprietary test language) from Estelle and SDL specifications, using either a 'single transition' test strategy or an extended transition tour strategy (extended to fit the EFSM concept). TVEDA is implemented in a CASE tool environment called Concerto. Thanks to this environment, which provides convenient means of establishing hypertext links, TVEDA automatically generates hypertext references between the test cases (in TTCN) and the corresponding parts of the formal specification (in Estelle). This has been a key feature for our approach to coverage computation.

3.2 Overview of coverage analyser

As in the case of test generation, we decided not to investigate sophisticated coverage measures. Instead, we focused on features that would make our tool better suited to the expectations of conformance test suite designers. We decided in particular to concentrate on:

1. Inputs: tailoring the ingredients of the coverage measurement
   a. Identifying the part of the specification considered

   b. Identifying the part of the test suite considered
   c. Choosing and tailoring a coverage measure
2. Outputs:
   a. Providing qualitative information which would be significant for the test designer
   b. Using appropriate display for an easy perception of coverage
3. Tool control: extra facilities to make the tool user-friendly and usable in the context of industrial test suite design.

In fact, all this work is mostly independent of the method used for computing coverage. As a matter of fact, the choice of a method (e.g. mutation analysis, data-flow cover...) is planned in the tool. However, since this was not our focus, we implemented only the transition coverage method so far, and prepared the ground for another one: an extension to EFSM of the proposal of [Yao 94]. This extension was described in [Renévot 95].

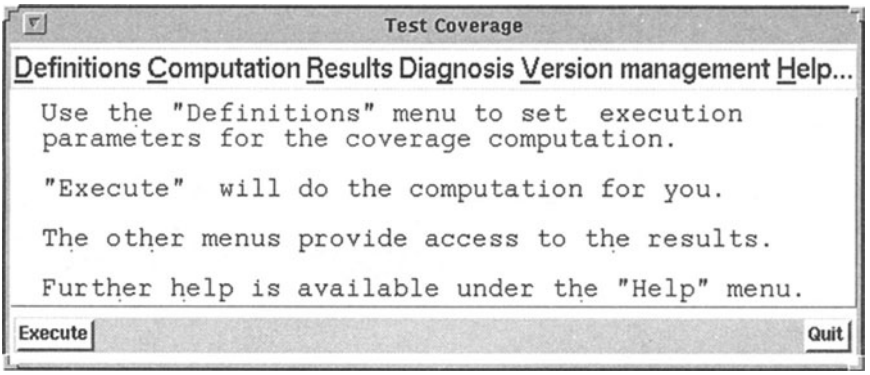

Figure 2 : Main menubar of coverage tool

Figure 2 shows the main window of the coverage tool, which gives a list of the main classes of functions offered. The most important ones are the 'definitions' and 'results', the others being more cosmetic in our current implementation.

3.3 Definitions

There are three main sub-menus; they correspond to the inputs mentioned in section 3.2.

1. Defining the subset of the specification considered. There are two main reasons for offering a choice of the part of the specification that will be the target for coverage. First, even though a test suite may be designed for a complete specification, it is important to be able to assess the coverage of a specific part (e.g. a critical function). Second, as was made clear by our first experiment with coverage tools (see section 4), it is quite common that a test suite is designed to cover only a part of a specification (e.g. because other parts are actually untestable); in that case, we would like to know how well it covers the part it was designed for.
   In our Estelle implementation, the subset is defined for one Estelle 'body', by providing a list of those elements that should be excluded from the computation: major states, state variables, input messages, input parameters, output messages.
2. Defining the subset of the test suite considered. This is probably less crucial than in the previous case, because a test suite is a sequence of test cases from which a sub-sequence could always be extracted 'manually' beforehand. Note that in the previous case (subset of speci-

fication), this is more difficult to do whilst keeping the specification semantically consistent. However, it is more convenient to start from a TTCN test suite, and our tool accepts a list of test cases and test groups to be excluded. This makes it possible to assess the coverage of any part of the test suite (either complete test groups or portions thereof).

3. Defining the method used for assessing coverage. At this point a choice can be offered between, e.g. our 'transition coverage' method (and all its variants listed in section 2.4), a mutation-based method [Renévot 95] (not implemented), a data-flow method etc. In any case, at this point, the precise parameters of the method must be given (although defaults are provided of course). For instance, in the case of mutations, a fault model can be parametrised with the probability associated with each class of faults (as is done in TESTL [Dubuc 91]).

3.4 Results and version management

A coverage tool should support different types of activities. Firstly, in the design phase of a test suite, it should provide a user-friendly interactive interface to quickly visualize the coverage of a given set of tests. Secondly, it should provide a detailed record of a test suite quality as measured by the tool.

Our tool addresses the first concern by providing a graphic form with associated navigation links, as shown on figure 3. On the second issue, it generates an automatic report on the results of coverage computation. This report contains detailed information on the reference inputs considered, i.e. specification version and subset considered, test suite and subset considered, parameters for computation and various 'administrative' details (such as date of issue, computation time etc.).

Figure 3 illustrates the information available interactively on the typical INRES protocol (actually, the interface is in French, but the choice of language is parameterized). The global coverage rate is shown as a pie chart in the upper left window. This window recalls the main parameters of the coverage computation (in this case the Estelle specification of the protocol, a test suite, and the transition coverage method), and offers two buttons. With these buttons, the user can choose to see either what is covered or what is not covered in the specification. In our session, the user chose the first button. Our tool opens a window on the Estelle specification (bottom left). Within this window, we have three graphical conventions, detailed in the small popup window (centre): the targeted Estelle 'body' is surrounded by a frame; the targeted transitions (i.e. those that have not been excluded from the computation according to the exclusion sets defined under the 'Definitions' menu) are greyed; and those that are actually covered are shown on a dark background. This provides a very intuitive perception of coverage, as befits the 'specification coverage' approach. Finally, following hypertext links, the user can easily find the TTCN test cases (right window) corresponding to a given transition[1].

3.5 Other functions

The 'Computation' menu offers some kind of task control facilities over the computation. In particular, it offers an a priori estimate of the expected duration of coverage computation. This should be particularly useful for mutation-based analysis which may be time-consuming (see for instance the CPU column in table 1).

The 'Diagnosis' function has not been implemented in our tool, but it could offer a useful complement to a coverage analysis tool. Two types of diagnosis have been envisioned.

1. Depending on the definition of coverage used, this should explain why e.g. such a transition is considered not covered. For instance, with a fault model, it could detail which faults have

1. In this session, the test cases presently shown do not correspond to the transitions currently visible in the lower left window. They would have been accessed by scrolling in that window.

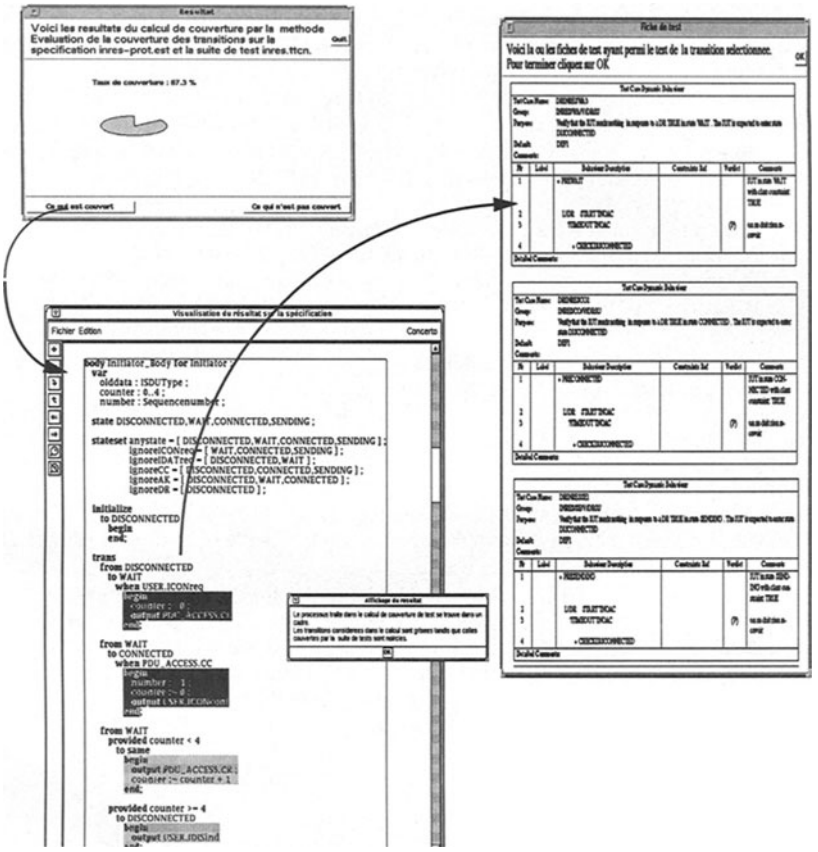

Figure 3 : Graphical/interactive output of tool showing navigation links

not been detected; of course, in that case, a tool should try to give some more synthetic information than a list of unkilled mutants.

2. In the case of automatically generated test suites, it could give hints on 'missing test cases', especially in the case of known limitations of the generation method. In our case, we know that transitions related to timers will not be tested.

4 EXPERIMENTING TWO COVERAGE TOOLS ON ISDN LAPD

4.1 Mutation analysis with TESTL

In joint work with the university of Montréal, we decided to apply their tool called TESTL [Dubuc 91] to test suites which are actually used in France Télécom test lab. for ISDN products. More precisely, we used TESTL's ability to take as input not only test suite definitions,

but also directly traces produced by protocol testers when actually carrying out the tests on a real implementation. This makes it possible to assess the actual coverage of a test campaign based on which paths were actually taken (a test suite in TTCN may allow several PASS, INCONClusive or FAIL branches due to the inability to have a strictly deterministic control of the implementation), and which test cases were actually carried out (some tests might not be executed due to restrictions in the implementation as stated in the PICS for instance).

The experiment was principally conducted on the LAPD protocol (link layer protocol for the ISDN D-channel, defined in recommendation Q.921 from ITU-T), for which we had an Estelle description (approximately 9000 lines for 800 transitions), and many traces available from test reports of different terminals. The test suite considered was the european standard 'PUB' test suite for the user (i.e. terminal) side. This test suite was implemented on K1197 protocol testers. TESTL was easily adapted to the output produced by those testers. However, due to TESTL restrictions, both the specification and the traces had to be modified. We give here a brief overview of the main modifications. Details can be found in [Salvail 94].

1. TESTL is restricted to a local test architecture. However, in our case, there is no upper tester as such and the traces do not report the interactions at the upper interface of the IUT (Implementation Under Test). The transitions triggered by inputs from the upper layer should be considered spontaneous, but this nondeterminism was not handled consistently by TESTL. A makeshift was found by introducing a responder in the Estelle specification. This responder mimics the behaviour of the actual upper layer in the implementation; writing this responder was no big task, because it was limited to responding to only the types of messages (mainly I-frames) actually used in the traces, and there were only a few instances of such messages.
2. TESTL does not take time into account. Therefore, all timers and transitions referring to timeouts had to be removed from the Estelle specification. Consistently, all tests related to timeouts of the specification (i.e. all those where the actual behaviour is triggered by a timeout; of course, all test cases in the TTCN test suite contain added timer settings for the tester, but this can be disregarded here) were removed from the traces.
3. TESTL can only take into account conforming traces. Therefore all FAILed tests had to be removed from the traces.

Test campaign	Number of PASS tests, states 4-7	Valid test cases for TESTL	Fault model	computation time	Number of mutants tested	Nb of mutants detected	coverage %
1001.pub	107	66	output	5h05	12597	2566	20.4%
1369.pub	209	135	output	20h01	12597	3592	28.5%
1369.pub	209	135	transfer	18h50	11130	1323	11.9%

Table 1: Results of test coverage computation

The results of a few coverage computations are shown in table 1. Column 3 amply testifies that TESTL introduced severe restrictions on the tests that could be taken into account. Out of approximately 450 tests in the original trace, only 66 could actually be handled by TESTL in the case of the first trace, and 135 in the second. The difference between the two campaigns lies in the fact that many more tests were 'FAIL' tests for 1001.pub. Column 2 provides an intermediate count on test cases that can be eliminated easily on syntactic checks : retain only PASS branches, and only for tests that involve neither states 1, 2, 3 (TEI management) nor state 8 (timer management) of the LAPD protocol specification.

The most striking result is the coverage percentage reached. This is extremely low. There are two main reasons.

1. Some states (namely sub-states 5_2, 6, 7_2, 7_3, 7_6 and 7_7) are not tested by the test suite: altogether, only 7 states out of 13 considered are considered as testable; the other states are regarded as unstable, and all the transitions starting from those states are purposely disregarded by the standardized conformance test suite. This limitation comes from the test suite, and is motivated by testability concerns.
2. Many test cases were removed from the test suites because they contained transitions triggered by timeouts. This limitation is purely due to TESTL.

4.2 Transition coverage with our tool

In order to make a relevant comparison between TESTL and our tool, the test suite used in both cases have to be the same. Contrary to TESTL, the current implementation of our tool needs TTCN test suites with references to the specification in the form of hypertext links. Of course the test suite used for the experiment with TESTL did not include those references. So we have had to generate with TVEDA a similar test suite including the references. We rely on TVEDA's ability to automatically reproduce a very close approximation of the PUB test suite (see [Phalippou 90]).

The specification used as input of TVEDA was the one used for TESTL (i.e. states 1, 2, 3, 8 removed from the LAPD complete specification). The parameters of TVEDA were also tuned to generate tests neither for the states 5_2, 6, 7_2, 7_3, 7_6, 7_7 and the error states nor for the PDU DL_UNIT_DATA_REQ and UI. As a result, the 198 automatically generated tests were very close to the 209 of the 1369.pub test suite (see Table 1).

The first experiment of coverage computation with our tool was performed with no additional parameters than the specification file and the TTCN file. So the whole specification was taken into account for the computation of the coverage. We obtain a rate around 30%. This result is very similar to the one we got with TESTL (28.5% with the 1369.pub test suite). The main reason for this low rate is that among the 13 states (plus two error states), we got tests for only 7 of them.

Since one of the facilities offered by our tool is the ability to focus on the significant part of the specification, we conducted another experiment. We gave parameters to the tool so that the transitions starting or ending on states 5_2, 6, 7_2, 7_3, 7_6, 7_7 (or error states) were ignored in the coverage computation and also the transitions firable on inputs DL_UNIT_DATA_REQ and UI. The rate grew up to 65.04% of transitions coverage. It is easy to locate the uncovered transitions since they appear in the analysis window pale grey highlighted. Figure 4 is a view of the analysis window (see section 3.4 for more detailed explanations on conventions used).

The information provided in this analysis allowed us to see why about 35% of the transitions were not covered. The first transition, which is not highlighted, is not targeted because it ends in a state which is not taken into account (error state). The four 'black' transitions are targeted transitions (their states and message are in the scope of the coverage computation) and tested. The grey one is targeted but there is no test in the suite which refers to it.

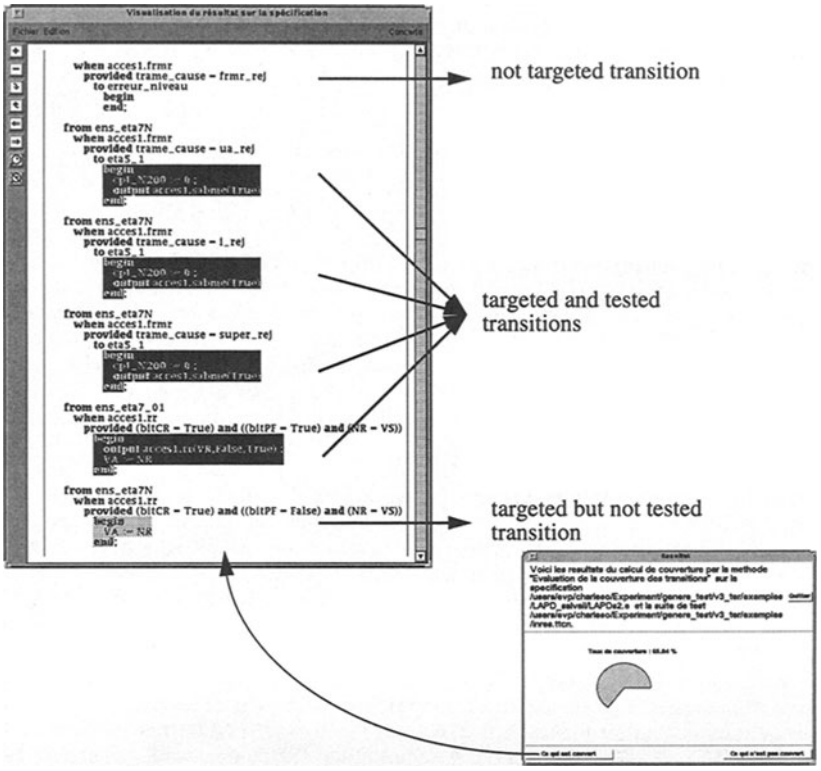

Figure 4 : Graphical/interactive output of tool on LAPD

By analysing the grey transitions, it can be noticed that the spontaneous transitions (with no *when* clause) and some firable transitions on input RR, RNR, REJ are not covered (for example the grey transition). It is obviously impossible to generate tests for the first type. TVEDA can generate test for the second type, but the obtained test suite is much bigger (415 against 198 tests) than the one used for the experiment with TESTL and then does not allow a comparison between the tools.

With the complete test suite generated with TVEDA we got 415 tests. The coverage of the test suite is 97.85%. The remaining uncovered transitions are still the spontaneous transitions.

4.3 Lessons from these experiments

1. The LAPD example shows clearly that just providing a figure (a percentage) is not enough. All the more so as this figure looks arbitrary (in the case of TESTL, it depends on the fault model and on the amount of valid tests). As long as it is not 100% or 0%, any number would do! In fact, we would like to know 'what' faults are not covered, not just the number of them. In short, *we want qualitative information, not just a number.* In the case of our tool the analyse function (that we used without restraint to correct the test suite up to 97.85% of transitions coverage) is a powerful facility that put one's finger on the untested transitions. It

also provides practical information on the way the test suite should be (manually or automatically) completed.

2. The test suite did not test purposefully some parts of the specification. So, computing the coverage on the whole specification leads to insignificant small rates (28.5%). What interpretation should be given to such a number? Is it small because the scope of the specification is too wide or because there are missing tests? By pointing out to our tool the exact part of the specification that should be taken into account, the alternative is reduced to the conclusion that some tests are missing.
3. Numbering killed mutants is far too brutal for a good appraisal of fault coverage. For instance, consider the case of output faults. TESTL will generate all possible variations of the output of a transition. If there are 20 types of messages, there will be 19 mutants. Now, if this transition is tested, all mutants will be killed; and if it is not, all mutants will survive. The impression is that the fault should be analysed once only, because replicating it 19 times is time consuming and it multiplies by 19 the weight of the transition in the coverage, which may be misleading as to the real weight of the corresponding test case (and of the fault).
4. As a consequence of item 3, the computation time of the coverage with our tool (less than 5 minutes) is much smaller than with TESTL (from 5 to 20 hours). This is because TESTL actually generates the mutants and test them. This drawback would disappear with the numbering scheme proposed in [Yao 94].
5. On the other hand, once the results of TESTL are reinterpreted with the help of our tool, TESTL results clearly show a drop in coverage when transfer faults are considered, which our tool could not show.

5 CONCLUSION

In this paper, we have presented a tool for assessing the coverage of test suites. We have focused on the practical needs expressed by conformance test suite designers and users. This has led us to define a rather simple form of coverage which we called 'transition coverage'. The notion itself is not very original (it can be seen as a specialized variant of branch coverage), but our approach has focused on the features that make a coverage tool useful. In fact it can be extended to more sophisticated definitions of coverage.

It might seem strange that we have aimed our first implementation of the coverage tool at automatically generated test suites. A typical misconception is that since we control the generation, the coverage reached is known implicitly from the test generation process and criteria. However, our experience shows that such a tool is an important complement to test generation (this is also acknowledged by [Vuong 93] in the design of Testgen+). First, it gives better confidence in the output of the test generation tool. Apart from bugs in test generation, it is important to note that test generation depends on several inputs: the formal specification, data on the test architecture and all sorts of parameters of the test generation algorithm; in the case of TVEDA, we can produce vastly different test suites starting from the same formal specification. With semi-automatic test generation tools such as STED [Ek 93], the extra input from the user of the test generation tool is even greater. Second, our test tool provides an easy visualization of what the test suite explicitly covers, which is much more helpful to the test designer than the implicit definition provided by generation parameters. Last, since automatic test generation still has many limitations, a test coverage tool is very important to identify the parts that escape automatic generation.

Finally, we should mention that our simple and pragmatic approach provides a convenient path to attract test designers to formal methods. As was the case in our previous experience with test

generation, it shows them that formal methods can be adapted to their primary needs; once this first psychological step is trodden, it becomes possible to introduce more refined notions of coverage (a very trivial example is provided in our LAPD example, where the combination of TESTL and our tool shows how poorly the standardized test suite deals with transfer faults).

REFERENCES

[Bochmann 91] G.v. Bochmann, A Das, R. Dssouli, M. Dubuc, A. Ghedamsi, G. Luo, *Fault Model in testing,* in Proceedings of the IWPTS IV - Int Workshop on Protocol Test Systems, Leinschendam, The Netherlands, 1991.

[Caouette 93] C. Caouette, *Evaluation du flux de données couvert par une suite de tests,* MSc thesis, Université de Montréal, 1993.

[Clatin 95] M. Clatin, R. Groz, M. Phalippou, Richard Thummel. *Two approaches linking a test generation tool with verification techniques.* in Proceedings of the IWPTS VIII - Int Workshop on Protocol Test Systems, Evry, France, 1995.

[Dubuc 91] M. Dubuc, *Sélection de tests pour les protocoles de communication,* MSc thesis, Université de Montréal, 1991.

[Ek 93] A. Ek, J. Ellsberger, A. Wiles, *Experiences with computer aided test suite generation,* proceedings of IWPTS 93, Pau, September 1993.

[Groz 95] R. Groz, M. Phalippou, *La génération automatique de tests est-elle possible ?,* proceedings of CFIP'95, Rennes, May 1995, pp 421-438.

[ISO 94] ISO, *Information Technology, Open Systems Interconnection, Conformance Testing Methodology and Framework,* International Standard IS-9646, 1994. Also available as recommendations X.290 (for Part 1 of ISO document), X.291 (Part 2) ... from ITU-T.

[ISO 95] ISO/IEC/JTC1/SC21/P.54 - ITU-T SG10 Q.8, *Formal Methods in Conformance Testing*, Working Draft, March 1995. To become Recommendation Z.500 from ITU-T.

[Myers 79] G. J. Myers. *The Art of Software Testing*. New York: John Wiley & Sons, 1979.

[Phalippou 90] M. Phalippou, R. Groz, *Evaluation of an empirical approach for computer-aided test cases generation*, proceedings of the 3rd IWPTS, Washington, October 1990.

[Phalippou 94] M. Phalippou, *Test sequence generation using Estelle or SDL structure information*, proceedings of FORTE'94, Bern, October 1994.

[Renévot 95] J. Renévot. *Définition et réalisation d'un outil d'analyse de couverture de test sous Concerto*. Rapport RP/LAA/EIA/77, CNET, Octobre 1995.

[Salvail 94] P. Salvail. *Calcul de couverture de tests pour le protocole D, niveaux 2 et 3*. Rapport RP/LAA/EIA/17, CNET, Février 1994.

[Sarikaya 85] B. Sarikaya, G. v. Bochmann, *Obtaining Normal Form Specifications for Protocols*, proceedings of COMNET'85, Budapest, October 1985, pp 601-612.

[Tretmans 93] J. Tretmans. *A Formal Approach to Conformance Testing*. proceedings of IWPTS 93, Pau, September 1993, Omar Rafiq ed, North Holland, pp 257-276.

[Vuong 91] S. T. Vuong, J, Curgus. *On Test coverage Metrics for Communication Protocols*. in Proceedings of the IWPTS IV - Int Workshop on Protocol Test Systems, Leinschendam, The Netherlands, 1991.

[Vuong 93] S. T. Vuong, S. Lee. *TESTGEN+: An Integrated Environment for Protocol Test Suite Generation, Selection and Validation*. in Proc. of the FORTE'93, Boston, USA, 1993.

[Yao 94] M. Yao, A. Petrenko, G. V. Bochmann, *A Structural Analysis to the Evaluation of Fault Coverage for Protocol Conformance Testing*. in Proceedings of the FORTE 94, Bern, Switzerland, October 1994.

[Zhu 94] J. Zhu, S. T. Chanson. *Toward Evaluating Fault Coverage of Protocol Test Sequences*. in Proceedings of the PSTV XIV, Vancouver, Canada, June 1994.