Computer Standards & Interfaces 31 (2009) 779–786

Contents lists available at ScienceDirect

Computer Standards & Interfaces

journal homepage: www.elsevier.com/locate/csi

Sdl2pml — Tool for automated generation of Promela model from SDL specification

Aleksander Vreže *, Boštjan Vlaovič, Zmago Brezočnik

University of Maribor, Faculty of Electrical Engineering and Computer Science, Smetanova ulica 17, SI-2000 Maribor, Slovenia

ARTICLE INFO

Article history:
Received 19 March 2007
Received in revised form 9 July 2008
Accepted 28 September 2008
Available online 25 October 2008

Keywords:
Promela
SDL
IUA
Automated generation of model

ABSTRACT

This paper presents research results from the field of automated generation of verification model from real-life SDL (Specification and Description Language) specification of the system. An award winning model checker Simple Promela Interpreter (Spin) was used for formal verification. Preparing a verification model from real-life specification is a hard task. We implement most of our research results in the tool named *sdl2pml* (SDL to Promela) in order to avoid human errors while building the verification model. This tool is used for automated generation of the model. We present its architecture and compare the implemented features with other existing tools. Additionally, we demonstrate its use on a real-life specification of an IUA (ISDN User Adaptation) protocol which is part of the SI3000 softswitch.

© 2008 Elsevier B.V. All rights reserved.

1. Introduction

Telecommunication systems usually describe very complex behaviour due to their many concurrent activities. Checking the correctness properties of such systems is a difficult task. We believe that formal verification using the model checking technique can help. It can automatically check whether the model of the system is in accordance with the system's required specifications.

SDL (Specification and Description Language) is used for the formal specification of concurrent, reactive, and distributed systems, and is standardized by the ITU (International Telecommunication Union) [15]. At the highest level of an SDL hierarchical specification there is a system object. It is the entry point to the SDL specification. It comprises a set of blocks and channels. The blocks can be connected to each other and with the environment by channels. A block consists of sub-blocks or a set of processes. A process is defined by a process graph which usually consists of several pages of state transitions. The processes describe the system's behaviour. A more detailed description of the SDL can be found in [15]. All real-life specifications from our industrial partners are extending SDL'93 specifications using ADT (Abstract Data Type) operators. These parts of the implementation are written in C programming language.

We chose the Simple Promela Interpreter (Spin) model checker for formal verification. It accepts the Promela (Process Meta-Language) specification language. In 2002, Spin was recognized by the ACM (Association for Computing Machinery) with its most prestigious Software System Award. We chose this tool because it is one of the most widely used logic model checkers in the world. There are two approaches when applying Spin in system design. The first approach is to use the tool to construct verification models that can be shown to have all the required system properties. Once the basic design of a system is shown to be logically sound, it can be implemented with confidence [8]. Unfortunately, the industry usually follows the second approach and starts directly with implementation. Therefore, in our research we focused on the semi-formal specification of a telecommunication system that represents implementation of the system as written in SDL and C.

Preparing a verification model from real-life specification is a difficult task. We implemented most of our research results in the tool named *sdl2pml* (SDL to Promela) in order to avoid human errors while building the verification model. It is used for algorithm-based generation of the model. It is our belief that this approach is the most promising alternative for industry-size specifications when using the implementation code.

Supertrace was the first tool to provide the model checking of an SDL specification [5]. It was an internal AT&T's experimental project that was used in the switch development environment and was later integrated into the Sdlvalid tool [6]. Unfortunately, these tools were available only to AT&T's engineers.

Another known approach that uses automated model generation is described in [2]. Firstly, the SDL specification is transformed into an intermediate format (IF) using a *sdl2if* tool. The main motivation for intermediate representation development is to provide sufficient expressiveness for mapping concepts of diverse modeling languages. It is used for easier interface regarding the various model-based validation tools [3]. It relies on an API of the ObjectGEODE (Telelogic) and is a product of VERIMAG (public research lab). Detailed studies of real-life industrial specifications have shown that this approach lacks support

* Corresponding author. Tel.: +386 2 2207267; fax: +386 2 2207272.
E-mail addresses: aleksander.vreze@uni-mb.si (A. Vreže), bostjan.vlaovic@uni-mb.si (B. Vlaovič), brezocnik@uni-mb.si (Z. Brezočnik).

0920-5489/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.csi.2008.09.005

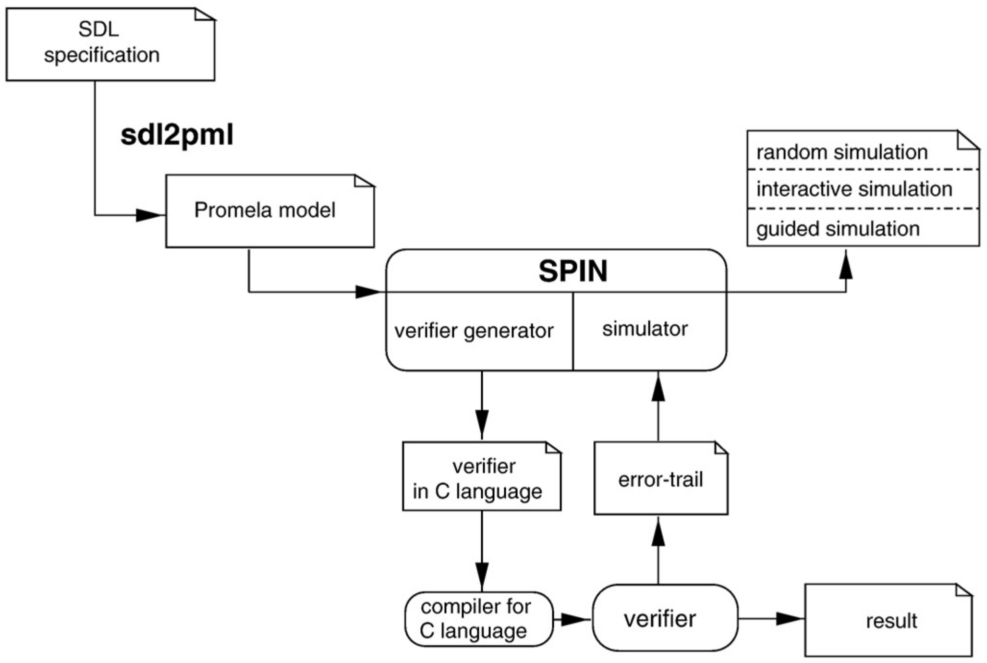

Fig. 1. Formal verification using a Spin tool.

for some of the important SDL features [12]. Most importantly, it does not support:

1. dynamic process creation,
2. communication with more that one process instance,
3. timer constructs with parameters,
4. charstring data type,
5. ADT operators written in C, and
6. has limited support for priority input.

In order to use Spin, the intermediate representation has to be transformed to Promela using the *if2pml* tool. The 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 extended Promela, with discrete-time features (timers and operations on timers). The obtained Promela models can be simulated and verified using DT Spin, a developed extension of the Spin model-checker within the Vires project [1]. Unfortunately, this extension is unavailable for all versions of Spin. There are further additional limitations introduced by the *if2pml*. It does not support:

1. SAVE construct,
2. enabling condition,
3. structure assignment, and
4. asterisk input (it is not in accordance with Z.100).

Based on these findings, we decided to propose a new algorithm-based translation from SDL to Promela. An overview of our approach is presented in [13]. This article focuses on the architecture and selected features of the *sdl2pml* tool. We present its use in the implementation of an IUA (ISDN User Adaptation) protocol from our industrial partners, where a mechanism for semi-automatic abstraction of ADT operators and a new model of discrete time are crucial for automated model generation. Detailed description of the theoretical background and algorithms is outside the scope of this paper.

This paper is organised as follows. Firstly, the overview is given of formal verification using the Spin tool. The design and features of the *sdl2pml* tool are presented in Section 3. Section 4 provides a case study regarding automated generation of the model for IUA protocol. In the Conclusion, we comment on our work and suggest directions for future research.

2. Spin tool

Spin[1] is a freely available open source tool. It can be used for the simulation and formal verification of a concurrent system. Given the system model in Promela, Spin can perform random, interactive, or guided simulation of the system executions. Furthermore, it can generate a verifier in C code, which performs online verification of the system's correctness properties. Spin is implemented in C and is portable across various operating systems, including 64-bit versions of Windows and Linux. Graphical user interface XSpin is provided by Tcl/Tk application.

The final goal of our project was formal verification of the model. Fig. 1 presents the verification process that includes automated model generation. If the formal verification run finds an error, the tool presents a full execution path for that run. Otherwise, it provides brief information about the verification run — number of global states, memory usage, etc. The error-trail can be analysed with the help of guided simulation. A detailed description of the tool can be found in [8].

3. Sdl2pml tool

Different development environments exist for system design and application development using SDL. Each of them extends the standard-based SDL using its own non-formal extensions, e.g., definition of user-defined abstract data type operators written in C. The use of such extensions results in semi-formal system specification. We used Telelogic's ObjectGeode v4.2 for our academic work and GEODE editor v.2.2.4 for development of the Iskratel's SI2000 V6 and SI3000 product family.

The *sdl2pml* tool is used for automated generation of Promela model with probes from SDL specification. Additionally, it supports some non-formal extensions of the Telelogic's tools. Probes provide an

[1] We used Spin version 4.2.6.

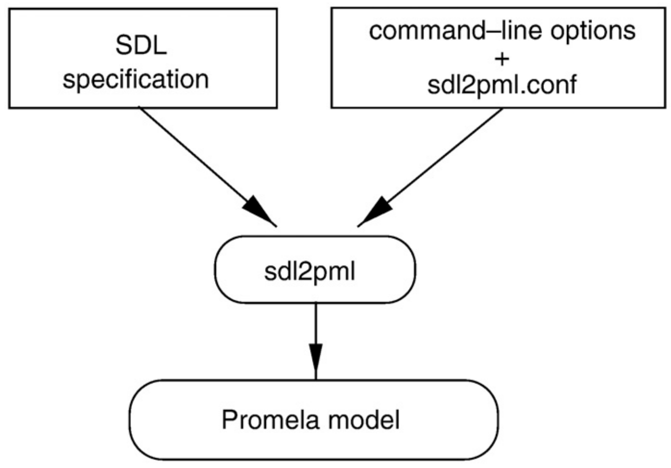

Fig. 2. Inputs and output of the *sdl2pml* tool.

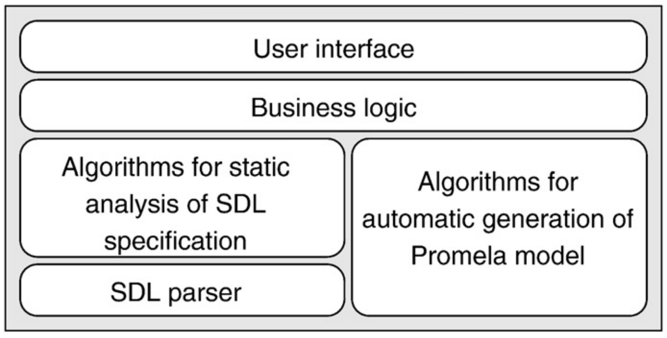
User interface
Business logic
Algorithms for static analysis of SDL specification
SDL parser
Algorithms for automatic generation of Promela model

Fig. 3. Design of the *sdl2pml* tool.

insight into the execution of the model and are mostly assertions on special variables [13].

Input to the *sdl2pml* consists of an SDL specification, command-line options, and a configuration file — sdl2pml.conf (Fig. 2). The latter can redefine values for some built-in parameters, e.g., the name of a file in which definitions of Promela signals are stored. Command-line options include support for predefined and user-defined probes. Additionally, a user can change several parameters regarding automated generation of the model — size of channel buffer, inclusion of abstract external operators, time modelling algorithms etc. Currently, *sdl2pml* only includes a simple textual user interface.

The design of the *sdl2pml* consists of five building blocks (Fig. 3). The SDL parser is implemented using ANTLR (ANother Tool for Language Recognition).[2] It is a tool for constructing recognisers, compilers, and translators. Input into the ANTRL is EBNF (Extended Backus–Naur Form) grammar. Currently, it can generate code in C++, C#, Java, and Python.

The parser is used by algorithms for static analysis of the SDL specification. It gathers all the relevant data needed by algorithms for automated model generation. Business logic is responsible for correctly executing algorithms, the creation of log records, and error handling.

The *sdl2pml* implements algorithms from [11,14] in C++:

- modelling of all predefined and user-defined SDL data types,
- support for the dynamic creation of processes,
- modelling of the priority signal,
- modelling of the implicit transition,
- modelling of the spontaneous transition,
- modelling of the save construct,
- modelling of the priority input,
- modelling of the enabling condition,

[2] We used ANTRL version 2.7.2.

```
[DBmIUA_AccessState|\
gu_T_DBSIGTRAN_dbmiua_accessstate|true]
/*INPUT PARAMETERS: T_SysInterfaceId,\
T_SysAccessId,T_AccessState*/
STARTPMLBODY;
{$INPUT_1}.val
{$INPUT_2}.val
{$INPUT_3}.val
/*OUTPUT PARAMETER: T_Err*/
{$OUTPUT}.F_ErrSts
{$OUTPUT}.F_StsSQL
ENDPMLBODY;
```

Fig. 4. Skeleton for abstraction of DBmIUA_AccessState operator.

- modelling of the asterisk input,
- modelling of direct (PId) and indirect addressing (name of the process, name of the signal route),
- support for path limitations introduced by the via statement,
- dynamic monitoring of the associated channel,
- modelling of the asterisk state,
- support for timers with parameters,
- introduction of probes for monitoring the system's behaviour during formal verification of the model,
- new model of discrete time in Promela,
- modelling of the SDL rational data types REAL, TIME, and DURATION,
- modelling for procedures,
- semi-automatic support of ADT operators.

All the implemented algorithms are formally specified using our own SDL-like pseudo language defined in [11]. Although all supported features are necessary for successfully automating generation of the model from real-life SDL specifications, we would like to emphasize the importance of semi-automatic inclusion of ADT operators. Our industrial partners use them in all SDL specifications.

3.1. Semi-automatic support for ADT operators

Real-life SDL specification often includes the use of external operators for various reasons (e.g., database access, algorithms in C). Therefore, their inclusion in the model is necessary for successful simulation or verification of the system under study.

We have developed a mechanism for the semi-automatic abstraction of external operators and their automatic inclusion in the model.

```
[DBmIUA_AccessState|\
gu_T_DBSIGTRAN_dbmiua_accessstate|true]
/*INPUT PARAMETERS: T_SysInterfaceId,\
T_SysAccessId,T_AccessState*/
STARTPMLBODY;
{$INPUT_1}.val
{$INPUT_2}.val
{$INPUT_3}.val
/*OUTPUT PARAMETER: T_Err*/
if
 ::{$INPUT_1}.val == 7 && {$INPUT_2}.val == 64->
   {$OUTPUT}.F_ErrSts = false;
   {$OUTPUT}.F_StsSQL
 ::{$INPUT_1}.val == 7 && {$INPUT_2}.val == 65->
   {$OUTPUT}.F_ErrSts = false;
   {$OUTPUT}.F_StsSQL
 ::else->
   {$OUTPUT}.F_ErrSts = true;
fi;
ENDPMLBODY;
```

Fig. 5. Abstraction of DBmIUA_AccessState operator in Promela.

```
V_Err:= DBmIUA_AccessState(VF_SysIntfId,\
         V_IUA_AccessData!F_SysAccessId,\
         V_IUA_AccState)
```

Fig. 6. Call of the DBmIUA_AccessState operator in SDL.

```
/*OUTPUT PARAMETER: T_Err*/
if
 :: VF_SysIntfId.val == 7 && \
   V_IUA_AccessData.F_SysAccessId.val == 64 ->
   V_Err.F_ErrSts = false;
 :: VF_SysIntfId.val == 7 && \
   V_IUA_AccessData.F_SysAccessId.val == 65 ->
   V_Err.F_ErrSts = false;
 :: else ->
   V_Err.F_ErrSts = true;
fi;
```

Fig. 7. Inclusion of DBmIUA_AccessState operator in a model.

The *sdl2pml* automatically prepares a skeleton for each external operator, based on its ADT operator definition. Then, an engineer can include implementation of the ADT operator or prepare its abstraction.

Fig. 4 shows a skeleton for the DBmIUA_AccessState operator. A skeleton consists of a header and body. The header is written between symbols [and] and is divided into three parts: SDL operator name, Promela operator name, and the flag which defines whether the body of the abstracted operator should be included in the model.

The body for operator abstraction starts with a reserved word STARTPMLBODY. The operator can have input and output parameters of different data types. User-defined data types are the most often used in this context. The *sdl2pml* explores data type definitions for all input and output parameters of each operator and prepares a skeleton for its abstraction by the engineer. If a parameter represents a structure, all elements of the structure will be extracted — e.g., output parameter T_Err (Fig. 4).

The suffixes of reserved words $INPUT and $OUTPUT define the operator's reference number. This ensures the independence of a parameter's reference from the actual parameter's name. When the operator is included in a system's model, the parameter reference, e.g., $INPUT_1, is replaced by the actual parameter name. The end of the operator's body is marked by ENDPMLBODY (Fig. 4).

Abstraction of the operator is prepared in Promela. In Fig. 5, abstraction of a DBmIUA_AccessState operator is shown, while Fig. 6 presents its use in an SDL specification. The user can decide which lines in the prepared skeleton for operator abstraction he/she will modify. User-defined statements should be syntactically correct. Based on the name of the operator and user-defined abstraction, *sdl2pml* prepares Promela code for the operator which will then be included in the model of the system (Fig. 7). It only considers comments and lines changed by the users (based on the reserved word =), others are ignored and can be used in the next versions of the abstraction.

Since each operator can be used in different processes, abstraction of the operator should enable the user to define its behaviour for each process of the system. An example of such behaviour is access to a database, where the result of a query depends on the process. The user can use the reserved word $PROCESS to differentiate abstractions of the operator during different SDL processes. Default value can be defined by the reserved word DEFAULT. Fig. 8 shows their usage. Operator abstraction defines values for output parameter F_IUA_Side that depend on the calling process.

If the user does not want to use abstraction for the selected operators, *sdl2pml* can automatically include them in the Promela model. It uses the embedded C code extension of the Promela that is present in versions of Spin 4.0, or higher. Currently, *sdl2pml* only supports limited inclusion of embedded C code within the model. When all algorithms from [14] have been implemented, it will support automatic inclusion of embedded C code. In the presented case study, we have used abstraction for all ADT operators.

3.2. Tool execution

Execution of the tool can be divided into two main phases:

1. static analysis of the SDL specification and
2. automated generation of the Promela model.

Firstly, some formatting of the SDL specification file is required. If the SDL specification is using MS DOS file format, new line characters “CR+LF” have to be replaced by “LF”. All C-like comments (/* */) are then extracted. Finally, all referenced sub-specifications are included in the main specification.

3.2.1. Static analysis

Static analysis of SDL specification examines synonyms, syntypes, newtypes, signals, signal lists, signal routes, channels, connect constructs, timers, processes, and procedures. Results are saved into plain text files, in accordance with the configuration file. We decided to use text files because users can easily perform enquiries using those standard pattern scanning and processing tools commonly used by the developers. Fig. 9 shows an extract from analysis results for variable declarations.

Each text file is divided into two sections — a header and a body. The context of the data is described in the header section. Additionally,

```
[DBeIUA_ASdata|gu_T_DBSIGTRAN_dbeiua_asdata|true]
/*INPUT PARAMETERS: T_IUA_AS_Id,T_Err*/
STARTPMLBODY;
{$INPUT_1}
{$INPUT_2}.F_ErrSts = false;
{$INPUT_2}.F_StsSQL

/*OUTPUTPARAMETER: T_IUA_ASdata*/
{$OUTPUT}.F_IUA_Side.val = ($PROCESS==IUAsg__IUA_LM)->
                T_IUA_Side__L_SG;
              ($PROCESS == IUAcs__IUA_LM)->\
                T_IUA_Side__L_MGCL3;
               (DEFAULT)-> T_IUA_Side__L_SG;
{$OUTPUT}.F_TrafficMode.val = CV_IUA_OverRide;
{$OUTPUT}.F_AS_State.val = CV_AS_Active;
{$OUTPUT}.F_TM_Recov = 4;
ENDPMLBODY;
```

Fig. 8. Usage of the $PROCESS reserved word in DBeIUA_ASdata operator.

```
/*=====================================*/
/* COLUMN 1 : SDL name          */
/* COLUMN 2 : SDL type          */
/* COLUMN 3 : Promela name       */
/* COLUMN 4 : Promela type       */
/* COLUMN 5 : default value      */
/* COLUMN 6 : declared          */
/*=====================================*/
V76par;V76paramTyp;dataLink__AtoB__V76par;\
V76paramTyp;NONE;NONE
V76par;V76paramTyp;dataLink__BtoA__V76par;\
V76paramTyp;NONE;NONE
DLCnum;DLCident;DLCa__dispatch__DLCnum;\
DLCident;NONE;NONE
DLCpeer;DLCident;DLCa__dispatch__DLCpeer;\
DLCident;NONE;NONE
uData;Integer;DLCa__dispatch__uData;int;10;\
NONE
```

Fig. 9. Results of a variable declaration analysis.

```
Init_IUALM_tab;IUAcs__IUAmng__Init_IUALM_tab;start;0;
Init_IUALM_tab;IUAcs__IUAmng__Init_IUALM_tab;start;7;V_IUA_LM_Inst :=0
Init_IUALM_tab;IUAcs__IUAmng__Init_IUALM_tab;start;6;J000_initIUAtab
Init_IUALM_tab;IUAcs__IUAmng__Init_IUALM_tab;start;10;V_IUA_LM_Inst
Init_IUALM_tab;IUAcs__IUAmng__Init_IUALM_tab;start;24;(>= CV_MaxIUA_AS_Id);
                     ...
```

Fig. 10. Static analyses results of process and procedure automata.

it includes information about input specification name and current date.

The results of SDL construct analysis are stored in the body section. In the example, presented in Fig. 9, each line in a body describes the declaration of a variable. The properties of an SDL construct are separated by semicolons. Detailed specification of the text file format and chosen properties for all SDL constructs can be found in [11,14].

The most complex part of static analysis is the examination of the communication infrastructure and the automata of processes and procedures. SDL processes communicate with signals. The communication infrastructure defines all possible routes for a signal with the use of channels and signal routes. The automaton of each process and procedure must be analysed for sound modelling of communication in Promela.

A process graph consists of states with unique names, including special state names, asterisk (*) and dash (-). The asterisk represents all states, except those that are explicitly excluded, while dash state represents the current state. Model generation using *sdl2pml* supports all types of state names, as defined by Z.100.

The input port of SDL process is a queue, which receives and retains signals in the order of arrival until the signals are consumed by an input construct. Input port is modelled using Promela channels. Defining the associated channel of a Promela process depends on the data types of all signals that are part of the valid input signal set. Additionally, the maximum number of process instances has to be considered.

Results regarding the static analysis of an automaton are stored in a file named body.var (Fig. 10). Each line has five columns and represents one SDL construct of a process graph. The first column describes the SDL-name of a process or procedure. The second column represents the process name in a Promela model. The SDL state's name is stored in the third column. The fourth column describes the type of SDL construct using its unique number. Finally, the fifth column includes an SDL construct's statement.

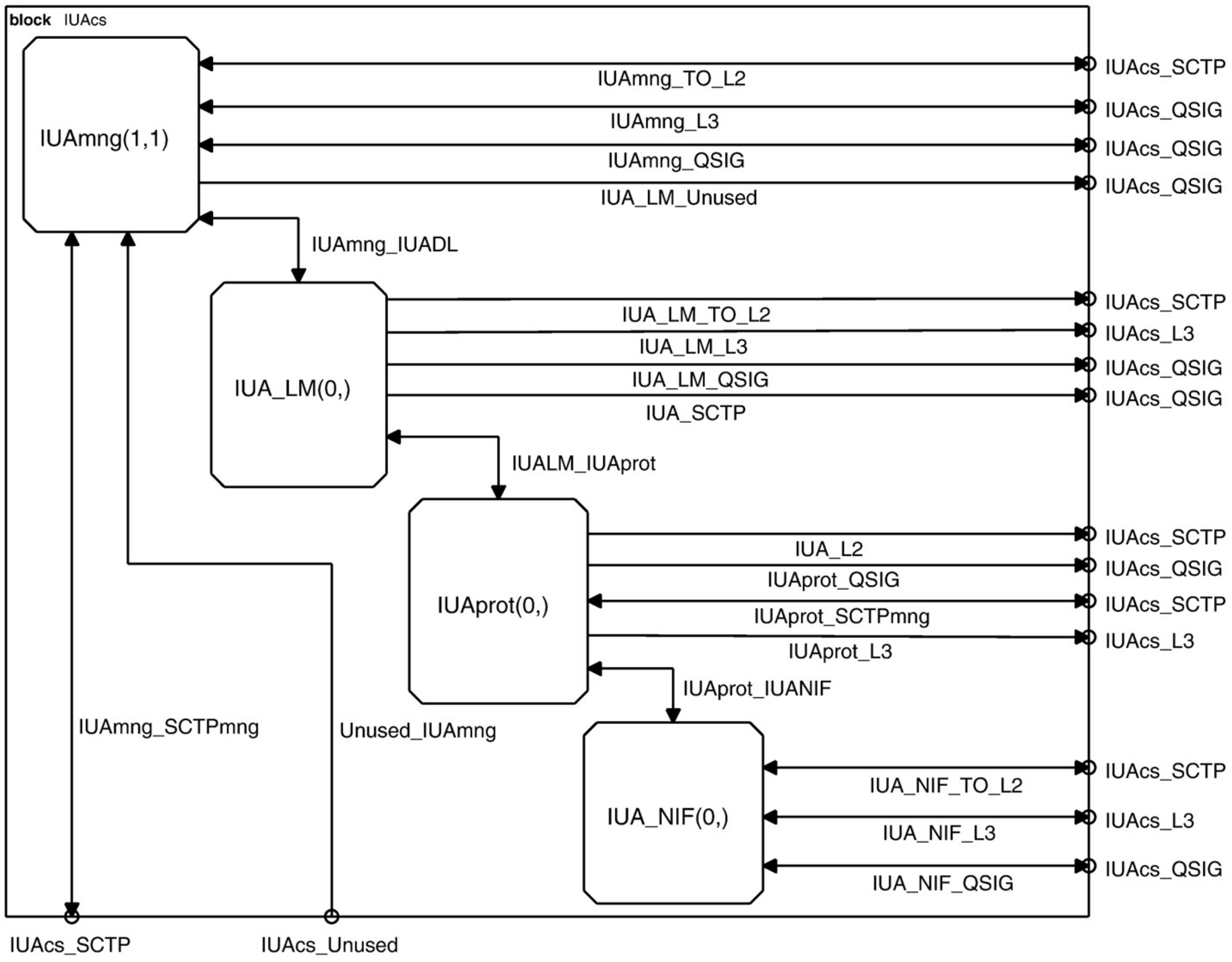

Fig. 11. Process chain in an IUAcs block.

3.2.2. Automated generation of Promela model

Automated model generation is based on the results of static analysis. Each process graph is written in a separate file. At the end of this phase all files are combined in a flat Promela file that models the whole SDL specification.

All SDL specifications from our industrial partner include timers and external operators. We developed a new model of discrete time in Promela that could be used within the mainstream version of Spin. Previously, we have used DT Spin [4], an extension of the Spin model-checker with discrete time, which had been developed within the Vires project. In order to check if the model is in accordance with SDL semantic, *sdl2pml* supports the automatic insertion of permanent probes. Predefined probes are included only on a user's demand. For example, they can be used to check if all variables are initialized before first being used. User-defined probes are introduced for the observation of selected variables and signals [13].

Promela does not support rational data types. Rational numbers can be modelled with fractions or with the use of external C code (only in Spin 4.0, or higher) [7]. *Sdl2pml* uses fractions to achieve backward compatibility with the Spin tool.

A detailed description of these features is outside the scope of this paper.

4. IUA protocol — a case study

The *sdl2pml* was tested within various SDL specifications [11,14]. We faced the greatest challenge with a real-life specification of an ISDN User Adaptation protocol. IUA protocol is used in the Next Generation Networks (NGN) between Signalling Gateway (SG) and Media Gateway Controller (MGC). The protocol specification is part of the SI2000 V6 digital softswitch [9,10].

IUA blocks at the SG and MGC are from two independent products. Because they differ in some implementation details, the model of the system must include both of them. They consist of four processes (Fig. 11). Process IUAmng is a static process and manages instances of the IUA_LM process. During connection establishment, the IUA_LM process instance creates peer IUAprot process instance, and manages its execution. IUAprot can execute in two modes. It works in a server mode if it is created at the SG side. If it is created at MGC side, it works in a client mode. An instance of the process manages the transport of DSS1 signalling over an IP network. Finally, each data link connection for user interface is assigned one instance of process IUA_NIF.

Firstly, the IUA-part of the SDL specification has to be prepared for automated model generation. Next, *sdl2pml* is used for automatically generating a Promela model of the newly prepared SDL system.

4.1. Preparation of the SDL specification

Preparation of the SDL specification consists of the following steps:

1. an automated extraction of the IUA-part from the SI2000 V6 SDL specification,
2. preparation of the extracted SDL specification's environment, and
3. abstraction of selected external operators.

Firstly, a new SDL system (IUAsystem) is extracted from the large SDL specification of the SI2000 V6. Its core concerns are processes from the block IUA. Additionally, those data types that are declared outside the IUA block must be redefined in the IUAsystem. Because the specification of the IUA block differs in systems that are designed for SG and MGC, it is extracted twice — once for the SG side of the communication and once for the MGC side (Fig. 12).

IUAsystem consists of eight blocks. The IUAcs block represents the MGC-version of the IUA block, while the IUAsg block describes the SG-version. Since simulation and verification using Spin require a complete system, we supplemented the specification with a model of its environment. It was defined using blocks L2, SCTP, L3, QSIGdummy, and Unused (Fig. 12). Each block included at least one process capable of receiving and sending all signals defined within the channels connected to it. Unfortunately, specification of processes, signal routes, and channels were carried-out manually. Clearly, we had to take great care to provide a proper environment, thus ensuring valid behaviour of the model. This phase is very time-consuming. We believe, that the major part of the environment specification could be automated.

Our objective was to check the establishment of an ISDN call between users A and B. Each user is connected to its SG. Signalling

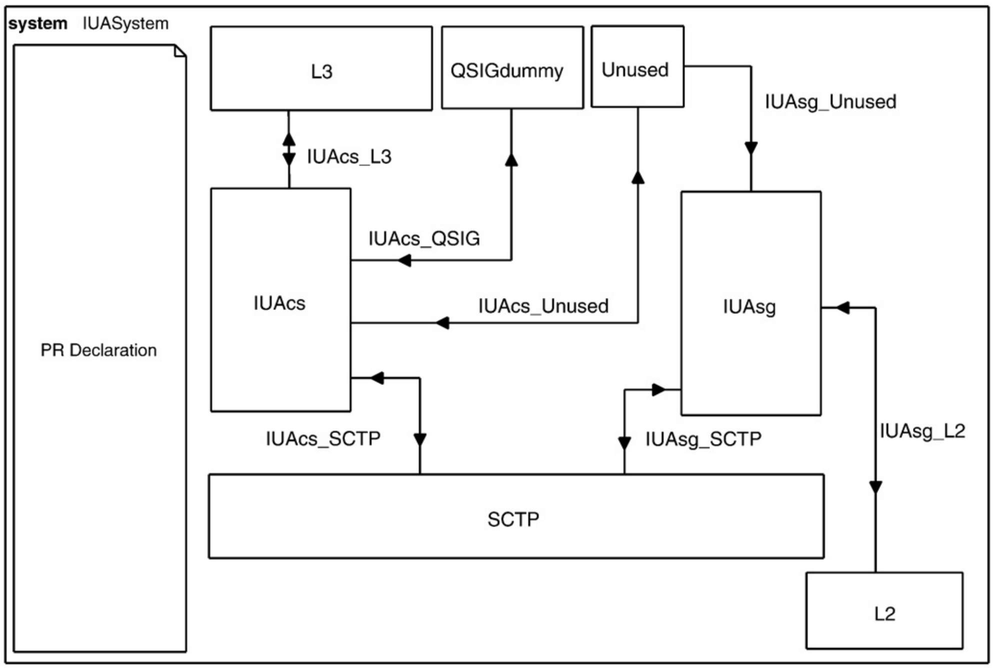

Fig. 12. SDL specification of the IUAsystem.

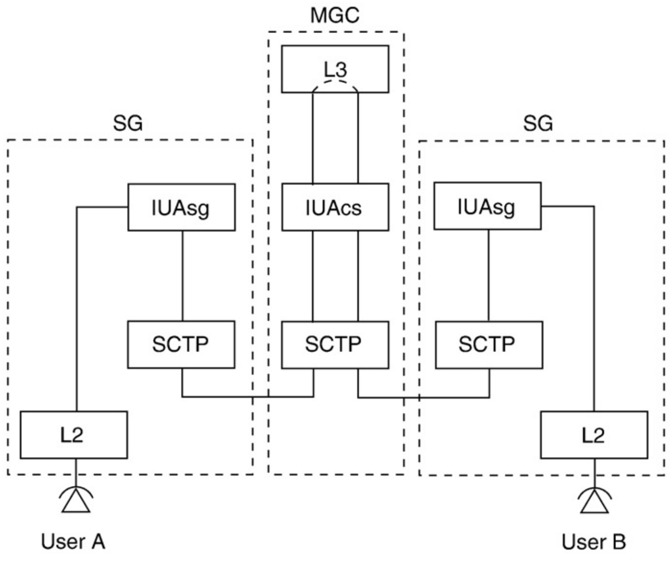

Fig. 13. Signalling path of an ISDN call.

gateways are connected to the MGC. The full signalling path is as follows: User A ↔ SG ↔ MGC ↔ SG ↔ User B (Fig. 13).

Fig. 14 shows MSC (Message Sequence Chart) from the beginning of connection establishment from User A to User B. For each process covered by a MSC there is an instance axis. The process names are written within the frame symbol. The name of the SDL block within the process resides is written above it. The MSC shows signal sequence following User A's commencement of connection establishment. L2UserAmng process sends a Create_L3dssm_Req signal to IUAmng process. It sends a request for IUA_NIF process creation through IUA_LM and IUAprot processes. Next, L2UserAmng process establishes data link connection with the IUA_NIF process (DL_EstablishInd signal). Data reception is indicated by DL_DataInd signal. SCTP_SendFrame signal is used for sending frames over the SCTP block to the User B.

Final specification of the IUAsystem without comments consists of 28,563 lines. This concludes the second step of the preparation of the SDL specification.

We used an ObjectGeode's simulator to check the prepared specification. During the simulation a user has to manually type return values for all external operators or write a special script to automate it.

Specification of the IUAsystem includes 87 external operators. They are used for database access (23), string manipulation, output to the console, storing temporary values, etc. Using the proposed mechanism for the abstraction of external operators, an engineer can define abstraction for the chosen external operators. For example, return values for the database access should be specified. *Sdl2pml* can prepare abstraction skeletons for the chosen operators. Unfortunately, abstractions can be done only by an engineer with a detailed understanding of the operator.

Spin does not support simulation of a model that includes C code. A model that uses the proposed abstraction mechanism can be simulated. Abstraction of the external operators is a very time-consuming task. Therefore, it deserves further research to decrease the required user effort.

4.2. Automated model generation

Specification of the IUAsystem was now prepared for the automated generation of Promela model using the *sdl2pml* tool. Generation of the model normally takes 120 min on AMD Athlon(tm) X2 Dual Core Processor 4200+. Due to the use of plain text files during static analysis of the system, we were able to decrease the generation time using the RAM-disk to 75 min. The generated Promela model consisted of more than 102,000 lines of code. The size of the text file was 6.5 MB and to our knowledge, this is the largest Promela model ever processed by the Spin tool.

After generation of Promela model we could perform either simulation or formal verification. We checked the correctness of the generated model using simulation.

4.3. Simulation of the model

Spin performs a syntactic check before the simulation run. During the development of algorithms and *sdl2pml* tool we struggled with warning and error reports. A model of this size did demonstrate that they could be improved by providing greater insight into the reason for the report.

During interactive simulation a user can obtain better understanding of the model's behaviour. We used graphical interface of the XSpin. Next, we started searching for unpredicted scenarios using random simulation. It took Spin 10 min to prepare the simulator and it consumed 500 MB of RAM. Each modification of the model resulted in another 10 minute wait. During simulation, XSpin drew an MSC. Again, it was hard to follow the diagrams for a model of this size.

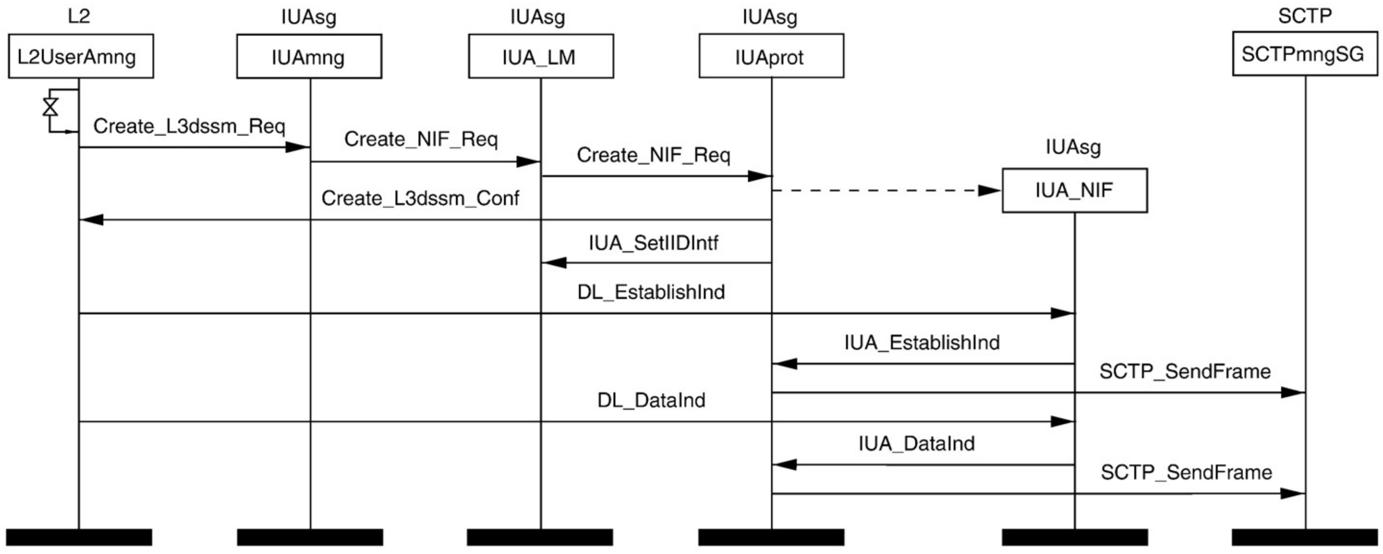

Fig. 14. MSC shows the beginning of connection establishment from User A to User B.

We managed to find some unexpected execution paths. Some of them were caused by the model of the environment and abstraction of the external operators. If we check the error trail against the original system, we can verify if the model of the system is sound. This kind of feedback makes the model checking technique very powerful. We also managed to find an execution path that was confirmed as an error by the developer of the protocol.

5. Conclusion

A tool was presented for automated generation of Promela model from an SDL specification. It is fully autonomous and does not rely on any external tools. The algorithms that formally specify its operation can be found in [11,14]. It supports automatic generation of models for Spin and DT Spin. Models for Spin can include embedded C code. Implementation in C++ language consists of more than 120,000 lines of code — partly generated by ANTRL.

Sdl2pml was successfully used on a real-life specification of the IUA protocol. Generated Promela model consists of more than 102,000 lines of code. To our knowledge, this is the largest Promela model ever processed by the Spin tool. This case study demonstrated that we managed to support most of the features that are used in industrial SDL specifications and shows its applicability. There is still room for an improvement. The abstraction of external operators is an error-prone and time-consuming task. It requires an engineer with detailed knowledge of the specification. In our future research we would like to extend the *sdl2pml* using a graphic user interface, provide a hierarchical view of the model, include support for the multi-threaded execution, port it to other operating systems and prepare methodology that will help users to prepare a model of the system. Additionally, we would like to focus on semi-automatic design of the environment.

Acknowledgements

This work was partly funded by the European Regional Development Fund and Slovenian Ministry of Higher Education, Science and Technology under the Information and Communication Technologies Centre of Excellence R&D project Correctness Verification of Communication System Functioning.

References

[1] J. Baete, Esprit Project 23498 — VIRES (Verifying Industrial Reactive Systems), URL: http://www.cordis.lu/esprit/src/23498.htm.
[2] 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 Proc. of SDL-FORUM'99, (Montreal, Canada, 1999).
[3] M. Bozga, S. Graf, L. Mounier, I. Ober, IF validation environment tutorial, Proc. of the 11th Int. SPIN Workshop on Model Checking of Software, Barcelona, Spain, Lecture Notes in Computer Science, vol. 2989, Springer, Berlin, 2004, pp. 306–307.
[4] D. Bošnački, Extending Promela and Spin with discrete time, Proc. of the VIII Conference on Logic and Computer Science, 1997.
[5] G.J. Holzmann, J. Patti, Validating SDL specifications: an experiment, Proc. 9th Int. Conf on Protocol Specification, Testing, and Verification, INWG/IFIP, North-Holland, Amsterdam, 1989, pp. 317–326.
[6] G.J. Holzmann, Practical methods for the formal validation of SDL specifications, Computer Communications 15 (2) (1992) 129–134.
[7] G.J. Holzmann, Logic verification of ANSI-C code with Spin, Proc. of the 7th Int. SPIN Workshop on SPIN Model Checking and Software Verification, Lecture Notes in Computer Science, vol. 1885, Springer, Berlin, 2000, pp. 131–147.
[8] G.J. Holzmann, The Spin Model Checker, Primer and Reference Manual, Addison-Wesley, Reading, 2004.
[9] R. Slatinek, Functional Specification of IUA Protocol (in Slovene), 2002.
[10] R. Slatinek, Design Specification of IUA Protocol (in Slovene), 2001.
[11] B. Vlaovič, Automatic generation of models with probes from the SDL system specification: Ph.D. Thesis (in Slovene), Faculty of Electrical Enginnering and Computer Science, University of Maribor, 2004.
[12] B. Vlaovič, A. Vreže, Z. Brezočnik, T. Kapus, Verification of an SDL specification — a case study, Electrotechnical Review 72 (1) (2005) 14–21.
[13] B. Vlaovič, A. Vreže, Z. Brezočnik, T. Kapus, Automated generation of Promela model from SDL specification, Computer Standards & Interfaces 29 (2007) 449–461.
[14] A. Vreže, Extending automatic modeling of SDL specifications in Promela with embedded C code and new model of discrete time: Ph.D. Thesis (in Slovene), Faculty of Electrical Enginnering and Computer Science, University of Maribor, 2006.
[15] ITU-T, Specification and Description Language (SDL), Recommendation Z.100, 1993.

Aleksander Vreže received his diploma and Ph.D. degrees from the Faculty of Electrical Engineering and Computer Science, University of Maribor, Slovenia, in 2001 and 2006, 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 automated generation of models from the SDL specification.

Boštjan Vlaovič 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 automated generation of models from the SDL specification.

Zmago Brezočnik 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 head of the Institute of Electronics and Telecommunications at the same faculty. His main research areas are formal hardware and protocol verification, especially symbolic model checking, and binary decision diagrams.