1

DIFFERENT APPROACHES TO PROTOCOL AND SERVICE TESTING

Ana Cavalli
Institut National des Telecommunications
9 rue Charles Fourier
F-91011 Evry Cedex
Ana.Cavalli@int-evry.fr

Abstract This paper presents an overview of the work we have developed at INT on different approaches to protocol and service testing. It is related to performing tests of communication protocols and telecommunication services based on techniques, such as, active test generation, passive testing, embedded testing and test execution.

Keywords: Conformance Testing, Test Generation, Embedded Testing, Passive Testing, SDL, CORBA.

1. INTRODUCTION

This paper presents an overview of the work we have developed at INT related to different approaches to performing tests of communication protocols and telecommunication services. It concerns active test generation, passive testing, embedded testing, test execution and their application to real protocols and services.

This work is mainly a work developed by my research group at INT, by myself and my PhD students [24, 9, 26, 27, 28, 33, 19, 25] but it is also the result of collaborations with colleagues from CNET [3, 6], Bell Labs [4, 10] and academic collaborations [5, 36, 35, 14]. It concerns different aspects of testing, that we have studied in the last years. This work covers all phases of the testing activity. We started by mainly studying the application of formal methods to test generation and we have developed a prototype tool, called TESTGEN. This tool allows automatic test generation from LOTOS and SDL specifications. In its first version it was strongly influenced by I/O FSMs test generation

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35567-2_25

G. Csopaki et al. (eds.), *Testing of Communicating Systems*

methods [5].
To solve the problem of combinatorial explosion of states and to perform embedded testing, we have integrated a new composition algorithm. This algorithm hides internal actions and controls the composition process in order to identify global transitions that reflect the behavior of the component under test [26, 28].

Following this work, we have defined and implemented new algorithms to optimize the resulting test sequences and to perform still embedded testing. We have then integrated two new algorithms: one to optimize the test generation [4] and the other to perform embedded testing using a transition coverage criteria of the embedded component [10]. Finally, we have integrated an algorithm that introduces a new approach to testing, passive testing [33].

While we were working on developing these testing methods, we started to work on open distributed architectures for telecommunications networks. We considered that it was interesting to execute the tests on a distributed environment in order to test systems implementations obtained from formal specifications. We have therefore developed an execution test platform into a CORBA environment [25].

In the following, I will provide a more detailed description of each of these steps of our work and give some previews of our future work.

2. THE TESTGEN TOOL

2.1 PRELIMINARIES

A program specification is composed of a control part and a data part. This section deals with the control part only. The control portion of a program, which will be referred to as a *program specification*, can be modelled as an Input/Output Finite State Machine (I/O FSM) with a finite set of states S=$\{s_1, s_2,..., s_n\}$, a finite set of inputs I=$\{a_1,..., a_k\}$, and a finite set of outputs O=$\{x_1,x_2,...,x_m\}$. Both sets include the symbol *null* used to represent *null* inputs or outputs. The next state (δ) and output (λ) relation are given by the relations $\delta : S \times I \rightarrow S$ and $\lambda : S \times I \rightarrow O$.

The I/O FSM is usually represented by a directed graph G=(V,E), where V=$\{v_1,v_2,...,v_n\}$ represents the set of states S, and a directed edge represents a transition from one state to another in the I/O FSM. Each edge in G is labelled by an input a_r and a corresponding output x_q. An edge in E from v_i to v_j which has label a_r/x_q means that the I/O FSM, in state s_i, upon receiving input a_r produces output x_q and moves to state s_j. A triple $(s_i,s_j,a_r/x_q)$ is used to denote a transition.

The graph representation is useful for describing and reasoning about test generation algorithms. Within this context, we give some basic definitions from graph theory. A graph is said to be strongly connected if for any pair of

distinct vertices v_i and v_j there is a walk which starts on v_i and ends on v_j. A walk W over a graph is a finite, non-empty sequence of consecutive edges. Head(W) and Tail(W) denote respectively the vertex where the walk W starts and the vertex where it ends. A path is a walk in which each edge of G appears exactly once.

An input/output sequence U = (a_{r1}/x_{q1},a_{r2}/x_{q2},...,a_{rn}/x_{qn}) is said to be specified for state s_i in a specification I/O FSM if there exists a walk W with origin s_i in the graph representation of the I/O FSM such that W = (s_i,s_{j1},a_{r1}/x_{q1}), (s_{j1},s_{j2},a_{r2}/x_{q2}),...,(s_{jn-1},s_{jn},a_{rn}/x_{qn}). An I/O FSM is said to be fully specified if from each state it has a transition for every input symbol; otherwise the I/O FSM is said to be partially specified. If an I/O FSM is partially specified and a non specified transition is applied, under the Completeness Assumption it is obtained that the I/O FSM will either stay in the same state without any output or signal an error. The initial state of an I/O FSM is the state the I/O FSM enters after power-up. An I/O FSM is said to have the reset capability if it can move from any state directly into the initial state with a single transition, denoted "*ri/null*" or simply "*ri*". State s_i is said to be weakly equivalent to state s_j if any specified input/output sequence for s_i is also specified for s_j.

A Unique Input/Ouput sequence (UIO sequence) for state s_i, denoted UIO(i), is an input/output sequence with origin s_i such that there is no $s_j \neq s_i$ for which UIO(i) is a specified sequence for starting state s_j [1].

A state possesses multiple UIO sequences if it possesses more than one UIO sequence. Some I/O FSMs do not possess UIO sequences for every state. The proposed approach to identify a state s_i that does not have an UIO is to use a signature, a sequence that distinguishes s_i from each of the remaining states one at a time. A signature for s_i uses (n-1) input/output sequences IO(i,m), each of which distinguishes s_i from one other state s_m. Before applying each IO sequence the I/O FSM must be put back into state s_i. Suppose that after the subsequence IO(i,m) is applied the specification is in state s_k. The method proposed in [1] is to use a transfer sequence Tr(k,i), which is the shortest path from s_k to s_i, in order to bring the I/O FSM back to state s_i. Therefore, the signature for state s_i is formed by concatenating the sequences $\mathrm{IO}(i,m)@Tr(Tail(\mathrm{IO}(i,m)),i)$ for all $m \neq i$, where @ indicates the concatenation of each IO sequence with the corresponding transfer sequence. The test generation method presented in the following sections uses a modified version of the concept of signature, based on the SIO sequences (Set of IO sequences), that we will denoted by Partial Unique Input/Ouput sequence (PUIO).

2.2 THE PROCEDURE FOR TESTING

The procedure for testing an implementation (whose behaviour can be described through an I/O FSM) is essentially divided into two parts [5]:

1 state identification: the implementation is forced to display the response of each state to the Unique Input/Output sequence (UIO) (or to the chosen signature) checking whether an implementation I/O FSM has n states and that each state identifier preserves its feature. This is carried out by repeating the following procedure for each state of the I/O FSM:

   - the implementation is put into state p_i, presumably isomorphic to s_i, by applying a reset (assumed reliable) followed by some path (already tested) from the initial state s_0 to s_i;
   - UIO (or other chosen signature) is applied to the implementation and the output is checked to verify whether it was expected or not.

2 transition testing: each transition is tested. The procedure for testing a transition t = (s_i, s_j, a_r/x_q) from state s_i to s_j with input a_r is the following:

   - the implementation is put into state p_i, known to be isomorphic to s_i, from the state identification part;
   - input a_r is applied and the output is checked to assure that it is x_q as expected;
   - the new state of the implementation is then tested to verify if it is isomorphic to state p_j. This is done by applying the UIO (or the chosen signature) as in the state identification part.

Any method based on UIOs, or a similar kind of signature, is valid if the UIOs or other signatures are unique both in the specification and the implementation. This uniqueness of the identification state must be verified. This can be done by putting the implementation into state p_i, presumable isomorphic to state s_i, and then applying UIO(j) to the implementation in order to verify that the resulting output is not what it was expected if UIO(j) was applied to s_j.

2.3 THE TOOL TESTGEN

In the first version of our tool TESTGEN, we have adapted the traditional test methods, based on finite automata [1, 32] to test generation and their application to real protocols. In the previous sections we have provided some basic concepts of these methods. These methods present two main advantages: they allow a totally automatic test generation, and they guarantee full coverage for certain types of faults (transfer and output faults) under certain test hypothesis.

In the following we will present test generation from a SDL specification [20], (for a description of test generation from LOTOS, see [7, 8]). The procedure to generate test sequences from a SDL specification can be decomposed in the following steps:

Step 1. Note that the method is applied to a SDL specification. This means that we start with a system of communicating extended finite state machines. This system will be simulated in order to obtain an accessibility graph. The accessibility graph that includes data and control parts of the original specification is computed using the simulator features of the ObjectGEODE tool [34]. This graph is restricted by fixing the values of the parameters in messages received from the environment by the system under test. An I/O FSM is obtained. The constraint on the arguments values is a very important optimization, since it permits the tool to obtain a graph even in the case of a real protocol;

Step 2. The obtained input/output machine is transformed: transitions labelled by internal actions and intermediate states are eliminated using an algorithm described briefly in the following section. We then obtain a *reference finite state machine* (see next section);

Step 3. Certain cases of non-determinism are eliminated using a classical determinization algorithm [2], while others cases, which may be significant for the test, are conserved. Furthermore, the *reference finite state machine* is transformed in a completely specified machine by the addition, on every state, of self-loops having as input labels the inputs not specified for that state and *null* as outputs labels. Notice that this transformation preserves the semantics of SDL. In fact, in certain cases it is not necessary to make every state complete. It is necessary only in certain states, so a unique identification sequence can be obtained;

Step 4. A minimization algorithm based on trace equivalence is applied. This minimization is important, since it makes it possible to obtain states identification sequences of reasonable size even for real protocols. In this step, we also verify that the finite state machine is strongly connected;

Step 5. UIOs , multiple UIOs (MUIOs) or partial UIOs (PUIOs) are defined for each state. TESTGEN allows the use of these different types of UIO sequences. By default, the type that generates the shortest test sequence is chosen;

Step 6. For each state of the minimized automaton, a preamble to put the implementation in that state is computed. Then, if possible, one or more UIO sequences for the state are computed. As some states may not have a UIO sequence, partial UIOs (PUOs) are computed for them.

Step 7. The test sequence is calculated. Once preambles and UIO (or PUIO) are obtained, the generation of the test sequence proceeds in two steps as follows:

- 7.1. First, it is verified that UIO and/or PUIO sequences are uniquely accepted by the states in the implementation;

- 7.2. Then, every transition in the specification is verified to have a corresponding transition in the implementation. This is done by applying to the implementation first the preamble, in order to put it at the start of the transition, then the transition itself, and finally the UIO sequences to verify that the transition put the implementation in the expected end state;

Step 8. The test generation can operate in two modes, optimized or non-optimized. In the optimized mode, one single test sequence is obtained by using an optimization algorithm (either Rural Chinese Postman Tour [1], or greedy algorithm with overlaps and multiple UIOs [11]). In the non-optimized mode, a set of test independent test cases is obtained. In both cases, the resulting test is translated into TTCN.

2.4 GENERATION OF THE REFERENCE FINITE STATE MACHINE

The global finite state machine (obtained using the ObjectGEODE tool) can have redundant states, as well as transitions labelled by internal or empty (*null*) messages. In order to reduce the size of the finite state machine, two transformations are made:

1. *Treatment of internal and empty signals.* In the global finite state machine, we distinguish actions that interact with the PCOs (Points of Control and Observation, which constitute the interface between the tester and the implementation under test) and actions that are not seen at the PCOs. The latter are called internal actions (denoted by τ). The test architecture defining the PCOs will determine the actions that will be considered as internal. Empty signals (denoted *null*) are used to model the input of spontaneous transitions, or the output of actions that do not produce events.

We apply a reduction algorithm to the automaton in order to eliminate part of the internal actions and empty signals, obtaining a finite state machine with fewer states. The principle of this reduction algorithm is to detect certain chains of transitions that can be grouped, for they are all internal. The simplest case is when two consecutive transitions have the form "in/τ" and "τ/out" where "in" and "out" are actions observable at the PCOs. In this case, these two transitions are replaced by a single transition labelled "in/out"; if there are no other incoming or outgoing transitions, the intermediate state is eliminated. *Null* signals are eliminated using the same method. But it must be noted that the algorithm treats "τ" and "*null*" separately and keeps them distinct in the resulting machine. This way, certain spontaneous actions are removed; for example those that appear after the start state in a SDL specification. On the other hand, there may be transitions of the form "in/τ" (a signal from the

environment followed by an internal signal), or "in/null" (a signal from the environment followed by a *null* signal) that cannot be removed. This is due to the fact that those transitions model the situations where the specified system accept an input signal but does not produce an observable answer. Transitions labelled with internal signals that cannot be deleted/hidden indicated a bad specified or incomplete machine. A more detailed description of this algorithm is given in [3, 28]. The experience of using this algorithm has shown that the resulting automata has fewer states than the original one. On the other hand, the number of transitions usually increases. The reasons for this is that when one state is eliminated the number of resulting transitions correspond to the number of incoming transitions multiplied by the number of outgoing transitions on that state. However, if the number of states eliminated is substantial, then the number of transitions also decreases. In the case of the MAP protocol [3], the algorithm reduces an automaton with 13.794 states and 45.893 transitions to an automaton with 6.193 states and 43.445 transitions. The application to the INRES protocol [15] gives the following results: it reduced an automaton of 175 states and 267 transitions to an automaton with 60 states and 154 transitions.

2. Minimization based on trace equivalence. This minimization eliminates redundancies and therefore reduces the number of states and transitions of the automata. In the case of the MAP protocol [3], this minimization produced an automaton with 25 states and 187 transitions, from an automaton with 6.193 states and 43.445 transitions.

3. EMBEDDING TESTING

Traditional testing methods proposed to test systems as a whole or to test their components in isolation. Testing systems as a whole becomes difficult due to their huge size. On the other hand, the purpose of testing can be to test only some system components in order to avoid the redundant testing of other parts of the system. In this case, we are concerned with embedded testing or testing in context [24, 26, 28, 30, 29, 23, 13].

Embedded testing is to test the component's behaviour in context and to detect if the component's implementation conforms to its specification. It is generally assumed that the tester does not have a direct access to the component interfaces. The access is then made through another component of the system, which acts as a sort of "filter". According to the standard: "if control and observation are applied through one or more implementations which are above the protocol to be tested, the testing methods are called embedded" [16].

Many of these systems can be formally specified as a system of distributed interacting extended finite state machines (EFSMs). EFSMs, which are finite state machines extended with variables, have emerged for the design and analy-

sis of both communication protocols and services. They are also the underlying model of some formal specification languages (as for example, SDL [20]).

Concerning embedded testing we have developed two approaches. The first one is based on a trace keeping algorithm for automaton composition. A detailed presentation of this algorithm is given in [26] and [28]. The generation of test sequences from formal specifications has been traditionally based on the exhaustive simulation of the specification in order to obtain an automaton that represents the global behaviour of the system. Since it is impossible, in most of the cases, to deal with the size of the automaton that represents the complete behaviour of these systems, a reasonable approach is to simulate the execution of the specification by controlling the range of values assigned to each internal variable and each parameter of input messages. The closer this range is to the real one, the more realistic and the larger the test will be. Obviously, there is always a compromise between accuracy (completeness) of the automaton and his size. However, even with an automaton of a "computable" size, the process of test sequence generation may not be able to cope with that automaton in a reasonable period of time.

In the first version of TESTGEN, to generate test sequences, what we have done is to take the "big" automaton (that is, the one which is as close to the specification as possible) and then, though the definition of view points (PCOs), abstract the signals which are irrelevant in the current consideration or view point. Then, we may proceed by minimizing the automaton using an algorithm described briefly in the previous section and presented in [5] and [28], which removes all internal signals (if the choice of the PCOs is well done). We thereby obtain an automaton that corresponds to the "big one", by abstracting details we do not yet want to consider. In general, this automaton has a reasonable size, and therefore it can be used as input for the process of deriving test sequences. However, even with a big automaton generated by simulation, the reduced one is often simpler than we would like.

To solve these problems we have considered the use of a composition algorithm in the following way. The idea is to avoid the initial automaton size explosion by dividing (which is often already done, if we are dealing with modular systems) the specification into smaller, interrelated modules which are then simulated to produce more complete or smaller automata. The simple composition of these automata, without taking into account any kind of abstraction (PCOs), would lead to the big automaton of the traditional case. However, if we use information about our abstraction level we are able to compose them and at the same time avoid the explosion of the model. In other words, we compute the global composition of two automata while removing internal actions.

The implementation of the proposed algorithm also includes mechanisms to tackle the following issues: multiple (simultaneous) outputs (i.e. when one signal from the environment stimulates several simultaneous outputs from the

system), deadlock and livelock detection (if components exchange messages indefinitely). A complex strategy for transition marking implements a mechanism of behaviour exploration that guarantees that all possible branches are examined, even in the presence of nondeterminism. This is implemented by allowing processes to warn the builder when there was a non deterministic choice for the last input, so that the builder can send the same signal a second time and a different transition will then be traversed. As a result, non deterministic machines can be produced.

The algorithm also allows the detection of undesired situations that may happen during the composition process and that are reported back by the tools as error or warning messages:

- incompatibility errors due that a process was not expecting a given internal signal from its peer machine at its current state. In this case, the internal signal is simply forwarded to the environment that instantiates a global transition with an error message (for it contains an internal output signal that was not supposed to reach the environment);

- unreachability warnings. During the machines join execution, some transitions of either machine may not be traversed and some states even may not be visited. This means that a part of the machine behavior was not exercised in the joint execution (e.g. one machine was not capable of using all the other machine's functionality). This kind of information can be useful, for instance, for the detection of some cases of feature interaction or to identify the executable part of a component in a given context.

We have also developed another approach to embedded testing that is based in a general procedure for embedded testing of EFSMs. This is a work that we have presented in [10]. This procedure can be applied to EFSMs, to Communicating Finite State Machines (CFSMs) and to Communicating Extended Finite State Machines (CEFSMs). The proposed algorithm is based on a "hit or jump" random walk that attempts to cover all transitions of the component EFSMÕs. This procedure has the advantage of avoiding the construction of the system reachability graph. As a matter of fact, the space required is determined by the user, and it is independent of the systems under consideration. The algorithm has been implemented on the ObjectGEODE tool. It takes advantage of some of the functionalities offered by this tool: for instance the use of the simulator to produce partial graphs that will be used to produce the test scenarios.

The first approach for embedding testing that we have developed has been applied to embedded testing of the GSM-MAP protocol [19]. Both approaches have been applied to the embedded testing of services on a telephone network. This case study corresponds to a real system that has been specified using the SDL language. In addition to the Basic Call Services (BCSs), two

other services are included: Originating Call Screening (OCS) and Call Forward Unconditional (CFU). These applications are described in [27, 28, 10] respectively.

4. PASSIVE TESTING

The objective of conformance testing is to evaluate whether a protocol implementation has the same behaviour as its specification. In active conformance testing, the implementation under test (IUT) is placed in a dedicated testing environment. According to predefined test sequences, one (or more) tester sends inputs to the IUT, observes the corresponding outputs and compares them with the expected outputs. On the other hand, passive testing consists in collecting traces of the messages exchanged between an operating implementation and its environment, and verifying whether these traces actually belong to the language accepted by the specification automaton. Though passive testing is sometimes mentioned as an alternative to active testing [17], only little effort has been devoted to this aspect of testing [31, 12]. However, we feel that passive testing is worth investigating, since (a) under certain circumstances, it may be the only type of test available, e.g. in network management, (b) it is relatively cheap and easy to implement, and (c) active testing is sometimes impractical due to the complexity of systems.

In passive testing [31], contrary to active testing, the tester does not control the implementation under test. The implementation is in operating condition, and the tester only observes the messages exchanged by the IUT and its environment, in order to check if they correspond to a behaviour compliant with the specification. The main difficulty in passive testing is that we have no knowledge of the state in which the implementation is at the beginning of the trace (no assumption is made about the moment when the recording of trace begins, and therefore it is not necessarily the initial state). Each input/output pair of the trace is assumed to represent a transition in the specification, and our objective is to match the transitions of the trace with those of the specification. The passive testing process can be decomposed into two steps: the passive homing sequence, in which the current state is found out, and the fault detection phase, in which the trace is compared with the specification.

We have defined an extension of the existing algorithms to consider the specification as an extended finite state machine (EFSM) that is presented in [33]. We also introduce the principles of passive testing, and an application to a real protocol, the GSM-MAP. The proposed algorithm is an extension of this of [12] in order to consider an extended finite state machine as the system specification. The SDL specification of the GSM-MAP that we used in our experiment has been obtained from ETSI standard; this specification has been completed in our group [19, 3]. For the purpose of conformance testing, the

specification is usually considered as an Extended Finite State Machine, the underlying model of SDL.

The first results were the detection of discrepancies between the global SDL specification and the behaviour of the FSM obtained through the decomposition/composition process. These discrepancies were detected within 10 input/output pairs after the end of the passive homing sequence and were reported as faults by the passive testing software. They turned out to be mostly due to some variable values that were inconsistent between the various processes. This allowed us to correct these inconsistencies in the specification. After correction, tests were rerun with the same traces, and no faults were detected, which was the expected outcome. In order to assess our tool, faults were also added manually in some of the traces. Those faults were detected within a few input/output pairs after they occurred.

Traces varied from 1.000 to about 12.500 input/output pairs. Computation time varied between 0.04 to 0.14 seconds on a Sun Ultra1. Transition coverage varied between 5% and 25% of the transitions, and state coverage varied between 56% and 71% of the states. The global coverage, obtained by taking into account all the states reached in the various experiments, and all the transitions, was 86% of the states and 46% of the transitions.

5. EXECUTION OF TESTS ON A CORBA PLATFORM

Intelligent Networks (IN) represented a remarkable step forward in the domain of telecommunications systems. The ability to add "intelligent" nodes (i.e. computers) into switching networks made it possible to introduce new telephonic services. However the service implementation was still dependent on the hardware and operating system upon which it was built. Nowadays, telecommunication services are migrating to open distributed environments where a layer, called middleware, located between the software application layer and the hardware layer (i.e. physical and low level programs like operating systems), assures transparency, interoperability and cooperation among the several service components in spite of the heterogeneous underlying systems. The Telecommunication Information Networking Architectures (TINA) specify the DPE (Distributed Processing Environment) layer which is responsible for providing facilities to service programmers, so that they do not need to cope with communication details.

There are many different validation methods for telecommunications systems as we have mentioned in the previous sections. All these methods correspond to what we call the Testing Generation Phase, which starts with the system specification and ends with the generation of test suites. However very little has been said about validation of telecommunication services, specifically, regarding distributed environments like ORB platforms. Thus, on the one hand

we see a world-wide move to using distributed platforms to implement telecommunications services and on the other hand the need for suitable methods to validate these systems.

Our works in this area aim at implementing and validating services on top of a TINA-CORBA like platform [25]. In the following we will provide a short presentation of the implementation and validation phases that we have developed in our test platform.

For the validation a tester (possibly distributed) must be implemented to send stimuli to the several components under test (which compose the System Under Test - SUT) and to gather information concerning their reaction to these stimuli. Eventually, the tester will give a verdict on the observed behaviour of the system. In doing so , we will be able to come up with a practical view on the following aspects :

- test generation and execution;
- distributed test execution and generation;
- how to structure the test architecture for TINA-CORBA-like system/objects.

The distributed platform chosen was Orbix (TM) [18], which is a full implementation of OMG-CORBA. The key advantage of such platforms is that they make application construction and integration much easier, since software interfaces are defined using a standard language, the Interface Description Language (IDL) which allows objects to be accessed from anywhere in a distributed system.

In this platform, all interactions (i.e. service requests) are performed by means of channels. Each channel is composed of a stub (on the client side) and a skeleton (on the server side) which are connected to the Object Request Broker (ORB). Channels allow a transparent communication between the client and the object implementation. The ORB in turn provides the functionality required to communicate across (possibly heterogeneous) platforms.

A service request consists of the following information : a target object, an operation, a list of parameters and an optional request context. "One way" operations (called announcements in the ODP terminology [22]) allow asynchronous message exchange amongst the various system components. It is important to point out that a client can simultaneously be a server to another object (or vice-versa). Objects must cooperate (or interoperate) in a useful way to achieve a given objective, which is exactly what TINA-CORBA provides.

Following the guidelines of classical test architectures for protocols as described in [16], we suggest that a test architecture for an open distributed object-oriented platform must have two sets of components: tester components and components under test that are possibly distributed over several sites. The

tester components receive information about the object configuration (i.e. the collection of objects able to interact at their interfaces) and the test suite which is obtained through some test generation method. Simple systems (with local architecture) are usually implemented using one single tester component and a single component under test.

The results of our work are encouraging, particularly taking into account that the majority of our colleagues working on the implementation of multimedia services on a TINA-CORBA platform do testing manually. Once the module representing a service has been implemented, they manually define a test script. They repeatedly express the need for a test environment where all phases of the testing are automated. The testing environment that we have presented in this paper is an effort in this direction: all phases are automated and the test results obtained are more complete. Another advantage is the possibility of using existing formal description techniques to describe the system (SDL), the test suite obtained (using TTCN) and the test script (using MSC) [21].

6. CONCLUSIONS AND FUTURE WORK

The main objective of all the work we have performed is the application of formal methods for conformance testing to real protocols and services. Our work also contributes to increase the use of formal description techniques and formal methods for testing in industry. In France, the use of the SDL language in industry has considerably increased in the last years and we realize, through our industrial collaborations, that the use of these techniques and associated tools is an accomplished fact.

In the future, we will continue our work related to the test of services on RI and TINA-CORBA architectures. We are applying our algorithms for embedded testing to the test of components of an audio and multimedia services specified following the requirements of this architecture. We are also working on extensions of the testing platform in order to be able to execute test on services implemented in these environments. We have also started to work in how to provide formal descriptions and validation methods for service administration. Furthermore, we will continue our collaborative work concerning more theoretical aspects of embedding testing, as presented in this conference [36, 35], on optimization techniques for test generation [4, 10] and on the test of timed systems [14].

References

[1] A. Aho, A. Dabhura, D. Lee, and U. Uyar. An optimization technique for protocol conformance test generation based on UIO sequences and rural chinese postman tours. In S. Aggarwal and K. Sabnani, editors, *PSTV'88*, pages 75–86, North Holland, 1988.

[2] A. Aho, R. Sethi, and J. Ulman. *Compilers, Principles, Techniques and Tools*. Addison-Wesley Publishing Company, 1986.

[3] R. Anido, A. Cavalli, L. Paula Lima, M. Clatin, and M. Phalippou. Engendrer des tests pour un vrai protocole grâce à des techniques éprouvées de vérification. In *Proceedings of CFIP'96*, Rabat, Marocco, 14-17 October 1996.

[4] C. Besse, A. Cavalli, and David Lee. Optimization techniques and automatic test generation for TCP/IP protocols. Research Report RR 99.06, INT, France, March 1999.

[5] A. Cavalli, B. M. Chin, and K. Chon. Testing methods for SDL systems. In *Computer Networks and ISDN Systems*, volume 28, pages 1669–1683, 1996.

[6] A. Cavalli, J. P. Favreau, and Marc Phalippou. Standardization of formal methods in conformance testing of communication protocols. In *Computer Networks and ISDN Systems*, volume 29, pages 3–14, December 1996.

[7] A. Cavalli, S. Kim, and P. Maigron. Automated protocol conformance test generation based on formal methods for LOTOS specifications. In *IWPTS'92*, Montreal, Canada, September 1992. Elsevier Science Publishers.

[8] A. Cavalli, S. Kim, and P. Maigron. Improving conformance testing for LOTOS. In *FORTE'93*, Boston, USA, October 1993.

[9] A. Cavalli, B. Lee, and T. Macavei. Test generation for the SSCOP-ATM networks protocol. In *SDL Forum'97*, France, September 1997.

[10] A. Cavalli, David Lee, Christian Rinderknecht, and F. Zaïdi. Hit-or-jump: An Algorithm For Embedded Testing With Applications To IN Services. Research Report RR 99.05, INT, France, March 1999.

[11] M. S. Chen, Y. Choi, and A. Kershenbaum. Approaches utilizing segment overlap to minimize test sequences. In *PSTV'90*, pages 67–84, Canada, June 1990.

[12] David. Lee et al. Passive testing and applications to network management. In *ICNP'97 International Conference on Network Protocols*, Atlanta, Georgia, 28-31 October 1997.

[13] M. A. Fecko, U. Uyar, A. S. Sethi, and P. Amer. Issues in conformance testing: Multiple semicontrollable interfaces. In *Proceedings of FORTE'97*, Paris, France, November 1998.

[14] T. Higashino, Nakata, K. Taniguchi, and A. Cavalli. Generating test cases for a timed I/O automaton model. In *Proceedings of the 12th IWTCS*, Budapest, Hungary, September 1999.

[15] D. Hogrefe. Osi formal specification case study: the inres protocol and service, revised. Technical report, Institut für Informatik Universität Bern, may 1992.

[16] International ISO/IEC multipart standard. *ISO/IEC 9646 Information Technology - OSI - Conformance Testing Methodology and Framework.*, 1994.

[17] International Standard 10746-2/ITU-T Recommendation X.902. *Open distributed Processing - Reference Model - Part 2: Foundations.*

[18] IONA Technologies Ltd. *Orbix 2 Programming Guide*, November 1995. P1.

[19] M. Ionescu and A. Cavalli. Test imbriqué du protocole MAP-GSM. In *Proceedings of CFIP'99*, Nancy, France, 26-29 April 1999.

[20] ITU-T, Geneva. *CCITT, Specification and Description Language, CCITT Z.100, International Consultative Committee on Telegraphy and Telephony*, 1992.

[21] ITU-T Rec. Z.120, Geneva. *Message Sequence Chart (MSC)*, 1996.

[22] ITU-T Recommendation X.903. *ISO/ITU-T - Open Distributed Processing - Reference Model - Part 3: Architecture - International Standard 10746-3*, 1995.

[23] D. Lee, K. Sabnani, D. Kristol, and S. Paul. Conformance testing of protocols specified as communicating finite state machines - a guided random walk based approach. In *IEEE Transactions on Communications*, volume 44, No.5, May 1996.

[24] L. P. Lima and A. Cavalli. An embedded testing approach. In *Proceedings of EUNICE Summer School*, Lausanne, Switzerland, 23-27 June 1996.

[25] L. P. Lima and A. Cavalli. Test execution of telecommunication service using CORBA. In *Proceedings FMOODS'97*, pages 409–422, Canterbury, United Kingdom, 21-23 July 1997.

[26] L. P. Lima and A. R. Cavalli. A pragmatic approach to generating test sequences for embedded systems. In *Proceedings of the 10th IWTCS*, pages 125–140, 1997.

[27] L. P. Lima and A. R. Cavalli. Application of embedded testing methods to service validation. In *2nd IEEE Intern. Conf. On Formal Engineering methods*, Brisbane, Australia, 1998.

[28] Luiz P. Lima. *Une Méthode Pragmatique de Génération de Séquences de Test pour les SystŁmes Imbriqués.* PhD thesis, Université d'Évry, France, November 1998.

[29] A. Petrenko and N. Yevtushenko. Testing faults in embedded components. In *Proceedings of IWTCS'97*, Cheju Island, Korea, September 1997.

[30] A. Petrenko, N. Yevtushenko, and G. V. Bochmann. Fault models for testing in context. In *Proceeding of FORTE*, Kaiserslatern, Germany, October 1996.

[31] Charles L. Seitz. *An Approach to designing checking experiments based on a dynamic model.* Z. Kohavi and A. Paz ED. Academic Press, 1972.

[32] D. P. Sidhu and T. K. Leung. Formal methods for protocol testinf: a detailed study. In *IEEE Transaction on Software Engineering*, volume 15, No.4, April 1989.

[33] M. Tabourier and A. Cavalli. Passive testing and application to the GSM-MAP protocol. *Journal of Information and Sotware Technology, Elsevier Science*, December 1999.

[34] Verilog, France. *Geode Editor - Reference Manual*, 1996.

[35] N. Yevtushenko, A. R. Cavalli, and R. Anido. Test suite minimization for embedded nondeterministic finite state machines. In *Proceedings of the 12th IWTCS*, Budapest, Hungary, September 1999.

[36] N. Yevtushenko, A. R. Cavalli, and L. P. Lima. Test suite minimization for testing in context. In *IWTCS'98*, Tomsk, Russia, August 1998.