Applying Automated Model Extraction for Simulation and Verification of Real-Life SDL Specification with Spin

Article *in* IEEE Access · March 2017

DOI: 10.1109/ACCESS.2017.2685238

CITATIONS
2

READS
156

3 authors, including:

Bostjan Vlaovic
University of Maribor
21 PUBLICATIONS **72** CITATIONS

SEE PROFILE

Zmago Brezočnik
University of Maribor
65 PUBLICATIONS **257** CITATIONS

SEE PROFILE

Applying automated model extraction for simulation and verification of real-life SDL specification with Spin

Boštjan Vlaovič, *Member, IEEE,* Aleksander Vreže, *Member, IEEE,* and Zmago Brezočnik, *Member, IEEE*
University of Maribor, Faculty of Electrical Engineering and Computer Science

***Abstract*—Formally defined Specification and Description Language (SDL) is used for the design and specification of complex safety-critical systems. Each change in the specification of the product should be immediately checked formally against the requirements' specification. This paper presents semi-automated system abstraction, automated model extraction, simulation, and formal verification of real-life complex SDL specification. Sound algorithms implemented in our *sdl2pml* automated model extraction tool preserve all properties of the SDL system. *Sdl2pml* includes our model of discrete time, abstraction, and support for all relevant SDL functionality and constructs such as dynamic process creation, rational data types, communication with more than one process instance, and others. To the best of our knowledge, most of them are not supported by any other known approach. We use our *SpinRCP* tool for simulation and formal verification of the extracted model with the Spin model checker. We demonstrate the applicability of our approach on ISDN User Adaptation Protocol from SI3000 Softswitch. The extracted Promela model is the largest one ever processed by Spin. We have shown that Spin simulation and model checking can be applied successfully to such huge models.**

***Index Terms*—Formal specifications, automated extraction, formal languages, simulation, formal verification, model checking, SDL, Promela, Spin, Sdl2pml, SpinRCP.**

I. Introduction

Specification and Description Language (SDL) is used for the design and specification of services, especially in the field of telecommunications. It is defined formally and can be utilized from the conceptual system design phase to the automated generation of implementations in the chosen programming language. SDL is regarded as one of the keystones of a stable and mature development environment. The latest version of SDL is SDL-2010. The most recent ITU-T recommendations for SDL-2010 were released in 2016 [1].

In an ideal development environment, each change in the specification of the product would be immediately checked formally against the requirements' specification. This check is not an easy task due to the scale of the complex real-life SDL specifications and various non-formal parts of the system that are usually developed in C language and included in the SDL via Abstract Data Type (ADT) notation. This practice was recognized, and non-standard semi-formal SDL Real-Time (SDL-RT) was introduced in 2002. The latest version supported by Pragmadev Studio was released in 2016 [2]. In the recommendation Z.104 [3], ITU-T defined the data and action features of the SDL-2010.

Manuscript received December 23, 2016;

In this paper, we focus on standardized versions of SDL88 and SDL92 that are still utilized actively by our industry partner Iskratel d.o.o on numerous products, e.g., IC1020AX (IMS Core), IE1020AX (IMS Edge), IA1020AX (IMS AS), MS1010AX (Media Server), CS6116AX (Call Server, Unified Communications), CE6111AX (Compact Call Server), and MG6114AX (Media Gateway). Specifications describe the implementation details and are used to build the production systems. Such specifications contain SDL constructs and additional extensions that enable developers to include operators that are implemented in other programming languages. The C programming language is used for low-level or processor intensive operations.

First, we had to decide which formal method would be the most suitable for checking the functional correctness of extremely large product specifications in SDL with some parts written in C [4]. From the very beginning, it was clear that the method should be automatized. Since one of the most prominent automatic formal verification techniques for assessing functional properties of concurrent and distributed systems is model checking [5], we selected this method. Model checking requires a model of the system under consideration and the desired property. Then it checks systematically whether the given model satisfies this property or not. The properties that can be checked describe either a safety (e.g. deadlock freedom, mutual exclusion, invariants) or liveness (e.g. starvation absence, request-response properties) of the system. If we have a sound model of the system and the property is not met, the model checking will provide a counterexample, i.e an execution path that violates the property.

Next, we had to choose one of the available model checkers. Based on studies in [4] and [6], we have decided to apply a Spin (Simple Promela Interpreter) model checker that uses Promela (Process Meta Language) model description language. The properties to be checked can also be specified in Promela, or they can more easily be generated automatically from LTL (Linear Temporal Logic) formulae. Specifications of the desired properties have to be written by expert developers who understand their true meaning fully.

Models are abstractions of real systems that facilitate analyses. Manual creation of a model is error prone and a very time-consuming task. Also, the construction and analysis of the model should be a less complex task than the construction of

the real system. Therefore, our research focuses on automating model extraction from SDL specification. After a detailed study of existing techniques, we decided to propose a new approach to automated model extraction from SDL to Promela that is based on formally specified algorithms [7], [8], [9], [10]. Sound modeling of SDL constructs is established on more than 60 formally written algorithms. As far as we know, *sdl2pml* is the first SDL to Promela converter that supports a full SDL system structure, predefined and user-defined data types, dynamic process creation and termination, timers with parameters, save construct, asterisk state, asterisk input, priority input, implicit transition, rational numbers, enabling condition, direct and indirect addressing, path limitations, procedures and some other less frequently used SDL concepts. We stress that all of the listed constructs must be supported to model SDL specifications with implementation details. Additionally, there is no notion of time in Promela, so we had to model it. We developed a new model of discrete-time that was modeled entirely with standard Promela [9]. The previously known solution required a special version of Spin [11]. This approach was not acceptable because we needed some of the features from the recent versions of Spin, especially the inclusion of embedded C code for the ADT operators.

Given the system model in Promela, Spin can perform random, interactive, or guided simulation of the model executions. Furthermore, it can generate a verifier in C code, which performs verification of the system's correctness properties. Spin is implemented in C and is portable across various operating systems, including 64-bit versions of Linux, OS X, and Windows [12].

II. Background and related work

Earlier, several other attempts were made to transform code that is written in real programming languages (e.g. SDL, C, or Java) into Promela, the input model description language of the Spin model checker.

In [13] the author of Spin and Promela, Gerard J. Holzmann, presents an experiment of validating SDL specification of the AT&T's 5ESS Switch software. Within this work, it was decided to extend the then validating language Argos and thus to introduce a new modeling language—Promela, that can support a significant fraction of the SDL features. The validation system was based on reachability analysis and was a predecessor of the later Spin model checker. It was able to complete analyses of over 250 million reachable composite system states. A similar translation from a dialect of SDL-88 (TNSDL – TeleNokia SDL) to Promela is presented in [14]. It covers more SDL features, but no practical application of the method is known. Another attempt to use the Spin model checker as a verification engine for SDL, with a particular focus put on the verification of timing properties of SDL models, is presented in [15]. To be able to model SDL timers in a quantitative way, the authors extended standard Promela to DT Promela and standard Spin to DT Spin, where DT stands for Discrete Time. The translation from SDL to DT Promela was done in two steps. The first step from SDL to intermediate language IF was developed at Verimag, Grenoble, and requires ObjectGEODE for its execution. The authors developed the second step from IF to DT Promela. The translation covers a subset of SDL with some essential omissions for complex real-life systems. In addition, the development of the DT Spin ceased.

One of the most successful tools is Modex [16] model extractor targeted to verification of multi-threaded software written in the C programming language. It extracts verification models automatically from implementation level C code under the guidance of a test-harness, specified by a user in a separate file. Those C statements that have no equivalent in Promela itself can be embedded directly as C code in Promela. In [17] it was shown how to handle further notable features of C, such as pointer structures and function calls.

Java PathFinder [18], JPF, is a translator from Java to Promela. A substantial subset of Java is covered by JPF that includes dynamic object creation, inheritance, exceptions, and thread operations. This work is part of a broader attempt to make formal methods applicable in the areas such as space, aviation, and robotics at NASA.

We have developed several verification tools including the following ones: SDL to Promela translator (*sdl2pml*) [10], Spin Trail to a Message Sequence Chart Conversion Tool (*st2msc*) [19], Eclipse plugin for Spin [19], and an integrated development environment for the Spin model checker (*SpinRCP*) [20]. We use *SpinRCP* to analyze the model by using simulation and formal verification. The final research goal is to build a framework for the systematic use of model checking. Available user-friendly tools can stimulate the integration of model checking in software development processes where the designers agree that its use is necessary. We hope that *SpinRCP* will become a major role of environments where Promela and Spin are used [21]. *SpinRCP* is freely available for 64- and 32-bit Windows, Linux, and Mac OS X operating systems [22].

This article demonstrates successful preparation of the sound model of the complex real-life system with the use of the *sdl2pml*. It is shown that we managed to support all relevant SDL functionality and constructs such as dynamic process creation, communication with more than one process instance, charstring and rational data types, support for priority input, SAVE construct, abstraction, and others. Our work is based on the cooperation with the telecommunication industry, but it could be applied to all domains where SDL is used, e.g., avionics, medical, automotive, industrial control systems, military, aerospace, and safety critical systems.

This paper is organized as follows. Section 2 presents an overview of the ISDN User Adaptation (IUA) protocol that has been selected as a test case for evaluation of our *sdl2pml* tool. Section 3 gives a detailed description of obtaining the IUA protocol model in Promela from the available specification of that protocol in SDL and some embedded C code. Simulation and formal verification of the extracted model with Spin model checker within the *SpinRCP* integrated development environment are discussed in Section 4. Finally, Section 5 contains the conclusions and plans for further work.

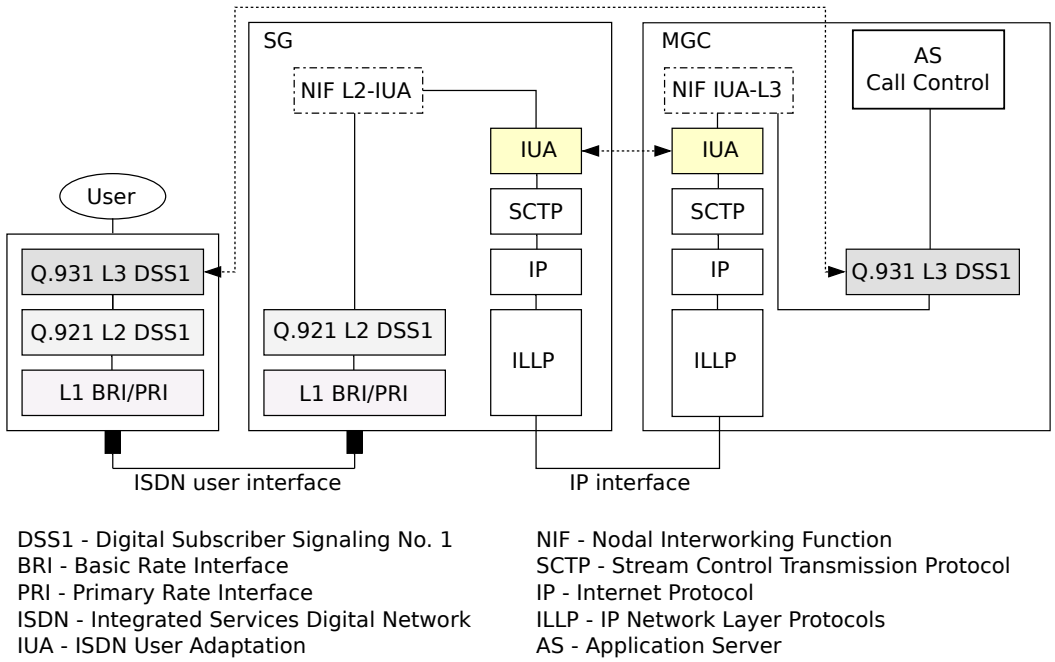

Fig. 1. IUA Protocol Architecture

III. ISDN User Adaptation Protocol

Detailed knowledge of all possible configurations and the functionality of the IUA (ISDN User Adaptation) protocol is not required, but an overview of its complexity might be interesting.

IUA protocol is used when there is a need for Switched Circuit Network (SCN) signaling protocol delivery over the IP network [23], [24]. In Figure 1, the user's terminal connects to the Signaling Gateway (SG) over a standard ISDN user interface. The specification supports both ISDN Basic Rate Access (BRA) and Primary Rate Access (PRA), including the support for point-to-point and point-to-multipoint modes of communication. The terminal's Q.931 signaling messages are transported via the nodal interworking function implemented in NIF L2-IUA and NIF IUA-L3 to the Media Gateway Controller (MGC) where the peer ISDN Q.931 protocol layer exists—the dotted line that connects the user and MGC in Figure 1 [24].

Application Server Process (ASP) is a process instance of an Application Server (AS) which is a logical entity serving a specific application instance. An example of an AS is a Call Control at MGC. It is handling the Q.931 and call processing for D channels terminated by the SG.

The IUA layer management at the SG maintains the availability state of all dynamically registered remote ASPs to manage the Stream Control Transmission Protocol (SCTP) associations and the traffic between the SG and ASPs. Each active connection is associated with one SG-ASP pair. Different ISDN terminals that connect to SG can use different SG-ASP pairs. Additionally, the active/inactive states are maintained of remote ASPs. Active ASPs are those currently receiving traffic from the SG.

In summary, the IUA protocol provides backhauling of ISDN Q.921 User messages over IP using the SCTP. The delivery mechanism should meet the following criteria:

- support for transport of the Q.921/Q.931 boundary primitives,
- support for communication between Layer Management modules on the SG and MGC,
- support for management of SCTP active associations between the SG and MGC.

For the purpose of our case study, we decided to use the simplest possible configuration connecting two ISDN users at the same SG (Figure 2). 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:

- IUAmng,
- IUA_LM,
- IUAprot, and
- IUA_NIF.

The singleton process IUAmng is created at the start of the system and never terminates (static process). It manages dynamic IUA Layer Management (IUA_LM) processes. Dynamic processes are created and terminated during the normal execution of the system. At the SG there is one instance of an IUA_LM process for each AS. It has the role of a server, manages SCTP associations between the SG and ASPs and provides fail-over functions to support a high availability of call processing capability. At the MGC IUA_LM has the role of a client. At activation, it initiates the connection to the SG and reports interface identifiers that it is configured to manage.

Process IUAprot implements the functionality of user adaptation protocol. It supports the transport of the Q.921/Q.931 boundary primitives. At the SG, some instances depend on

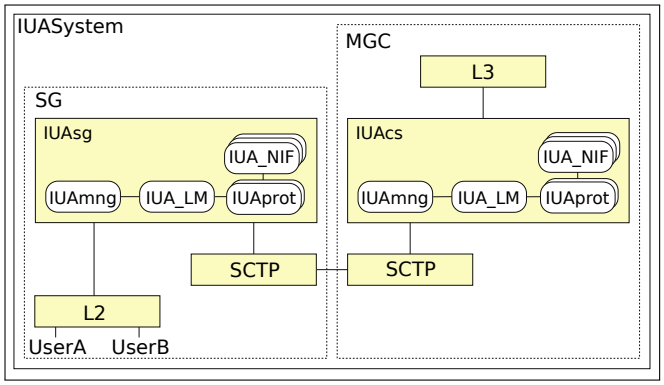

Fig. 2. Chosen configuration of the verification system

the failover and traffic distribution capabilities. The MGC has one instance of IUAprot for each connected SG. Each IUAprot may create up to nine instances of the IUA Nodal Interworking Function (IUA_NIF) processes.

IUA_NIF processes provide a seamless mapping between the L2/L3/QSIG and the IUAprot. They manage data connections of the ISDN user interface—eight for the ISDN terminal equipment and one for the point-to-multipoint mode of communication.

The telecommunication company Iskratel d.o.o. developed this SDL specification as a part of SI3000 Softswitch. IUA protocol and supporting blocks represent only a small part of the softswitch and are from two different products, SG and MGC. Both systems consist of $931,539$ lines of code. Blocks that are shown in Figure 1 consist of $97,388$ lines of SDL'93 code without comments and blank lines. An abstracted version of these blocks that was prepared for the automated extraction of the model consists of $28,563$ lines of SDL code. The extracted Promela model that was used for the purpose of this paper consists of $87,948$ lines of code without comments and blank lines.

The specification includes all SDL constructs that are used in the specification of the softswitch. Additionally, it includes many external operators written in C with the use of extended Abstract Data Type (ADT) definitions. Therefore, it is very suitable to demonstrate the applicability of our approach in greater detail. To our knowledge, this is still the largest Promela model ever processed by the Spin tool.

IV. Automating Model Extraction

To extract the model of the selected part of the SDL specification successfully, we had to:

- prepare the SDL system for the verification,
- define a mechanism for semi-automatic abstraction of ADT operators,
- define the sound modeling of the SDL specification,
- define a mechanism for inclusion of permanent, predefined, and user-defined probes.

Simulation and verification with Spin require a complete system. A new SDL system for verification has to be constructed to model only part of the treated SDL specification (Figure 3). It consists of:

- part of the SDL specification that is under inspection (e.g., IUA protocol) and
- model of its environment (everything else).

We managed to automate some steps of the process, but there is still room for improvement.

IUA protocol implementation includes 87 external ADT operators. They are used for the access to the database, string manipulations, trace and debugging operations, and for writing to the management console. The execution of the system depends on the configuration data from the database. It includes all necessary data for successful establishment and termination of the ISDN call. Abstraction of operators depends on the chosen setup of the system and the requirement specification.

If we want to check some properties of the system, we have to insert probes into the model. Probes provide an insight into the execution of the model. We support permanent, predefined, and user-defined probes. Permanent probes are included in all models. They are used to check a model's accordance with the SDL semantics. Predefined probes are included only on a user's demand. They are used for checking properties which would contribute significantly to the model's complexity or could result in an undesired behavior. User-defined probes are used for observation of selected variables or signals. Each inserted probe—a variable or group of variables—can be used in requirement specifications (assertions, never claims or LTL formulae). They can be included mechanically by the developer with the use of comments in the SDL specification or via a configuration file of the *sdl2pml* tool [7], [8], [10]. Use of comments is preferred since it provides additional information to all participating developers and promotes inclusion of probes during the development process.

A. Preparation of the SDL System for Verification

The IUASystem consists of unchanged blocks that implement the IUA protocol and supporting environment (Figure 3). The environment should provide the behavior that is required to check the requirement specification successfully. We managed to automate many steps of the process, but the abstraction remains in the hands of an expert who understands the functionality of the system and the requirement specification. However, if the requirement specification and interface primitives do not change during the development of the system, the environment can remain unchanged.

Based on the requirement specification, we decided that the environment of IUAsg and IUAcs should include SDL blocks that implement:

- DSS1 L2 at the SG,
- DSS1 L3 at the MGC,
- SCTP at SG and MG, and
- other functionalities that are not part of the requirement specification.

To ease the investigation of unexpected execution paths that might be exposed during the model checking phase, we strived to use the original names for the abstracted blocks and processes. An abstraction of the DSS1 L2 was prepared in the SDL block L2. It consists of two processes—L2UserAmng

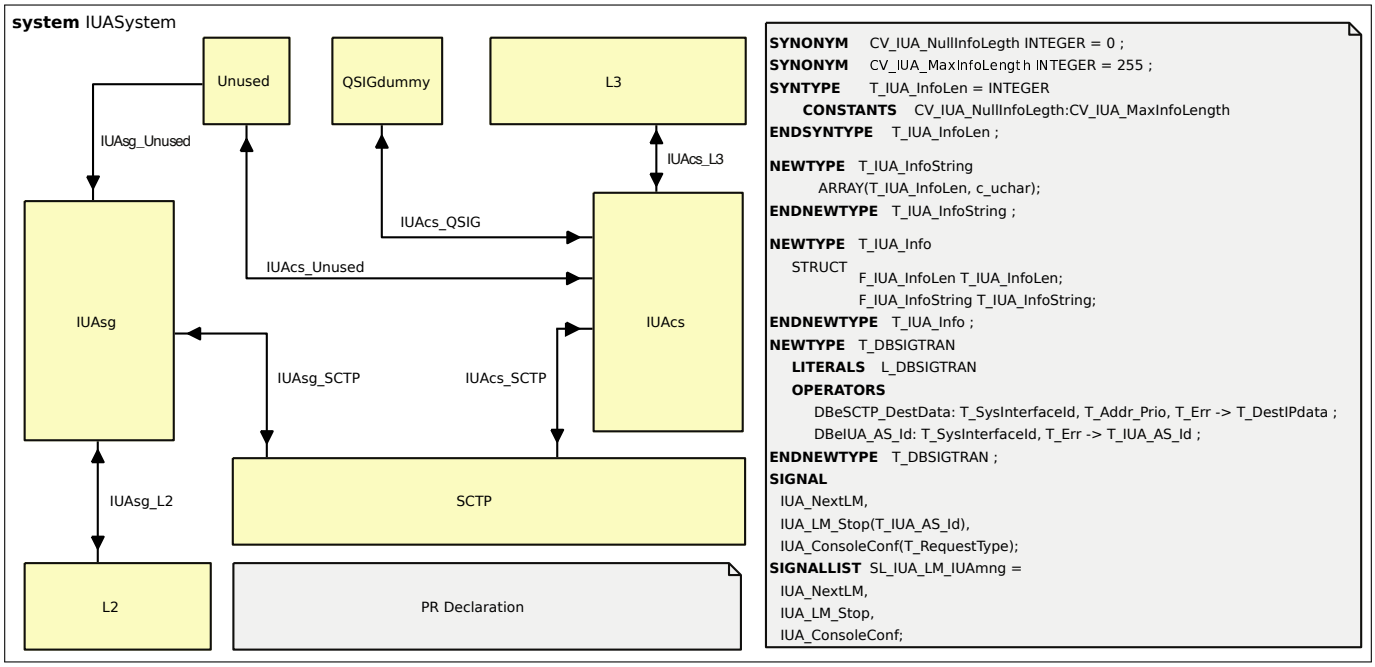

Fig. 3. SDL System for the verification of IUA Protocol

and L2UserBmng. The former models the behavior of the ISDN terminal that initiates and terminates the ISDN call. The latter models the receiving terminal. Figure 4 shows the Finite State Machine (FSM) that defines the behavior of the process L2UserAmng. It is created at the start of the system. At the beginning of the execution, it waits for 32 seconds for the initialization of the system. The constant value CV_sec defines the number of ticks per second in the used hardware platform. Expiration of the timer TM_Act results in the reception of a signal with the same name. The reception of the signal is followed by the initialization of variables that define the module, port, and interface of the subscriber. Next, the initiation of the ISDN call begins. Immediately after successful call establishment L2UserAmng terminates the connection and kills itself. These actions demonstrate how the abstracted environment defines the behavior of the verification system. The environment can be "cooperating" or "hostile." A "hostile" environment would not follow the expected behavior patterns and would include nondeterminism. The degree of "hostility" influences the number of possible execution paths greatly and can lead to state space explosion problems during the verification. Therefore, we usually start with the "cooperating" environment as presented in Figure 4.

The abstraction of DSS1 L3 was prepared in the block L3. It consists of two static processes—one for each subscriber. Processes include only the basic functionality that is needed for successful call establishment. They are adapted exclusively for the chosen modules, interfaces, and ports. Again, the abstraction of processes is bound to the requirement specification.

The abstraction of the SCTP block is presented in Figure 5. It consists of two static processes and some signal routes that connect processes to channels. Processes model the transfer of IUA primitives between the SG and MGC and abstract the lower layers from the original specification (Figure 1). They are replaced by signal route MngA_MngB which enables the direct transfer of signals between the SG and MGC. Other signal routes are derived from the original specification. We did not change the communication infrastructure of the blocks under inspection, e.g., IUAsg and IUAcs. Consequently, the abstraction of surrounding blocks must include all channels and signal routes that are tied to these processes. We had to provide proper termination for all channels that connect to SDL blocks. QSIGdummy and Unused terminate channels from SDL blocks that are not included in the verification system. Both have a static process which implements the functionality of a black hole—in SDL terminology this is called implicit transition.

To build a complete and valid SDL system, we must support interface primitives and declarations for all included blocks. First, all relevant SYNONYM, SYNTYPE, NEWTYPE, SIGNAL, SIGNAL LIST, and OPERATOR declarations should be identified and included in the new system. Additionally, we must include data types that define parameters of timers, formal parameters of processes and procedures. A lot of these constructs are defined in the upper levels of the original SDL systems. Remember that IUAsg and IUAcs are from different systems and consist of $931,539$ lines of code. Therefore, this is not an easy task. We managed to develop many algorithms that take into account the inheritance rules and automate almost the whole process. Table I shows that a significant percentage of declarations in the IUASystem had to be transferred from the upper levels of the original SDL system.

These activities conclude the preparation of the SDL specification for the verification. Table II compares lines of code without comments and blank lines for both products and

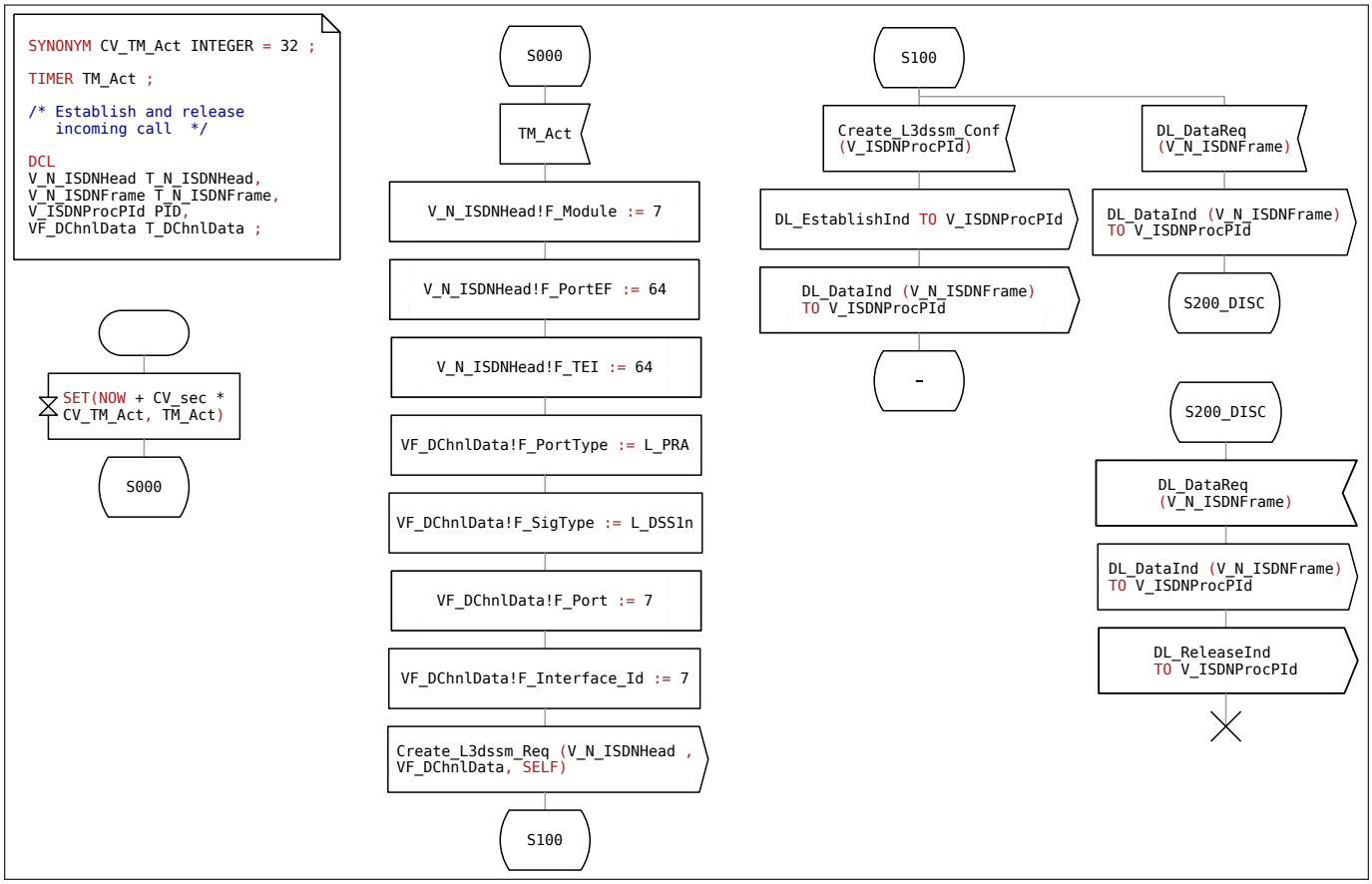

Fig. 4. Finite State Machine of the process L2UserAmng

TABLE I
IUASYSTEM DECLARATIONS

Declaration	Total	Upper levels	Percentage
SYNONYM	361	144	40%
SYNTYPE	129	85	66%
NEWTYPE	100	65	65%
SIGNAL	144	75	52%
SIGNAL LIST	17	11	64%
OPERATOR	87	58	67%

TABLE II
LINES OF CODE WITHOUT COMMENTS AND BLANK LINES

Block	SG	MGC	IUASystem
L1 + L2	12,678	0	126
L3	883	27,254	160
ILLP + IP	5,025	5,062	0
SCTP	4,024	3,997	279
IUA	14,522	14,078	26,204
Complete SDL System	193,397	738,142	28,563

emphasizes blocks that are included in the verification system. If we take into account only these blocks, we see that abstraction reduced the size of the system to 27 % of the original specification.

B. ADT operators with C code

External ADT operators are an important part of the system. They are useful when implementation in C can be more efficient (e.g., parser) or we need to access some system-related functions such as string manipulations and access to the database. Current versions of Spin support embedding of C code into Promela. However, it should be noted that extended models cannot be simulated and that they are open to the full power, and also the dangers of arbitrary C code. The embedded code fragments cannot be checked by Spin, either in the parsing phase or the verification phase. They are trusted to be without errors and copied through from the text of the model into the code of the verifier that Spin generates [12].

Algorithms for the automatic inclusion of embedded C code into a Promela model of the SDL system are given in [9]. Only the simplified description will be presented in this section.

Construction of the skeleton for the inclusion of external ADT operator into a model of the system is based on its SDL declaration. We use the *sdl2pml* tool to automate this process. Next, it is up to the verification engineer to define the extent of abstraction for each operator—he/she can decide to include complete C implementation of the operator or prepare its abstraction in Promela.

After careful analysis of ADT operators in the IUASystem, we decided to use abstraction for all operators. Figure 6 shows a skeleton that was prepared by the *sd2pml* tool for one of the

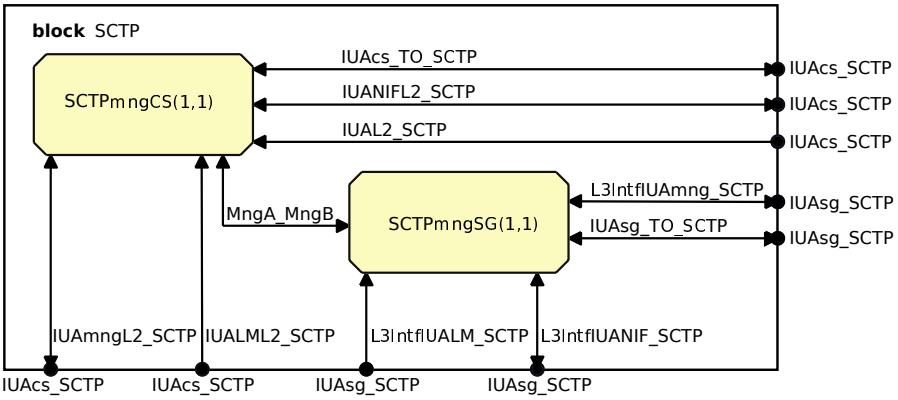

Fig. 5. Block SCTP

```
[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. 6. Skeleton for the abstraction of operator

operators that access the database. The skeleton consists of a header and a body. The header is written between the square brackets 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 or not.

The body for operator abstraction starts with a reserved word `STARTPMLBODY`. The operator can have input and output parameters of different data types. The suffixes of reserved words `$INPUT` and `$OUTPUT` define the operator's reference number. They ensure that the parameter's reference is independent of the actual parameter's name. When the operator is included in the model, the parameter reference, e.g., `$INPUT_1`, is replaced by the actual parameter name. `ENDPMLBODY` marks the end of the operator's body. A detailed description of the skeleton and explanation of an example can be found in [10].

After abstraction of all ADT operators, an automated extraction of the model was performed with the use of *sdl2pml*. The tool implements formally specified algorithms that define sound modeling of the SDL specification in standard Promela.

C. Sound modeling of SDL specification

Sound modeling of the SDL specification in Promela required some innovative solutions. There are some fundamental differences between these languages, e.g., SDL supports hierarchical structure, whereas Promela uses flat design. Some data types and many SDL constructs cannot be represented trivially in Promela. Therefore, properties that are required for proper modeling of an SDL object should include additional information about the object's position within the hierarchy of the system, inherited default values, communication infrastructure data, etc. [8].

The most important parts of our approach can be grouped as follows:

- selection of the relevant attributes for all SDL data types and constructs and their formal definition,
- algorithm-based analysis of the specification,
- sound modeling of predefined and user-defined data types,
- sound modeling of SDL constructs,
- sound modeling of communication,
- sound modeling of rational numbers,
- sound modeling of procedures,
- a new discrete-time model for Promela,
- algorithm-based mechanical extraction of a Promela model with probes.

A detailed description of our approach is out of the scope of this article. An overview of modeling predefined and user-defined data, hierarchical structure, processes, communication, and use of probes is given in [8]. An introduction to modeling rational numbers, procedures and presentation of a new Discrete Time Model for Promela (DTMP) follows.

Telecommunication systems describe complex behaviors with many distributed and concurrent activities. In SDL specifications rational numbers are used quite frequently—data types REAL, TIME, or DURATION. Promela supports only integer values. Therefore, rational values should be modeled. There are at least three possible approaches:

- abstraction with the use of integer data types,
- use of Promela primitives for the inclusion of C code,
- the introduction of rational data type to Promela.

The abstraction of rational values with the use of integer data types can cause that the prepared models are not sound. This fact is problematic especially for variables of data types REAL and DURATION that are used with timers—the order of timeouts can be changed. The advantage of the second approach is an easier implementation of the arithmetic operations that result in smaller models. However, simulation of these

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2685238, IEEE Access

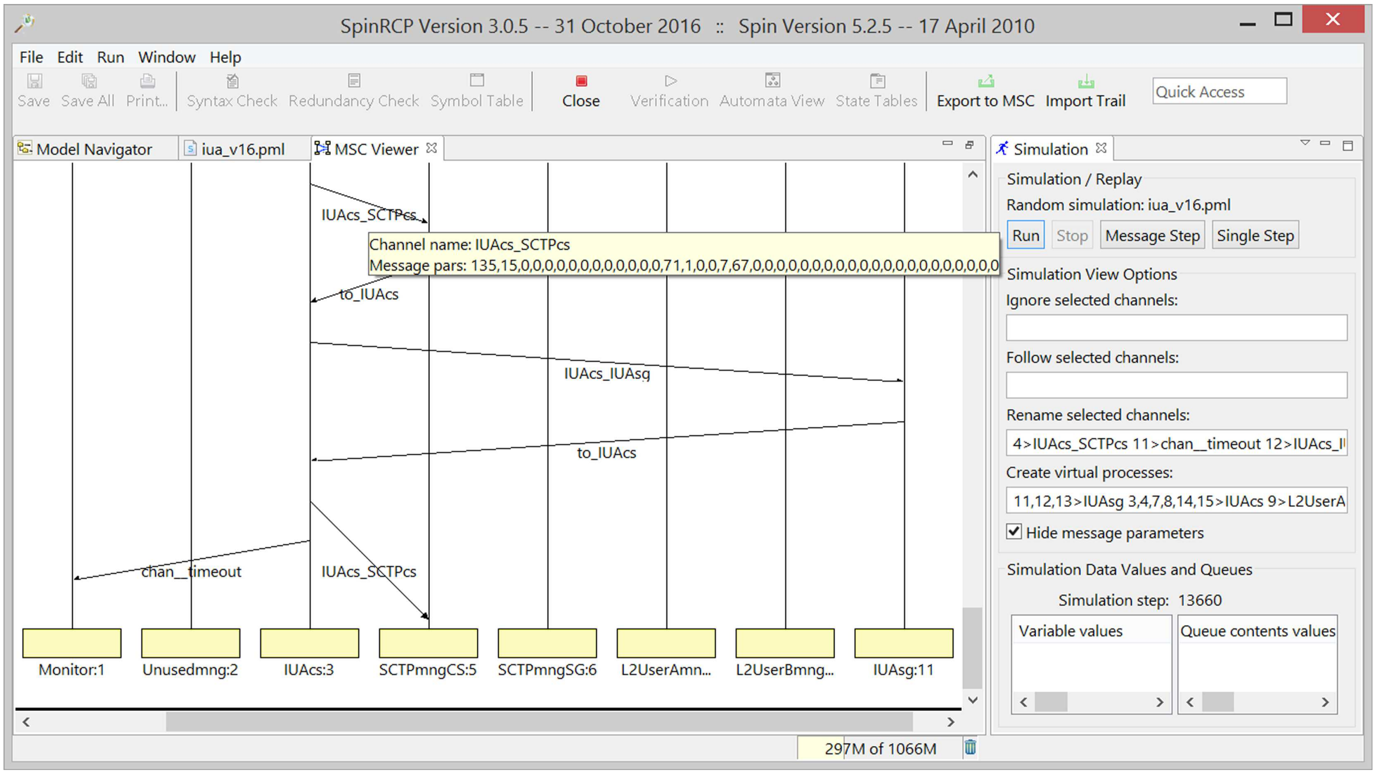

Fig. 7. Random simulation with *SpinRCP*

models would not be possible. We believe that simulation that is guided by the results of the formal verification is crucial. Presentation of a counterexample is essential for real understanding why the model does not satisfy the requirement. Therefore, we decided to introduce rational data type to Promela [9]. We modeled it with two integer values. We have prepared sound models for all operators that are defined by the SDL standard: $+$, $-$, $*$, $/$, $=$, $/=$, $>$, $>=$, $<$, $<=$, $:=$, $Float$, and Fix.

In SDL, a procedure represents an FSM that can be used within different processes to perform some task. It supports formal parameters and variables. The latter are local variables within the procedure that can neither be revealed, nor viewed, exported, or imported. They are created when the procedure start node is interpreted, and they cease to exist when the return node of the procedure graph is interpreted [25]. Specifications use procedures quite frequently and they can perform quite complex tasks. There are at least three approaches to modeling procedures in Promela:

- use of Promela `proctype` definitions,
- inline inclusion in the model of the calling process, and
- use of Promela macro definitions.

Use of `proctype` definitions can be problematic when we model SDL specifications with a large number of processes and procedures, because Promela limits the number of active processes to 255 [7]. If we include the procedure body in the model of the calling process for each procedure call, we increase the size of the model dramatically and reduce its readability. This fact is important when we are searching for reasons for unexpected execution of the model. During the preparation of the model of the IUA protocol, we came to the conclusion that the third approach was best suited for our needs [9].

In SDL, timer constructs are usually used to handle some time-dependable events or unusual behaviors. Promela does not have any notion of time. Consequently, it does not support timer definitions. *Sdl2pml* implements a DTMP that was proposed in [9]. Each `proctype` can include timers that are presented with a unique signal name. We introduced a Monitor process that manages all active timers in the model of the system. At each timeout event, more than one timer can expire, and the Monitor process sends timer signals to all relevant processes. Communication between processes and the Monitor is synchronous. Experimental results confirmed that this approach is suitable for:

- systems that use timers frequently and
- timers with a large range of values.

We have used DTMP successfully in the automated extraction of a model for the IUASystem.

V. Simulation and Formal Verification

After automatic extraction of the model, called `iua_v16.pml`, we use our *SpinRCP* integrated development environment for inspection of Promela code, its syntax check, simulation, and formal verification with the Spin model checker [21], [20]. Figure 7 shows a small part of the Message Sequence Chart (MSC) drawn in the MSC Viewer and the Simulation View of the *SpinRCP* during

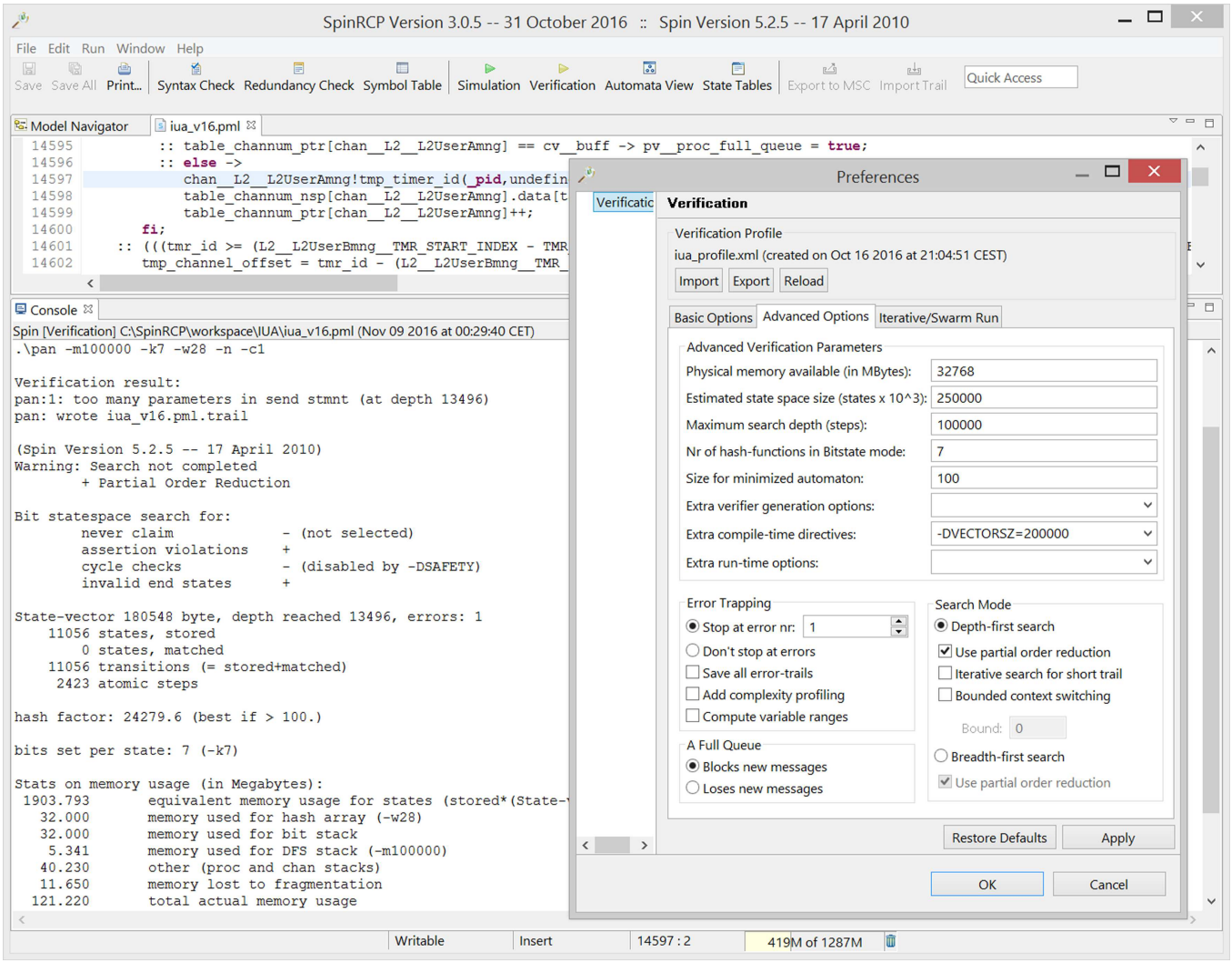

Fig. 8. Formal verification with Spin and *SpinRCP*

random simulation of the model. To facilitate the analysis of this MSC we used two options that are available in the Simulation View. The first one was to create virtual processes by joining two or more origin processes. Such abstraction is very useful if a user wants to analyze huge MSCs in a top-down manner to hide details that are not interesting at a given abstraction level and would otherwise contain an unnecessary complexity. We joined all three processes within the signaling gateway (IUAsg__IUAmng, IUAsg__IUA_LM, and IUAsg__IUAprot) into a virtual process IUAsg and all processes within the gateway controller (IUAcs__IUAmng, IUAcs__IUA_LM, and IUAcs__IUAprot) together with the surrounding processes (L3__L3netA, L3__L3netB, and QSIGdummy__QSIGmng) into a virtual process IUAcs. For the sake of MSC clarity, we shortened the names of several other origin processes by using the same option with an identification of the process before the ”>” character and the shortened name of the process afterward. The second Simulation View option was used to rename selected channels from numerical identifications to self-descriptive names. The messages transferred across channels between a sending and a receiving process have many parameters. To avoid lack of clarity in MSCs, we decided to hide message parameters by checking the corresponding checkbox in the Simulation View. However, if the user hovers the mouse pointer over the channel name as displayed for channel IUAcs_SCTPcs in Figure 7, message parameters will disclose.

In the past, we have run the simulation of the IUA model by using XSpin graphical interface and managed to discover an unknown, invalid execution path [10]. Some additional unexpected execution paths were found during the initial random simulation. It was revealed that they were caused by the model of the environment or the prepared abstractions for the external operators. This showed that mechanical extraction of the Promela model was successful. After that, we tried to perform formal verification of the extracted model. The Spin generated the verifier file `pan.c` in the length of more than 288,000 lines of C code for this model successfully, but the GCC compiler could not compile it. Current versions of compiler accomplish this task, thus providing us with the

opportunity for more detailed analysis of the model.

First, we tried to check basic safety properties such as assertion violations and invalid end states for the extracted model. In models that are based on complex real-life implementations, it is expected that the prepared environment for formal verification will not include all possible configurations. Therefore, reporting of unreachable code is not meaningful. Due to the huge size of the model, the exhaustive storage mode was not feasible and we had to resort to the bitstate hashing method. Within the *SpinRCP* integrated development environment, all verification parameters were set or selected in the Verification Preference Page. After verification being started, *SpinRCP* first ran Spin to generate a model-specific `pan.c` verificator file, then called the C compiler to generate the executable file `pan`, and last, it runs the verificator `pan` itself. The result of the first run of formal verification of `iua_v16.pml` with Spin within the *SpinRCP* integrated development environment is shown in Figure 8. In front of the *SpinRCP* window, the Verification Preference Page with the Advanced Options tag is opened. To cope with the huge model we increased the values of physical memory available to 32,768 (32 GiB), estimated state space size to 250,000, maximum search depth to 100,000 steps and maximum size for the state vector to 200,000. Other parameters were left at their default values, e.g. depth-first search with partial order reduction and stop the verification after the first error found. We experimented with two different compilation modes—without any optimization and with the `-O1` optimization option. Compilation without optimization took about 25 minutes and produced an executable verifier `pan` of the size 133 MiB. On the other hand, compiling with the `-O1` optimization option lasted over 9 hours, but produced a more compact code of the size 88 MiB. In addition, the optimized executable code was more than five times faster.

The very first run of formal verification found a flaw in the model. It reported an error "`too many parameters in send stmnt (at depth 13,496)`". The generated counterexample execution trail was used for guided simulation of the model. It revealed that an error was triggered by timer expiration in the Promela `proctype` L2__L2UserAmng which is the mechanically extracted model of the SDL process L2UserAmng (Figure 4). Modeling of communication in Promela requires that channels which are associated with the model of the SDL process include the union of all data types that can be used in receiving SDL signals. Process L2UserAmng can receive signals TM_ACT with a data type TIMER, CreateL3dssm_Conf with data type PID, DL_DataReq and DL_DataReq with data type T_N_ISDNFrame. Analysis showed that channel declaration was made correctly, but the timer expiration send statement used variable undefined__T_N_ISDNHead, which is of the wrong data type. Typedef T_N_ISDNFrame includes two elements, while T_N_ISDNHead has four. Therefore, formal verification has produced a real error in the model of the system. This variable is only a placeholder in the timer expiration model. It enables the send statement to conform to the definition of the channel but does not carry any data which is also indicated with the prefix "undefined." After we had replaced it with undefined__T_N_ISDNFrame, the verification did not report any additional problems with parameters in the send statement. The state vector containing the complete description of a single global system state of the corrected model required 182,108 bytes. The longest depth-first search path reached 22,633 transitions from the initial system state.

The total number of reachable states for this model is unknown. To at least estimate it, we ran subsequent verifications for varying hash-arena sizes from 1 MiB (runtime parameter `-w23`) to 128 MiB (`-w30`) and two different numbers of hash functions (runtime parameters `-k3` and `-k7`). The number of unique states reached, verification runtime, hash factor (the ratio between the size of hash-array in bits and the number of states stored), and the total number of memory usage are shown in Figure 9, Figure 10, Figure 11, and Figure 12, respectively. Taking into account the logarithmic scale on the abscissa, we can find out that if the size of the hash-array is doubled, both the states reached and the runtime increase a little less than a factor of two. It means that we are still far away from the complete coverage. Another reason for this conclusion is the fact that the hash factor is low and increases only slowly for increasing the hash size (Figure 11). For a very reliable hash operation (low probabilities of hash collisions) the hash factor should be over 100. From the graphs in Figure 10 and Figure 12, it is evident that the verification bottleneck is not the available amount of memory but the acceptable runtime. For example, by using the hash-array of the size 128 MiB, only a little bit more than 300 MiB was consumed, which is less than 1% of the available physical memory available in our system (32 GiB), but the verification took about 12 hours (for `-k3`) to complete. We ran the verification also with a larger hash-array, but we canceled it after 4 days and 6 hours when 3.893 billion states had been reached. It was interesting that the depth reached increased to 22,902 after 1.712 billion unique states had been visited and stored.

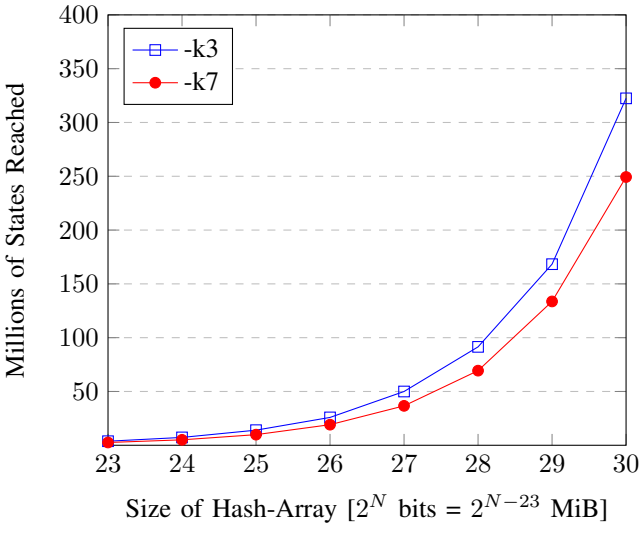

Fig. 9. Number of unique states reached for varying hash-arena sizes

A reasonable solution for tackling the computing complexity is to use a swarm verification [26], i.e. to run a great number of smaller verification tasks on multi-core CPUs in parallel with highly randomized parameters to spread the

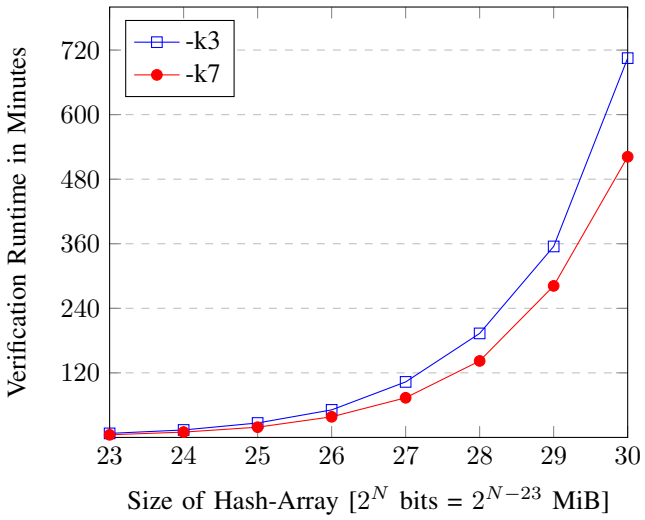

Fig. 10. Verification runtime for varying hash-arena sizes

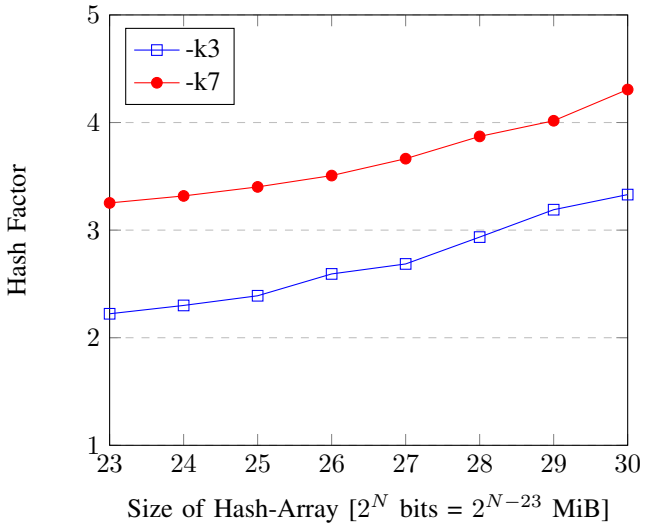

Fig. 11. Hash factor for varying hash-arena sizes

search to different parts of the giant search space. Spin swarm verification is supported in *SpinRCP* from Spin Version 6.4. However, currently, mechanically extracted models with *sdl2pml* can be analyzed fully with the mainstream versions of *Spin* up to version 5.2.5. Later *Spin* versions require modifications of some algorithms in *sdl2pml*. We use the name of the array (scalar) as a placeholder for the entire array in inline calls, e.g., parameter x is used in the body of the inline as `x[offset]`. This triggers error "`arrays were referenced as scalars`." because a change has been made in the Spin Version 6 syntax check to prevent accidental references to an array as a scalar which can be very tough to diagnose. Additionally, we do not differentiate between `x` and `x[offset]` when the offset is 0. This fact enabled the use of the same communication modeling algorithms for singleton SDL processes and those with multiple instances. Now these constructs require different Promela interpretation and are the focus of our current work. Furthermore, some changes will be

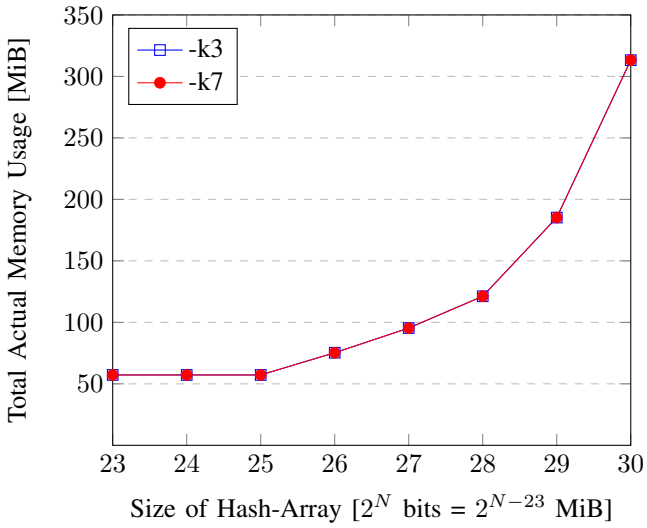

Fig. 12. Total Actual Memory Usage for varying hash-arena sizes

necessary when defining channel assertions.

VI. Conclusion

Automating model extraction for an SDL specification of this size and complexity showed that verification engineers need additional help from the verification tools—especially during the preparation of the environment and analysis of the reported errors from the verifier. We believe that our approach is promising, but there is still much room for improvements. We are satisfied that we managed to find an error during the simulation and formal verification of the model.

The model checking technique also verifies the correctness of the mechanically extracted model. The verification engineer can distinguish between real errors and false positives. With the help of the error trail, we can trace back to the actual source of the problem as was shown in this paper. If soundness of the model is compromised, improvements of the algorithms that define the automated extraction of the model are necessary.

We are currently developing new algorithms for *sdl2pml* that will enable the use of the latest versions of Spin with included support for inline specification of LTL properties, multiple never claims, parallel breadth-first search algorithm, iterative and swarm verification, and other techniques. Support for the SDL-2010 presents the next great challenge [27], [28]. We will have to extend our DTMP to support activation-delay for the output construct, provide a proper model for priority order, and many other features. However, new developments in SDL standardization present additional motivation. They have the potential to get fully standards-compliant SDL descriptions, with parts that are written in C programming language, and verified formally by Spin with the use of the mechanically extracted model.

Hopefully, we managed to demonstrate that automated extraction of models promises greater utilization of formal techniques for real-life SDL specifications. Due to increasing complexity and human dependence on such systems, each change in the specification of the product should be formally

checked immediately against the requirements specification. Therefore, we strive to provide methods and tools to make this possible.

ACKNOWLEDGMENT

This work was funded partly by the European Regional Development Fund and Slovenian Ministry of Higher Education, Science and Technology under the R&D project Correctness Verification of Communication System Functioning in the framework of the Centre of Excellence for Information and Communication Technologies, Iskratel d.o.o, and 4S d.o.o. The authors acknowledge the financial support from the Slovenian Research Agency (research core funding No. P2-0069) by the research program Advanced methods of interaction in telecommunication.

REFERENCES

[1] International Telecommunication Union, "Z.100: Specification and description language - overview of sdl-2010," 2016, series Z: Programming Languages.
[2] Pragmadev, "SDL-RT specification & description language - real time," 2016. [Online]. Available: http://www.sdl-rt.org
[3] International Telecommunication Union, "Z.104: Specification and description language - data and action language in SDL-2010," 2016, series Z: Programming Languages.
[4] B. Vlaovič, A. Vreže, Z. Brezočnik, and T. Kapus, "Verification of an SDL Specification — a Case Study," *Electrotechnical Review*, vol. 72, pp. 14–21, 2005.
[5] C. Baier and J.-P. Katoen, *Principles of Model Checking*. The MIT Press, 2008.
[6] A. Vreže, B. Vlaovič, Z. Brezočnik, and T. Kapus, "Development of MGCP protocol stack for SI2000 digital switch node," *Electrotechnical Review*, vol. 72, pp. 22–29, 2005.
[7] B. Vlaovič, *Automatic Generation of Models with Probes from the SDL System Specification: Ph.D. dissertation (in Slovene)*. Maribor, Slovenia: University of Maribor, Faculty of Electrical Engineering and Computer Science, September 2004.
[8] B. Vlaovič, A. Vreže, Z. Brezočnik, and T. Kapus, "Automated Generation of Promela Model from SDL Specification," *Computer Standards & Interfaces*, vol. 29, pp. 449–461, 2006. [Online]. Available: http://dx.doi.org/10.1016/j.csi.2006.10.001
[9] A. Vreže, *Extending Automatic Modeling of SDL Specifications in Promela with Embedded C code and a New Model of Discrete Time: Ph.D. dissertation (in Slovene)*. Faculty of Electrical Engineering and Computer Science, University of Maribor, 2006.
[10] A. Vreže, B. Vlaovič, and Z. Brezočnik, "Sdl2pml-Tool for Automated Generation of Promela Model from SDL Specification," *Computer Standards & Interfaces*, vol. 31, pp. 779–786, 2009. [Online]. Available: http://dx.doi.org/10.1016/j.csi.2008.09.005
[11] D. Bošnački and D. Dams, "Discrete Time Promela and Spin," *Lecture Notes in Computer Science*, pp. 307–310, 1998.
[12] G. J. Holzmann, *The SPIN model checker: Primer and reference manual*. Addison Wesley, 2003.
[13] G. J. Holzmann and J. Patti, "Validating SDL specifications: an experiment," in *Proceedings of the 9th International Workshop on Protocol Specification, Testing, and Verification, Enschede, The Netherlands, 6-9 June, 1989*. North-Holland, 1989, pp. 317–326.
[14] H. Tuominen, "Embedding a Dialect of SDL in PROMELA," *6th International SPIN Workshop on Model Checking of Software, Lecture Notes in Computer Science*, pp. 245–260, 1999. [Online]. Available: http://dx.doi.org/10.1007/3-540-48234-2_19
[15] D. Bošnački, D. Dams, L. Holenderski, and N. Sidorova, "Model Checking SDL with Spin," *Lecture Notes in Computer Science*, pp. 363–377, 2000.
[16] G. J. Holzmann and M. H. Smith, "Software model checking: extracting verification models from source code," *Software testing, verification, and reliability*, vol. 11, no. 2, pp. 65–79, 2001.
[17] K. Jiang and B. Jonnson, "Using SPIN to model check concurrent algorithms, using a translation from C to Promela," in *Proceedings of the 2nd Swedish Workshop on Multi-Core Computing, Uppsala, Sweden, November 26–27*, 2009.
[18] K. Havelund and T. Pressburger, "Model checking Java programs using Java PathFinder," *International Journal on Software Tools for Technology Transfer*, vol. 2, no. 4, pp. 366–381, April 2000.
[19] T. Kovše, B. Vlaovič, A. Vreže, and Z. Brezočnik, "Eclipse plug-in for Spin and st2msc tools - tool presentation," in *Proceedings of the 16th International SPIN Workshop, Grenoble, France, June 26-28, 2009*. Springer Berlin Heidelberg, 2009, pp. 143–147. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-02652-2_13
[20] Z. Brezočnik, B. Vlaovič, Boštjan, and A. Vreže, "SpinRCP: The Eclipse rich client platform integrated development environment for the Spin model checker," in *Proceedings of the 2014 International SPIN Symposium on Model Checking of Software*. New York, NY, USA: ACM, 2014, pp. 125–128. [Online]. Available: http://doi.acm.org/10.1145/2632362.2632380
[21] Z. Brezočnik, B. Vlaovič, and A. Vreže, "Model checking using Spin and SpinRCP," *Informacije MIDEM, Journal of Microelectronics, Electronic Components and Materials*, vol. 43, no. 4, pp. 235–250, 2013. [Online]. Available: http://www.midem-drustvo.si/Journal%20papers/MIDEM_43(2013)4p235.pdf
[22] Z. Brezočnik and T. Kovše, "SpinRCP," 2015, Faculty of Electrical Engineering and Computer Science, University of Maribor, Slovenia. [Online]. Available: http://lms.uni-mb.si/spinrcp/
[23] IETF Network Working Group, "Framework Architecture for Signaling Transport," 1999.
[24] IETF Network Working Group, "Integrated Services Digital Network (ISDN) Q.921-User Adaptation Layer," 2006.
[25] ITU-T Recommendation Z.100, "CCITT Specification and Description Language (SDL)," 1993, series Z: Programming Languages.
[26] G. J. Holzmann, R. Joshi, and G. Alex, "Swarm Verification Techniques," *IEEE Transactions on Software Engineering*, vol. 37, no. 6, pp. 845–857, 2011.
[27] International Telecommunication Union, "Z.101: Specification and description language - basic sdl–2010," April 2016, series Z: Programming Languages.
[28] International Telecommunication Union, "Z.102: Specification and description language - comprehensive sdl-2010," April 2016, series Z: Programming Languages.

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 an Assistant Professor at the same institution. His main research areas are in the field of formal verification. His special interests cover formal protocol verification with model checking, especially automated extraction of models from the SDL-specification.

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. His main research areas are in the field of formal verification. His special interests cover formal protocol verification with model checking, especially automated extraction 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 a Full Professor, Head of the Laboratory for Microcomputer Systems, and Deputy Head of the Institute of Electronics and Telecommunications at the same institution. His main research areas are formal methods and tools for software and protocol verification and binary decision diagrams.