Elektrotehniški vestnik 72(1): 14–21, 2005
Electrotechnical Review, Ljubljana, Slovenija

Verification of an SDL Specification — a Case Study

Boštjan Vlaovič, Aleksander Vreže, Zmago Brezočnik, Tatjana Kapus

University of Maribor, Faculty of Electrical Engineering and Computer Science,
Smetanova ulica 17, 2000 Maribor, Slovenia
e-mail: {*bostjan.vlaovic, aleksander.vreze, brezocnik, kapus*}*@uni-mb.si*

Abstract. This paper presents practical experience gained by an attempt to mechanically extract a model of the Inres service with the *go-back-n* extension and verify it with the use of simulation and formal verification based on the model checking technique. The service specification is written in the Specification and Description Language (SDL). The model is obtained mechanically with the application of the `sdl2if` and `if2pml` tools. Transformation to discrete-time Promela is followed by simulation and an example of formal verification of a property described in Linear Temporal Logic (LTL) with the Spin model checker. Description of each step of the process is followed with the noticed shortcomings of the procedure, limitations of the tools and discovered faults in the specification of the service. We conclude with a discussion of further research activities and some results from the domain of mechanical extraction of models from SDL system specifications.

Key words: Simulation, Formal Verification, Model Checking, SDL, Spin, Promela, IF

Primer verifikacije specifikacije v jeziku SDL

Povzetek. Prispevek predstavlja praktične izkušnje, ki smo jih pridobili ob poskusu avtomatske tvorbe modela storitve Inres s protokolom "vrni se za N" ter preverjanjem pravilnosti delovanja z uporabo simulacije in formalne verifikacije na osnovi tehnike preverjanja modelov. Specifikacija sistema je zapisana v opisnem jeziku SDL. Model sistema je pridobljen avtomatsko z uporabo programskih orodij `sdl2if` in `if2pml`. Pridobljeni model je opisan v jeziku Promela z razširitvijo za podporo diskretnega časa. Tvorbi modela sistema sledi opis simulacije in primer formalne verifikacije lastnosti, ki je zapisana z linearno temporalno logiko. Uporabljeno je orodje za preverjanje modelov Spin z razširitvijo za podporo diskretnega časa. Ob opisu vsakega koraka so izpostavljene opažene pomanjkljivosti postopka, omejitve orodij ter odkrite napake v specifikaciji storitve. Zaključimo s predstavitvijo nadaljnjega raziskovalnega dela in nekaterimi rezultati s področja avtomatske tvorbe modelov iz specifikacij sistemov v jeziku SDL.

Ključne besede: simulacija, formalna verifikacija, preverjanje modelov, SDL, Spin, Promela, IF

1 Introduction

The main objective during the development of computer systems is to reduce reliance on human intuition and judgment in evaluating arguments. Formal methods refer to the use of logic and discrete mathematics during the development process. They are not a guarantee of a superior product. Realistic expectations are a function of the designated role and extent of formal methods use and of project resources allocated to the formal methods activity. Skillful application of formal methods can detect faults that would be hard to discover with the test methods that are still prevailing in most projects.

In this paper, only a small sub-field of the general area that is covered by formal methods will be used: the application of automated verification techniques referred to as model checking. In this approach, specifications are expressed in a propositional temporal logic, and design under consideration is modelled as a state-transition system. An efficient search procedure is used to determine if the specification is true for the transition system. In other words, the transition system is checked to see whether it is a model of the specification [1].

This paper presents a practical experience gained by an attempt to mechanically apply model checking method to the specification of the Inres service with the *go-back-n* extension. It was written in SDL (Specification and Description Language), converted to intermediate format (IF), which was converted to Promela (Protocol/Process Meta Language), and checked by simulation and formal verification.

This paper is organized as follows. Section 2 defines the problem. In section 3, introduction to model checking is presented. Section 4 describes specification of the protocol in SDL. Next, the model extraction technique is shown. Section 6 discusses the simulation and verification of the extracted model with the Spin model checking tool. Section 7 summarises findings of the case study. We conclude with a discussion of further research activities

Received 30 January 2004
Accepted 14 February 2005

and some results from the domain of mechanical extraction of models from SDL system specifications.

2 Problem Definition

The main objective of the research was to test current state-of-the-art tools and methods for mechanical formal verification of telecommunication protocols developed in SDL. In order to eliminate knowledge of the specification in advance, we decided to verify a specification which has been written by a third party.

The optimal solution would consist of black box approach based on the system specification and strong prohibition of any alteration of the original SDL code. As we will present in the paper, this was not possible due to the limitations of the tools and particularities of the specification. Additional research activities which would provide a better solution will be discussed at the end of each section.

3 Model Checking

Traditional model checking technique developed by Ed Clarke and Joseph Sifakis is a two-pass verification process. In the first pass, the reachable state space of a concurrent system is computed and the essential features of it are encoded into a data-structure that is stored in a computer memory. In the second pass, the validity of a correctness requirement, formalized in a temporal logic formula, is verified for the given global state space structure [1].

The other approach is described as on-the-fly verification. It has its roots in the work of Colin West. The main objective of this method is to prove every property of interest for a concurrent system in a single pass of the state space exploration algorithm. To our knowledge, the first efficient implementation of such a system was Spin by Gerard Holzmann [2].

Model checking method enjoys two remarkable advantages. First, it is fully automatic, and its application requires no user supervision or expertise in mathematical disciplines such as logic or theorem proving. Second, when the design fails to satisfy a desired property, the process of model checking can produce a counterexample that demonstrates a behaviour which falsifies the property. This fault trace provides insight to understanding the real reason for the failure [1].

The main disadvantage of model checking is state explosion which can occur if the system being verified has many components that can make transitions in parallel. The number of global system states may grow exponentially with the number of processes. In the case of asynchronous protocols, it is possible to decrease the size of the state space by the use of techniques such as partial order reduction.

4 Specification

The object of our study is an SDL specification of the Inres service. A similar system was used in [3] as an introduction to the OSI (Open Systems Interconnection) concepts and system description with SDL. Specification under study is described in [4]. It extends original service with the *go-back-n* functionality. The specification was written in Telelogic's ObjectGEODE, which is used in our research work.

4.1 SDL

SDL is standardized by ITU (International Telecommunication Union). It is based on Extended Communicating Finite State Machines (ECFSM), but it uses graphical representation of flowcharts to show allowed transitions. In the development cycle, SDL is employed for the formal specification and design of the system. It supports specification and description of structural and behavioral aspects of the application under development.

SDL may serve a number of purposes, from reasoning about systems at an abstract level to the automatic derivation of implementations. Nowadays, several commercial and academic tools are available that support the development of systems with SDL. Tool support comprises graphical editing, validation, verification, simulation, animation, code generation, and testing.

4.2 Inres service

The Inres service is offered to the user in the environment (Figure 1). Users of the system communicate with the Inres protocol. It is connection-oriented. The user who wishes to communicate with another user via the service must first initiate a connection before exchanging data (Figure 2). For simplification purposes, the service is not symmetrical. The user A can initiate a connection and later send data. User B can accept the connection or reject it. After acceptance it can receive data from the initiating user until it decides to terminate the connection.

The hierarchic structure of the SDL system and its environment is shown in Figure 1. It is composed of three layers. The Inres service encompasses layer 2 of the system within blocks `Ini_Station` and `Res_Station`. Due to the difference between interfaces of the first and the third layer recoding of the messages is required. Coding and decoding is performed by processes `Coder_Ini` and `Coder_Resp`. The core functionality of the Inres service is described in the processes `Initiator` and `Responder`. The physical layer is described within the `Medium` block. Processes `MSAP_Man1` and `MSAP_Man2` offer lossy transmission of messages.

The original specification of the Inres protocol included simple repetition of the lost messages in layer 2.

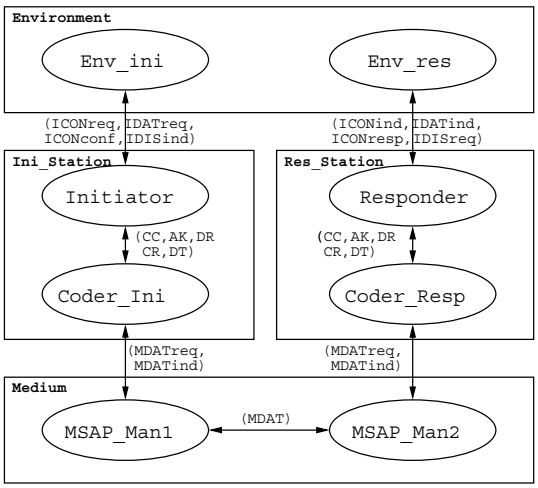

Figure 1. SDL system description

After four unsuccessful attempts connection was terminated. The presented SDL description supplements this simple functionality with the *go-back-n* protocol, but only for the transmission of the `IDATreq` signal which carries user's data frame. It is specified in the processes `Initiator` and `Responder`. Window length of the *go-back-n* protocol is three.

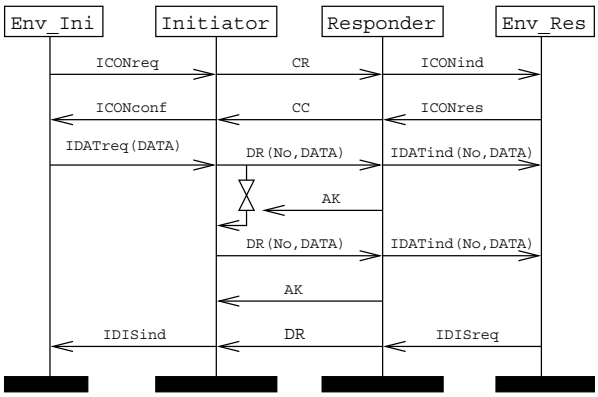

Figure 2. Message flow of the Inres protocol.

5 Model Extraction

Conversion of the system design to a formalism accepted by the model checking tool is of crucial importance. A model can be obtained from the system specification or implementation. For the purpose of this article, we are interested only in the former. The best results can be obtained by the automatic model extraction – given that a transformation method does not change the system design.

Usually, a model of the system is extracted by an expert for the formal verification tool that is used. This procedure is heavily dependent on the verificator and his or her skills. We tried to mechanically extract the model of the specification with two tools: `sdl2if` and `if2pml`. The first tool converts an SDL description of the system to the intermediate format. Next, conversion to Promela is performed by the `if2pml` tool.

5.1 Intermediate Format

IF can be considered as a common representation model for other existing languages such as Promela or for a combination of languages adopting different description styles.

`sdl2if` is a translator of SDL specifications into the IF intermediate representation. It relies on an API of the `ObjectGEODE` (Telelogic) and is a product of VERIMAG (public research lab) [5].

The translation pointed out the inexactness of the specification. In SDL, each process can have initial and maximal number of instances. If no explicit limits are defined, it is implicitly assumed that the initial number has a value 1 and the maximal number is unbounded. Therefore, the translation would not be rigorous if it implicitly assumed other values. In our example, the correct assumption for both values would be 1, in some other cases it could be something else. The translator writes a warning to the log file, but handles this inexactness to our favor. In spite of that, we would suggest that this warning is also printed to the standard output. Before the actual translation ObjectGEODE's SDL checker is used to verify compliance of the current SDL system with the selected version of the standard. Its output is printed to the standard output. It can be misleading. A process declaration without explicit limits has completely valid SDL syntax. Therefore, the SDL checker does not produce any errors or warnings.

According to [5], `sdl2if` should translate each local variable in an SDL process to a local variable of the corresponding IF process. SDL predefined data types Boolean, Integer, Real, Natural, PId, Time, and Duration have corresponding data types in IF. Data type Character is translated into an IF user defined type range $[1..256]$. We noticed that variables of the predefined type `charstring` were not translated. A confirmation of our "discovery" was found in [6]. Data type `charstring` covers characters of any length enclosed within apostrophes, e.g. "Hello World". Currently it is translated to an abstract data type. The treated SDL specification included only one such variable definition and it could be easily changed.

An acknowledged limitation of the `sdl2if` is its lack of full support for the enabling condition. It is only translated if the condition does not involve parameters of the input signal.

5.2 Promela

Protocol/Process Meta Language is a modelling language which allows dynamic creation of concurrent processes. It is used by the Spin model checker. The description of a concurrent system consists of one or more user-defined process templates or proctype definitions and at least one process instantiation.

Process templates are used to define a finite automaton of the system. Next, computation of asynchronous interleaving product of automata gives global behaviour of the system (state-space of the system, reachability graph). It describes a finite systems, which implies:

- no unbounded data,
- no unbounded channels,
- no unbounded processes,
- no unbounded process creation.

Real-time properties of the specifications cannot be expressed in standard Promela and Spin in a quantitative manner. Consequently, the IF specifications are actually translated into DT Promela, an extension of Promela, with discrete-time features (timers and operations on timers) [7].

The obtained DT Promela models can be simulated and verified with DT Spin, an extension of the Spin model-checker with discrete time that has been developed within the Vires project [8]. It is currently based on version 3.3.10 of Spin from the year 2000. It seems that the project is frozen. Current version of the mainstream version of Spin is 4.0.6.

Next, system description was translated to Promela by `if2pml`. Its acknowledged limitations are [9]:

- save operations are not implemented,
- enabling conditions are not translated.

During our research we ran into additional limitations. Structures are not supported in assignments. Promela does support structures, but each element of the structure has to be individually assigned a value. It is not possible to assign values to all of the elements in one "structure assignment". It could be realized with the introduction of a special channel [10], but `if2pml` does not handle this requirement correctly. Translated SDL specifications are therefore invalid if they use structures in assignments. We solved this problem with the expansion of the structures. Extension of Promela with support for the transparent use of structures would greatly simplify the process of transformation. Experiences from the industrial projects show that most of the communication in SDL is performed via signals which include variables of the type `struct`.

During the modification of the environment model we discovered an improper transformation of the SDL construct `input` when an asterisk ("`*`") is used as a signal descriptor. It is interpreted in the same way as `any` descriptor for a `decision` construct, which is not in accordance with the SDL standard. Asterisk implies that all signals that have not been specified explicitly for that state in an `input` or `save` construct are accepted. Next, some minor problems concerning correct `timer` and `reset` statements were found and corrected.

6 Simulation and Verification

Successful translation was followed by the simulation and verification of the system behaviour with the Spin model checker.

6.1 Spin

Spin is a tool for analyzing logical correctness of concurrent systems, specifically of data communication protocols [11]. Its first version under the name Pan appeared in 1980. Currently, in its 4^{th} version, it is a mature project with a lot of users and contributors.

Spin's wide support from the research community is due to its free availability. It supports efficient model checking, invariant assertions, and temporal properties expressed in a subset of Linear Temporal Logic (LTL). Promela adopts strong formal basis established, like SDL, in ECFSM theory. The similar basis of SDL and Promela makes the translation between different representations feasible. Our future research activities will focus on direct translation between the two.

Given the model of the system specified in Promela, Spin can perform random, interactive, or guided simulation of the system executions. Further, it can generate a C program which performs online verification of the system's correctness properties:

- safety properties,
- liveness properties,
- general temporal properties.

It checks for the absence of deadlocks, unspecified receptions, unexecutable code, and it can find non-progress execution cycles.

6.2 Model of the Environment

Simulation and verification of protocols require modelling of the environment. The environment is often unpredictable, and its effective modelling requires the use of non-deterministic choices. Based on the assumption that a developer has better knowledge of SDL than Promela, the environment was described in SDL. The environment had to satisfy the requirements for the simulation and verification of safety and liveness properties. To prove liveness properties we had to add two environment processes: the first to submit data to be transferred, and the second to check that the data accepted by the receiver match the data that were submitted by the sender.

The original system description already included unpredictable message transfer in the physical layer. For each message a non-deterministic choice was made to forward or ignore the message. It was treated as an existing environment model that was included in the specification because of the system verification by the ObjectGEODE tools [4].

6.3 Simulation

During the random simulation several inconsistencies about the behavior of the system and some additional limitations of the `if2pml` tool were discovered. In the *go-back-n* protocol, no more than a maximum number of unacknowledged frames should be outstanding at any time. In order to enforce the flow control, layer 2 should be able to block the sender. The informal specification [4] poses a requirement on the Initiator to store packets when transmission window is full, but does not elaborate on it in any detail. The specification blocks the sender with the enabling condition construct. When the enabling condition is not fulfilled, signal should be saved. Unsupported save construct and enabling condition are known limitations of the `if2pml` translator. It should be noted that different interpretations of the save construct are possible also in IF. Implementation of the `sdl2if` translator handles that correctly. It would be recommended to warn the validator on the occurrence of the enabling condition. Its current transformation greatly influences the behaviour of the system. Figure 3 demonstrates the transformation from SDL to IF and Promela.

SDL:

```
STATE Connected;
INPUT IDATreq(B_ISDU);
 PROVIDED V_queue<2;
NEXTSTATE Sending;
```

IF:

```
connected
 save
  idatreq in q_initiator_i0 \
      if not (v_queue < 2);
end;
```

Promela:

```
connected:
if
:: ((v_queue<2)) == true ->
    q_initiator_i0?\
    idatreq(sender,b_isdu) ->
    v_d.v_isdu = b_isdu ->
    goto con25;
fi
```

Figure 3. Transformation of the enabling condition from SDL to IF and Promela

It shows that the description in Promela is not properly formed. The transition is enabled as long as variable `v_queue` is smaller than 2. When signal `idatreq` is not in the input queue, execution of the process could be blocked until its reception. Reception of the acknowledgements is unintentionally blocked. This results in a deadlock when the input buffer becomes full. Description of the SDL system and environment was altered to solve this problem. A better solution would be to properly translate the enabling condition construct, but this is outside the scope of this paper.

During a simulation of the *go-back-n* protocol we noticed that after four unsuccessful retransmissions of the signal, retransmissions are stopped and a disconnection is requested [4]. The specification follows the requirements.

Next, a wrong specification of the retransmission procedure was found. Due to the miss-configured index of the last outstanding packet, incorrect retransmission was performed. Part of the original MSC generated by the `xdtspin` is shown in Figure 4. The first retransmission is regular. During the second retransmission an unspecified or old value in the first "empty" slot in the sending window is sent to the receiver.

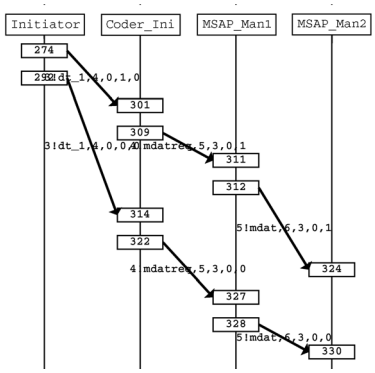

Figure 4. Double-retransmission procedure

We could not afford to ignore these discoveries, so SDL code modification was necessary. Modifications included elimination of the maximum number of retransmissions and retransmission procedure corrections. The first correction would require change of the functional specification and is questionable. Our intention was to verify the functionality of the *go-back-n* protocol, so we decided to make the change.

The protocol specification also included transmission of the acknowledgement frames for the retransmitted frames. A simulation run of the new model discovered a case where an acknowledgement was not sent by the process `Responder`. Visual inspection of the SDL description discovered a wrong specification of the acknowledgement transmission. After the correction we did not find any additional errors with the following simulation runs.

6.4 Safety

The safety property capabilities of Spin check assertions and invalid end states. The assert statement can be used to formalize system invariants, i.e., boolean conditions that are required to be invariantly true in all reachable system states. We did not define any invariants. Focus of the safety properties check was on the invalid end-states.

Spin supports exhaustive, supertrace, and hash-compact search modes. Exhaustive mode represents a full state space exploration. This mode is always preferable, but is not always possible, because all unique states must be stored in memory. Our system ran out of memory during the exhaustive verification. Much research has been directed to reduce the state size by storing states in a compact form [12]. Spin's supertrace mode involves a controlled partial search. The technique is based on hashing with collision detection. Spin provides an estimation of the state coverage with a hash factor. It is a ratio between the number of the available bits and reached states. For each system state only one bit of memory is used. When using this method one should check if the coverage was adequate. If not, reports of the unreachable code can be inaccurate and some errors might be missed. Reasonably good results can be obtained with the hash factor of 100 or greater which represents coverage greater than 99%. Hash-compact search mode collapses state vector sizes down using a version of the Wolper's hash-compact method [13]. Additionally, several reduction techniques and compressions are available.

We will present only the results of the supertrace search and partial order reduction algorithm. Partial order reduction exploits the independence of concurrently executed events. Most of the activities in concurrent software are performed independently, without a global synchronizing clock.

Some particularities of the XSpin environment were found. Most notable is a wrong interpretation of the specified advanced options. A good estimation of the state space size is crucial for the good coverage of the supertrace search algorithm. XSpin accepts estimated number in a decimal representation of the number of states. The closest power of two is taken by XSpin to use it during the verification run. This calculation does not check if the memory limit is exceeded, so additional care is needed. In some cases estimated number was not correct. It is recommended to check the used value in the command line of the verifier. If the calculation is not correct, direct usage of the verifier can solve the problem.

There were no invalid end-states found. Since none of the states were assigned predicate of the valid end-state, this confirms that system is dead-lock free. After we modified the "interminable" model of the environment to include request for the disconnection, the system terminated and invalid end-states were reported. Proper labeling of the valid end-states produced a verification run without any error.

An example of unreachable code report was performed with the "interminable" model of the environment. For each process, report of an unexecutable code is provided as a result of the state-space exploration. For example, in the process `Responder` four unexecutable statements were found. Three of them were caused by the lack of the disconnection request from the environment. The fourth statement exposed that the process expects reception of the signal `ICONreq` in the state `Connected`. Verification showed that this cannot occur with the specified model of the environment and that it could be safely removed from the system description. Importance of a proper model of the environment is evident.

6.5 Liveness

The input data sequence of our environment consists of three different data items. We can imagine that each data item has a different colour. Our model of the environment send one red and one blue data item inserted randomly in an infinite sequence of white data items [14, 15].

To monitor the behaviour of the system we supplemented the model of the system with probes, which are global monitor variables. For each data item send and receive probes were defined (sr = send red, rr = receive red, etc.). While adding the probes to the model, one should be very careful and take into account the concurrent execution of processes. Therefore, send and receive statements should be indivisible with the assignment of the probes. During the conversion, `if2pml` defines process transitions from one stable state to another as atomic actions. This transformation is based on the definition of SDL. Probes have to be added manually by the verifier each time a new version of the model is extracted. We believe probe insertion can be done mechanically based on the LTL formula.

As an example of formal verification of liveness properties we checked that the red data item cannot disappear from the data sequence with the following LTL formula: `[](sr -> <> rr)`. The formula claims that always, if the red data item is sent by the sender, it is eventually received at the receiver.

The verification disclosed where the specified method of acknowledgement reception in the process `Initiator` would fail. Inspection of the resulting counterexample simulation run revealed that the acknowledgement number was not correctly checked. This resulted in unintentional acknowledgements of frames that were lost. Correction was checked with another verification run which did not produce any errors.

7 Findings

The case study showed that there are many things that influence the quality of the verification. Formal verification of the SDL system specification with DT Spin consists of many steps. Each of them has its own particularities which the verifier should be aware of. Our intention was to present our experience and discoveries during the research activities. We managed to reveal some interesting and erroneous system behaviors which were not discovered by the original author. Some of them were revealed already during the simulation runs. More subtle errors, like checking of the acknowledgement number, were found only with the verification of the LTL-expressed property during the verification of liveness.

8 Further Research

Based on the experience gathered during this case study, our research group decided to expand research activities to the domain of mechanical extraction of models from SDL system specifications. We decided to use formal verification tool Spin and Promela language for the description of the models. Since the manual model preparation process is prone to incorrect modelling of system properties and requires highly skilled professionals, we believe that mechanical extraction of models could promote formal techniques in the industry. In this section we present challenges, research directions, and first results in the automatic generation of models from SDL specifications.

We decided to use Spin due to its support for modelling of external operators written in programming language C and wide acceptance from the research community [16, 17]. Every SDL specification from our industry projects uses such operators. Their mechanical inclusion to the model is our long-term goal.

Each SDL Abstract Data Type (ADT) can contain one or more operators. The body of the operators can be implemented with SDL or external functions written in C language. ObjectGEODE's SDL C code generator provides a predefined interface between the SDL specification and operators that are implemented in C. External files can include complex functions and define new data structures and header files.

Mechanical generation of models with support for the external operators presents a big challenge. Our current approach is divided into two phases. First, exhaustive analysis of the SDL specification and C code is performed. This phase involves analysis of data structures, global variables, function calls, and all external files which are included through the ADT operator mechanism. Next, we build a model of the specification [18].

The complexity of external operators can be unmanageable for the formal verification due to the access to a data base, complex parsing operations or similar. In our further research we will try to mechanically reduce the complexity of external operators with their abstraction in the generated model of the specification [19].

First results of our research are presented in [20], where we address the following issues:

1. modelling of the SDL data types,
2. support for the dynamic creation of processes,
3. modelling of the priority signal,
4. modelling of the implicit transition,
5. modelling of the spontaneous transition,
6. modelling of the save construct,
7. modelling of the priority input,
8. modelling of the enabling condition,
9. modelling of the asterisk input,
10. modelling of direct (PId) and indirect addressing (name of the process, name of the signal route),
11. support for path limitations introduced by the via statement,
12. dynamic monitoring of the associated channel,
13. modelling of the asterisk state,
14. support for timers with parameters,
15. introduction of probes for monitoring of the system behaviour during formal verification of the model.

One of the hardest challenges was modelling of communication. Each process has an associated channel. For proper modelling of the save construct and priority input dynamic monitoring of the associated channel is required. Each process keeps track of its input queue. Evaluation of the signal reception is done by a special skeleton in the body of the process. Signal reception order is defined by the input constructs of the current state of the process. First results, obtained from the automatic generation of a model from an SDL specification in [21], are promising. Unfortunately, further discussion of our work exceeds the scope of this paper.

9 Conclusion

In the first part of this paper we wanted to focus more on the formal verification, but we decided to present the whole experience. We did not want to leave out the discoveries obtained during the transformation procedures and simulation runs which inspired us to focus our research activities in this area. Currently many research

activities are focused on the use of formal methods in an industrial setting. Section 8 describes part of our efforts in the last months. We believe there is still room for an improvement in the algorithms for mechanical extraction of the model, verification tools, and design methodologies that would include formal verification as an equally important part of the development process.

10 References

[1] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. The MIT Press, third printing ed., 2001.

[2] G. J. Holzmann, "Proving the Value of Formal Methods," in *7th Int. Conference on Formal Description Techniques (FORTE94)*, (Bern, Switzerland), 1994.

[3] J. Ellsberger, D. Hogrefe, and A. Sarma, *SDL Formal Object-oriented Language for Communicating Systems*. Prentice Hall Europe, 1997.

[4] B. Lesjak, *Design of a communication protocol using ObjectGEODE (in Slovene)*. Faculty of Electrical Engineering and Computer Science, 2002.

[5] M. Bozga, J. Fernandez, L. Ghirvu, S. Graf, J. Krimm, L. Mounier, and J. Sifakis, "If: An Intermediate Representation for SDL and its Applications," in *Proceedings of SDL-FORUM'99, Montreal, Canada*, June 1999.

[6] M. Bozga, L. Ghirvu, S. Graf, L. Mounier, and J. Sifakis, "The Intermediate Representation IF: Syntax and semantics," tech. rep., Vérimag, Grenoble, 1999.

[7] D. Bošnački, "Extending Promela and Spin with Discrete Time," in *Proceedings of the VIII Conference on Logic and Computer Science*, 1997.

[8] J. Baeten, "Esprit Project 23498 - VIRES (Verifying Industrial Reactive Systems)." URL: <http://www.cordis.lu/esprit/src/23498.htm>.

[9] "if2pml." URL: <http://www.win.tue.nl/ sidorova/Vires/if2pml.html>.

[10] "Promela Reference — typedef." URL: http://spinroot.com/spin/Man/typedef.html.

[11] G. J. Holzmann, *The SPIN model checker: Primer and reference manual*. Addison Wesley, 2003.

[12] J. Geldenhuys and P. de Villiers, "Runtime Efficient State Compaction in SPIN," in *5th SPIN workshop*, July 1999.

[13] P. Wolper and D. Leroy, "Reliable Hashing Without Collision Detection," in *Computer Aided Verification, Proc. 5th Int. Workshop*, vol. 697 of *Lecture Notes in Computer Science*, (Elounda, Crete), pp. 59–70, Springer-Verlag, June 1993.

[14] G. Holzmann, "The model checker SPIN," in *IEEE Transactions on Software Engineering*, vol. 23, pp. 279–295, May 1997.

[15] P. Wolper, "Expressing Interesting Properties of Programs in Propositional Temporal Logic," in *Proc. 13th ACM Symp. on Principles of Programming Languages*, (St. Petersburgh), pp. 184–192, January 1986.

[16] G. J. Holzmann, "From Code to Models," in *Proc. 2nd Int. Conf. on Applications of Concurrency to System Design*, pp. 3–10, 2001.

[17] G. Holzmann, "Logic Verification of ANSI-C Code with SPIN," pp. 131–147, Springer Verlag / LNCS 1885, Sep. 2000.

[18] A. Vreže, *Extending automatic modelling of SDL specifications in Promela with embedded C code and a new model of discrete time*. Work in progress, Faculty of Electrical Engineering and Computer Science, University of Maribor, 2005.

[19] F. Tip, "A survey of program slicing techniques," *Journal of Programming Languages*, vol. 3, pp. 121–189, Sept. 1995.

[20] B. Vlaovič, *Automatic generation of SDL models with probes from the system specification*. Ph.D. thesis, Faculty of Electrical Engineering and Computer Science, University of Maribor, 2004.

[21] L. Doldi, *Validation of Communications Systems with SDL: The Art of Simulation and Reachability Analysis*. John Wiley & Sons, Ltd, 2003.

Boštjan Vlaovič (Member, IEEE, ACM) received his diploma and Ph.D. degrees from the Faculty of Electrical Engineering and Computer Science, University of Maribor, Slovenia, in 1999 and 2004, respectively. He is a teaching assistant at the same faculty. His main research areas are in the field of formal verification. His special interests cover formal protocol verification with model checking, especially mechanical extraction of models from the SDL specification.

Aleksander Vreže (Student Member, IEEE) received diploma degree in Computer Science from the Faculty of Electrical Engineering and Computer Science, University of Maribor, Slovenia, in 2001. He is a Ph.D. candidate at the same faculty and works as a researcher in the field of formal verification. His special interests cover formal protocol verification.

Zmago Brezočnik (Member, IEEE, ACM) received his M.Sc. and Ph.D. degrees from the University of Maribor, Faculty of Electrical Engineering and Computer Science, in 1986 and 1992, respectively. He is professor, head of Laboratory for Microcomputer Systems, and Deputy Dean of Education at the same faculty. His main research areas are formal hardware and protocol verification, especially symbolic model checking, and binary decision diagrams.

Tatjana Kapus (Member, IEEE, ACM) received the M.Sc. and Ph.D. degrees from the Faculty of Electrical Engineering and Computer Science, University of Maribor, Slovenia, in 1991 and 1994, respectively. She is now an associate professor there. Her research interests are in the area of temporal logic, process-algebraic, and other formalisms and tools for specification and verification of reactive systems, such as, for example, communication protocols.