Testing ISDN Services Based on SDL Specifications

Marjeta Pučko
Jožef Stefan Institute
University of Ljubljana
Jamova 39, 1001 Ljubljana, Slovenia
marjeta.pucko@ijs.si

Abstract

Experiences in using formal methods in the practical example of automated derivation of test sequences for testing telecommunications services are described. The input of the first phase of the complete derivation process, where a logically correct SDL specification is generated, is an informal ETSI description, specifying the external behaviour of a given ISDN service. The informal-to-formal transformations in this phase are not completely automated. The output of its second phase is a set of automatically derived test sequences described in TTCN. Test sequences are generated using FSM-based methods.

1 Introduction

Generation of test scenarios for testing telecommunications software is often a complex and time-consuming task which can be considerably simplified by automation, based on the use of formal methods. Automation of the complete derivation process of test scenarios for ISDN services is the aim of a current industrial project[1] for Iskratel Ltd, Telecommunications Systems. On the basis of ETSI recommendations and Iskratel informal descriptions, SDL specifications of ISDN services are to be generated, and test scenarios for the services automatically derived.

Several approaches to the derivation of test scenarios from SDL specifications can be found in the literature. Ural and Williams [17] derived scenarios from flowgraphs for SDL processes or procedures. Luo, Das and Bochmann [9, 10] represented SDL processes and procedures by SDL-machines, transformed them into FSMs and derived scenarios by using existing FSM-based methods. Heinrich and Pospiech [4] concentrated only on chains of dependent variable values. Except for [17], these approaches dealt with a relatively small subset of SDL constructs. All the approaches also derived test scenarios from an FSM representing a single process or procedure.

After careful study of the existing approaches, we developed a new approach which offers the possibility to automate the test scenario derivation process, and to do this also for SDL specifications with a wider set of SDL constructs and several involved processes. In the first phase of the approach an SDL specification of the service under test is generated from informal specifications. In the second phase we translate the SDL specification into a suitable form for application of test generation methods, and derive the test scenarios. While the functionality of the first phase is already covered by existing commercially available tools, the second phase requires development of a new tool. For a restricted class of SDL specifications, we have already implemented the second phase by an initial version of a tool. Since the tool is still under development, we present some preliminary results, especially our experiences in using formal methods.

In what follows, we explain for each formal approach used or description technique the main reasons why it has been selected for use in our tool, and discuss the advantages and disadvantages which have been demonstrated in the development and the use of the tool. After that we illustrate the use of selected formal techniques by an example of the activation procedure for the ISDN service "Call Forwarding Unconditional". Finally, we present some conclusions and directions for future work.

2 The approach

2.1 Formal description of a service

Automation of test scenario generation requires an input which formally describes the relevant functionality of the service under test, i.e. the external behaviour of the service. ETSI recommendations for ISDN services may be taken as an appropriate basis for construction of the formal description. Since we need a description of a service functionality from the user's standpoint, it suffices to use an ETSI Stage 1 overall service description. Information on the service options implemented can be found in the Iskratel

[1]The project is financially supported by Iskratel Ltd, Telecommunications Systems.

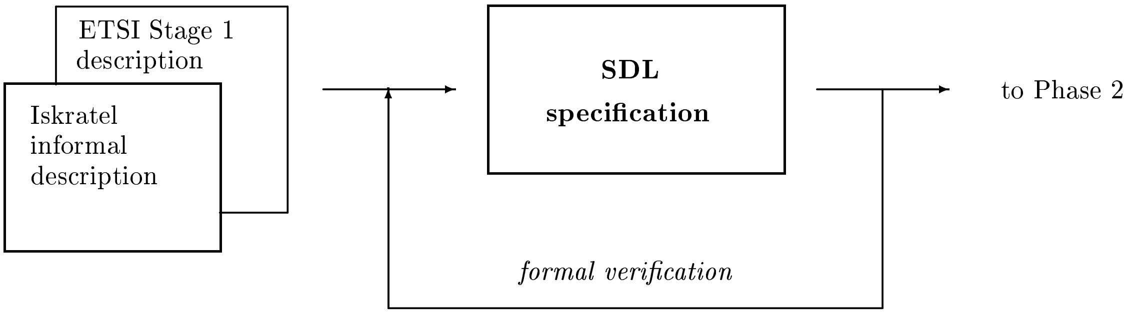

Figure 1: Formal description of a service

informal description.

Formally specifying a service, we assume the same architectural view of the system and the service as ETSI Stage 1 descriptions and the Iskratel informal description (for example [3], and [13]), and include the same functionality. A system executing a service, in this case the system SI2000, communicates with its users over channels. If the external service behaviour of the service is to be tested as seen by its users, we are obliged to observe action sequences passing the channel between a user and the system. A formal specification should therefore contain all possible user requests from the system and system answers to users, and vice versa. On the other hand, it is not desired to describe the nonvisible internal behaviour of the system. A formal specification has also to meet the following requirements:

- The description of the system-users interaction in the scope of a service needs the addition of an interactive description of the basic call processing which has invoked the service execution. Only successful call establishments may here be taken into account. In this context, the formal specification is not always independent of the service implementation.
- User and system actions should be denoted by concrete names, used by the system and service developers.
- It should be made in SDL (ITU Specification and Description Language [2]) using the commercial tool Verilog GEODE, because the system and service developers in Iskratel usually use this computer-supported environment to create formal specifications.

The first phase of our approach in which we construct a formal description of a service proceeds in two steps:

s1.1. *Specification in SDL.* The step of transforming an informal ETSI description and the Iskratel informal description into a corresponding formal SDL specification, due to its context-dependent nature, is not completely automated. As the basic behavioural representation of the specified service, we consider a control or operation procedure as specified by ETSI [3]. We structure the behaviour of the system executing the procedure in the behaviours of the communication subsystems. Here, we adopt the structuring principles for informal-to-formal transformations suggested in [12]. The actual creation of the SDL description is done using the GEODE tool and considering the Iskratel methodology for the use of SDL [16].

s1.2. *Verification of the SDL specification.* The specification from the step s1.1 is formally checked for syntactic and semantic correctness. The verification is performed using the GEODE Simulator. If any errors are detected, the simulation is followed by correction of the specification (as shown in Figure 1). Consequently, this step has to be repeated until the SDL specification is error-free.

The advantages of SDL, which became apparent on use, are its status as a well-standardized language, and the resulting sufficient support by commercially available tools for specification and verification. Another advantage is the underlying asynchronous communication model which is able to represent communication between processes in telecommunications services in a very natural way. Among its disadvantages, we place the lack of abstraction (in comparison to LOTOS, for example) when a service behaviour should be described at a high level of abstraction, and the lack of well-defined formal semantics (SDL is therefore often considered as a semiformal language).

2.2 Test scenario generation

For actual test scenario generation, we principally selected FSM-based methods which cannot directly be applied to EFSM-based SDL specifications. However, by using abstraction and composition techniques we can construct an FSM, describing the SDL-specified behaviour. Another problem arises from the global view of external service behaviour in generation of test scenarios. If the generated scenarios involve more than one service user and they are to be executed on testing equipment, assuming a distributed view, introduction of synchronization messages is needed to assure the correct ordering of actions during execution of a distributed scenario.

The second phase, where we generate test scenarios and eventually distribute them among the users involved, proceeds in the following steps:

s2.1. *Abstraction to FSMs.* Each SDL process, in the SDL service specification represented as an EFSM, is abstracted to a corresponding FSM. The compiler implementing the step also includes the FSMs of procedures, acting inside an SDL process, into the FSM for the whole process. To provide a deterministic FSM, a user of the tool has the possibility to insert fixed parameter values, and to specify the circumstances of test scenario execution by selection of branches in the SDL specification. Otherwise, the resulting FSM may be nondeterministic.

s2.2. *Composition of FSMs.* The FSMs from Step s2.1 are composed together into a combined FSM, modelling the complete SDL-specified behaviour. Here, we suggest the use of parallel asynchronous composition. The underlying FIFO queueing mechanism and the contents of the original SDL specification may also be taken into account. This is particularly important, if the SDL specification contains constructs which affect the ordering of signals (save or timer, for example). Observation of process input queues is not needed in the case of non-delaying channels, or in the absence of logical errors (such as deadlocks because of noncontrollable delays in channels) checked by simulation.

s2.3. *Generation of test sequences.* From the combined FSM, test sequences are generated using well-known UIO-based methods, or test generation methods for nondeterministic protocol machines. Selection of the method depends on the properties of the FSM. The test sequence generator [8] implements the methods of Aho, Dabhura, Lee, Uyar [1], Yao, Petrenko, Bochmann [18], and Jiren, Jun [6] for deterministic FSMs, and the method of Luo, Petrenko, Bochmann [11] for nondeterministic FSMs.

s2.4. *Translation of test sequences into TTCN.* The test sequences, generated in Step s2.3 and described by states, inputs and expected outputs, are translated into a recommended TTCN description [5]. The TTCN form is required for eventual automated further translation of the generated test scenarios into a form executable on the real object.

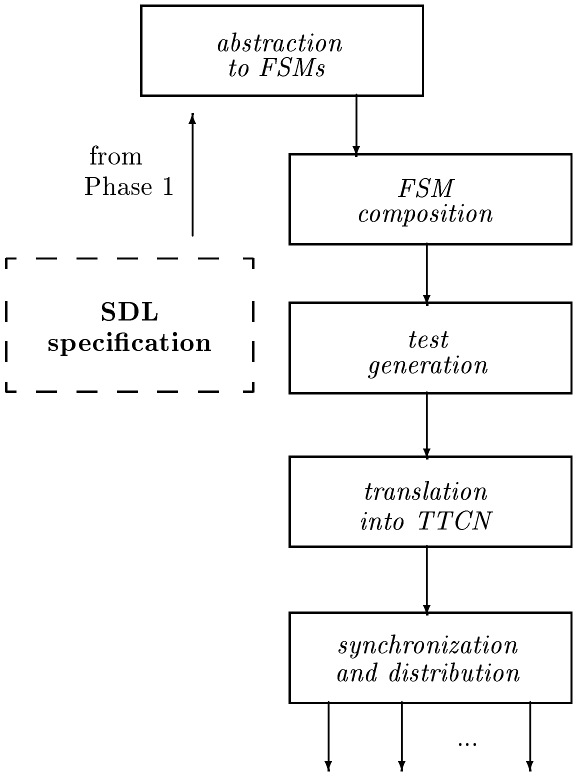

Figure 2: Test scenario generation

s2.5. *Synchronization and distribution.* If the testing equipment executing the test scenarios in practice assumes a distributed view to the external service behaviour, the scenarios have to be correctly distributed among the service users involved. Here, we suggest the use of correctness preserving transformations which automatically distribute the scenarios and derive the required synchronization. The corresponding transformations for LOTOS-like languages are suggested in [7] and [14]. The latter can be efficiently applied in specific cases where a scenario is executed only for a subset of the service users involved. The step is optional.

The main reason for the use of the FSM model in our approach is the availability of many FSM-based methods for generation of test sequences which cover a wide spectrum of FSM properties. The model is also very close to the EFSM model of SDL. Other advantages of the FSM model are the natural representation of process behaviour and the well-defined theoretical background. However, system runs in this model are not explicitly visible. This is a disadvantage in the case when execution ordering of actions has to be observed. The disadvantage of the selected FSM-based methods for generation of test sequences is their applicability to a restricted class of FSMs. FSMs for real services often do not have the required properties (for example, strong connectedness or completeness), as well as the fact that the system is not always able to provide a reliable reset capability.

The advantages of TTCN are the availability of standards, and its easily understandable and well-structured notations. Among the disadvantages, we found that it allows no additional information on test scenarios exceeding the general format of dynamic behaviour. It also provides no efficient form for specification of nondeterministic cases which result in long sequences of pairs, consisting of the same input and a different output.

3 An example

The ISDN supplementary service "Call Forwarding" provides the possibility to a served user of forwarding incoming calls to a desired number. ETSI recommendations specify three variants of the service [3]:

1. *Call Forwarding Unconditional* (in the following CFU), where all incoming calls are forwarded without being offered to the served user. The served user, as a subscription option, may receive notification of the call forwarding but shall not be able to answer the incoming call.
2. *Call Forwarding Busy*, where an incoming call to the served user shall be forwarded if the served user is network or user determined as busy. In the case of the user determined busy condition, the call is first offered to the served user together with provision of normal call set-up information. The network determined busy condition results in forwarding the call without it being offered to the served user. Both cases optionally provide the forwarding notification to the served user.
3. *Call Forwarding No Reply*, which offers all incoming calls to the served user. If the served user does not reply within the specified time interval, the call shall be forwarded. The forwarding notification is also optionally available.

Stage 1 description of all the service variants specifies the external service behaviour with a set of service procedures. While activation, deactivation and interrogation of the service are given by user control procedures, the actual service behaviour during its execution is prescribed by an invocation and operation procedure. In this paper, we present in detail the *activation procedure* for the CFU service since its functionality is not so complex that it would require extensive SDL specifications.

If the served user has subscribed to the CFU service, he/she shall use the activation procedure [3] before the actual use of the service. For service activation the served user shall supply:

- the number to which the call is forwarded,
- information whether all calls, or calls of a specified basic service, should be forwarded,
- if used, the multiple subscriber number.

When the served user activates the service, the service provider shall return notification of acceptance, or rejection of the request. In our example we represent a simplified activation procedure without the forwarding notification and with simple tone indications of system responses.

In SDL, we specify the complete interaction between the system (`ISDN_system`) and the served user (`USER_B`) during the service activation by the SDL object system `CFU_Activation`. As depicted in Figure 3, it consists of two blocks:

- The block `ISDN_system` describes the part of the interaction concerning system requests and answers to the served user. It contains the process `Main_process` whose functionality is to receive the `OFF_Hook` signal from the served user, to send the user the `L_ToneDial` signal and to wait for the next user action.
- The block `USERS` specifies user requests and answers to the system. The process `USER_B` inside the block first sends the `OFF_Hook` signal to the system and waits for the `L_ToneDial` signal. After its reception it may either initiate the `ON_Hook` signal or send the activation request `send(prefix_OK,number_OK)` with the prefix for the CFU activation and the forwarded-to number. If the served user is subscribed to the service with the selected prefix and the selected forwarded-to number is valid and not reserved for special service codes, the service may be activated. The activation is confirmed by the `L_ToneConf` signal from `Main_process`. In the case of an unsuccessful activation, `USER_B` receives from `Main_process` the `L_ToneSIT` signal.

The blocks communicate over the channel `to_from_USER`. We describe the activation procedure with an SDL specification in the following textual form:

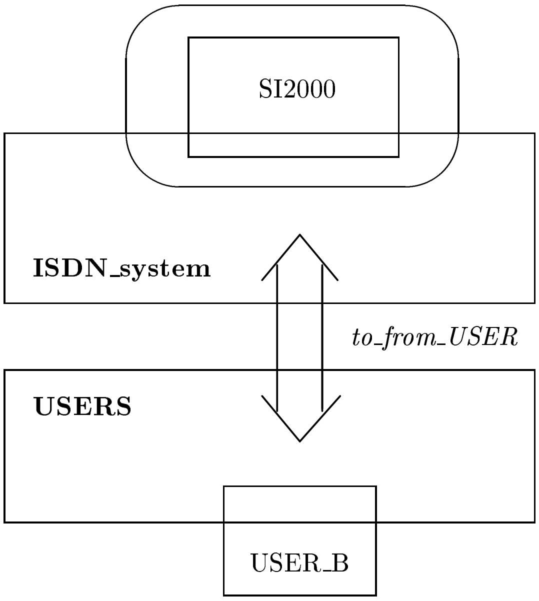

Figure 3: Interaction between the system and users in testing CFU activation

```
SYSTEM CFU_Activation;
SIGNAL
   OFF_Hook,
   ON_Hook,
   L_ToneDial,
   L_ToneConf,
   L_ToneSIT,
   Out_Sign,
   Send(BOOLEAN,BOOLEAN);
  CHANNEL to_from_USER
    FROM ISDN_system TO USERS WITH
         L_ToneDial, L_ToneSIT, L_ToneConf;
    FROM USERS TO ISDN_system WITH
         OFF_Hook, ON_Hook, Send;
  ENDCHANNEL to_from_USER;
  CHANNEL from_env
    FROM ENV TO USERS WITH Out_Sign;
  ENDCHANNEL from_env;
  BLOCK ISDN_system;
    SIGNALROUTE Check_Signal
      FROM Main_process TO ENV WITH
           L_ToneDial, L_ToneConf, L_ToneSIT;
      FROM ENV TO Main_process WITH
           OFF_Hook, ON_Hook;
    SIGNALROUTE Main_route
      FROM ENV TO Main_process WITH Send;
    CONNECT to_from_USER AND Check_Signal,
                             Main_route;
    PROCESS Main_process(1,1);
DCL prefix_OK BOOLEAN,
    number_OK BOOLEAN;
      START ;
        NEXTSTATE Idle_state;
      STATE Wait_state;
        INPUT ON_Hook;
          STOP;
        INPUT Send(prefix_OK, number_OK);
          DECISION prefix_OK;
            (True) :
              DECISION number_OK;
                (True) :
                  OUTPUT L_ToneConf;
                  NEXTSTATE Wait_state;
                (False) :
                  OUTPUT L_ToneSIT;
                  NEXTSTATE Wait_state;
              ENDDECISION;
            (False) :
              OUTPUT L_ToneSIT;
              NEXTSTATE Wait_state;
          ENDDECISION;
      ENDSTATE;
      STATE Idle_state;
        INPUT OFF_Hook;
          OUTPUT L_ToneDial;
          NEXTSTATE Wait_state;
      ENDSTATE;
    ENDPROCESS Main_process;
  ENDBLOCK ISDN_system;
  BLOCK USERS;
    SIGNALROUTE tofrom_B
      FROM USER_B TO ENV WITH OFF_Hook,
           ON_Hook, Send;
      FROM ENV TO USER_B WITH L_ToneDial,
           L_ToneSIT, L_ToneConf;
    SIGNALROUTE from_env
      FROM ENV TO USER_B WITH Out_Sign;
    CONNECT from_env AND from_env;
    CONNECT to_from_USER AND tofrom_B;
    PROCESS USER_B(1,1);
DCL prefix BOOLEAN,
    number BOOLEAN;
      START ;
        NEXTSTATE ON_Hook_state;
      STATE Wait_prefix_state;
        INPUT L_ToneSIT;
          OUTPUT ON_Hook;
          STOP;
        INPUT L_ToneConf;
          OUTPUT ON_Hook;
          STOP;
      ENDSTATE;
      STATE ON_Hook_state;
        INPUT Out_sign;
          OUTPUT OFF_Hook;
          NEXTSTATE OFF_Hook_state;
      ENDSTATE;
      STATE OFF_Hook_state;
```

```
      INPUT L_ToneDial;
        TASK prefix:=True, number:=True;
        OUTPUT Send(prefix, number);
        NEXTSTATE Wait_prefix_state;
    ENDSTATE;
  ENDPROCESS USER_B;
 ENDBLOCK USERS;
ENDSYSTEM CF_Activation;
```

The processes involved in the SDL specification are first abstracted to the corresponding FSMs. In Table 1 we present the FSM of `Main_process`. The FSM of `USER_B` is not presented, since for generation of test scenarios it suffices to use only the specification of the interaction from the side of the system.

The FSM representing `Main_process` after the abstraction is not in the final form for test scenario generation. Due to the single process in the block `ISDN_system`, composition of pocesses is not needed, but the FSM is still nondeterministic. Despite the possibility of also generating test scenarios for nondeterministic cases, it is more convenient to make the FSM deterministic by more detailed specification of circumstances, where the scenarios are going to be executed, and by selection of parameter values.

Now let test scenarios be generated only for the case of a successful CFU activation, and for the values of `prefix_OK=true` and `number_OK=true`. Under such circumstances, the last transition in Table 1 may be omitted and the parameters in the third transition set to `true`. The resulting final FSM which is used as an input to test sequence derivation is shown in Table 2.

Test sequences in this example are generated by the algorithm of Aho, Dabhura, Lee, Uyar [1], assuming completeness of the FSM, a reliable reset capability and an upper bound on the length of UIO sequences equal to 3 [15]. The following test sequences were constructed:

Test sequence 1:

```
Idle_state
Send_prefix(true,true) NULL
Idle_state
reset NULL
```

Test sequence 2:

```
Idle_state
OFF_Hook L_ToneDial
Wait_state
OFF_Hook NULL
Wait_state
reset NULL
```

Test sequence 3:

```
Idle_state
OFF_Hook L_ToneDial
Wait_state
Send_prefix(true,true) L_ToneConf
Wait_state
reset NULL
```

Test sequence 4:

```
Idle_state
OFF_Hook L_ToneDial
Wait_state
ON_Hook NULL
Idle_state
OFF_Hook L_ToneDial
Wait_state
Send_prefix(true,true) L_ToneConf
Wait_state
Send_prefix(true,true) L_ToneConf
Wait_state
reset NULL
Idle_state
reset NULL
```

The generated test sequences were also translated into the following TTCN form:

Tree 1:

```
{
  ! NULL
  { Idle_state
    ? reset
    {
      ! NULL Pass
      ! OTHERWISE Fail
    }
    ? OTHERWISE Inconc
  }
  ! OTHERWISE Fail
}
```

Tree 2:

```
{ Wait_state
  ? reset
  {
    ! NULL Pass
    ! OTHERWISE Fail
  }
  ? OTHERWISE Inconc
}
```

Tree3:

```
{ Wait_state
  ? ON_Hook
  {
```

Table 1: The FSM representing `Main_process`

state	*input*	*output*	*nextstate*
Idle_state	OFF_Hook	L_ToneDial	Wait_state
Wait_state	ON_Hook	NULL	Idle_state
Wait_state	Send(prefix_OK,number_OK)	L_ToneConf	Wait_state
Wait_state	Send(prefix_OK,number_OK)	L_ToneSIT	Wait_state

Table 2: The final FSM

state	*input*	*output*	*nextstate*
Idle_state	OFF_Hook	L_ToneDial	Wait_state
Wait_state	ON_Hook	NULL	Idle_state
Wait_state	Send(true,true)	L_ToneConf	Wait_state

```
  ! NULL
  { Idle_state
    ? OFF_Hook
    {
      ! L_ToneDial
      { Wait_state
        ? Send_prefix(true,true)
        {
          ! L_ToneConf
          { Wait_state
            ? Send_prefix(true,true)
            {
              ! L_ToneConf
              { Wait_state
                ? reset
                  + Tree1
                ? OTHERWISE Inconc
              }
              ! OTHERWISE Fail
            }
            ? OTHERWISE Inconc
          }
          ! OTHERWISE Fail
        }
        ? OTHERWISE Inconc
      }
      ! OTHERWISE Fail
    }
    ? OTHERWISE Inconc
  }
  ! OTHERWISE Fail
}
? Send_prefix(true,true)
{
  ! L_ToneConf
    + Tree2
  ! OTHERWISE Fail
}
? OFF_Hook
{
  ! NULL
    + Tree2
  ! OTHERWISE Fail
}
? OTHERWISE Inconc
}
```

Tree 4:

```
{ Idle_state
  ? Send_prefix(true,true)
    + Tree1
  ? OFF_Hook
  {
    ! L_ToneDial
      + Tree3
    ! OTHERWISE Fail
  }
  ? OTHERWISE Inconc
}
```

Since only one user is involved in the interaction with the system, distribution of the generated test sequences among participating users and introduction of synchronization messages are not needed in this example. On execution of the test scenarios on the real system, the assumed circumstances have to be provided by service data management functions (reliable reset capability, settings of the subscription data, etc.).

4 Conclusion

We suggested an approach to automated generation of test scenarios for ISDN services. In the paper, much attention was given to presenting our experiences in using formal techniques in the development and the use of the tool. The use of selected formal techniques was illustrated by an example of generating test scenarios for CFU (Call Forwarding Unconditional) service activation.

In our future work, we plan to extend the use of the tool to a considerably wider set of SDL constructs and their combinations, including the construct subset which Iskratel specifies for its developers as relevant. The procedure of selecting testing circumstances and parameter values should be further automated to decrease the degree of user knowledge required of an SDL specification contents. In addition, we shall extend the presented approach by adding the possibility of compositional grey-box testing. Here methods can be applied which are based on pretested correct implementation of individual embedded service components.

Acknowledgements

I would like to express my gratitude to the project participants M. Kapus – Kolar, V. Avbelj, R. Verlič and B. Slivnik, who have contributed to the tool in an essential way. I am also grateful to M. Stojsavljevič, D. Kodrič and R. Magdalenič from Iskratel for comments and provision of a platform for test scenario execution on the real system.

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, pp. 1604 – 1615, 1991.

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

[3] ETSI, *Integrated Services Digital Network (ISDN); Call Forwarding Unconditional (CFU) supplementary service, Call Forwarding Busy (CFB) supplementary service, Call Forwarding No Reply (CFNR) supplementary service* – Service descriptions, 1992.

[4] R. Heinrich, F. Pospiech, "Control flow relevance analysis – a new approach for avoiding the state space explosion problem", *Inform. Inf. Rep.* (Germany), No. 10, pp. 115 – 123, 1990.

[5] ISO, IS 9646 Part 3: Tree and Tabular Combined Notation (TTCN).

[6] 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, pp. 373 – 381, 1994.

[7] M. Kapus – Kolar, "A protocol derivation tool for services supporting mobile users", *Proc. of the 4th Electrotechnical and Computer Science Conf. ERK'95*, Portorož, Slovenia, 1995, Slovenia Section IEEE, pp. 51 – 54, 1995.

[8] M. Kapus – Kolar, "Test-sequence generator for finite state machines", in M. Pučko et al., *Generation of test scenarios*, Technical report, Iskratel, Telecommunication Systems Ltd., 1996.

[9] G. Luo, A. Das, G. v. Bochmann, "Software testing based on SDL specifications with Save", *IEEE Trans. on Software Engineering*, Vol. 20, No. 1, pp. 72 – 87, 1994.

[10] G. Luo, A. Das, G. v. Bochmann, "Generating tests for control portion of SDL specifications", in O. Rafiq (Ed.), *Protocol Test Systems, VI (C-19)*, Elsevier Science B.V. (North-Holland), IFIP, 1994.

[11] G. Luo, A. Petrenko, G. v. Bochmann, *Selecting Test Sequences for Partially-Specified Nondeterministic Finite State Machines*, Technical report, Département d'IRO, Université de Montréal, Canada, 1994.

[12] G. Németh, "The missing link in software specification", *Proc. of the ConTEL 95 Conf.*, Zagreb, Croatia, pp. 167 – 178, 1995.

[13] K. Pasinović, K. Pogačnik, *Call Forwarding*, Requirements Specification (in Slovene), Iskratel, 1994.

[14] M. Pučko, "Automated development of Intelligent Network services", *Proc. of the 21st EUROMICRO Conf.*, Como, Italy, 1995, IEEE Computer Society Press, Los Alamitos, California, pp. 387 – 394, 1995.

[15] M. Pučko, M. Kapus - Kolar, V. Avbelj, R. Verlič, B. Slivnik, *Automation of test scenario generation procedures for SI2000*, Technical report (in Slovene), Iskratel, Telecommunication Systems Ltd., 1996.

[16] A. Robnik, "Experience of using SDL in Iskratel", in Z. Brezočnik, T. Kapus (Eds.), *Proc. of the COST 247 Int. Workshop on Applied Formal Methods in System Design*, Maribor, Slovenia, June 1996, University of Maribor, Slovenia, 1996.

[17] H. Ural, A. Williams, "Test generation by exposing control and data dependencies within system specifications in SDL", *IFIP Trans. C, Commun. Syst.*, Vol. C-22, pp. 335 – 350, 1994.

[18] M. Yao, A. Petrenko, G. v. Bochmann, "Conformance testing of protocol machines without reset", in A. Danthine, G. Leduc, P. Wolper (Eds.), *Protocol Specification, Testing and Verification XIII*, Elsevier Science Publishers, pp. 241 – 256, 1993.