# Toward Automatic Generation of Promela Models from SDL Specification

Boštjan Vlaovič, Aleksander Vreže, Zmago Brezočnik, Tatjana Kapus
University of Maribor, Faculty of EE & CS, Maribor, Slovenia
E-mail: {bostjan.vlaovic, aleksander.vreze, brezocnik, kapus}@uni-mb.si

***Abstract*— This paper presents our research in the domain of mechanical extraction of a model from an SDL (Specification and Description Language) specification of a system. We use formal verification tool Spin (Simple Promela Interpreter) and Promela (Process Meta-Language) language for the description of the model. With the model checking technique the model's accordance with the system correctness requirements can be established with mathematical accuracy. The model can be generated manually or mechanically. If it is to be prepared manually, we will need an expert with the detailed knowledge of the system and both languages. The quality of the model is directly influenced by the expert. The process is prone to the incorrect modelling of the system's properties. In this paper we present the most critical parts of mechanical creation of the models in Promela. Additionally, we present challenges, research directions and some solutions for the automatic generation of models from an SDL specification.**

## I. INTRODUCTION

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

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

For successful formal verification of an SDL specification a model has to be prepared for the chosen verification program. In this paper we use Spin (Simple Promela Interpreter) with its input language Promela (Process Meta-Language). Promela adopts a strong formal basis established, like SDL, in ECFSM theory. The similar basis of SDL and Promela makes the translation between different representations feasible. Similar to SDL, Promela allows dynamic creation of concurrent processes. The description of a concurrent system consists of one or more user-defined process templates or proctype definitions and at least one process instantiation. Process templates are used to define a finite automata of the system. Computation of asynchronous interleaving product of the automata gives global behaviour of the system (state-space of the system, reachability graph).

Spin is a tool for analyzing logical correctness of concurrent systems, specifically of data communications protocols. Its first version under the name Pan appeared in 1980. Currently, in its $4^{th}$ version, it is a mature project with a lot of users and contributors. Given the system model in Promela, Spin can perform random, interactive, or guided simulation of the system executions. Further, it can generate a verifier in C code which performs online verification of the system's correctness properties (safety properties, liveness properties, and general temporal properties). It checks for the absence of deadlocks, unspecified receptions, unexecutable code, and it can find non-progress execution cycles. It supports efficient model checking, invariant assertions, and temporal properties expressed in a subset of LTL (Linear Temporal Logic). Checking of system properties requires the use of probes in the model. Probes provide us with an insight to the model execution and are mostly present as assertions on special variables.

This paper is organized as follows. Section 2 defines the problem. In Section 3, specification of the V.76 protocol in SDL is presented. Section 4 presents the mechanical generation of models. We discuss the modelling of SDL data types, processes, variables, communication, and timers. We conclude with a discussion and directions for further research.

## II. PROBLEM DEFINITION

A model of an SDL specification can be generated manually or mechanically. If it is to be prepared manually, we will need an expert with the detailed knowledge of the system and the language. The quality of the model is directly influenced by the expert. The process is prone to the incorrect modelling of the system's properties. Therefore, automatic generation of models from the SDL specification would help with the introduction of formal methods to the development cycle. Different approaches to generation of models from SDL specifications are described in [1], [2], [3], [4], [5], [6], [7], [8].

In this paper specification of the V.76 protocol from [9] will be used as an example. Using this SDL specification we will try to show the steps needed to automatically generate a valid verification model and present an overview of research results from [10] for the solution of some issues.

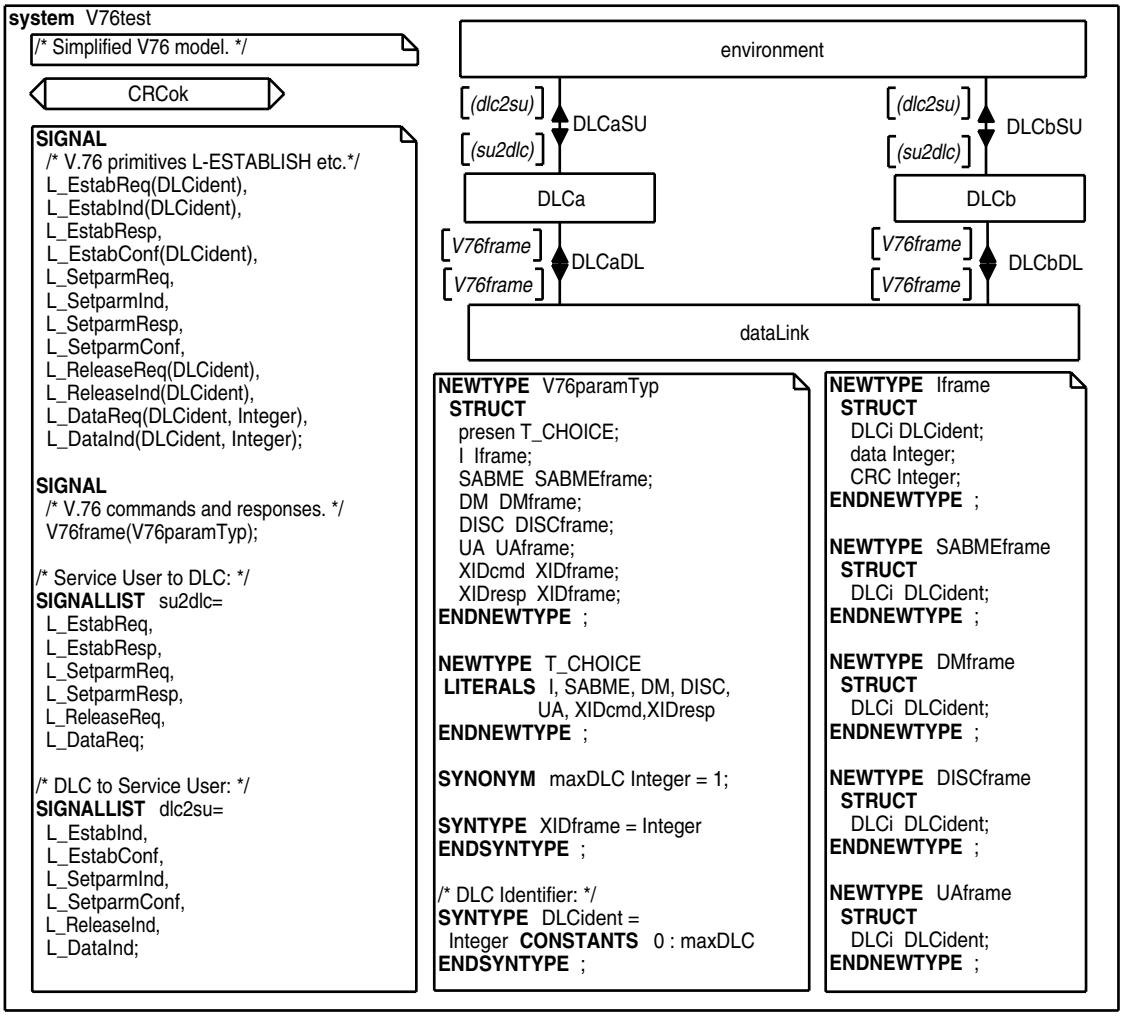


Fig. 1. Graphical description of V.76 protocol and its environment

## III. SPECIFICATION OF V.76 PROTOCOL

The system used in [9] is a simplified version of the protocol described in [11]. ITU-T V.76 Recommendation describes a protocol to establish Data Link Connections (DLCs) between two modems and to transfer data over these connections (Fig. 3). The V.76 SDL specification and associated files can be downloaded in ObjectGeode and Tau SDL formats from the Internet [9]. The specification from [9] includes one ASN.1 (Abstract Syntax Notation One) data type definition. We decided to replace it with SDL96-compliant definition of data types to avoid the ASN.1 extensions of the Z.100 standard (Fig. 1).

Fig. 3 shows communication between two service users. Several connections can exist in parallel: SUA may establish DLC number 0 to transmit voice and DLC number 1 to transmit data to or from SUB. A request on one side is generally followed by an indication on the other side of the connection. Fig. 2 shows four stages of the expected connection. In the first stage SUs can optionally perform exchange identification procedures. Next, establishment of a data link connection is expected. On receipt of an L-ESTABLISH request primitive (`L_EstabReq`) from its SU, the V.76 shall attempt to establish the DLC.

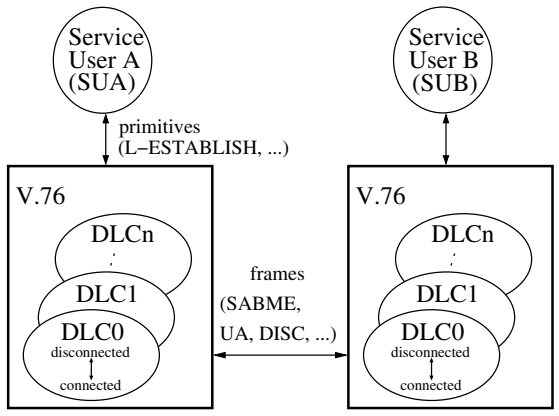


Fig. 3. Communication between SUA and SUB through V.76 protocol

The DLC entity transmits a Set Asynchronous Balanced Mode Extended (SABME) frame, the retransmission counter is reset, and timer T320 [11] is started [9]. If a peer DLC entity, based on the response from its SU (`L_EstabResp`), is able to establish the DLC, it shall respond with Unnumbered Acknowledge (`UA`) and enter the connected state. Otherwise it should respond

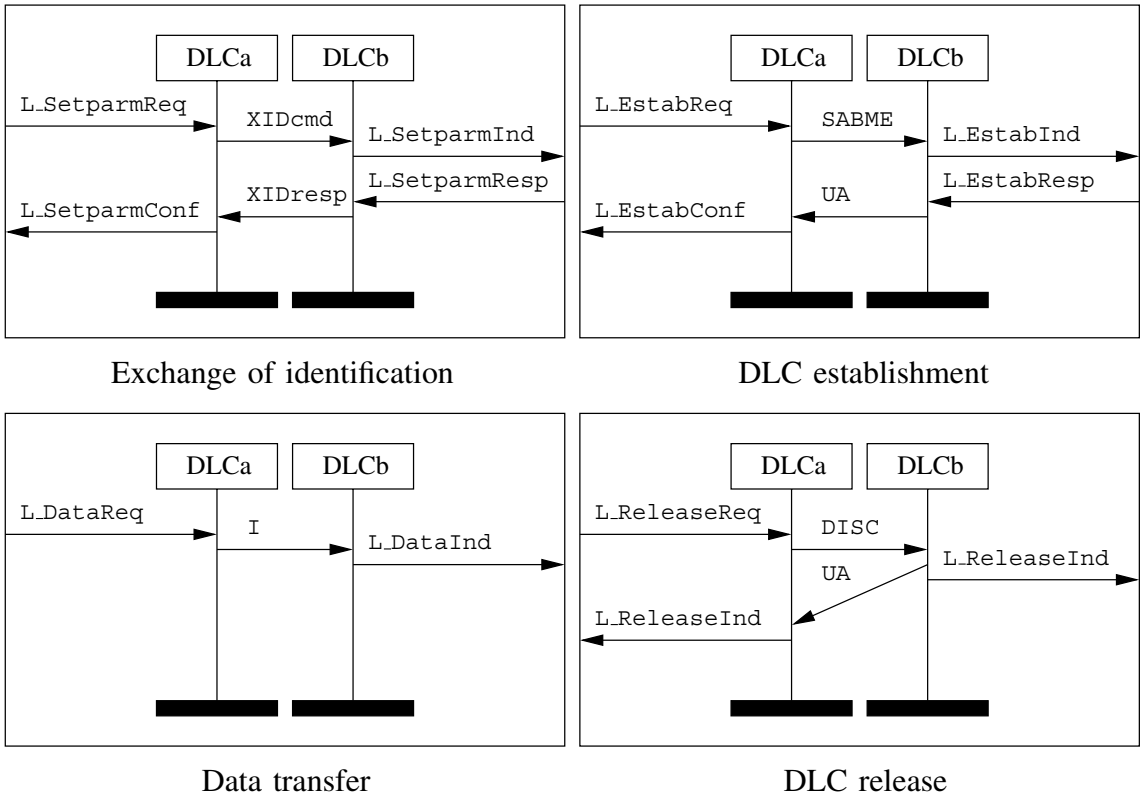


Fig. 2. Four stages of V.76 protocol connection between SUs

with Disconnect Mode (DM). Once in the connected state, information transfer may begin. DLC receives data from SU with the use of an L-DATA request primitive (`L_DataReq`). Data are transmitted in an I frame. Structure of the I frame is shown in Fig. 1 with the definition of data type `Iframe`. Communication is terminated with the L-RELEASE request primitive (`L_ReleaseReq`) from any SU. Description of the protocol in greater detail is outside the scope of this paper and can be found in [9], [11].

Since verification with Spin requires a complete system, we included environment processes `SUa` and `SUb`, which can receive and send all signals that are defined at the channels `DLCaSU` and `DLCbSU`. Fig. 1 shows a graphical description of the `V76test` SDL system that is used in this paper as a reference SDL specification. Blocks `DLCa` and `DLCb` describe the V.76 protocol and are identical. Each block consists of two processes—`dispatch` and `DLC` (Fig. 4). Process `dispatch` creates new DLCs on request (`L_EstabReq`) from SU or refuses new connections with `L_ReleaseInd`. Each DLC is managed by its own `DLC` process. Unreliable transfer of frames is modelled in the `dataLink` block (Fig. 5). All choices that were in [9] specified as an informal text in processes `AtoB` and `BtoA` were replaced with the nondeterministic decision (Fig. 6). No other changes to the original system specification were made.

## IV. AUTOMATIC GENERATION OF MODELS

The SDL specification of the V.76 protocol includes most of the elements that are used in real-life specifications in the industry:

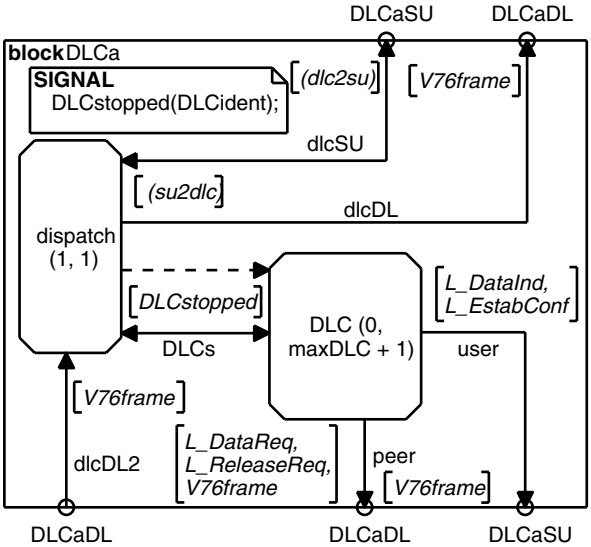


Fig. 4. Graphical description of DLCa process

- SDL data types and expressions,
- dynamic process creation,
- priority signal,
- implicit transition,
- spontaneous transition,
- save construct,
- priority input,
- enabling condition,

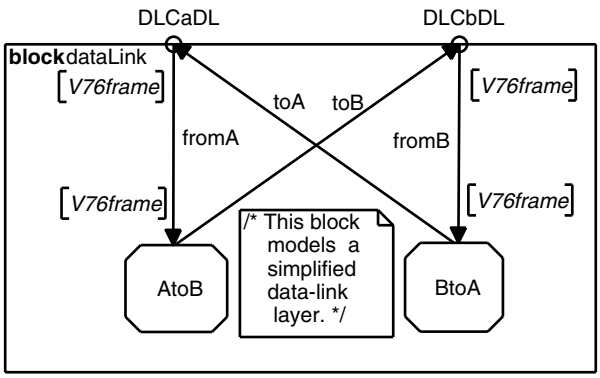


Fig. 5. Block `dataLink`

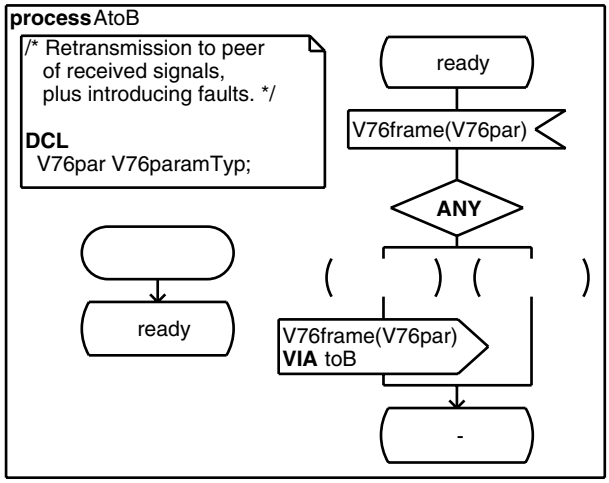


Fig. 6. Process `AtoB`

- asterisk input,
- direct (PId) and indirect addressing (name of the process, name of the signal route),
- path limitations introduced by the via statement,
- asterisk state,
- timers and
- procedures.

Most of them can not be properly modelled with existing approaches. We believe that their support in the mechanical generation of models is crucial for the recognition of formal verification as an equally important step of the product development activities in the industry.

### A. Data Types

Data in SDL are based on the concept of ADT (Abstract Data Type). ASN.1 can also be used as a data type notation in combination with SDL, but it is not included in our research activities. An ADT describes the functional properties of data objects. It defines the result of operations on data objects without constraining how this result is obtained. The latter is up to the implementation of a data object [12]. Predefined data sorts and data generators are defined in Annex D to the recommendation Z.100 [13].

Our algorithms for modelling the predefined data types and operators are described in [10]. We define algorithms for automatic modelling of *INTEGER, BOOLEAN, PId, NATURAL, CHARACTER, CHARSTRING, REAL, TIME,* and *DURATION* data types. All rational numbers are currently modelled with integers. Their accurate modelling is part of our current research activities [14].

```
#define   NUL          0
#define   SOH          1
          .
          .
          .
#define   ZERO        48
#define   ONE         49
          .
          .
          .
#define   ASTERISK    42
#define   PLUS        49
          .
          .
          .
#define   CHAR_A      65
#define   CHAR_B      65
```

Fig. 7. Some definitions of *CHARACTER* data type literals

For the `CHARACTER` data type we use the Promela `#define` directive for all literals that are defined in [13] (Fig. 7). Data type *CHARSTRING* is by default limited to 32 characters. Fig. 8 shows modelling of both data type and `strcpy` operator with the use of the inline statement.

```
typedef pt__character {unsigned char : 7 = 0}

#define pcv__charstring_max 33

typedef pt__charstring {byte char[pcv__charstring_max]}

inline strcpy(pfv__stringB,pfv__stringA){
  d_step{
    pfv__tmp=1;
    do
      :: ((pfv__tmp<=pcv__charstring_max) &&
            pfv__stringA.char[pfv__tmp]!=NUL) ->
            pfv__stringB.char[pfv__tmp]=
            pfv__stringA.char[pfv__tmp];pfv__tmp++;
      :: else -> break
    od
  }
}
```

Fig. 8. Modelling of *CHARSTRING* data type and `strcpy` operator

Fig. 9 shows the model of the `T_CHOICE` and `Iframe` data types defined in the `V76test` system (Fig. 1). During the static analysis of the system we check the range of each data type and try to optimize its representation in the Promela model. Explicitly declared literals are defined as constant values. Such representation allows redefinition of some predefined values such as 1, 2, ...

Each SDL ADT can contain one or more operators. Tools used in the industry provide interfaces and skeletons for the implementation of the ADT operators in C programming language. External files can include complex functions and define new data structures and header files. One of the reasons we decided to use Spin is its support for embedded C code in a model ([15], [16]). Every SDL specification from our industry projects uses such operators.

```
typedef T_CHOICE {byte val}

#define T_CHOICE__I 1
#define T_CHOICE__SABME 2
#define T_CHOICE__DM 3
#define T_CHOICE__DISC 4
#define T_CHOICE__UA 5
#define T_CHOICE__XIDcmd 6
#define T_CHOICE__XIDresp 7
#define T_CHOICE__undefined__value 8

typedef Iframe {
  DLCident DLCi;
  int data;
  int CRC
}

typedef DLCident {byte val}
```

Fig. 9. Some definitions of data types from `V76test` system

Mechanical generation of models with support for the external operators presents a big challenge. Our current approach is divided into two phases. First, exhaustive analysis of the SDL specification and C code is performed. This phase involves analysis of data structures, global variables, function calls, and all external files which are included through the ADT operator mechanism. Next, we build a model of the specification [14]. First results of our research are promising.

### *B. Processes and Variables*

SDL processes are communicating extended finite-state machines and can be modelled with Promela proctypes. Algorithms for the automatic generation of SDL process models with the use of Promela's proctype definitions should support dynamic process creation, expressions in parameters, process communication and their termination.

SDL processes are either created at system initialisation time or later in the lifetime of the system by other processes in the same block. An arbitrary number of instances of a process can be active at the same time. Each process instance can be in a different state and therefore react differently to a signal at a given time. The number of processes may change during the lifetime of the system. The specification defines the initial and the maximum number of processes.

When a process instance is created, all variables of the process are also created—including predefined variables `parent`, `offspring`, and `self`. They are set to an initial value if specified. All variables which do not have an initial value remain undefined until they are first assigned a value in a transition [12]. Promela uses a different approach and always initializes all variables to some explicit or implicit predefined values. Special care has to be taken during the generation of the model to explicitly model the undefined value.

In a mechanically generated model, which is based on algorithms presented in [10], initial processes are created by the special process `init`. Fig. 10 shows part of the generated Promela model. The creation of the `AtoB` process demonstrates the interdependence of various parts of the model. The complexity of the model is greatly influenced by the explicit modelling of semantics of the

```
init{
pt__pid offspring;
  atomic{
    if
    :: table__dataLink__AtoB__free <
         table__dataLink__AtoB__max ->
       offspring = run
       dataLink__AtoB(chan__dataLink__AtoB[\
         table__dataLink__AtoB__free],_pid);
       table_pid_channum[offspring] =\
       chan__dataLink__AtoB[\
         table__dataLink__AtoB__free];
       table_pid_channame[offspring] =\
         chan__dataLink__AtoB__select +
       table__dataLink__AtoB__free;
       table_channame_channum[\
         chan__dataLink__AtoB__select+\
         table__dataLink__AtoB__free] =
       chan__dataLink__AtoB[\
         table__dataLink__AtoB__free];
       if
         ::(offspring==0) ->
           pv__runtime_error = true;
         ::(offspring!=0) ->
           table__dataLink__AtoB__free++;
       fi;
    :: else  -> pv__runtime_error = true;
    fi;
                        .
                        .
                        .
  }
}
```

Fig. 10. Part of the generated `init` process

SDL. Detailed explanation of all elements in Fig. 10 and presentation of the process modelling, even for the simple process `AtoB`, is unfortunately outside the scope of this paper.

The biggest remaining challenge for complete modelling of the dynamic process creation is a different interpretation of the process termination in SDL and Promela. An SDL process terminates at the end of its execution. At that time all associated resources are released—most notably PId. Promela distinguishes between the end of the process execution and its termination. Its resources are released at its termination, which can occur only when all younger processes have terminated first. The maximum number of simultaneously running processes is 255. Since each active process is guaranteed to have a unique PId within the system, it can be reused only after the process terminates. This difference creates a problem when an SDL specification is modelled where processes do not terminate in the reverse order of their creation. This issue requires further research activities.

### *C. Communication*

Each SDL process has an associated input queue. In [10] each proctype has an associated channel. The number of defined channels depends on the number of expected process instances during the system execution. If the initial and maximum number of processes is not explicitly specified, the mechanically generated model which is based on the algorithms presented in [10] is not in accordance with [17]. Both values are set to one, while recommendation Z.100 in this case defines the maximum

number to be unlimited.

The SDL static structure specifies which processes can communicate with each other. The number and type of channel parameters are acquired by static analysis of the specification. To avoid state space explosion, each channel parameter should be used by more than one signal. Each associated channel has potentially different set of parameters. If a process can send the same signal to two different processes, the send statement potentially has to be modelled differently—based on the concrete model of the receiver's associated channel.

Most of the specifications from the industry we know of use all of the available SDL communication constructs. The most critical constructs were itemized at the beginning of this section. For their simultaneous support a special skeleton for the process body and monitoring of the input queue had to be developed [10]. Support for the save construct, priority signal, and all forms of addressing is especially important.

### D. Timers

In standard Promela and Spin, timing properties of the specifications can not be expressed in a quantitative manner. Consequently, we actually verify the specifications with DT Spin ([18], [19]), an extension of the Spin model-checker with discrete time which has been developed within the Vires project. In [10], modelling of timers is extended with support for timer parameters. Expiration of a timer is modelled in accordance with [17] as a reception of a signal.

## V. CONCLUSION

The results from [10] are implemented in the `sdl2pml` tool. It mechanically generates Promela models from specifications in SDL. During the development of the tool we used the specification of the V.76 protocol as one of the test specifications. The SDL specification of the V.76 protocol consists of 1304 lines, while the model consists of between 4627 and 5034 lines of code. The final length of the model depends on the number of probes which are mechanically included in the model for the verification of various properties of the system.

Our intention was to give a reader an impression of the complexity of the specification and present the most critical parts of the mechanical creation of the model in Promela. A detailed description of algorithms and implementation details are outside the scope of this paper. During the verification of the mechanically generated model all errors that are presented in [9] were found.

Next, we want to test the `sdl2pml` on a real-life industrial specification. Our future research will be based on this experience. We believe that there are still a lot of unresolved issues. Our main motivation is the promotion and inclusion of formal verification as an equally important part of the development process at our industrial partners.

## ACKNOWLEDGMENT

This work was partly funded by the EU under the Information and Communication Technologies Centre of Excellence R&D project Correctness Verification of Communication System Functioning.

## REFERENCES

[1] G. J. Holzmann and M. Smith, "Automating Software Feature Verification," *Bell Labs Technical Journal*, vol. 5, pp. 72–87, 2000.

[2] G. J. Holzmann and M. H. Smith, "An Automated Verification Method for Distributed Systems Software Based on Model Extraction," *IEEE Transactions on Software Engineering*, vol. 28, April 2002.

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

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

[5] M. Bozga, S. Graf, and L. Mounier, "Automated validation of distributed software using the IF environment," *Electronic Notes in Theoretical Computer Science*, 2001.

[6] D. Bošnački, D. Dams, L. Holenderski, and N. Sidorova, "Model Checking SDL with Spin," *Lecture Notes in Computer Science*, pp. 363–377, 2000.

[7] A. Prigent, F. Cassez, P. Dhaussy, and O. Roux, "Extending the translation from SDL to Promela," in *9th International SPIN Workshop on Model Checking of Software (SPIN'02)*, vol. 2318 of *Lecture Notes in Computer Science*, (Grenoble, France), pp. 400–414, Springer-Verlag, Mar. 2002.

[8] N. Sidorova and M. Steffen, "Verifying Large SDL-Specifications Using Model Checking," in *SDL 2001: Meeting UML: 10th International SDL Forum*, Lecture Notes in Computer Science, Springer Verlag, 2001.

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

[10] B. Vlaovič, *Automatic Generation of Models with Probes from the SDL System Specification*. Ph.D. thesis, Faculty of Electrical Engineering and Computer Science, University of Maribor, 2004.

[11] ITU-T Recommendation V.76, "Generic multiplexer using V.42 LAPM-based procedures," 1996. Series V: Data communication over the telephone network.

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

[13] I. Recommendation Z.100 Annexes C and D, "Initial algebra model and SDL predefined data," 1993. Series Z: Programming Languages, Specification and Description Language (SDL).

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

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

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

[17] I. R. Z.100, "CCITT Specification and Description Language (SDL)," 1993. Series Z: Programming Languages.

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

[19] D. Bošnački and D. Dams, "Discrete Time Promela and Spin," *Lecture Notes in Computer Science*, pp. 307–310, 1998.