AUTOMATED GENERATION OF INTEROPERABILITY TESTS*

Cédric Besse*, Ana Cavalli*, Myungchul Kim**, Fatiha Zadi*

**Institut National des Telecommunications*

9 rue Charles Fourier

F-91011 Evry Cedex, France

Email : { Cedric.Besse, Ana.Cavalli, Fatiha.Zadi } @int-evry.fr

***Information and Communications University*

Hwaam-dong 58-4, Yusung-gu

Taejon 305-732, Korea

Email : mckim@icu.ac.kr

Abstract In this paper we present two methods for automated interoperability tests generation. The tests are applied to different entities implementing the same protocol and are peer-to-peer tests. They can be executed on test architectures either including only Points of Control and Observation (PCOs), or including both PCOs and Point of Observation (POs) to facilitate the diagnosis after the test execution. Both methods are applied in a case study to the interoperability test of the TCP/IP protocol. The protocol used for the experiment is TCP-Reno, which is specified in SDL. It is one of the most commonly referenced implementations. In the paper we present the tests sequences obtained and compare the experimental results of both methods.

Keywords: Conformance testing, interoperability testing, automated test sequences generation, fault coverage, SDL, TCP/IP protocol.

1. INTRODUCTION

Conformance testing of a protocol component means to test the conformance of the component to its specification, but this is not enough when this component needs to interact and communicate with other components. Moreover, with the development of networks and, particularly of the Internet, a large family of protocols and new services are

*This work has been partially supported by Korean Science and Engineering Foundation and ARIEL-Conference des Grandes Ecoles, France (No. 995-0900-007-2).

coming up. If two or more components that are a part of different communicating systems must interact between them to perform a task, it must be validated that they can interact as expected. The evaluation of this interoperability is one of the essential aspects of the correctness of the integrated systems.

Interoperability tests need to be performed to evaluate the interoperability of the integrated components that can be implemented by different providers. In addition, although each component is usually considered as a black-box, a finer degree of observability is needed to enable a diagnosis, which determines the causes of non-interoperability, after tests execution.

Several methods have been proposed on interoperability testing [13, 9, 11, 4]. One of the main issues in these works is the definition of a test case selection criterion that guarantees the coverage of faults related to components interoperability. Some of these works only provide testing of the control part of a system under test [13, 11].

The aim of the work presented in this paper is to provide an answer to the previous issues. We first propose two methods for automated the interoperability test generation. The tests are peer-to-peer tests. The first method is based on an algorithm designed originally for automated test generation for embedded components of a complex system [5]. It uses the interactions of the embedded component as a guide for test sequence generation. We have adapted this algorithm for interoperability test generation: we now use the interactions between the interacting entities as a criterion for the selection of test cases and as a guide for the test sequences generation. Thus we do not generate redundant tests. The second method we present is based on an algorithm designed originally for the optimization of test suites [2]. The algorithm is based on the coloring of some transitions of the original specification. Each transition representing an interaction of an entity corresponds to several transitions of the global graph. We assign the same color to all such transitions. In this way, we define a criteria for the selection of interoperability tests: we produce a minimum set of paths that covers all the colored transitions. In addition, both methods take data into account when generating tests (test sequences include parameters values).

There are a number of test architectures for interoperability testing [4]. Some of them include Points of Control and Observation(PCOs) only and some of them include both PCOs and Point of Observations (POs). Tests generated by the methods presented in this paper can be applied to both test architectures. However, we consider the test architectures that include both PCOs and POs, which have more observability, in order to facilitate a fault diagnosis.

We have applied both methods to generate interoperability tests of the TCP/IP protocol. The protocol used for the experiment is TCP-Reno [10], which is specified in SDL [7]. It is one of the most commonly referenced implementations. These implementations can be supplied by different providers. As a result of our work we can generate two sequences of tests, generated by the two proposed methods. The generated test sequences can be used for interoperability testing between two entities of the TCP/IP protocol. Note that for TCP/IP tests peer-to-peer are end-to-end tests, which means that they perform testing of entities that are at the extremities of the network and need to exchange information.

The sequences of tests obtained by the two methods are different. We have analyzed and compared the results, and then found that the two methods are complementary. The first method that uses interactions as a guide for simulation provides better results in testing control parts. The second method that uses a coloring technique permits an easier data differentiation, particularly in branches where decisions are embedded. This complementarity constitutes for us a proof of the interest of our methods: users can choose one of the best fitted to their needs.

The rest of the paper is organized as follows. In Section 2, some basic definitions are given. This section also introduces the criteria used for test selection to guarantee interoperability fault coverage and diagnosis. Section 3 presents the two methods for generating interoperability tests. Section 4 gives an outline of TCP/IP protocol and presents its SDL specification. Section 5 presents the experimental results of the application of the two methods. Finally, Section 6 gives the conclusion and perspectives of this work.

2. RELATED WORK

Conformance testing cannot guarantee that two components or more will interoperate properly within a communicating system. Specific interoperability testing strategies have to be developed, once conformance testing techniques have been applied and conformance has been stated.

Some works have been performed in regard to interoperability, in order to furnish either a formal framework, or experimental results. The authors of [13] were the first to propose a solution for interoperability tests generation. Their work lies on the generation of the reachability graph of the communicating system. [4] has formalized the relation of interoperability by adapting the existing conformance relations. [11] proposed algorithms to generate a test suite. For this purpose, their composition algorithm considers stable states and only tests the control

part of the system. These previous methods can be classified in methods based on reachability analysis. Thus, they are faced to the well-known problem of state space explosion, for real or simply realistic case studies. Other methods have been also proposed, such [3, 16]. All of these methods are based on experimentations, and no coherent framework is proposed. [9] based their work on interoperability by proposing a test generation technique from the joint behaviour of end-users interfaces. This method, in order to avoid the problem of combinatorial explosion, classifies the edges concerning interoperation as black edges and those concerning local activities as white edges. Then, they propose an efficient algorithm to retrieve the significant transitions, in other words the black edges.

Another drawback of all the previous methods is that they only consider the control part of the protocol. In our case, we deal with control and data parts. The generated tests comprehend parameters values in messages and consider variables in graph generation.

3. BASICS

3.1 Extended Finite State Machines

Protocols can be properly modeled by extended finite state machines [12]. In this paper, we have chosen this formalism to describe the studied systems and the test generation algorithms.

Definition. An *extended finite state machine* (EFSM) is a quintuple $M = (I, O, S, \vec{x}, T)$ where I , O , S , $\vec{x}$, and T are finite sets of input symbols, output symbols, states, variables, and transitions, respectively. Each transition t in the set T is a 6-tuple:

$$t = (s_t, q_t, a_t, o_t, P_t, A_t)$$

where s_t , q_t , a_t , and o_t are the current state, next state, input, and output, respectively. $P_t(\vec{x})$ is a predicate on the current variable values, and $A_t(\vec{x})$ specifies an action on the variable values.

Initially, the machine is in an initial state $s^{(0)} \in S$ with initial variable values: $\vec{x}^{(0)}$. Suppose that the machine is in state s_t with the current variable values $\vec{x}$ and an outgoing transition t from s_t . Upon input a_t , if $\vec{x}$ is valid for P_t , i.e., $P_t(\vec{x}) = \text{TRUE}$, then the machine follows the transition t , outputs o_t , changes the current variable values by action $\vec{x} := A_t(\vec{x})$, and moves to state q_t .

3.2 Fault Coverage

The definition of a test case selection criterion that guarantees a fault coverage related to components interoperability is one of the main is-

sues in research work related to interoperability testing. In this paper we propose the following test selection criterion: we select in the protocol specification the transitions that represent an interaction between protocol entities under test. This selection is performed in the first method by considering the set of transitions, which are involved in entities interoperability, as test purposes. These transitions will be used as a guide for simulation. In the second method the selection is performed by coloring the interoperability transitions. In other words, for interoperability tests generation the coverage criterion is reduced to cover some parts of the specification that are related to the communication between interworking entities. In the work presented in this paper, interoperability concerns the test of distinct communicating entities of the same layer.

Another important issue to perform interoperability testing is that since some of the interfaces of the components under test are not directly accessible, interactions through these interfaces can only be observed. To take into account this issue, we use a test architecture (see figure 1) with Points of Control and Observation (PCO) and Points of Observation (POs) below each protocol entity. The POs allow to observe interactions we cannot directly access to. In particular for the TCP/IP protocol it is possible to observe execution traces. Both methods presented in this

Figure 1. Test architecture

paper allows to detect all output faults in covered transitions. The first method generates a test sequence that covers all the transitions that represent entities interactions. The second method generates a set of paths that also covers these transitions.

4. TWO DIFFERENT METHODS FOR AUTOMATED INTEROPERABILITY TESTS GENERATION

In this section, we present for the two methods the algorithms for interoperability test sequences generation.

4.1 Method 1: Interoperability based on interactions as test purposes

As mentioned in section 3 our test purpose covers all the interactions from the entity A to the entity B in the case of a communication protocol. Indeed we will focus on the interactions between two protocol entities. The test purposes cover all the interactions. Note that we obtain a test case for each interaction.

The algorithm presented in this section is based on the generation of partial reachability graphs [5]. It offers an alternative to the second method presented in this paper, which is based on the generation of a global reachability graph. Sometimes it is not possible or impractical to construct such a global reachability graph due to the state-space explosion problem. To overcome this problem we propose a technique which avoids the construction of the complete system reachability graph. Figure 2 presents the algorithm. The technique we propose conducts, at any moment, a local search from the current state in a neighborhood of the reachability graph (step 2 of figure 2). If an untested part is found, we test that part (steps 3 to 7) and continue the process from there (step 8). Otherwise we move randomly to the frontier of the neighborhood searched (steps 10 to 13) and continue the process from there (step 14). A condition for the application of the algorithm is the definition of the tests purposes we want to test. A pre-defined limitation for the search must be defined by the user. For example, it can be a depth limit. Due to this limit we avoid the construction of the complete reachability graph. In fact it's just a bet on non-locality, we want to move to untested parts of the specification, and thus to avoid to be trapped in certain part of the specification. The algorithm terminates when all the interactions present in the set are found. Moreover, the exhaustive search can be a depth-first, breadth-first or bounded depth-first.

Algorithm

inputs. two CEFSMs A and B that are in their initial states, as well as the variables values.

predicate. set of interactions to cover, *interact_seq_set*.

an interaction is a global transition where both entities exchange messages

output. interoperability test case

1. **while** *interact_seq_set* $\neq \emptyset$ **do begin**
2. From the current state conduct an exhaustive search
3. **if** one of the *interact_seq_set* = true
4. **then**
5. Stop the exhaustive search;
6. Keep track of the path from the current node to the interaction reached (extend the current sequence with the path from the root to the interaction we got);
7. Remove the true interaction from *interact_seq_set*;
8. Go to the new current node (reached after the interruption of the exhaustive search on the found interaction).
9. **else**
10. We stopped the exhaustive search on a pre-defined limit;
11. Consider one spanning tree of the partially deployed FSM;
12. Examine all the leaf nodes of the tree, and select one uniformly and randomly;
13. Include the path from the root to the selected leaf in the test sequence;
14. Go to the new current state.
15. **end if**
16. **end while**
17. Return the test sequence.

Figure 2. Description of the algorithm used by method 1

4.2 Method 2: Interoperability based on the coloring of the interactions

We presented in [2] an automatic and optimized technique for test generation of communication protocol control and data portions with the goal of minimizing the number of tests with a guaranteed fault coverage. This coverage corresponds to a set of entities that we want to cover (transitions of the specification, variable values, etc). Different colors are assigned to such entities, to cover interactions transitions. Each interaction transition is involved in a number of transitions of the global

reachability graph. Those colors remain in the reachability graph of the protocol produced by simulation. The algorithm then finds a minimal number of paths from the source node of the graph covering each one a maximum number of uncovered colors. Each path is counted as one sequence, and the set of tests has to cover all the colors.

The algorithm is in fact applied to a Direct Acyclic Graph (DAG), obtained from the colored reachability graph. Construction of the DAG consists in shrinking each Strongly Connected Component (SCC) of the graph into what becomes a node of the DAG. That node contains all the colors of the SCC, and be referred to as an SCC-node of the DAG. Once paths are generated, these are completed to take into account all the colors of each SCC-node involved in a path. It is done by finding in the corresponding SCC a colors tour, which is added to the path.

Algorithm

input. a colored DAG $G = \langle V, E \rangle$ with V and E respectively sets of vertices and edges of G , and the source node v_0 .

output. an approximate maximal color path from v_0 .

1. topologically sort nodes in V : $v_0, v_1, \dots, v_{n-1}$;
2. **for** $i = n - 1, \dots, 0$ /* bottom-up */
3. $\kappa(v_i) = \emptyset$; /* $\kappa(v_i)$: computed color set of chosen path from v_i */
4. $p(v_i) = \text{NULL}$; /* $p(\cdot)$: pointer to next node on chosen path */
5. let outgoing edges from v_i be: $(v_i, w_1), \dots, (v_i, w_r)$;
6. **for** $j = 1, \dots, r$
7. **if** $|c(v_i) \cup c(v_i, w_j) \cup \kappa(w_j)| > |\kappa(v_i)|$
8. $\kappa(v_i) = c(v_i) \cup c(v_i, w_j) \cup \kappa(w_j)$,
- $p(v_i) = w_j$; /* pick up winner */
9. $\text{path} = v_0$; /* path : constructed max-color path */
10. $v = v_0$; /* construct path top-down */
11. **while** $p(v) \neq \text{NULL}$
12. $v = p(v)$; /* get to next node on path */
13. $\text{path} = \text{path} \circ v$;
14. **return** path ;

Figure 3. Description of the algorithm used by method 2

Given a colored DAG G , each node v and edge (u, v) is associated with a set of colors: $c(v), c(u, v) \subseteq C$. We want to find a path (test) from the source node v_0 , covering a maximal number of colors. Since the graph G is a DAG, we topologically sort the nodes [8], and compute a desired path bottom-up in a reverse topological order as follows.

Note that $c(\cdot)$ denotes the color set associated with a node or edge, which is given by the testing criteria and is fixed. We denote $\kappa(v_i)$ is

the color set of the max-color path from node v_i , constructed by the algorithm. When we process a node v_i at Line 3-8 and consider all the outgoing edges (v_i, w_j) , we take the union of the colors of node v_i , edge (v_i, w_j) , and $\kappa(w_j)$. We compare the resulting color sets from all the outgoing edges from v_i and keep one with the largest cardinality. Line 9-13 collect nodes top-down from v_0 along the constructed path. Since we examine each edge once and take unions of and compare color sets of no more than k colors, the time and space complexity of the algorithm is $O(km)$ where m is the number of edges of G .

This **Maximum Color Path** algorithm has to be modified to be used for interoperability testing. We assign a color to all the transitions of the specification. This allows us to keep a trace of the part of the specification we have to pass through to obtain the required interoperability test sequences, and measure the algorithm efficiency in terms of coverage. We give a **priority** to each color that is either an input or an output of one of the interoperating systems. The major effect on the algorithm is that the set of tests has now to cover not all the colors, but **at least** all the colors with priority.

In such a manner, we generate interoperability tests. The algorithm used by this method is presented in Figure 3.

5. THE TCP/IP PROTOCOL

The TCP protocol is a highly reliable host-to-host protocol in packet-switched computer communication networks, and in interconnected systems of such networks [14]. For the description of the protocol we have used the SDL language, standardized by ITU-T [7], which is widely used for the specification of protocols. As it is mentioned in [10], the SDL specification of the TCP includes the description of the protocol architecture. This architecture is organized in three layers: application, communication and network layers. The application layer includes a socket layer with two different variants: Client and Server. They are described as SDL blocks that represent the behavior of the environment and some applications. The SDL description follows the TCP Transition Diagram presented in [14]. The states represented in the SDL specification are the same as that in the TCP Transitions Diagram. **Experiments**

The experiments are performed on the TCP-Reno specification. As the complete reachability graph of the specification is too big, and considering the fact that we are especially interested in the part of the system that describes the interworking between the two TCP A and B, at the TCP Layers, we have reshaped the specification, moving Socket and Client Layers outside the system, so that Socket Layer becomes the new

system environment, in accordance with the test architecture presented in section 3.

5.1 Method 1 experiments

In this section we describe the results obtained after applying method 1 to the TCP-Reno specification. To use this method we need to set some prerequisites. First of all we have to select the test purposes, as mentioned in section 3.2. We want to test the interactions between TCP A and TCP B. With this method we can define test purposes, which in that case are the interactions that we want to test. In the following we explain how to define the test purposes.

Figure 4. Test purposes definition

Description of test purposes based on interactions between TCP A and TCP B

Figure 4 illustrates how to select the test purposes: on the left side you can find the graphical SDL of the two TCP entities and on the right side the test purposes. To express the test purposes the algorithm uses a special syntax close to the SDL syntax. The algorithm has been implemented in such a way that it manages the simulator of ObjectGEODE, and we use some features of this tool to write the test purposes. The

interested readers can refer to [5] for a more detailed description of the implementation of this algorithm.

Concerning the test purposes, to facilitate the understanding of their meaning, we describe them in the following using natural language. The test purpose 1 in figure 4 represents each part of an interaction that we want to test. The test purpose 1 exhibits an end-to-end interaction between the two distinct entities. The test purpose 2 shows how it is possible, using this formalism, to distinguish two decision branches on the variables. These two test purposes express the same interaction to test. This interaction is the following: TCP A is in state *Closed*, when it receives a *tcp_packet* (a in figure 4), it tests the *rst* flag (d), if it is equal to false, TCP A calls the procedure *ResetRequest* (b), which sends *tcp_packet* to TCP B. TCP B receives this packet in state *Listen* (c) and calls the procedure *checkchecksum*, which allows to check the correctness of the *tcp* header.

The application of the method to the specification of TCP-Reno produced the following results. We covered **36** interactions concerning the two different TCP entities. Indeed we obtain test sequences that represent a path from one of entity to the other, it is what we call an end-to-end interoperability test sequence.

To illustrate this application we present in the figure 5 one of the test sequences that we have obtained. This one is a simplified version of the original sequence, that illustrates the connection establishing. The TCP connection is done with a three-way handshake. In this example TCP A is the initiator of the connection, at the beginning it is in *passive* (listening) state, it receives an *active open* request from the environment (the socket layer). At the same time the TCP B receives a *passive open* with an unspecified foreign socket (represented by the *upo* signal). This specification and most APIs (Application Programming Interface) do not provide a way to treat this. Thus TCP B must leave the foreign socket and wait for a connection to arrive, it moves then to the state *Listen*. TCP A sends a TCP segment with the flag *SYN*, it means that the synchronization flag is set. TCP A remembers it has sent this initial *SYN* packet by entering the *Synsent* state. In response TCP B after receiving this TCP segment enters the state *Synreceived* and responds with its own TCP segment and acknowledging receipt of the initial *SYN* packet by putting the *ACK* flag. In figure 5, the *tcp_packet* fields flag *SYN* and *ACK* are equal to true. TCP A acknowledges receipt of the *SYN* from TCP B and enters the state *Established*. And when TCP B receives this it enters also the *Established* state. This test sequence has been obtained by the concatenation of the two test purposes, the first one to obtain the interaction between the peer TCP in their *Closed*

states, and a second one for the interaction between the *Synsent* state (TCP A) and *Synreceived* state (TCP B).

Figure 5. A generated simplified test sequence for the connexion establishing

Fault diagnosis

The PCOs that appear in figure 1, permit to detect the output faults. Indeed the testers according to the response of the system under test can establish a verdict. Moreover, if a fault occurs during the interaction of the TCP entities, thanks to the POs, we will be able to detect this fault and to have a finer verdict.

5.2 Method 2 experiments

All transitions are colored and to obtain interoperability test sequences priority is given to the transitions representing the exchanges between TCP A and TCP B. A reachability graph with good cover rates of transitions and states can be generated by an exhaustive simulation, see Table 1.

We obtain a minimal reachability graph, from which we apply the maximal color path algorithm repeatedly until all the transitions with priority are covered. The results are shown in Table 2.

In total, to cover the **36** colors with priority deployed in the graph, **11** paths are generated.

Table 1. Exhaustive Simulation of the TCP specification

# of States	14008
# of Transitions	46695
Maximum depth reached	21
Maximum breadth reached	3146

Table 2. Final results for the minimal reachability graph

# of Nodes	1926
# of Edges	14767
# of Maximum Color Paths	11

As illustrated by table 3 the first path covers **16** colors, among which are **5** colors with priority. The next three paths cover **5** colors with priority, the next one **4**, and so on. Given the large number of nodes

Table 3. Created paths

Path	# of covered colors	# of colors with priority	Length
0	16	5	18
1	6	5	17
2	6	5	17
3	6	5	17
4	5	4	17
5	4	4	16
6	4	2	17
7	3	2	18
8	4	2	17
9	1	1	17
10	1	1	17

and edges in the minimal reachability graph, our algorithm efficiently constructs a rather small number of tests, **11**, that cover the required parts of the specification.

The figure 6 presents the MSC obtained from the path number 0. This path correspond to a connection establishment phase presented in figure 6, initiated by TCP A. This can be expressed as follows. An *active open* is sent by the Socket_A layer to TCP A, that synchronizes its sequence number with TCP B sequence number, and wait for a connection acknowledgement from TCP B. Once it is obtained, TCP A sends its acknowledgement to TCP B and asks the Socket_A Layer for a socket to be opened (*open socket* signal) on its side. TCP B then receives TCP A acknowledgement and sends an *open socket* signal to the Socket B Layer.

At this point of the path execution, connection is fully established between TCP A and TCP B (both TCP are in the *Established* state). The end of the path corresponds to the beginning of a connection termination phase.

Figure 6. Connection establishment phase

5.3 Comparison of the obtained results

In this section, we compare the results of the two methods. Both methods have similar goals, i.e. obtain tests for the interoperability parts of the specification and both have been applied successfully. Nevertheless there are some differences between the two methods. Indeed, some interactions found in method 1 cannot be detected using method 2, and reciprocally some interactions found by method 2 cannot be detected using method 1.

Method 2 can easily distinguish interactions according to variable values, because we can assign different colors to different branches of a transition, as a decision based on data values might modify the parameters of a sent signal. Most of the time is spent on the generation of the reachability graph. Once the graph is obtained, the paths are automatically and quickly generated by an algorithm developed at INT and integrated to our tool TESTGEN. The cost is fairly high in term of resource consuming, as we first compute the whole reachability graph to obtain the DAG.

On the other hand, applying method 1, we do not have to obtain a global reachability graph. Method 1 requires to spend more time to express test purposes as they are until now handcrafted, whereas they can be generated automatically in method 2. Both methods demand a relative knowledge of the protocol to test. This is more significant in

method 1, since writing test purposes can be more complex than the coloring of transitions. Once the interactions are located and formally expressed, we can focus the efforts on the complexity of test generation. We mean the launching of the tool developed at the INT, for each interaction. Finally, we can say that method 1 is time consuming in term of the characterization of interactions to test, but not in term of execution. In term of resource consuming, we can say that this cost is very low in method 1, as we only compute at each step a partially reachability graph and we do not keep in memory track of this graph.

Comparison can also be made between length of the tests. A sequence of 1,500 transitions (joining all the sequences) is obtained using method 1, whereas method 2 generates 11 paths, each one having about 17 transitions, for a total of 188 transitions. This awesome difference (about 8 times less transitions) is caused because paths construction is intended to maximize the number of interactions in it. On the contrary, a sequence in method 1 only contain one interaction.

The coverage of the 36 transitions covered by both methods, is complete with respect to the interoperability requirements presented in the specification. In fact, these 36 transitions represent all the interactions between TCP A and TCP B on a total number of 282 transitions for the whole specification.

6. CONCLUSION

To summarize, the contribution of our work consists in the development of two methods for interoperability tests generation and their application to a real protocol, TCP/IP. This application illustrates their suitability. Both methods have been applied in a test architecture that enables to control and to observe the entities interactions. This is an important factor for the faults diagnosis.

The two methods use the same basic idea for generating interoperability test sequences: to select in the protocol specification the transitions that represent an interaction between the protocol entities under test. The experiments showed that both methods are complementary, one is well adapted to test more different control parts and the other permits an easier data differentiation. We have also compared the length of the test sequence and the concatenation of the set of paths.

Both methods have been integrated in a tool set for automated test generation, TESTGEN [6, 5], that we have developed in our research group at INT.

The outcomes of this work will be used in a future work related to the development of a platform for the interoperability testing of services and protocols, with an application to the wireless access to Internet.

References

- [1] *RFC 793: Transmission Control Protocol*, September 1981.
- [2] C. Besse, A. Cavalli, and D. Lee. An automatic and optimized test generation technique. applying to TCP/IP protocol. In *ASE'99 (14th IEEE International Conference on Automated Software Engineering)*, october 12-15 1999.
- [3] Grard Bonnes. IBM OSI Interoperability Services. In *Protocol Test Systems, III*. North-Holland, 1991.
- [4] R. Castanet and O. Kone. Deriving coordinated testers for interoperability. *Protocol Test Systems*, VI (C-19):331–345, 1994. Elsevier Science Publ B. V.(North-Holland).
- [5] A. Cavalli, D. Lee, Ch. Rinderknecht, and F. Zadi. Hit-or-Jump: An Algorithm for Embedded Testing with Applications to IN Services. In *Formal Methods for Protocol Engineering And Distributed Systems*, pages 41–56. Kluwer Academic, October 1999.
- [6] A. R. Cavalli, B. Chin, and K. Chon. Testing methods for SDL Systems. In *Computer Networks and ISDN Systems*, volume 28, pages 1669–1683, 1996.
- [7] CCITT, ITU-T General Secretariat, Geneve, Switzerland. *Recommendation Z.100: Specification and Description Language (SDL)*, 1992.
- [8] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.
- [9] N. Griffeth, R. Hao, D. Lee, and R. K. Sinha. Integrated system interoperability testing with applications to VOIP. In *FORTE/PSTV'00*, October 2000.
- [10] J. Hintelman and R. Westerfeld. Performance analysis of tcp's flow control mechanisms using queueing SDL. In *In Proc. of SDL Forum'97*. Elsevier, September 1997.
- [11] S. Kang and M. Kim. Interoperability test suite derivation for symmetric communication protocols. In *FORTE/PSTV'97*, 1997.
- [12] D. Lee and M. Yannakakis. Principles and Methods of Testing Finite State Machines - A Survey. *Proc. of the IEEE*, 84(8):1090–1123, August 1996.
- [13] O. Rafiq and R. Castanet. From conformance testing to interoperability testing. *The 3rd Int. Workshop on Protocol Test Systems*, 1990.
- [14] W. R. Stevens. *TCP/IP Illustrated*. The protocols. Addison-Wesley, 1994.
- [15] Verilog. *ObjectGEODE Simulator, Reference manual*, 1997.
- [16] G.S. Vermeer and H. Blik. Interoperability testing : Basis for the acceptance of communication systems: Theory and practice. In *Protocol Test Systems, VI (C-19)*. North-Holland, 1994.