22

AUTOMATIC TEST CASE GENERATION FROM THE INDUSTRIAL POINT OF VIEW: CONFORMANCE TESTING IN ISKRATEL

Marjeta Frey-Pučko
IskraTEL, Telecommunications Systems, Ltd.
Ljubljanska cesta 24a, SI-4000 Kranj, Slovenia
pucko@iskratel.si

Monika Kapus-Kolar
Jožef Stefan Institute
Jamova 39, P.O.B. 3000, SI-1001 Ljubljana, Slovenia
monika.kapus-kolar@ijs.si

Roman Novak
Jožef Stefan Institute
Jamova 39, P.O.B. 3000, SI-1001 Ljubljana, Slovenia
roman.novak@ijs.si

Abstract In this paper we present the tool iATS (integrated Automatic Test Sequence generator) for generation of conformance tests in development of digital switching systems SI2000. The tool generates test cases in TTCN form from an SDL specification of the service or protocol under test. Test cases are derived by implemented FSM-based methods. We also describe practical experience of using the tool, illustrated by some quantitative results.

Keywords: Practice of testing, test tools, conformance testing, automatic test case generation.

1. INTRODUCTION

IskraTEL is a company specialised for production of telecommunications systems. Among other products, the company produces digital switching

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*

systems SI2000. One of the key issues for the success of the product is conformance of the implemented protocols and services with standards and requirements of specific market needs. Since the time consumption in manual test case generation and provision of adequate test coverage proved to be the most critical factors in conformance testing of the system SI2000, a need was detected for the automation of conformance test generation procedures.

In order to improve the software development process of SI2000 systems, a general decision to use formal languages was made already at the beginning of eighties. The first one actually used in the company was an SDL-like language called Specification Language One (SL1) [5]. We migrated to the SDL language after it reached its mature stage and its use was supported by commercially available tools [16]. Now SDL is successfully used in different phases and activities of the software development process. As a result of positive experience collected during the long-term use of formal languages in the company, the idea appeared in the middle of nineties to automate the generation of test cases for conformance testing by means of formal languages and methods. The goal was a tool for automatic generation of test cases in the standard TTCN (Tree and Tabular Combined Notation) form [9] from an SDL specification of the service or protocol to be tested.

Since no appropriate commercial tools were available on the market, the decision was made to develop our own tool adapted to the specific needs of conformance test generation for the system SI2000. The tool iATS (integrated Automatic Test Sequence generator) presented in this paper was then developed in the scope of the research and development project "Automation of test scenario generation for the system SI2000". The partners of the collaborative project were the Jožef Stefan Institute (academic partner) and IskraTEL (industrial partner).

The paper is organised as follows: section 2. presents the general testing framework in IskraTEL and exposes the main objectives in the development of the tool. Section 3. gives a detailed description of the tool. Section 4. contains basic guidelines for writing SDL specifications used as input of the tool. Section 5. summarises the results of practical use of the tool in test case generation for ISDN services and signalling protocols. Finally, section 6. discusses the advantages and shortcomings of the tool, and analyses perspectives for its improvement in the future.

2. FRAMEWORK AND OBJECTIVES

Conformance testing is an important activity in the verification and validation phase in the development process of SI2000 systems. The aim of conformance testing is to prove the conformity of implemented protocols and services with standards or specific customer requirements. The testing architecture used

corresponds to the ITU-T X.290 recommendation where the basic test configuration is represented by the upper and lower tester interacting with the IUT (Implementation Under Test). The interaction is performed by means of the execution of test suites and cases respectively. We use the terms "test case" and "test suite" as defined by ISO OSI terminology [9]. The definition of a similar term used in the following – test sequences – is adopted from [15]: test suites selected by test case generation methods are a (test) sequence of input-event and expected output-event pairs.

Test cases for conformance testing are coming from several sources. There is a set of conformance test cases collected in the past decade, which have been generated manually. Other test cases are either adopted from ETSI (after some minor corrections) or generated automatically by iATS. While the manually generated test suites are described in their abstract form (Abstract Test Suite – ATS [9]) mainly in MSC (Message Sequence Charts), the ETSI and the automatically generated test suites are described in TTCN. For actual execution on the system SI2000 they have to be translated into an executable form (Executable Test Suite – ETS). Afterwards they are executed on different testing platforms using the testing equipment of several providers, not allowing the execution of the same ETS on all platforms.

As the main objective of iATS development, the automatically generated ATSs should have been described in a unique form for all testing platforms possibly supported later by commercial tools for automatic ETS generation. Therefore the standardised TTCN form was selected. The second important objective was to integrate the form of the input formal specification into the existing formal specification environment adequately supported by existing commercial tools. For this reason, SDL was selected. Finally, the tool should have been designed also to derive benefit from previous testing results. In order to reuse the testing results for pre-tested components, the context test generation feature was defined for the cases where only the context should be tested in which the components are currently operating.

3. THE IATS TOOL

At the very beginning of the tool development we decided to use methods for actual test case generation based on the FSM (Finite State Machine) model. The reason was the availability of many FSM-based methods with a well-defined mathematical background, which cover a wide spectrum of FSM properties. Another reason was that the model is very close to the EFSM (Extended Finite State Machine) model of SDL. Afterwards the remaining functionality of the tool was designed to support the selected FSM-based concept of test case generation. The tool differs from the test generation tools for conformance testing like TGV, TVEDA, TTCgeN or Autolink [4, 6, 17] basically in the

approach because it is based on implicit test derivation methods without defining test purposes.

The development of the tool started in 1995. As a result of the pilot project, the first prototype was available in 1996. The functionality of the prototype included a simple compiler generating an FSM from a particular SDL specification of an ISDN service, and a test case generator implementing several FSM-based methods. It provided output in the form of test sequences described by transitions of the FSM, and in a TTCN-like notation. From this list of features it is evident that it had very limited applicability. Its purpose was only to prove the feasibility of the approach in the particular service example. We evaluated the prototype in real testing of the selected ISDN service and identified possibilities for improvement and generalisation of the tool to be applicable to a larger set of services and protocols. After that we started the second phase of the tool development which ended in 1998. The result was the tool with the functionality described herein.

Using iATS, test cases are generated in the next three steps:

- *Abstraction to FSM.* Each EFSM (i.e. process in the SDL specification) is first abstracted to a corresponding FSM. Afterwards, the FSMs for all processes involved are composed into a combined FSM modelling the complete SDL-specified behaviour.

- *Test sequence generation.* From the combined FSM, test sequences are generated using well-known UIO (Unique Input-Output) methods, or test generation methods for non-deterministic protocol machines. The selection of the method depends on the properties of the FSM.

- *Translation to TTCN.* The generated test sequences are automatically translated into TTCN test cases. While the behaviour part of the TTCN description is generated completely automatically, constraints in the declaration part have to be inserted manually.

The functionality of each step is covered by a corresponding tool component as follows:

1 *SDL-FSM compiler-simulator.* This component is used for abstracting each process involved in the given SDL specification to a FSM. Values of parameters are inserted by the user of the tool and afterwards considered as fixed.

2 *Tool for composition and construction of approximate machines.* From the FSMs constructed by abstraction from SDL processes, a composed FSM is generated. Construction of an approximate machine is possible if some constructs in the SDL specification have been declared by the

user as hidden, or if the SDL specification describes only the behaviour of the context of some pre-tested and correctly-working components.

3 *Test sequence generator and compiler into TTCN.* This tool generates test sequences from a given FSM on the basis of the selected test derivation method. The user may choose between the method suggested by the tool as default, and other implemented methods.

4 *Graphical user interface.* It integrates the first three components into a single tool in X-Windows environment, supports interaction with the user and provides a system of friendly help.

Although all the components have been developed specially for iATS, the first three components can also be used as standalone tools. The iATS tool has been developed for HP-UNIX and X-Windows environment. It is owned and used as an in-house tool by IskraTEL. We describe the functionality and theoretical foundations of the first three components from the list above more precisely in the following subsections. iATS is described in detail in [7].

As the input of the tool, any SDL specification in the textual form may be used, being created in correspondence with the methodology briefly described in section 4.. The tool generates two main outputs: a set of generated test sequences described by transitions of a FSM, and a TTCN description of test cases in the standardised form. As an auxiliary output, another description in style of TTCN is generated.

3.1 CONSTRUCTION OF FSM

We selected FSMs based on the Mealy machines. Since the set of all languages defined by FSMs is a subset of the set of all languages defined by SDL specifications based on the EFSM model, not all SDL specifications can be translated into FSMs. The language of an FSM [8] or an SDL specification is a set of all valid input/output sequences. An FSM and an SDL specification are considered behaviourally equivalent if their input/output behaviour is the same, i.e. the output sequences match for each possible sequence of input symbols.

The compiler consists of its front-end and back-end in the meaning defined in [1]. The front-end performs lexical, syntax and semantic analysis of the SDL specification, while the back-end actually translates the SDL specification into the FSM. The syntax analysis is adapted to the SDL-88 syntax definition of [19, 20]. The compiler's back end is actually a specially designed simulator of SDL processes.

The basic block of our translation is a single SDL process. Each process is translated separately and, at the end of the compilation, merged with the rest of the translated processes through FSM composition procedures. The translation

of the input and output signals into the input and output symbols is almost straightforward. Signal parameters must be handled separately. In order to avoid state explosion problem, the set of allowed parameter values should be given in advance. No additional help, except the information about the type of the parameter, is given to the user in giving the right values by the system.

The mapping from SDL states to FSM states is not so clear since the state of the SDL process in execution can not be characterised only by the SDL state name. The state of the SDL process is determined by the state of all variables, signal parameters, timers, and by the content of the procedure call stack. In order to reduce the FSM only to the set of states, which actually appear during the SDL process execution, simulation is required. To a certain degree, we can avoid the state explosion problem and still get valuable test sequences by limiting the depth of the SDL process simulation. Still, all possible paths of execution need to be systematically simulated. The redundant states and transitions are removed from the final FSM. The output of the compiler is a set of FSMs where each FSM corresponds to one SDL process. The FSMs are composed in one FSM representing the input/output behaviour of the source SDL specification by sequential and parallel composition techniques.

A set of limitations is imposed on the SDL specification to make the translation into an FSM possible. Some limitations are introduced simply to reduce the complexity of the compilation process. These limitations, by our experience, do not strongly affect the expressive power of SDL when telecommunications services are in question. The input of the compiler may be an SDL specification in the textual form with the contents corresponding to the limitations described in the following.

The tool can handle the constructs "start", "state", "nextstate", "stop", "decision", "label" and "join" without any constraints. All transitions should have non-empty inputs except the first transition after the construct "start". The construct "output" is forbidden in the first transition following the construct "start", otherwise there are no restrictions. The resulting FSM may have empty output symbols on transitions. Multiple signals may be outputted on the same transition; they result in one new output symbol in the FSM output alphabet. Signal decomposition is forbidden. The construct "create" should be used without parameters. Timers should have no parameters. Variables can be of any predefined sort. User defined sorts are currently not supported. Supported are all predefined operators on variables of supported sorts. Procedures may have parameters called by value or by reference, and local variables. Nesting of procedures is forbidden. At the present stage, among frequently used constructs the constructs "save", "continuous signal", "import", "export", "viewed", "revealed", and "alternative" are forbidden. They are not supported for different reasons. In the case of "save", the translation into FSMs using an existing method [12] would be possible only for SDL processes with known contents of

the input queue. To provide correct results, the same order of signals should be guaranteed during test case execution on a real object. This is in the example of the system SI2000 testing not possible. "continuous signal" would result in a transition with an empty (NULL) input symbol. Consequently, the generated FSM would be an unusable input for the test sequence generator. "import", "export", "viewed" and "revealed" would require knowledge of current values for variables used in different processes. This is not possible since the translation is based on simulation of one SDL process at a time. "alternative" was omitted from the list of supported constructs because it is practically never used in specifications relevant to conformance testing of SI2000.

In figure 1 an example of SDL to FSM conversion is given. User defines the parameter of the signal A to be 0. Only actual values of the variable x contribute to new states.

3.2 TEST SEQUENCE GENERATION

The test sequence generator takes as input a file containing the FSM for which the sequences are being generated, and a file with control information. The FSM file is a text file specifying the initial state of the FSM and listing for each transition its source, its destination, its input symbol and the resulting actions. An input symbol is either a sequence of ordinary input signals or a timer signal. The resulting actions specify the output symbol of the transition (a sequence of output signals) and an arbitrary number of settings and/or resettings of timers. If the FSM is non-deterministic, the use of timers is forbidden for the present, because they are difficult to handle during test generation.

Optionally, the FSM file can be edited to specify for each transition its cost, so that the generator can enhance the implemented test-sequence generation methods with additional cost minimisation. The cost of an individual transition can be an arbitrary non-negative number properly reflecting the difficulties associated with execution of the transition, for example the necessary time or resources. By editing the FSM file, one can also indicate which of the transitions have already been successfully tested. When building a test sequence as a composition of tests for individual transitions, the pre-tested transitions shall be ignored and the resulting composite sequence shorter.

The control information file can be generated with the help of a graphical interface that suggests which methods (and their parameters) are worth trying in the next run of the test sequence generator. The suggestions are based on the diagnostic information resulting from the previous runs. A possible suggestion might also be to make the FSM complete or to introduce the reliable reset capability. That can be done automatically, by introducing in each state the missing input/NULL loops in the first case, or a reset transition with user-defined input and output symbols in the second case.

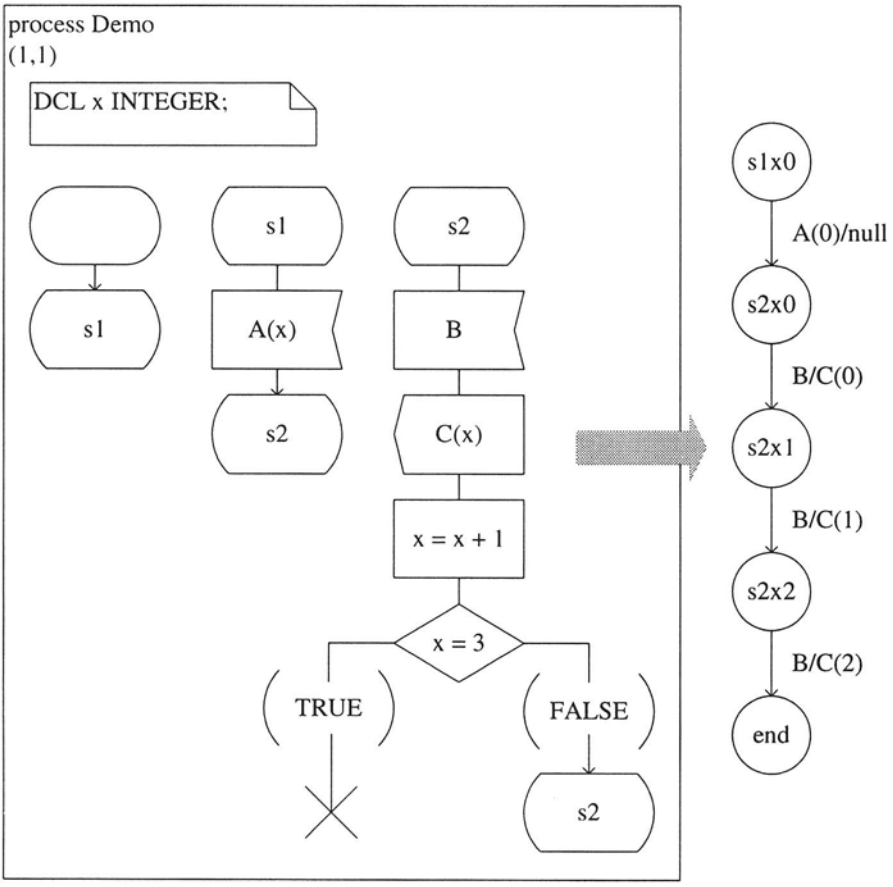

Figure 1 An example of SDL to FSM conversion

Based on the observations in [3, 15], we have elected to implement four existing test-sequence generation methods, for different purposes. For FSMs that are strongly connected, deterministic, with proper UIOs and with the correct number of states in the implementation, the methods proposed in [2, 11] have been implemented for transition testing. Method [2] is applicable to most practically interesting FSMs, while method [11] is seldom applicable, but generates extremely short sequences. If the reliable reset capability exists, both methods can be optionally preceded by testing of state and UIO implementation. In the absence of the reliable reset capability, the available method is [21], optionally without its state and UIO testing parts, for they might require catastrophically long test sequences. For a FSM that is non-deterministic (or even non-observable), only weakly connected, without proper UIOs or with an

incorrect number of states in the implementation, the method proposed in [13] might help, if the reliable reset capability exists. To cope with states with very long characterisation sequences, the test sequences can be interactively optimised by gradually increasing the allowed length of the state-characterisation sequences as long as the so-called fuzziness degree [13] decreases.

The generated test sequences are basically represented by the corresponding sequences of input symbols of the FSM. For a deterministic FSM, that output file of the tool also specifies for each transition its source state, its output symbol and its cost. In addition, the cumulative cost is given for each test sequence and for their entire set. Another human-readable representation of the test cases is a tree in the TTCN [9] style, nicely divided into subtrees to fit into the designated page width. The representation is particularly convenient for a non-deterministic FSM, as each node of the tree is labelled with a list of the potentially corresponding states, and each output symbol with the cost of the most expensive among the transitions to which it potentially belongs.

For a deterministic FSM, the tool also represents the test cases in the standard machine-readable TTCN format. The file also specifies the necessary timer actions. Besides the timers specified in the given FSM, there is a special timer for each pair of a state and an input symbol. Expiration of the timer indicates that IUT reacted on the input symbol with a NULL output symbol. The duration of such a timer is a parameter of the test specification, so that it can be set to suit individual testing needs.

3.3 TESTING OF COMPONENTS

The black-box view of the IUT considered in the previous subsection is not appropriate when IUT is embedded within a complex system under test. In that case, grey-box testing methods are necessary where internal structure of the complete system under test is known in the sense of components structure. Test cases have to be generated for one particular component operating in the context of the remaining part of the system, which is assumed to be correctly implemented.

Basically we selected the model of a system with an embedded component given in [14]. We applied it as basis of test case generation for the context of pre-tested components with some minor simplifications and differences in interpretation. Firstly, the roles of the embedded component and context are changed: the embedded component actually to be tested is in our case the context represented by a context FSM (so-called component machine in [14]), and the rest of the system is a set of components represented by a single product FSM (in [14] called the context machine). Secondly, the set of input symbols (X) for the complete system under test equals to the set of input symbols of its components, i.e. no internal inputs are assumed. The same is assumed for the

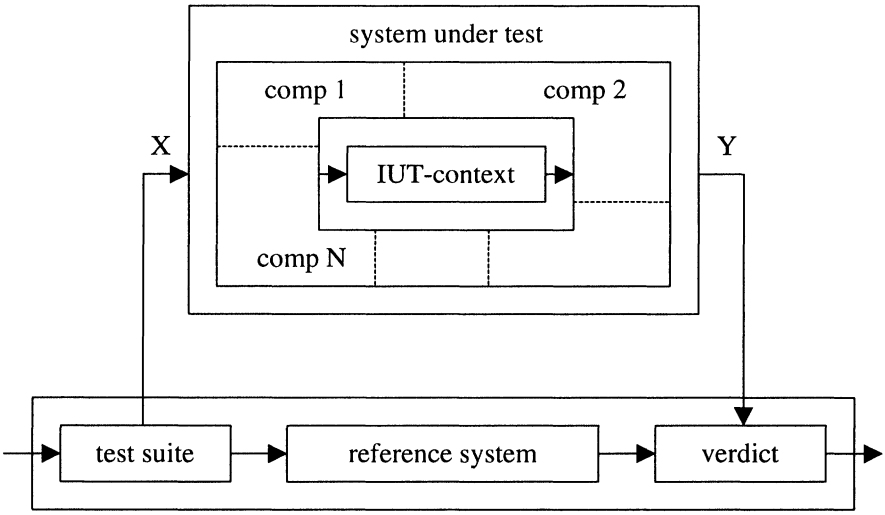

Figure 2 Architecture for testing of context

set of output symbols (Y). The applied architectural model is shown in figure 2. The described assumptions significantly reduce general applicability of the model, but they adapt it for the use as a theoretical basis for test case generation method implemented in iATS.

We generate test cases for the context of pre-tested components as follows. As input, an SDL specification has to be made for each of the pre-tested components and the context to be tested. SDL specifications are translated into FSMs. The transitions of pre-tested components are marked as already successfully tested (in the meaning described in the previous subsection), and the FSMs of components (comp 1,...,comp N) are composed into a product FSM. Afterwards an approximate machine for the context is constructed based on composition of the product FSM and the context FSM (adaptation of method of [14] in accordance with our assumptions about the model). For the constructed approximate machine, test sequences are generated using the generator described in the previous section.

Besides limitations resulting from the assumptions described above, the method has limited applicability also because of the combinatorial explosion problem caused by use of FSM composition. The problem is not to construct an approximate machine with considerable size, but to apply it as input of the test sequence generator. Therefore it is strongly recommended to specify each component by a single block containing one process and to restrict the number of components. Another problem is that it is often difficult to pre-

test a component of a real system separately to prove its correct operation. However, the method can, to our experience, be successfully used for testing ISDN services in context, for example in testing interaction between services.

4. WRITING SDL SPECIFICATIONS

The main purpose of an SDL specification used as an input of a test case generation tool is to describe the complete behaviour to be tested. To write an SDL specification we need a precisely defined input and output, methodological rules and an appropriate tool support. The input is adequate information on the functionality to be specified. For the case of ISDN services, information can be obtained from informal service specifications available in standards or internal documents of the system SI2000. The output is an SDL specification of the functionality meeting some requirements about its form and contents. Methodological rules define how from the input an output should be generated. Among software tools supporting the specification activity, an editor (preferably graphical) and a syntax checker are the most essential. A semantic checker proves to be also a very valuable tool to check the specification against logical errors, such as deadlocks for example. We used Verilog GEODE Editor and Simulator.

An SDL specification used as an input of the iATS tool can give prospective results in test scenario generation if the abstraction level and precision in the specification of the functionality to be tested is appropriate for the derivation of conformance test cases, and if the structure of the specification and the properties of the contained EFSM assure optimal results regarding to the implemented test derivation methods.

The requirements belonging to different specification problem domains imply different methodological rules. A very important question is how to reach the appropriate abstraction level and which details of behaviour to present in the specification. For the generation of conformance test cases of ISDN services, for example, we need to specify the behaviour of the system under test communicating with a user where all the observable communication should be described. The system actions are therefore observed only at the user-system interface but actually they are the result of sequences of actions executed in different parts of the system: switching system, fixed network, related mobile network etc.

Our methodology is based on a set of rules. Due to different specification problem domains the rules are divided into five categories:

1 *Abstraction rules* define at which abstraction level the SDL specification describes the given functionality.

2 *Naming rules* specify how names of all elements of the SDL specification are defined. The rules impose restrictions on the structure and contents of

names of the complete specification, blocks, processes, signals, channels and signal routes.

3 *Structure rules* define how the specification has to be structured into blocks and processes and how communication paths between them have to be specified. They also recommend how the specified functionality should be structured into components in means of test scenario generation for testing of single components.

4 *Adaptation rules* define how the form and contents of the specification have to be adapted to the properties of the test scenario generation tool. While use of a subset of those rules is mandatory to provide an acceptable input for the iATS tool, other rules may be used optionally to provide input for iATS giving more optimal results in test scenario generation. The mandatory rules impose restrictions particularly on use of some SDL constructs and their combinations. The optional rules deal more with the number of SDL processes and properties of the EFSM inside the SDL specification that are expected to give the best result in the sense of test suite length and test coverage.

5 *Mixed rules* concern more than one specification domain. The intersected domains are adaptation, structure and naming.

Defining the rules we considered two existing methodologies: ITU basic methodological guidelines for use of SDL [10] and the IskraTEL SDL methodology [16]. The reason for consideration of the latter was the need to tailor iATS to the previously existing SDL specifications, which have been developed in accordance with that methodology. Although we adopted some of its naming and adaptation rules, the most of the rules of our methodology were newly defined.

5. FROM SDL TO ETS

In the last year the tool was used for generation of test cases for seven ISDN services and for some parts of the SS7 and DSS1 signalling protocols. Our experience is described as follows:

Writing SDL specifications. SDL specifications were already available for all services and the selected protocol parts from the design phase. They were not usable for test case generation for two reasons: firstly the specifications contained all the details of internal system behaviour and secondly they served as formal basis for automatic code generation in the implementation phase. Therefore we created new SDL specifications describing the observable service behaviour at a much higher abstraction level (the user-system interface). The length and complexity of the specifications were dependent on the complexity of the service functionality and the number of users involved. The

Table 1 Some quantitative results for the services CLIP and 3PTY

	service CLIP	*service 3PTY*
SDL specification length	349 lines	995 lines
number of FSM states	8	36
effort of writing SDL spec.	0.25 man/month	0.4 man/month
time for test suite generation by iATS – TTCN, no constraints	8 seconds	0.3 hour
total time of test suite generation – TTCN with constraints	1 hour	2.3 hours
time for automatic ETS generation	30 seconds	45 seconds

specified behaviour was described by a single SDL block containing a single SDL process. Some data on the length of the specification and effort required for the services CLIP and Three-Party Service (3PTY) can be found in table 1.

TTCN test case generation. For the created SDL specifications we generated test cases using iATS. As the common usable method for test sequence generation from automatically generated FSMs proved the method of [13]. The actual fault coverage was as defined for the method of [13]. The method of [11] was applicable for none of the automatically generated FSMs from the set of the created SDL specifications. Since the functionality was described in the specifications by a single SDL process, the FSM was small enough to generate practically usable test cases. Time for generation depended completely on the properties of the FSM and, consequently, the selection test sequence generation method. The TTCN constraints were manually inserted into the automatically generated test cases. In table 1 we present some quantitative results on time required for generation for the services CLIP and 3PTY.

Translation into the executable code. The way of translation of test cases in TTCN into the executable code depended on the testing platform and equipment. While the translation for the test cases later executed on the Tekelec Chameleon 32 equipment was performed automatically using the Expert TTCN-C compiler and an in-house tool, the form for the Alcatel 8610 equipment required quite a lot of manual work. Since there is no tool for automatic translation into the Alcatel 8610 form available for the present, iATS has not yet become the main tool for designing tests within IskraTEL.

Time spent and time saved. Generally, we saved by automatic generation of test cases in TTCN (ATS) between 20% and 50% of the total time needed for complete manual test case generation. For the cases where also the ETS was generated automatically, we saved additional 20% of the total time.

6. CONCLUDING REMARKS

We have presented the tool iATS for automatic generation of test cases for conformance testing. The tool has been developed to generate conformance test cases for services and protocols in the development of digital switching systems SI2000. Its main advantage is that it generates a TTCN test suite from the given SDL specification automatically except the selection of (fixed) signal parameter values and the definition of TTCN constraints. These are currently also the main disadvantages of the tool. Practical use of the tool has shown that the first disadvantage decreases the general applicability of the tool, while the second one results only in time consumption for completion of a generated test suite. Developing the tool, we evaluated several existing methods for test sequence derivation and developed a methodology for writing SDL specifications used as the input of the tool.

Our main goal for the future is to improve the iATS tool by removing the limitation of fixed parameter values in test sequences derivation. In this way iATS is currently being enhanced with an additional test case generation method based on the EFSM model [18]. Here some theoretical work is also being done, trying to make the method work for more than one SDL process. We also plan to build into the tool the possibility of generating the TTCN constraints more automatically using ADTs.

To our experience, formal description techniques, especially SDL and TTCN, are successfully used and becoming well accepted by the industry. The most important contributing factors for their success in the industrial use are the appropriate tool support in all steps from writing specifications to generation of the executable code, and the systematic training of system developers possibly already at the undergraduate level.

Acknowledgments

We wish to thank the Ministry of Science and Technology of the Republic of Slovenia, and IskraTEL for financing this project. Next we wish to thank A. Ciglič, D. Kodrič, N. Maloku and M. Stojsavljevič from IskraTEL for constructive comments and provision of a platform for test scenario execution on the real system. Finally we would like to express our gratitude to the project participants V. Avbelj, B. Močnik, B. Slivnik and R. Verlič, who contributed to the iATS tool in an essential way.

References

[1] Aho, A.V., Sethi, R., and J.D. Ullman. (1986). *Compilers: Principles, Techniques, and Tools*, Addison-Wesley Series in Computer Science, Addison-Wesley Publishing Company, Bell Telephone Laboratories.

[2] Aho, A.V., Dabhura, A.T., Lee, D., and M.U. Uyar. (1991). An optimiza-

tion technique for protocol conformance test generation based on UIO sequences and rural Chinese postman tours, *IEEE Transactions on Communications*, vol. 39, no. 11, pp. 1604-1615.

[3] Anido, R., and A.R. Cavalli. (1995). Guaranteeing full fault coverage for UIO-based testing methods, in *Proceedings of the 8th Int. Workshop on Protocol Test Systems*, Evry, France, pp. 221-236.

[4] Doldi, L., et al. (1996). Assessment of automatic generation methods of conformance test suites in an industrial context, in *Testing of Communicating Systems*, Chapman & Hall, pp. 347-361.

[5] Exel, M., Popovič, B., and F. Prijatelj. (1982). SL1 language - A specification and design tool for switching systems software development, *IEEE Transactions on Communications* (Special Issue on Comm. Software).

[6] Fernandez, J.-C., Jard, C., Jeron, T., and C. Viho. (1997). An experiment in automatic generation of test suites for protocols with verification technology, *Science of Computer Programming*, vol. 29, no. 1-2, pp. 123-145.

[7] Frey-Pučko, M., Kapus-Kolar, M., Novak, R., Verlič, R., Močnik, B., Slivnik, B., and V. Avbelj. (1998). *Automatic Test Scenario Generation for the System SI2000*, Final report, IskraTEL (in Slovene).

[8] Hopcroft, J.E., and J.D. Ullman. (1979). *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley Publishing Company.

[9] ISO. (1997). *ISO/IEC 9646-3, Tree and Tabular Combined Notation (TTCN)*, Second Edition.

[10] ITU-T. (1993). *Z.100*, Appendix I, SDL Methodology Guidelines.

[11] Jiren, L., and D. Jun. (1994). A new approach to protocol conformance test generation based upon UIO sequences, *Chinese Journal of Advanced Software Research*, vol. 1, no. 4, pp. 373-381.

[12] Luo, G., Das, A., and G. von Bochmann. (1994). Software testing based on SDL specifications with Save, *IEEE Transactions on Software Engineering*, vol. 20, no. 1, pp. 72-87.

[13] Luo, A., Petrenko, A., and G. von Bochmann. (1994). *Selecting Test Sequences for Partially-Specified Nondeterministic Finite State Machines*, Technical Report, Departement d'IRO, Univ. de Montreal, Canada.

[14] Petrenko. A., Yevtuschenko, N., and G. von Bochmann. (1996). Fault models for testing in context, in *Formal Description Techniques IX*, Chapman & Hall, pp. 163-178.

[15] Ramalingam, T., Das, A., and K. Thulasiraman. (1995) Fault detection and diagnosis capabilities of test sequence selection methods based on the FSM model, *Computer Communications*, vol. 18, no. 2, pp. 113-122.

[16] Robnik, A. (1995). Experiences of using SDL collected in IskraTEL SDL methodology, in *Formal Description Techniques VIII*, North-Holland Elsevier, pp. 221-236.

[17] Schmitt, M., Ek., A., Grabowski, J., Hogrefe, D., and B. Koch. (1998) Autolink – Putting SDL-based test generation into practice, *Testing of Communicating Systems*, Kluwer Academic Publishers, pp. 227-243.

[18] Ural, H., and A. Williams. (1994). Test generation by exposing control and data dependencies within specifications in SDL, in *Formal Description Techniques VII*, North-Holland Elsevier, pp. 335-350.

[19] Verilog. (1993). *GEODE Product Documentation*, Appendix C: SDL Concrete Syntax.

[20] Verilog. (1993). *GEODE Product Documentation*, Chapter 5: SDL Extensions/Restrictions.

[21] Yao, M., Petrenko, A., and G. von Bochmann. (1993). Conformance testing of protocol machines without reset, in *Protocol Specification, Testing, and Verification XIII*, Elsevier Science Publishers, pp. 241-256.

Marjeta Frey-Pučko received the B.S., M.S. and Ph.D. degrees in computer science from the University of Ljubljana, Slovenia, in 1988, 1991 and 1995, respectively. In 1988 she joined the Department of Digital Communications and Networks at the Jožef Stefan Institute in Ljubljana. In the years 1995-1998 she was the manager of the project "Automation of test scenario generation for system SI2000". Since 1998 she has been with IskraTEL where she is responsible for process improvement in the development of the system SI2000. She is also with the Jožef Stefan Institute as a part-time researcher. Her main research interests concern communications systems engineering, verification and validation techniques, and use of formal methods.

Monika Kapus-Kolar received the B.S. degree in electrical engineering from the University of Maribor, Slovenia, in 1981, and the M.S. and Ph.D. degrees in computer science from the University of Ljubljana, Slovenia, in 1984 and 1989, respectively. Since 1981 she has been with the Jožef Stefan Institute, Ljubljana, where she is currently a researcher at the Department of Digital Communications and Networks. She is also a part-time lecturer at the University of Maribor. Her current research interests include formal specification techniques and methods for development of distributed systems and computer networks.

Roman Novak received his B.S., M.S. and Ph.D. in computer science from the University of Ljubljana in 1992, 1995 and 1998, respectively. He works as a researcher at the Department of Digital Communications and Networks at the Jožef Stefan Institute in Ljubljana, Slovenia. Since 1997 he has been also a part-time lecturer at the Faculty of Computer and Information Science at the University of Ljubljana in the fields of programming techniques, algorithms and data structures. His current research interests include distributed systems, communication protocols and security.