SDL-Based Automatic Test Generation for GSM Services

Aleš Švigelj[1], Marjeta Frey-Pučko[1,2] and Gorazd Kandus[1]

[1] Jožef Stefan Institute, Ljubljana, Slovenia
[2] IskraTEL, Ltd., Kranj, Slovenia

The paper introduces automatic test scenario generation for GSM (Global System for Mobile communications) services. The first step of the test scenario generation, which cannot be fully automated, is to create an SDL (Specification and Description Language) description of a GSM service. We propose a methodology for transforming an informal service description to an SDL specification. The form and contents of the resulting specification are adapted to the properties of the test derivation method. The next three steps of the test scenario generation are abstraction of an EFSM (Extended Finite State Machine) to a FSM (Finite State Machine), generation of test sequences and translation to TTCN (Tree and Tabular Combined Notation). These steps are completely automated in the tool iATS. The process of the automatic test scenario generation is illustrated by the example of test scenario generation for the GSM call setup.

Keywords: telecommunication software engineering, conformance testing, automatic test generation, SDL specification, GSM.

1. Introduction

The GSM industry is able to offer an ever-increasing number of services and value-added features to its subscribers. However, it is up to operators to decide the technological strategy used to implement and test new advanced services. Reliable services are a key issue for operators and service providers on the way to success.

To ensure reliability, testing is a very important but complex and time-consuming activity in the service development process. The main contributing factors to the time complexity are the amount of time required to study the functionality to be tested and the manual generation of test scenarios. Another disadvantage of manual test scenario generation is the appropriate test coverage, which, especially for complex services, can be difficult to achieve. The time needed for test scenario generation can be reduced and test coverage improved if automatic test scenario generation is used. However, an automated approach requires firstly a formal specification of the service behaviour to be tested and secondly a tool for deriving test scenarios from the specification. Since derivation algorithms are based on the mathematical properties implied by the specification semantics, it is particularly important that the form and contents of the specification are adapted to the mathematical background of the derivation tool. Only in this case, can automatic derivation of test scenarios give optimal results.

In this paper we present our experience with automatic test scenario generation for GSM services. Over the last three years we have developed a test scenario derivation tool. The tool automatically derives test scenarios in the standard TTCN (Tree and Tabular Combined Notation) from a given specification in the SDL language. Although the tool named iATS - integrated Automatic Test Scenario (generator) - was developed principally for generating test scenarios for ISDN (Integrated Services Digital Network) services, it has wider applicability. Our idea was to transfer the know-how of automatic test scenario generation from the area of fixed telephone networks to the area of mobile (GSM) networks. The main attention has been given to the development of methodology for writing SDL specifications of GSM services.

The paper is organised as follows: Section 2 presents some related work referring to the use of SDL in different development activities for mobile communication systems. In section 3 we describe in detail our methodology for writing SDL specifications. In section 4 we present main parts of the test derivation methodology implemented in the tool iATS. The use of the methodology as well as the generation of test scenarios using iATS are illustrated by the example of GSM basic call. In section 5 we give conclusions and directions for future work.

2. Using SDL in Developing Mobile Communication Systems

SDL is a standardised formal specification language [4] widely used in developing telecommunications systems. Creating an SDL description of a system, a designer is obliged to structure the system hierarchically by elements called systems, blocks, channels, signal routes and signals. In this way a complex system can be described by a set of more understandable and manageable modules which together describe the behaviour of the complete system. The underlying communication model assumes that signals between SDL processes are sent asynchronously and are, after arrival, stored in a separate FIFO (First-In-First-Out) queue for each process. A process is formally represented by an EFSM (Extended Finite State Machine).

The language is widely used and well accepted by designers, for two reasons:

1. Its constructs and the underlying communication model are well suited to describe the behaviour of telecommunications systems.
2. Several commercial CASE tools are available to support a designer in different development activities.

Formal descriptions in SDL are also used in different activities of mobile systems development:

- In the specification phase they provide an unambiguous description of the functionality. Mitchell and Lu [13] have described the Inmarsat-Aero System protocols in SDL. They decomposed the protocols on the basis of physical separation and functionality and represented the communicating partitions in SDL. Jager [10] also used SDL to represent GSM protocols, with particular emphasis on interprocess communication in the application layer.
- Supported by appropriate CASE tools they can be used successfully to generate a simulation executable for use as a validation tool (Mitchell and Lu [13]) or for performance evaluation. Böhmer [2] used it to evaluate the performance of the LAPDm protocol.
- Similarly, C code can be automatically generated from an SDL specification. In this case, the SDL specification should describe all implementation details, as in the example of Zaim and Calikoglu who used that approach for implementing trunked radio systems [16].
- Finally, SDL can be used to provide a system or service behaviour description as an input to automatic test scenario generation. Ek et al. specified the GSM Data Link protocol in SDL and generated test scenarios in TTCN [5]. Their approach is similar to ours except in two basic differences: first, their test derivation process uses a different formal background, and second, it involves more than one software tool. Boullier et al. evaluated some existing tools for test scenario generation using the example of the GSM MAP protocol [3]. They started with an SDL specification provided already by ETSI.

3. Writing SDL Specifications

The main purpose of an SDL specification for use as the input of a test scenario generation tool is to describe the complete behaviour to be tested. In our example the specification is used for generation of conformance tests. Conformance testing is a step in the service validation process where the conformance of a service implementation is checked against the given service specification. This type of testing can be characterised as black box testing, where the testing system interacts with the implementation under test (IUT) only through inputs and

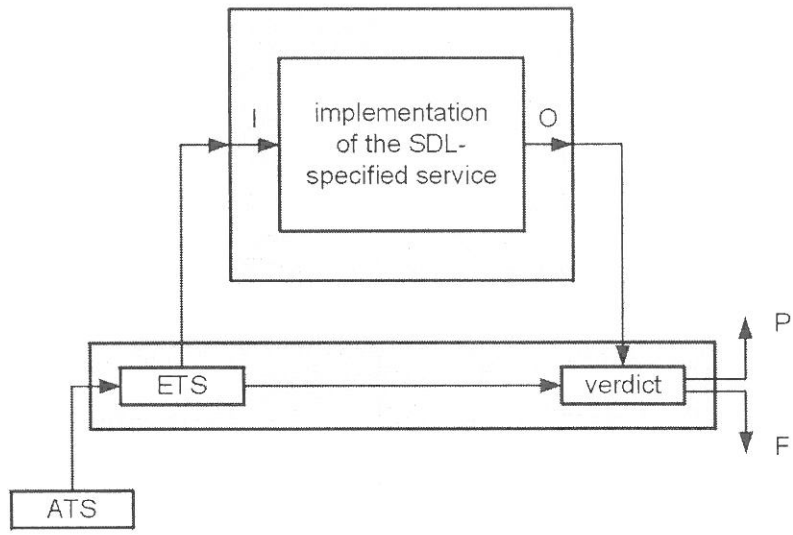

Fig. 1. Testing architecture.

outputs. The testing architecture is shown in Figure 1.

The set of test scenarios automatically derived from the SDL specification and described in TTCN language represents an abstract test suite (ATS) which is independent of the actual testing system. To be executed by the testing system, an ATS has to be transformed into the executable test suite (ETS). The verdict in each step of a test scenario execution depends on the output sent by the IUT to the testing system after it has received an input from the testing system. If the actual outputs in all steps of the test scenario execution match the expected ones, the test scenario passes (P), otherwise it fails (F). If all scenarios needed to prove conformance pass, the IUT conforms to the specification. The terminology used is defined by the ISO/IEC 9646 recommendations for a conformance testing framework [8].

To write an SDL specification we need precisely defined input and output, methodological rules and an appropriate tool support:

- The input comprises adequate information on the functionality to be specified. For the example of GSM services, information can be obtained from informal service specifications available in standards or internal documents of a service provider.
- The output is an SDL specification of the functionality meeting certain requirements about its form and contents.
- Methodological rules define how an output should be generated from the input.
- Among software tools supporting the specification activity, an editor (preferably graphical) and a syntax checker are the most necessary. A further, very valuable tool is a semantic checker to check the specification against logical errors, such as deadlocks.

An SDL specification of a GSM service used as an input of the iATS tool can give prospective results in test scenario generation if it fulfils the following requirements about the form and contents:

- First, the abstraction level and precision in the specification should be appropriate for deriving conformance test scenarios.
- Second, structure of the specification and properties of the contained EFSM should ensure optimal results, referring to the test derivation methods implemented in the iATS. The requirements belonging to different specification problem domains imply the methodological rules described in the next subsection.

A very important question is how to reach the appropriate abstraction level and which details of behaviour to present in the specification. To generate conformance test scenarios we need to specify the behaviour of the system under test, i.e. the complete observable interface with the user. The system actions are therefore observed only at the user-system interface of the mobile station but actually they are the result

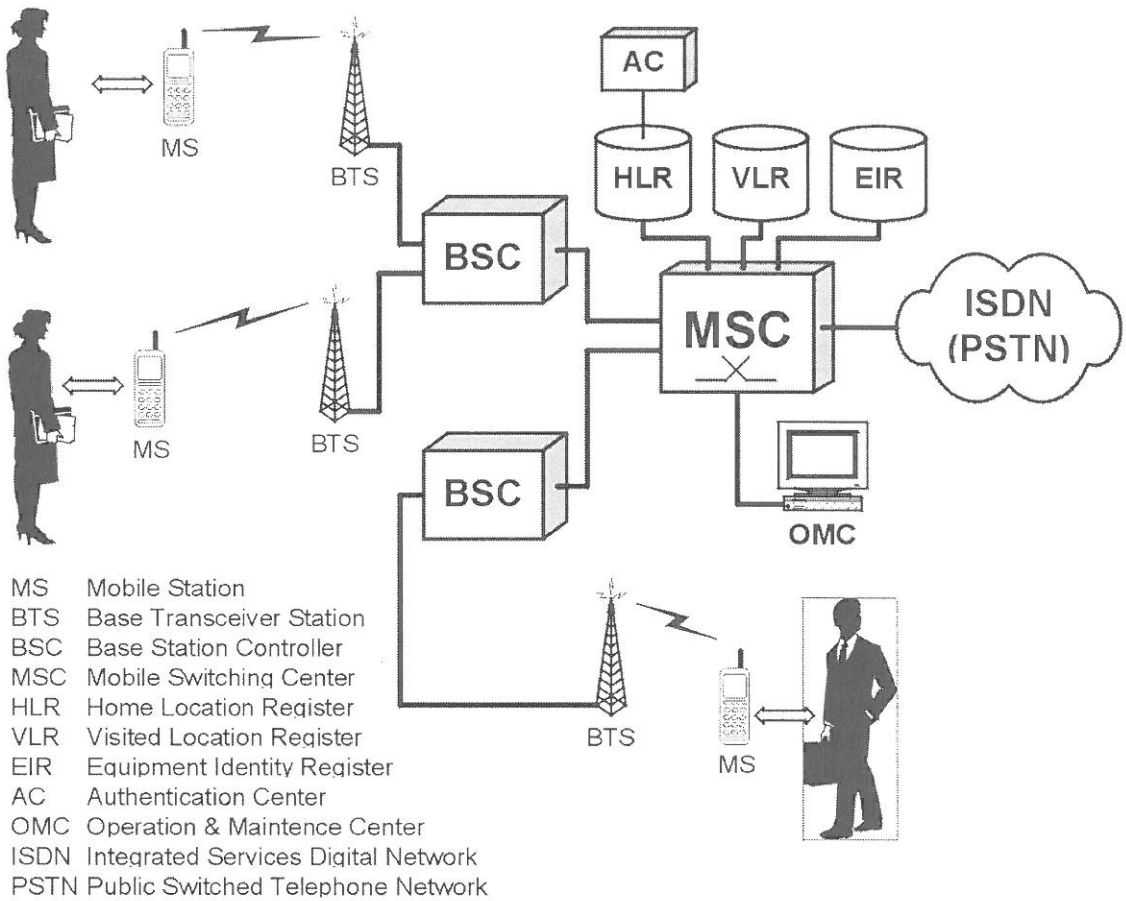

Fig. 2. Communication in a GSM system.

of sequences of actions executed in different parts of the system: mobile station, switching subsystems, base station system, related fixed network etc. (shown in Figure 2). We follow the same approach for representing system actions in both mobile and fixed networks. The only differences are the actual (physical) parts and applied technologies of the system whose actions we describe in SDL in such an abstract way.

3.1. Methodology

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 on which abstraction level the SDL specification describes the given functionality.
2. *Naming rules* specify how names of all the 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 by means of test scenario generation for compositional testing.
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 in order to provide an acceptable input for the iATS tool, other rules may be used optionally to provide an input for iATS which gives better results during test scenario generation. Mandatory rules impose restrictions, in particular on the use of certain SDL constructs and their combinations. Optional rules deal more with

the number of SDL processes and properties of the EFSM inside the SDL specification which are expected to give the best result in terms of test scenario length and test coverage.

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

In developing the rules we considered two existing methodologies: ITU basic methodological guidelines for use of SDL [9] and the IskraTEL SDL methodology (shown in Figure 3) [15]. The reason for considering the latter was the need to tailor iATS to the previously existing SDL specifications, which were developed in accordance with that methodology. Although we adopted some of its naming and adaptation rules, most of the rules of our methodology IJS ATGB (Institut Jožef Stefan Abstract Test Generation Basic Guidelines) have been newly defined.

3.2. Allowed Contents for Test Scenario Generation

The allowed input of the iATS tool may be any SDL specification in textual form created by means of the proposed methodology. Here we should emphasize that the mandatory adaptation rules impose some limitations on the use of SDL constructs. The most important are:

- Each transition has to contain an input signal. An output signal may not be used immediately after the *start* construct.
- The constructs *save, continuous signal, import, export, view, revealed* and *alternative* are not allowed.
- Declaration of subprograms inside other subprograms is not allowed.
- Use of timers with parameters is not allowed.
- Newly defined and some complex data types are not allowed.

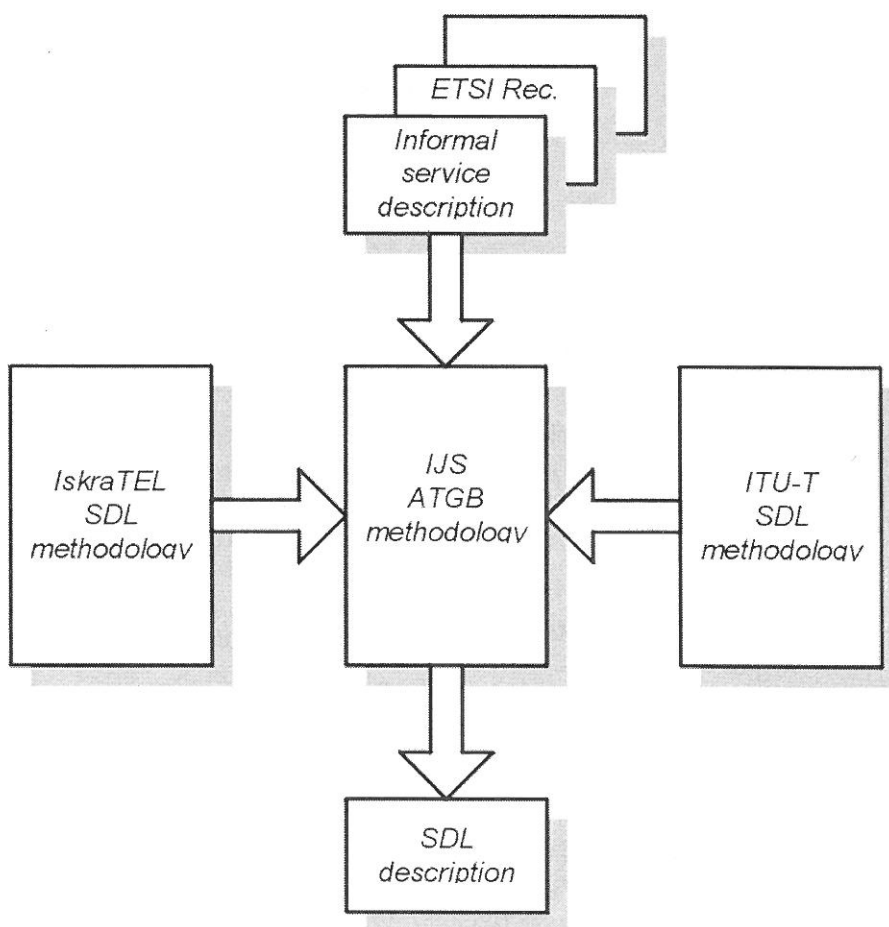

Fig. 3. Methodology for writing SDL specifications.

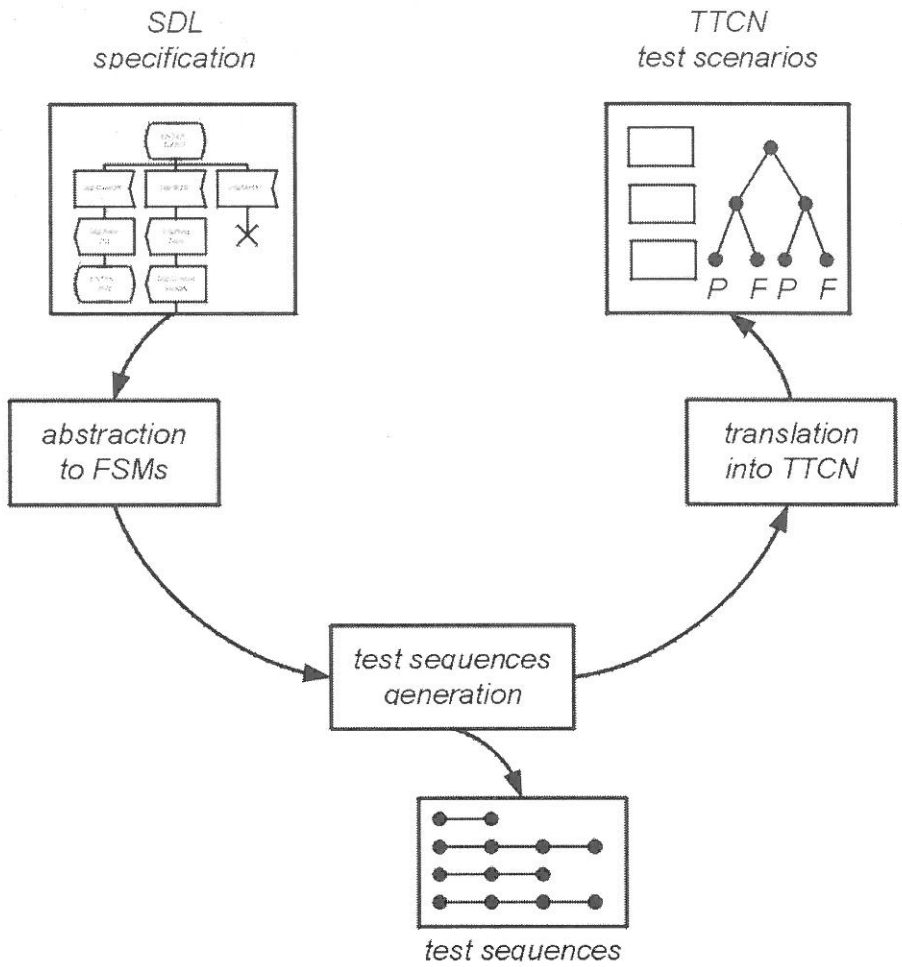

Fig. 4. Test scenario generation.

- In one generation of an FSM from the SDL specification only one value of an input signal parameter may be used. The signals may not be further structured.

Our experience shows that the set of allowed constructs and their combinations suffices to specify most of the services of mobile and fixed networks.

4. Automatic Test Scenario Generation

For actual test scenario generation we principally selected test generation methods based on the FSM (Finite State Machine) model. The reason is the availability of many FSM-based methods with a well-defined mathematical background, which cover a wide spectrum of FSM properties. Another reason is that the model is very close to the EFSM (Extended Finite State Machine) model of SDL. Transformation of an SDL description into an FSM specifying an equivalent behaviour is performed using abstraction and composition techniques. As shown in Figure 4, test scenarios are generated in the following 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 [1, 11], or test generation methods for non-deterministic protocol machines [12]. Selection of the method depends on the properties of the FSM.
- *Translation to TTCN.* The generated test sequences are automatically translated into

TTCN test scenarios. While the behaviour part of the TTCN description is generated completely automatically, constraints in the declaration part have to be inserted manually. The scenarios are described in a standardised form, which is accepted and can be further used by commercial TTCN editors and compilers.

The terms "test sequences" and "test scenarios" are not used as synonyms. A test sequence, in the literature also called a "test", denotes a sequence of inputs and expected outputs uniquely identifying a state in a FSM. A test scenario (from an ATS or ETS) has a wider meaning; it contains the complete information on the structure and on verdicts. In the example of ETS, it contains the actual (physical) contents of a test sequence, executed in a real testing environment.

The steps described above are implemented in the iATS tool, developed for HP-UNIX and X-Windows environment. It is owned and used as an in-house tool by IskraTEL Ltd., Telecommunications Systems. As the input of the tool, any SDL specification in the textual form may be used, being created in correspondence with the methodology described in the previous section. The tool generates two main outputs: a set of generated test sequences described by transitions of a FSM, and a TTCN description of test scenarios in the standardised form. As an auxiliary output, another description in style of TTCN is generated. It contains less information on test scenarios, especially on the declaration part, but it provides much more readable information on the behaviour part. They are also used for describing non-deterministic test scenarios where, because of the internal representation of timers, the standardised TTCN form cannot be generated. iATS consists of the following components:

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, after which they are considered as fixed.
2. *Tool for composition and construction of approximate automata.* From the FSMs constructed by abstraction from SDL processes, a composed FSM is generated. Construction of an approximate automata is possible if some constructs in the SDL specification have been declared by the user as hidden, if some SDL processes are proved to be independent, 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. During the test generation appropriateness of the selected method is evaluated automatically; if an alternative which could give better results is detected, the tool automatically starts test generation with another method and settings.
4. *Graphical user interface.* This 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 especially for iATS, the first three components can also be used as stand alone tools. A detailed description of iATS is given in [14].

5. Example: GSM Call Setup

Generation of test scenarios for GSM call setup was selected as an example because it is a basic service and is not too complex to be presented in a paper. We first present the functionality as described in [6], after that we give its description in SDL and finally explain some test cases.

5.1. Specified Functionality

The call setup procedure is performed as follows: In the first state (*MS_OFF*) the user has to switch the mobile on with the signal U@MsOn. This action leads in the main process to the state named *ENTER_CARD* and the mobile phone is activated. Even if there is no SIM card inserted in the mobile phone, the user can establish a SOS call. This is a standard feature of mobile phones. The SOS call can also be established in almost all of the following states. When the

SIM card is inserted and the PIN is correct, the user has to select PLMN. Selection of PLMN is important only in case there is more than one network available. At this moment the handy is ready to start with the call setup. There are two possibilities of call setup when the call is mobile-originated:

- User first executes Off Hook with signal *U@OffHook*. The dial tone is presented and the process proceeds to the state *CALL_INI1*. The user has then to enter the calling number. Afterwards, the *Timer_U_ini* timer is activated and the process is transferred to the state CALLING1. Now the user may press the Send key. If the key is pressed, the timer is reset and the user gets the ring tone with a signal *U@RingTone*. After the other party has answered the user's call, the call is established and the user receives the signal *U@ConnectionOK*. If the Send key is not pressed, the call is established automatically after the timeout delay of five seconds. The user may drop the call at any time using the signal *U@OnHook*.
- The user only has to enter the number using the signal *U@EntryNumber*. The process is transferred to the state *CALLING2*. In this case the user must press the Send key. As in the previous case, the ring tone and the signal *U@ConnectionOK* are received.

The call is established and the system is in the state *CALL_ON*. When the user wants to release the call, there are two possibilities, depending on how the call has been established. In one case the user can release the call pressing the OnHook key and in the other pressing the End key.

Behaviour at the GSM call setup actually consists of four very similar sub behaviours. In our example we only describe the most complicated one. It is the case where a call is mobile-originated and mobile-terminated. The corresponding SDL specification is graphically depicted in Figure 5. The process has ten states. Names of states are self-explanatory and denote the action which has to be performed in the next transition. As recommended by our methodology, we represent the behaviour of a specified basic service by a single SDL process.

Four different types of calls are possible for each user:

- mobile-originated call – mobile-terminated call,
- mobile-originated call – B party-terminated call,
- B party-originated call – mobile-terminated call,
- B party-originated call – B party-terminated call.

To generate test scenarios and test the system for all types, we represented the behaviour of the system by four different system descriptions for each type of call. Using this solution the undesired information about the type of call is excluded from the test scenarios.

There are also two different types of call establishment in mobile communications: the first is where a mobile user enters the calling number and then presses the Send-function key, and the second one is where a mobile user first lifts the hand-set "Off Hook", when the dial tone is present. The user enters the number and presses the Send-function key. If the Send key is not pressed, the call set up may be automatically initiated after expiry of the 5 second time-out. Both possibilities have been taken into account in all system descriptions.

Since the main purpose of using naming rules is to make the SDL description as readable and understandable as possible, we have used the following names:

- The system name denoting functionality is *gsm_call_setup*.
- Since only one process is used, its name is *main_process*.
- The number of channels and the number of participating users are the same. When there was only one user we used *USER_U*. The names of signal lists depend on the direction of signals on the list. If a signal is sent from the user to the system, the name is *from_USER_U*. In the opposite direction the name is *to_USER_U*. The names of signals are always written as *user@descriptionofsignal* (i.e. *U@off_hook*). We have taken most of the signal names from the GSM standards [6, 7]. The same principle has been used for the names of states.

The functionality described has been specified in SDL using the commercial tool Verilog

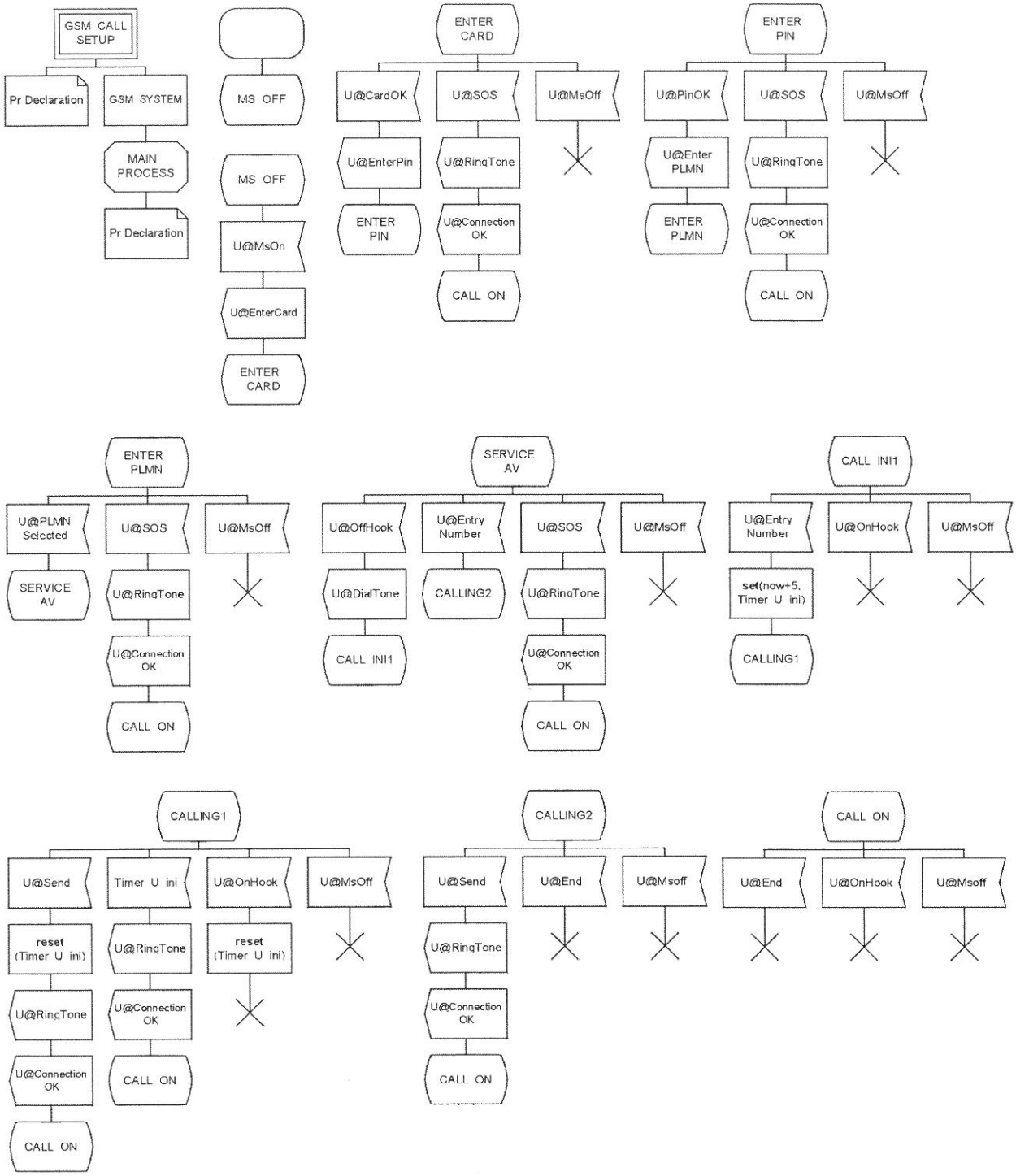

Fig. 5. SDL specification of the call setup procedure.

GEODE. Part of the specification is graphically represented in Figure 5. Two tools from the GEODE toolbox have been used: the GEODE Editor as a framework for writing specifications and syntax checking, and the GEODE simulator for verifying the created specification. The GEODE Editor automatically generates textual form from the graphical one and vice versa.

5.2. Generation of Test Scenarios

We successfully used the SDL specification presented above for actual test scenario generation. Three different outputs resulted from this: a set of test sequences described by the transitions

Description	*Lines of description*
SDL specification of GSM call setup in textual form	488
ATS in the test sequence form	244
ATS in the standard TTCN form	3879
The selected test scenario in the test sequence form	9
The selected test scenario in the standard TTCN form	129

Table 1. Length of SDL specification and generated scenarios

of a FSM, a TTCN-like description of the scenario behaviour, and a standard TTCN description of scenarios. While the first two provide easily readable information concerning actual test sequences, the latter gives a complete ATS description which can be used as an input to TTCN compilers for translation into ETS. It already contains all the information apart from the definitions of TTCN constraints. A selected test scenario from the generated ATS is given in the Appendix, in test-sequence form and in the standard TTCN form. Comparing both forms, the difference in amount and representation of information can be clearly seen. The test sequence form actually presents the test sequences <state, input signal, output signal, cost of the tested transition, next state> generated from the FSM. The cost is a non-negative number reflecting the difficulties associated with execution of a transition, for example the necessary time or resources. The information on verdict is implicit, i.e. the verdict is P (pass) if the complete test sequence is successfully executed. The standard TTCN form as defined in [8] explicitly describes possible verdicts in each step of scenario execution. It contains no state names except in the names of timer signals. Due to the length, only the test behaviour is presented in the Appendix, without definitions in the TTCN constraints. Some quantitative results referring to the length of test scenarios are given in Table 1.

6. Conclusion

We have proposed an approach to the automatic generation of test scenarios for GSM services. Particular attention has been paid to the methodology of writing SDL specifications. For actual test generation we use the iATS tool which gives, as its output, test scenarios in a standard TTCN form. Advantages of this approach are the considerably decreased time required for test scenario generation and the better test coverage. Consequently, a highly reliable service can be developed despite the reduced time needed for testing.

In our future work we shall consider the appropriateness of the proposed methodologies for creating SDL specifications and SDL-based automatic generation of test scenarios for a wider set of GSM services. In addition, we also plan to apply our approach to test scenario generation for testing GSM protocols.

References

[1] A. V. AHO, A. T. DABHURA, D. LEE, M. U. UYAR, An optimization technique for protocol conformance test generation based on UIO sequences and rural Chinese postman tours, *IEEE Transactions on Communications*, vol. 39, no. 11, 1991, pp. 1604–1615.

[2] S. BÖHMER, Performance analysis of the GSM signalling protocol LAPDm, *Creating Tomorrows Mobile Systems: Proceedings of the IEEE 44th Vehicular Technology Conference, Stockholm*, Sweden, 1994, IEEE, New York, USA, 1994, pp. 719–723.

[3] L. BOULLIER, B. KELLY, M. PHALIPPOU, A. ROUGER, N. WEBSTER, Evaluation of some test generation tools on a real protocol example, *Protocol Test Systems, 7th IFIP WG 6.1 Int., Workshop on Protocol Test Systems, Tokyo*, Chapman & Hall, London, 1995, pp. 141–154.

[4] CCITT, Revised Recommendation Z.100 CCITT Specification and Description Language (SDL), 1992.

[5] A. EK, J. ELLSBERGER, A. WILES, Experiences with computer-aided test suite generation, Protocol Test Systems, VI(C–19), O. Rafiq (Ed.), Elsevier Science B.V. (North–Holland), 1994, IFIP, pp. 181–195.

[6] ETSI, *Digital cellular telecommunications system (Phase 2+); Man-Machine Interface (MMI) of the Mobile Station (MS)*, GSM 02.30 version 5.6.0, Part A: Release 96 and Release 97, 1997.

[7] ETSI, *Digital cellular telecommunications system (Phase 2+); General description of a GSM Public Land Mobile Network (PLMN)*, GSM 01.02, 1996.

[8] ISO/IEC 9646–3, Second Edition, 1997.

[9] ITU–T, Z.100, *Appendix I, SDL Methodology Guidelines*, 1993.

[10] R. JAGER, A communication mechanism for GSM, Telecommunications, Vol. 26, No. 5, 1992, pp. 107–117.

[11] L. JIREN, D. JUN, A new approach to protocol conformance test generation based upon UIO sequences, *Chinese Journal of Advanced Software Research*, vol. 1, no. 4, 1994, pp. 373–381.

[12] G. LUO, A. PETRENKO, G. V. BOCHMANN, *Selecting Test Sequences for Partially-Specified Nondeterministic Finite State Machines*, Technical Report, 35 p., Departement d'IRO, Universite de Montreal, Canada, 1994.

[13] L. MITCHELL, S.–C. LU, Specifications and validations of Inmarsat Aeronautical system protocols, SDL '93: Using Objects, O. Faergemand, A. Sarma (Eds.), Elsevier Science Publishers B.V., 1993, pp. 51–63.

[14] M. FREY–PUČKO, M. KAPUS–KOLAR, R. NOVAK, R. VERLIČ, B. MOČNIK, B. SLIVNIK, V. AVBELJ, Automatic Test Scenario Generation for the System SI2000, Final report, IskraTEL, 1998 (in Slovene).

[15] A. ROBNIK, Experiences of using SDL collected in IskraTEL SDL methodology, *FORTE'95*, North–Holland Elsevier, 1995.

[16] A. ZAIM, F. CALIKOGLU, Using SDL in a commercially available wide area coverage trunking mobile radio system development, *SDL '93: Using Objects*, O. Faergemand, A. Sarma (Eds.), Elsevier Science Publishers B.V., 1993, pp. 41–49.

Appendix: An example of a generated test scenario in TTCN

1. An example of a generated scenario in the test sequence form

```
@MS_OFF_1
U@MSON U@ENTERCARD 1.000000
@ENTER_CARD2
U@SOS U@RINGTONE:U@CONNECTIONOK 1.000000
@CALL_ON_4
U@MSOFF NULL 1.000000
@end_5
user@reset NULL 1.000000
--- 4.000000
```

2. The same scenario in the standard TTCN form

```
$Begin_TestCase
    $TestCaseId gsm_i11_4
    $TestGroupRef
    $TestPurpose /* */
    $DefaultsRef
    $BehaviourDescription
        $BehaviourLine
            $LabelId
            $Line [0] U!MSON START NullInput_MS_OFF_1_U_MSON
            $Cref
            $VerdictId
            $Comment /* */
        $End_BehaviourLine
        $BehaviourLine
            $LabelId
            $Line [1] U?ENTERCARD CANCEL NullInput_MS_OFF_1_U_MSON
            $Cref
            $VerdictId
            $Comment /* */
        $End_BehaviourLine
```

```
        $BehaviourLine
            $LabelId
            $Line [2] U!SOS START NullInput_ENTER_CARD_2_U_SOS
            $Cref
            $VerdictId
            $Comment /* */
        $End_BehaviourLine
        $BehaviourLine
            $LabelId
            $Line [3] U?RINGTONE CANCEL NullInput_ENTER_CARD_2_U_SOS,
START NullInput_ENTER_CARD_2_U_SOS
            $Cref
            $VerdictId
            $Comment /* */
        $End_BehaviourLine
        $BehaviourLine
            $LabelId
            $Line [4] U?CONNECTIONOK CANCEL NullInput_ENTER_CARD_2_U_SOS
            $Cref
            $VerdictId
            $Comment /* */
        $End_BehaviourLine
        $BehaviourLine
            $LabelId
            $Line [5] U!MSOFF START NullInput_CALL_ON_4_U_MSOFF
            $Cref
            $VerdictId
            $Comment /* */
        $End_BehaviourLine
        $BehaviourLine
            $LabelId
            $Line [6] ?TIMEOUT NullInput_CALL_ON_4_U_MSOFF
            $Cref
            $VerdictId
            $Comment /* */
        $End_BehaviourLine
        $BehaviourLine
            $LabelId
            $Line [7] user!reset START NullInput_end_5_user_reset
            $Cref
            $VerdictId
            $Comment /* */
        $End_BehaviourLine
        $BehaviourLine
            $LabelId
            $Line [8] ?TIMEOUT NullInput_end_5_user_reset
            $Cref
            $VerdictId
            $Comment /* */
        $End_BehaviourLine
        $BehaviourLine
            $LabelId
            $Line [8] U?OTHERWISE CANCEL NullInput_end_5_user_reset
            $Cref
            $VerdictId F
            $Comment /* */
        $End_BehaviourLine
```

```
        $BehaviourLine
            $LabelId
            $Line [6] U?OTHERWISE CANCEL NullInput_CALL_ON_4_U_MSOFF
            $Cref
            $VerdictId F
            $Comment /* */
        $End_BehaviourLine
        $BehaviourLine
            $LabelId
            $Line [4] ?TIMEOUT NullInput_ENTER_CARD_2_U_SOS
            $Cref
            $VerdictId F
            $Comment /* */
        $End_BehaviourLine
        $BehaviourLine
            $LabelId
            $Line [4] U?OTHERWISE CANCEL NullInput_ENTER_CARD_2_U_SOS
            $Cref
            $VerdictId F
            $Comment /* */
        $End_BehaviourLine
        $BehaviourLine
            $LabelId
            $Line [3] ?TIMEOUT NullInput_ENTER_CARD_2_U_SOS
            $Cref
            $VerdictId F
            $Comment /* */
        $End_BehaviourLine
        $BehaviourLine
            $LabelId
            $Line [3] U?OTHERWISE CANCEL NullInput_ENTER_CARD_2_U_SOS
            $Cref
            $VerdictId F
            $Comment /* */
        $End_BehaviourLine
        $BehaviourLine
            $LabelId
            $Line [1] ?TIMEOUT NullInput_MS_OFF_1_U_MSON
            $Cref
            $VerdictId F
            $Comment /* */
        $End_BehaviourLine
        $BehaviourLine
            $LabelId
            $Line [1] U?OTHERWISE CANCEL NullInput_MS_OFF_1_U_MSON
            $Cref
            $VerdictId F
            $Comment /* */
        $End_BehaviourLine
    $End_BehaviourDescription
$End_TestCase
```

Received: March, 1999
Revised: October, 1999
Accepted: October, 1999

Contact address:

Aleš Švigelj
Institut Jožef Stefan
Jamova 39
SI-1000 Ljubljana
Slovenija
phone: ++386 61 1773 379
fax: ++386 61 1262 102
e-mail: ales.svigelj@ijs.si

Marjeta Frey-Pučko
IskraTEL, Ltd.
Ljubljanska 24a
SI-4000 Kranj and
Institut Jožef Stefan
Jamova 39
SI-1000 Ljubljana
Slovenija
e-mail: pucko@iskratel.si, marjeta.pucko@ijs.si

Gorazd Kandus
Institut Jožef Stefan
Jamova 39
SI-1000 Ljubljana
Slovenija
e-mail: gorazd.kandus@ijs.si

ALEŠ ŠVIGELJ received his B.Sc. degree in electrical engineering from the University of Ljubljana, Ljubljana, Slovenia, in 1997. Currently he is a research assistant in the Department of Digital Communications and Networks at the Jožef Stefan Institute and an M.Sc. student of telecommunications at the Faculty of Electrical Engineering at the University of Ljubljana. His research interests concern the development of mobile communication systems.

MARJETA FREY-PUČKO received her Diploma, M.Sc. and Ph.D. degrees in computer science from the University of Ljubljana, Slovenia, in 1988, 1991 and 1995, respectively. In 1988 she joined the Digital Communications and Networks Department at the Jožef Stefan Institute in Ljubljana. Since 1998 she has been with IskraTEL where she is responsible for process improvement in the development of the SI2000 system. She is also with 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.

GORAZD KANDUS received his B.Sc., M.Sc. and Ph.D. degrees in electrical engineering from University of Ljubljana in 1971, 1974 and 1991 respectively. He is currently the head of the Department of Digital Communications and Computer Networks at Jožef Stefan Institute, Ljubljana, Slovenia. His main interests involve telecommunications systems including services, network protocols and architectures for satellite and terrestrial mobile communication systems.