Automated Generation of Promela Model from SDL Specification

Boštjan Vlaovič [*], Aleksander Vreže, Zmago Brezočnik, Tatjana Kapus

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

Abstract

This paper presents our research in the domain of automated generation of a model from an SDL (Specification and Description Language) system specification. We use the Spin (Simple Promela Interpreter) formal verification tool and the Promela (Process Meta-Language) language for the description of the model. If the model is prepared manually, we need an expert with the detailed knowledge of the system, specification language, and modelling language. The quality of the model is directly influenced by the expert and is prone to the incorrect modelling of the system's properties due to the human error. Therefore, automatic generation of the model from the SDL specification is desired. In this paper we present our approach to the automated generation of the model in Promela. Additionally, we present challenges and future research directions.

Key words: Formal specification techniques, SDL, Promela, Automated generation of model

1. Introduction

Specification and Description Language (SDL) is standardized in the ITU-T Recommendation Z.100 [13]. It can be used from the abstract system description design phase to the automatic generation of implementations. For successful formal verification of an SDL specification a model has to be prepared for the chosen verification tool. If it is prepared manually, we need an expert with the detailed knowledge of the system, specification language, and modelling language. The quality of the model is directly influenced by the expert and is prone to the incorrect modelling of the system's properties due to the human error. Therefore, automatic generation of the model from the SDL specification is desired.

We decided to use verification tool Spin (Simple Promela Interpreter) with its input language Promela (Process Meta-Language). Like SDL, Promela adopts a strong formal basis established in the ECFSM (Extended Communicating Finite

[*] Tel.: +386 2 2207217; Fax: +386 2 2207272

Email addresses: bostjan.vlaovic@uni-mb.si (Boštjan Vlaovič), aleksander.vreze@uni-mb.si (Aleksander Vreže), brezocnik@uni-mb.si (Zmago Brezočnik), kapus@uni-mb.si (Tatjana Kapus).

State Machine) theory. It was designed during the development of an experimental Supertrace tool which focused on the exhaustive validation of a closed SDL system [5].

Similar to SDL, Promela allows dynamic creation of concurrent processes, an arbitrary number of message parameters, and multiple data types. A finite automaton of the system is defined with process templates. 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. To check some properties of the system, probes have to be inserted into the model. Probes provide an insight into the execution of the model and are mostly assertions on special variables.

The Supertrace was the first tool that provided a automatically generated model of an SDL specification [5]. It was introduced into AT&T's switch development environment as an experimental tool and was later integrated into tool Sdlvalid [4]. Unfortunately, these tools are not freely available. Another approach to the automated generation of model is described in [2].First, the SDL specification is transformed to an intermediate format (IF) with the *sdl2if* tool. The main motivation for the development of the intermediate representation was to provide easier interface to the various tools for formal verification. Next, the intermediate representation is transformed to Promela with the *if2pml* tool. We have studied this approach in [10].

Detailed study of real-life industrial specifications [8,9,12] showed that this approach lacks support for some of the important SDL features. Additionally, it requires a licence for the ObjectGeode toolset. These findings resulted in the development of an autonomous tool for direct automated generation of a Promela model with probes from an SDL specification—*sdl2pml*. This paper is an overview of the research results presented in [7].

The paper is organized as follows. Section 2 is an overview of our approach to the automated generation of a model. In Section 3, a simplified SDL specification of the V.76 protocol is given. In this paper we present our research results with help of this specification. Section 4 presents modelling of SDL data. In Section 5, modelling of SDL constructs and communication is described. Section 6 presents formal verification of the automatically generated model with Spin. We describe the use of probes with system requirement specifications expressed in Linear Temporal Logic (LTL). We conclude with a discussion and directions for further research.

2. Automated generation of model

Automated generation of a Promela model with probes from an SDL system specification is not a trivial task. 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 modelling of an SDL object usually include additional information about the object's position within the hierarchy of the system, inherited default values, communication infrastructure data, etc.

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

1. selection of relevant attributes for all SDL data types and constructs and their formal definition,
2. algorithm-based analysis of the specification,
3. sound modelling of predefined and user-defined data types,
4. sound modelling of SDL constructs,
5. sound modelling of communication,
6. algorithm-based automated generation of a Promela model with probes.

First, detailed analysis of the SDL and Promela syntax and semantics was performed. It resulted in many definitions that formally describe properties of the SDL data types and constructs which have to be explicitly modelled in Promela. These definitions are crucial for the formal specification of algorithms for the analysis of the specification.

Next, algorithms for sound modelling of predefined and user-defined data types were developed. Our research does not include ASN.1 (Abstract

Syntax Notation One) which can also be used as a data type notation in combination with SDL. This was followed by the development of algorithms for sound modelling of the SDL constructs.

Due to the expressive power of the SDL constructs for specification of communication, sound modelling of communication presented one of the biggest challenges. The proposed solution is based on:

– complete analysis of communication infrastructure—interconnection and hierarchy of channels and signal routes—which provides all possible routes for each signal that can be sent by any process,
– the generation of a special process skeleton which provides signal reception that is in accordance with the ITU-T Recommendation Z.100 [13].

It includes support for direct and indirect addressing, priority signals, implicit transitions, the save construct, enabling conditions, the asterisk input and path limitations.

The algorithms for the automated generation of a Promela model support most of the SDL constructs. Additionally, they include support for permanent, predefined, and user-defined probes.

Permanent probes are included in all models. They are used to check model's accordance with the SDL semantics, e.g., detection of violations of the maximum number of process instances.

Predefined probes are used for checking properties which would significally contribute to the model's complexity. They are included only on user's demand. For example, it can be checked if the range conditions of the answer parts of a decision statement are mutually exclusive for all possible execution paths. Predefined probes can also be used to check for execution paths which are in accordance with the SDL semantics, but can result in undesired behaviour, e.g., implicit transitions—reception of signals that are not included in explicitly defined input or save constructs. An implicit transition contains no action and leads directly back to the same state of the process. Therefore, this behaviour might present underspecification that results in undesired behaviour of the system.

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, neverclaims or LTL formulae). They can be automatically included with the use of comments in the SDL specification or via a configuration file of the *sdl2pml* tool.

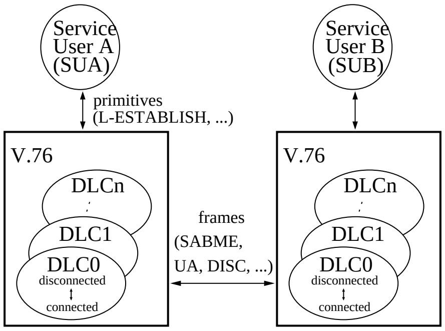


Figure 1. Communication between SUA and SUB through V.76 protocol

In this paper we present our research results with help of an SDL specification of a simplified version of the V.76 protocol. The user employs the service of protocol V.76 [6] to establish one or more Data Link Connections (DLCs) between two modems and to transfer data over these connections (Fig. 1).

3. Specification of system `V76test`

The specification of the SDL system `V76test` and associated files were published in [3] and can be downloaded in ObjectGeode and Tau SDL formats from the Internet [1].

Fig. 1 shows communication between two service users. Specification of system `V76test` supports two parallel connections: SUA may establish DLC number 0 (DLC0) to transmit voice and DLC1 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 connection.

[1] ftp://ftp.wiley.co.uk/pub/books/ldoldi/v76_ftp.zip

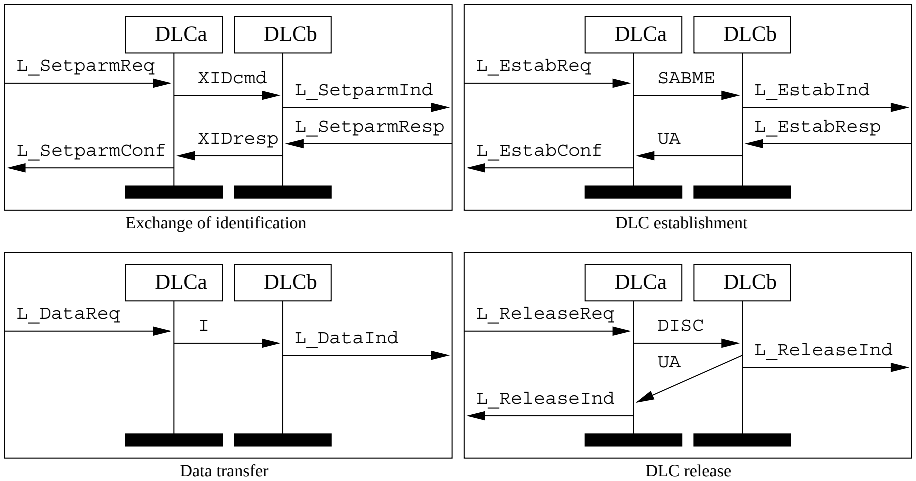


Figure 2. Four stages of V.76 protocol connection between SUs

3.1. *DLC establishment*

In the first stage SUs can optionally perform identification exchange procedures. Next, establishment of a data link connection is expected. On receipt of an L-ESTABLISH request primitive (signal `L_EstabReq`) from its SU, the V.76 shall attempt to establish the DLC. The DLC entity transmits a Set Asynchronous Balanced Mode Extended (SABME) frame, the retransmission counter is reset, and timer T320 [6] is started [3]. If the peer DLC entity, based on the response from its SU (signal `L_EstabResp`), is able to establish the DLC, it shall respond with Unnumbered Acknowledge (`UA`) and enter the connected state. Otherwise it should respond with Disconnect Mode (`DM`). Once in the connected state, information transfer may begin.

3.2. *Data transfer*

The DLC receives data from SU with the use of an L-DATA request primitive (signal `L_DataReq`). Data are transmitted in an I frame. Structure of the I frame is defined by the definition of the data type `Iframe` (Fig. 3).

3.3. *DLC release*

Communication is terminated with the L-RELEASE request primitive (signal `L_ReleaseReq`) from any SU. Description of the protocol in greater detail is outside the scope of this paper and can be found in [3] and [6].

3.4. *Modifications of specification*

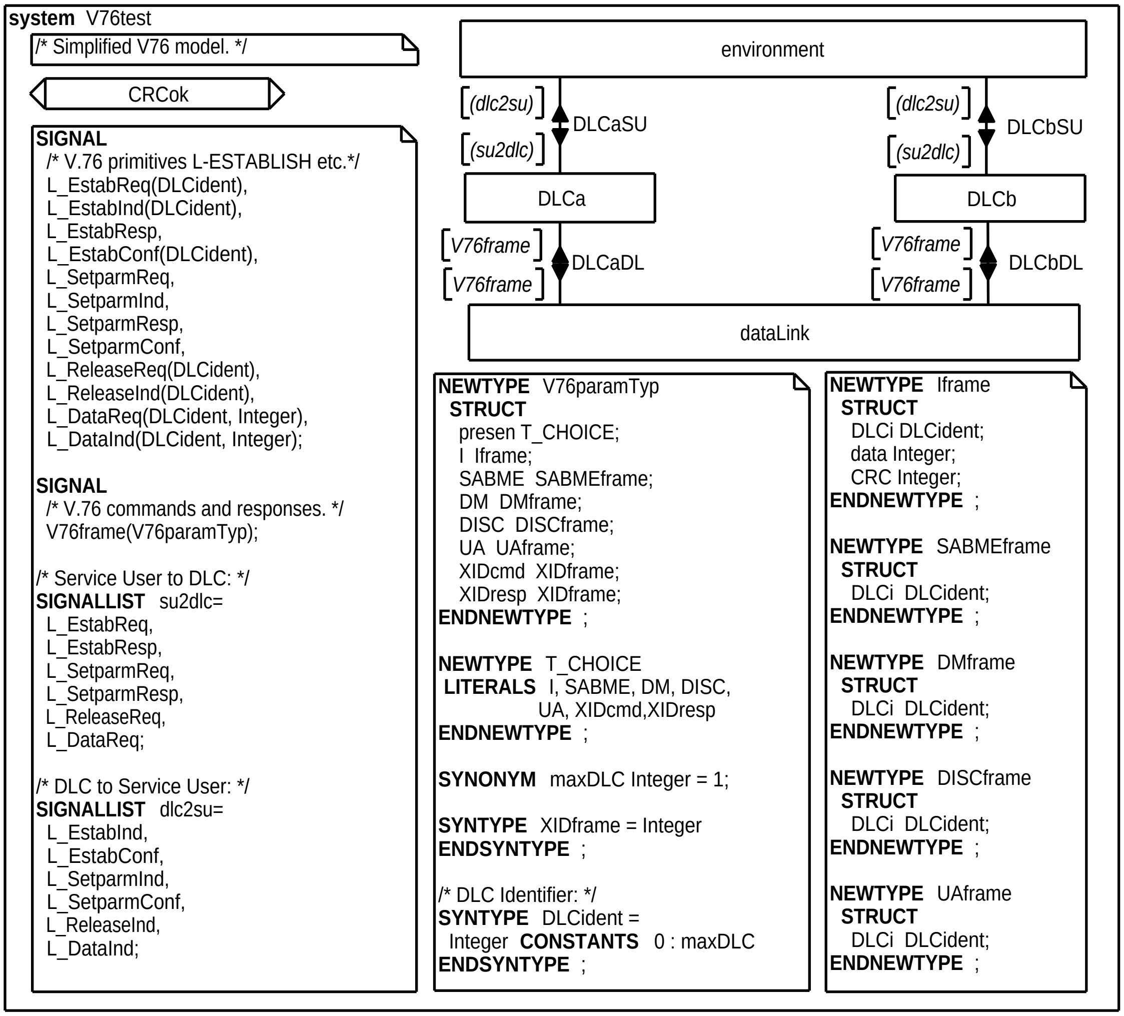
The SDL specification of system `V76test` includes one definition of ASN.1 data type `choice`. We decided to replace it with an SDL'96-compliant definition of data type `T_CHOICE` to avoid the ASN.1 extensions of the ITU-T Recommendation Z.100 (Fig. 3).

Since simulation and verification with Spin require a complete system, we supplemented the specification with a model of its environment. Fig. 3 shows a graphical SDL description of system `V76test` that is used in this paper as a reference SDL specification.

To complete the formal specification of the system and eliminate the need for user's intervention during the simulation, all decision statements which were specified as an informal text that abstractly represented user's arbitrary decisions were replaced with nondeterministic decision statements (Fig. 6). No other changes of the original `V76test` system specification were made.

3.5. *System structure*

Block `environment` (Fig. 3) consists of processes `SUa` and `SUb`. They can receive and send all sig-



```
SIGNAL
  /* V.76 primitives L-ESTABLISH etc.*/
  L_EstabReq(DLCident),
  L_EstabInd(DLCident),
  L_EstabResp,
  L_EstabConf(DLCident),
  L_SetparmReq,
  L_SetparmInd,
  L_SetparmResp,
  L_SetparmConf,
  L_ReleaseReq(DLCident),
  L_ReleaseInd(DLCident),
  L_DataReq(DLCident, Integer),
  L_DataInd(DLCident, Integer);

SIGNAL
  /* V.76 commands and responses. */
  V76frame(V76paramTyp);

/* Service User to DLC: */
SIGNALLIST  su2dlc=
  L_EstabReq,
  L_EstabResp,
  L_SetparmReq,
  L_SetparmResp,
  L_ReleaseReq,
  L_DataReq;

/* DLC to Service User: */
SIGNALLIST  dlc2su=
  L_EstabInd,
  L_EstabConf,
  L_SetparmInd,
  L_SetparmConf,
  L_ReleaseInd,
  L_DataInd;
```

```
NEWTYPE  V76paramTyp
  STRUCT
    presen T_CHOICE;
    I  Iframe;
    SABME  SABMEframe;
    DM  DMframe;
    DISC  DISCframe;
    UA  UAframe;
    XIDcmd  XIDframe;
    XIDresp  XIDframe;
ENDNEWTYPE  ;

NEWTYPE  T_CHOICE
 LITERALS  I, SABME, DM, DISC,
           UA, XIDcmd,XIDresp
ENDNEWTYPE  ;

SYNONYM  maxDLC Integer = 1;

SYNTYPE  XIDframe = Integer
ENDSYNTYPE  ;

/* DLC Identifier: */
SYNTYPE  DLCident =
  Integer  CONSTANTS   0 : maxDLC
ENDSYNTYPE  ;
```

```
NEWTYPE  Iframe
  STRUCT
    DLCi DLCident;
    data Integer;
    CRC Integer;
ENDNEWTYPE  ;

NEWTYPE  SABMEframe
  STRUCT
    DLCi  DLCident;
ENDNEWTYPE  ;

NEWTYPE  DMframe
  STRUCT
    DLCi  DLCident;
ENDNEWTYPE  ;

NEWTYPE  DISCframe
  STRUCT
    DLCi  DLCident;
ENDNEWTYPE  ;

NEWTYPE  UAframe
  STRUCT
    DLCi  DLCident;
ENDNEWTYPE  ;
```

Figure 3. Graphical SDL description of V.76 protocol and its environment

nals (protocol primitives) that are defined at the channels DLCaSU and DLCbSU. The model of the users defines all possible execution paths that can be checked with formal verification of the model of the specification. The complexity of the formal verification can be directly influenced by different versions of the model of the users.

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 performs management tasks. It can create a new instance of the DLC process on user's request (signal L_EstabReq) or refuse the new connection with signal L_ReleaseInd (Fig. 12). Each data link connection is managed by its own DLC process.

Block dataLink consists of two processes (Fig. 5). Processes AtoB and BtoA model a simplified data-link layer which provides an unreliable frame transfer service (Fig. 6).

3.6. *Additional information*

The SDL specification without comments consists of 1304 lines of code. It includes most of the

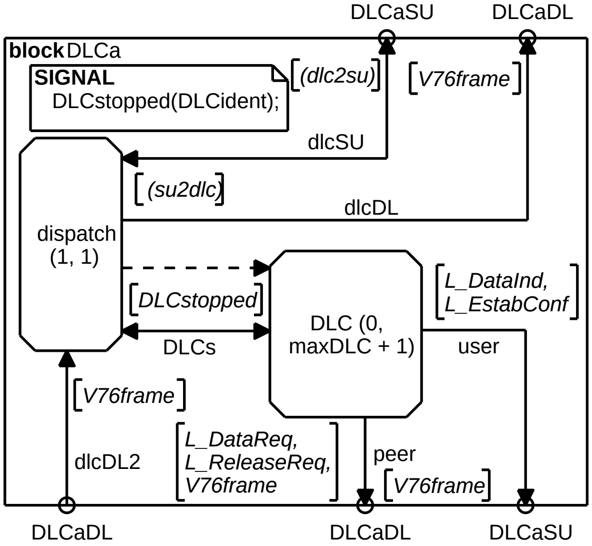


Figure 4. Graphical description of block DLCa

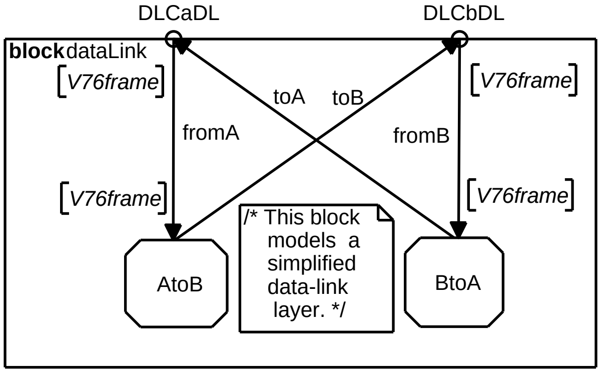


Figure 5. Block `dataLink`

SDL constructs that are used in the real-life specifications of telecommunications protocols:

- SDL data types and expressions,
- dynamic process creation,
- priority signals,
- implicit transitions,
- spontaneous transitions,
- save construct,
- priority inputs,
- enabling conditions,
- asterisk inputs,

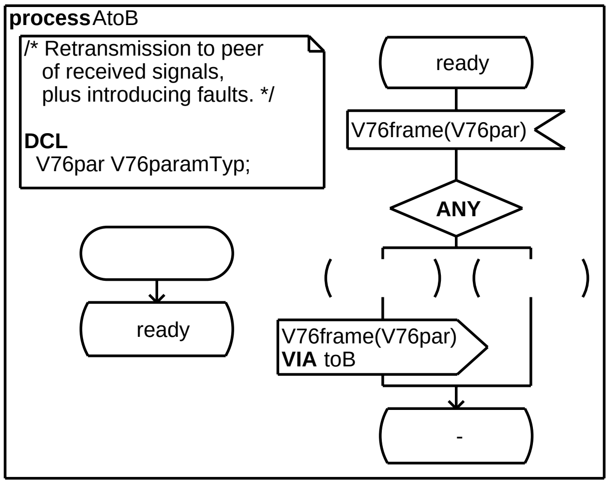


Figure 6. Process `AtoB`

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

We believe that their support is crucial for the adoption of formal verification of SDL-based designs in the industry. To our knowledge, the *sdl2pml* tool is the only tool which supports the modelling of all the named constructs.

The size of the automatically generated Promela model depends on the number of included probes. During our experiments the size of the model was between 4627 and 5034 lines of code.

4. Modelling of SDL data

Data in SDL are based on the Abstract Data Type (ADT) concept. 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. Predefined data types and data generators are defined in Annex D to the ITU-T Recommendation Z.100.

A set of values with certain characteristics is called a sort. Operators are defined from and to sorts. Each sort defines the collection of all the possible values which can be generated by the related set of operators. Each value belongs to exactly one sort. That is, sorts never have values in common. For most sorts there are literal forms to denote values of the sort. Literals are a special case of operators without arguments. Each SDL specification has a number of partial type definitions, each of which defines a sort and operators and algebraic rules associated with that sort.

Although all the data types are abstract, and the predefined data facilities may even be overridden by the user, SDL attempts to provide a set of predefined data facilities which are familiar in both their behaviour and syntax. The following are predefined: *INTEGER*, *BOOLEAN*, *PId*, *NATURAL*, *REAL*, *TIME*, *DURATION*, *CHARACTER*, *STRING*, *CHARSTRING*, *ARRAY*, and *POWERSET* [13].

4.1. *Variables*

Variables are objects which can be associated with a value by assignment. When a variable is accessed, the associated value is returned. A variable access in SDL has a value "error" if the variable is undefined. If the value is "error", the further behaviour of the system is undefined. The same holds for cases where variable is assigned a value which is outside the defined range [13].

Since Promela always initializes variables to zero, if there is no user-specified predefined value, all data types in the generated model include an explicit literal that represents the undefined value, if possible. For the predefined sorts the highest value of the sort is used [2]. For user-defined sorts we model the undefined value with the explicit literal (Fig. 9). This is required for the sound modelling of the SDL specification and detection of undesired behaviour related to the access to the undefined variable.

The proposed algorithms include predefined probes that check for the access to the undefined variable and perform range check on every assignment statement.

4.2. *Predefined Data*

Our algorithms support the following predefined sorts: *INTEGER, BOOLEAN, PId, NATURAL, REAL, TIME, DURATION, CHARACTER*, *CHARSTRING*, and *ARRAY* [7].

For the first three sorts corresponding data types in Promela are used—*int*, *bool* and *pid*.

The *NATURAL* sort is modelled with the *unsigned* data type. Unfortunately, the *unsigned* data type cannot be used in the definition of a channel. Therefore, we use the *int* data type for the channel definitions. To keep the same names for sorts in a specification and a model, we decided to use the following structure: `typedef natural` {`unsigned val :31`}. This solution provides a model for 31-bit variables of the *NATURAL* sort. Promela's syntax would allow 32-bit values, but Spin version 4.1.1, which was used during our research, reports an error: "width-field val too large". In later version 4.2.6 usage of a 32-bit value does not produce an error during the simulation and generation of a verifier.

Sorts *REAL*, *TIME* and *DURATION* represent rational numbers. Currently they are modelled with integers. Their accurate modelling is part of our current research activities [11].

For the *CHARACTER* sort we use Promela's `#define` directive for all literals that are defined in Annex D to the ITU-T Recommendation Z.100. For each character we have used an unambiguous mnemonic defined in RFC1345 (Fig. 7).

The *CHARSTRING* sort is modelled as an array of characters which is indexed in the SDL style—the index begins with 1. It is limited to 32 characters [3] by default, but this can be changed by the user. All operators associated with the *CHARSTRING* sort are modelled with the use of the inline statements. Fig. 8 shows the model of the sort and `strcpy` operator.

Modelling of the *ARRAY* generator is presented in the next section.

[2] *BOOLEAN* sort is an exception

[3] ObjectGeode's SDL C code generator is limited to 510 characters

```
#define NUL          0     #define CHAR_A      65
#define SOH          1     #define CHAR_B      66
           ⋮                           ⋮
#define ZERO         48    #define CHAR_a      97
#define ONE          49    #define CHAR_b      98
           ⋮                           ⋮
#define LESS_THAN    60    #define TILDE       126
#define EQUALS_SIGN  61    #define DEL         127
           ⋮
```

Figure 7. Some definitions of *CHARACTER* sort literals

```
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
  }
}
```

Figure 8. Modelling of *CHARSTRING* sort and `strcpy` operator

4.3. *User-defined Data*

Modelling of user-defined sorts presented many challenges. We describe the proposed solution, which consists of twelve definitions and seven algorithms for:

1. analysis and modelling of *NEWTYPE* definitions,
2. analysis and modelling of *SYNTYPE* definitions,
3. analysis and modelling of *SYNONYM*s,
4. range of values analysis,
5. modelling of expressions,
6. analysis of *TIMERs* and
7. analysis of variables.

During the static analysis of the system we check the definition of each sort and try to optimize its representation in the Promela model—the data type used in the model is based on the range of values. The aim is to model variables of every sort with the smallest possible number of bits. Additionaly, we take into account the following inherited attributes: range of values and default initialization.

A *NEWTYPE* introduces a partial data type definition which defines a distinct new sort. Explicitly declared literals are defined as constant values with the use of the `#define` directive. Fig. 9 shows the model of the `T_CHOICE` and `Iframe` data types defined in the `V76test` system (Fig. 3).

A *SYNTYPE* specifies a set of values of a sort. A syntype sort has the same semantics as the sort referenced by the syntype, but might include only a subset of values compared to the referenced sort. Syntype `DLCident` defined in the `V76test` system references predefined sort *INTEGER*, but limits the range of values to 0:maxDLC (Fig. 3). The model of the syntype `DLCident` is shown in Fig. 9.

A *SYNONYM* gives a name to an expression which represents one of the values of a sort. The value which the *SYNONYM* represents is determined by the context in which the synonym definition appears. A *SYNONYM* has a sort which is the sort of the ground term. For example, the ground term for the unity Integer value can be written "1". Usually there are several ground terms which denote the same value, e.g., the Integer ground terms "0 + 1", "3 − 2" and "$\frac{(7+5)}{12}$" [13]. The algorithm provides a normal form of the ground term (in this case "1") for denoting the value. *SYNONYM*s are modelled with the use of the `define` directive (Fig. 9).

In standard Promela and Spin, timing properties of the specifications cannot be expressed in a quantitative manner. Consequently, we actually verify the specifications with DT Spin [1], an extension of the Spin model-checker with discrete time, which has been developed within the Vires project. Our modelling of timers is extended with support for timer parameters [7]. Additionaly, expiration of a timer is modelled in accordance with [13] as a reception of a signal. This enables the use of the timer in the priority input and save constructs.

DT Spin is not able to follow the pace of Spin development. Therefore, our current research in-

```
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}

#define maxDLC 1
```

Figure 9. Modelling of user-defined *NEWTYPE* and *SYNTYPE* sorts and *SYNONYM* from `V76test` system

cludes a search for a new model of discrete time in Promela that could be used with the mainstream version of Spin [11].

4.4. *ADT operators*

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. Every SDL specification from our industry projects uses such operators.

Automated generation of a model 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 [11]. The first results of our research are promising.

System `V76test` does not include any operator definitions, therefore, description of these algorithms is outside the scope of this paper.

5. SDL constructs and communication

Our analysis of the ITU-T Recommendation Z.100 resulted in 16 additional definitions and 33 algorithms for the modelling of SDL constructs and communication. In this paper only a high-level presentation of the main concepts will be given based on the system `V76test` presented in Section 3.

5.1. *Hierarchical Structure*

System `V76test` consists of four SDL blocks. Each block has more than one process. A process is connected to other processes in the same block with signal routes or to processes in other blocks with the use of channels that connect these blocks. Fig. 4 shows block `DLCa` with processes `dispatch` and `DLC`. The processes are connected with signal route `DLCs`. Process `DLC` can communicate with the system user through signal route `user`, which connects to channel `DLCaSU`.

5.2. *Processes*

SDL processes are either created at system initialization time, e.g., process `dispatch`, or later in the lifetime of the system by other processes in the same block, e.g., process `DLC`. Each process instance can be in a different state and therefore react differently to a signal at a given time. The number of process instances may change during the lifetime of the system. The specification defines the initial and the maximum number of process instances. An arbitrary number of instances of a process can be active at the same time if it is not explicitly specified otherwise[4]. The maximum number of instances of process `DLC` is limited by expression `maxDLC + 1`. `MaxDLC` is defined in Fig. 3.

SDL processes are extended communicating finite-state machines and can be modelled with Promela proctypes. Our algorithms for modelling SDL process with the use of Promela's proctype

[4] The maximum number of active processes in a real-life system depends on the target hardware and configuration.

```
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;
              ⋮
 }
}
```

Figure 10. Part of generated init process

definition supports dynamic process creation, expressions in parameters, communication between processes and their termination.

Process variables are created at the creation of the process instance—including predefined variables `parent`, `offspring`, and `self`. The variables are set to an initial value if specified. In SDL, variables which do not have an initial value remain undefined until they are first assigned a value in a transition. Promela uses a different approach and always initializes all variables to some explicit or implicit predefined value. Special care had to be taken during the construction of the algorithms to explicitly model the undefined values in a automatically generated model (Fig. 9).

In a model based on our approach all processes that have the initial number of instances greater than zero are created by the special process `init`. Fig. 10 shows a part of the generated model which creates one instance of process `AtoB`. Each process is allocated a unique process instantiation number and input queue which is modelled with the Promela channel (Section 5.3). The new process executes asynchronously with the existing active processes from this point on. When the run operator completes, the new process need not have executed any statements. Most of the code in Fig. 10 updates various tables that are part of the model's skeleton, which provides a proper infrastructure for sound modelling of the SDL semantics. To fully understand the model of the simple process `AtoB` (Figs. 10 and 11) would require a detailed explanation of process modelling. This is, unfortunately, outside the scope of this paper. Details are given in [7].

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 might create a problem when we model an SDL specification where processes do not terminate in the reverse order of their creation. This issue requires further research activities.

5.3. *Communication*

Each SDL process has an associated input queue. In the Promela model 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 are not explicitly specified, the automatically generated model is not in accordance with [13]. Both values are set to one, while [13] in this case defines the maximum number to be unlimited.

The SDL communication infrastructure specifies

```
proctype dataLink__AtoB(pt__chan input;
pt__pid parent){
pt__pid offspring, sender;
byte pv__ptr, pv__cur;
xr input;

V76paramTyp V76par;

goto ready;
ready:
end_1:
do
:: table_channum_ptr[input] > pv__cur ->
  table_channum_prio[input]=false;
  pv__cur++;
  pv__ptr=0;
  atomic{
  do
  :: pv__ptr <= cv__buff-1 ->
    if
    :: else -> set__clear();
    fi;
    pv__ptr++ ;
  :: else -> goto ready_start;
  od;
  }
ready_start:
  if
  :: table_channum_prio[input]==true ->
    pv__ptr=0;
    do
    :: (pv__ptr <= cv__buff-1)
    && (table_channum_nsp[input].\
 data[pv__ptr].prio==true) ->
      if
      :: else  -> skip;
      fi;
    :: (pv__ptr == cv__buff) -> break;
    :: else -> pv__ptr++
    od;
  :: else ->
    pv__ptr=0;
    do
    :: (pv__ptr <= cv__buff-1) ->
      if
      :: skip__save()
      :: else ->
        if
        ::atomic{recv__sig(true,V76frame) ->
        input??V76frame(sender,V76par);
        update_chan_tab();
        }
        if
        :: true ->
         d_step{
          pfv_tmp=1;
          do
          :: pfv_tmp < table__DLCb__dispatch__free ->
            pfv_tmp++ ;
          :: break;
          od;
          pfv_tmp--;
         };
         atomic{
         chan__DLCb__dispatch[pfv_tmp]!V76frame(_pid,\
          pcv__null,V76par,pcv__null);
         table_channum_nsp[table_channame_channum[\
          chan__DLCb__dispatch__select+pfv_tmp]].data[\
          table_channum_ptr[table_channame_channum[\
          chan__DLCb__dispatch__select+pfv_tmp]]].name=\
          V76frame;
         table_channum_ptr[table_channame_channum[\
          chan__DLCb__dispatch__select+pfv_tmp]]++;
         }
        fi;
        goto ready;
        :: else ->
          if
          :: else -> goto ready;
          fi;
        fi;
      fi;
    od;
  fi;
  goto ready;
od;
end: skip
}
```

Figure 11. Generated AtoB process

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 can be used by more than one signal. Each associated channel has a potentially different set or order of parameters. Consequently, 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 use all of the available SDL communication constructs. The most critical constructs were itemized in Section 3.6. For their simultaneous support a special skeleton for the process body and monitoring of the input queue had to be developed [7]. Support for the save construct, priority signal, and

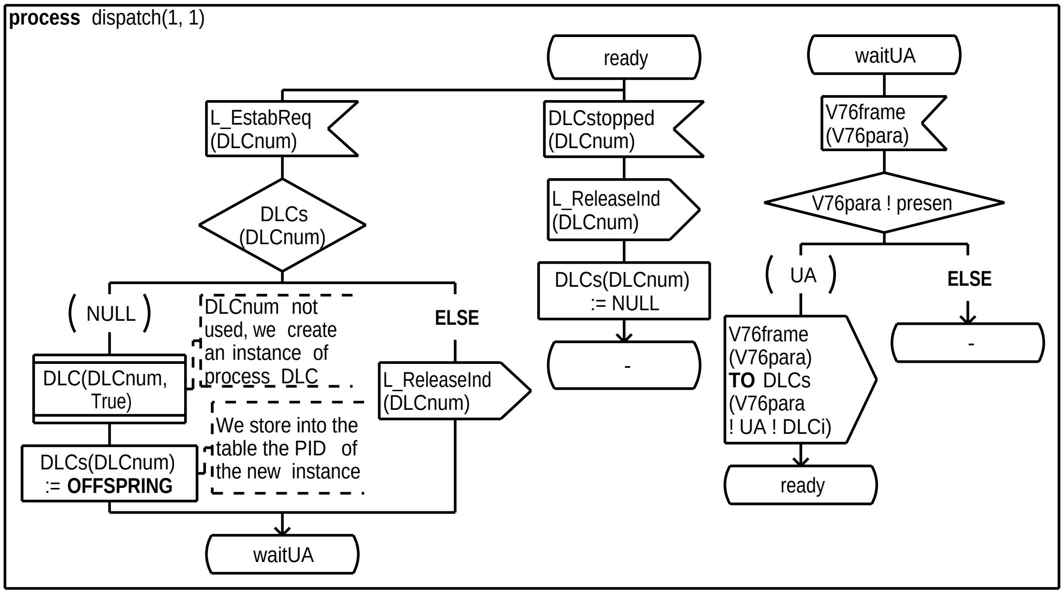


Figure 12. Part of SDL specification of process `dispatch`

all forms of addressing is especially important.

6. Verification of model with probes

The model of the environment defines all possible execution paths that can be checked with the formal verification of the model of the specification. The *sdl2pml* tool supports two approaches:

1. the environment is modelled within the SDL specification or
2. the environment is modelled in Promela.

With the second approach the *sdl2pml* tool prepares needed communication infrastructure based on the channel and signal route definitions. We chose the first approach (Fig. 3).

We started formal verification with the most general model of the environment. Processes `SUa` and `SUb` in block `environment` could receive and send all signals (protocol primitives) that are defined at the channels `DLCaSU` and `DLCbSU` with one important limitation that prevented aggressive sending of signals: any signal can be sent only after the reception of any signal. The order of signals was not imposed.

6.1. *Depth-first search*

The formal verification of this model stops after 37 ms in the 997th step and reports invalid end state[5]. The size of the state-vector is 1940 bytes and total actual memory usage is 4260 kB. The examination of the counterexample shows that the model has infinite execution paths. Therefore, all regular wait states should be labelled as valid end states. The formal verification of the supplemented model again terminates due to an invalid end state.

The examination of the counterexample (Fig. 13) exposes problems in the processes `dispatch` (Fig. 12). If users `SUa` and `SUb` begin with the DLC establishment (Fig. 2) "simultaneously", processes `dispatch` remain at the end of execution in the state `waitUA`. They are endlessly waiting for the reception of the signal `V76frame` with command `UA` which would acknowledge successful DLC establishment. A detailed explanation of the execution path follows. On receipt of an L-ESTABLISH request primitive (signal `L_EstabReq`) from its SU, the processes `dispatch` attempt to establish the

[5] Intel(R) Xeon(TM) CPU 2.40GHz (4778.59 bogomips) with 2 GB of RAM.

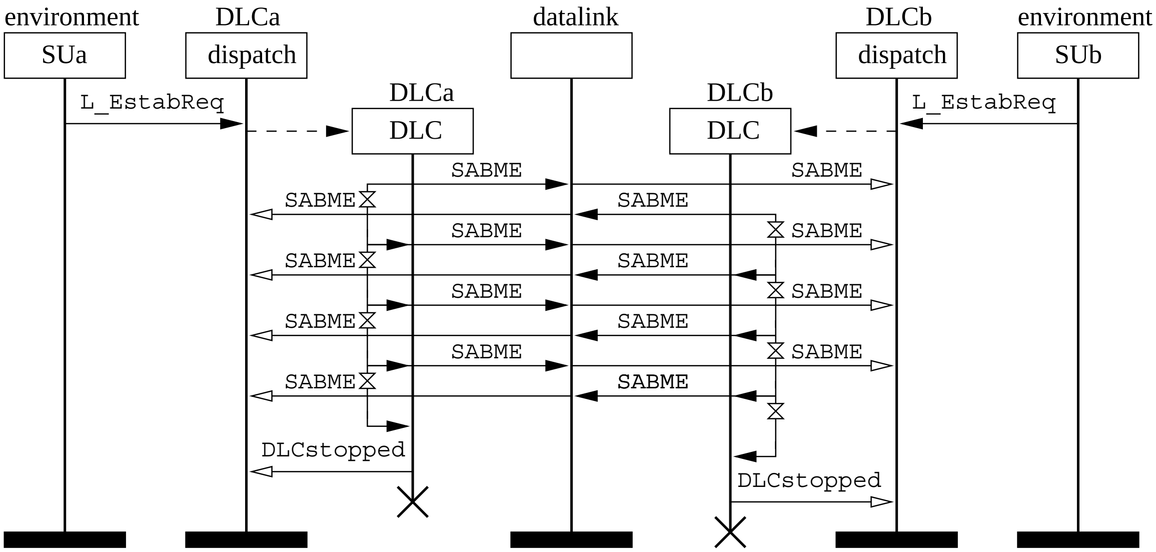


Figure 13. MSC of the “dispatch’s end state”counterexample

DLC. Both processes create a new instance of process `DLC` and go to the `waitUA` state. The DLC establishment continues with the SABME frame. If a peer DLC entity, based on the response from its SU (signal `L_EstabResp`), is able to establish the DLC, it shall respond with command `UA` and enter the connected state. Otherwise it should respond with Disconnect Mode (`DM`). This never happens, because all `SABME` signals result in implicit transition in the processes `dispatch`. After three unsuccessful retransmissions processes `DLC` terminate. Notification signal `DLCstopped` also results in implicit transition and the system remains in a deadlock. The system would arrive to the same deadlock on recursive signal loss in the `dataLink` block. This behaviour is not in accordance with the ITU-T Recommendation V.76. Therefore, we supplemented the processes `dispatch` with the reception of the signal `DLCstopped` in the `waitUA` state.

6.2. *Breadth-first search*

The formal verification of the supplemented model with the default depth-first search terminates in the 2590th step due to an invalid end state. The examination of relatively complex counterexample exposes problems with the associated input queue of the `dispatch` processes. The problems are independent of the number of slots in the input queue and are triggered by the user’s rejection of the DLC establishment.

Breadth-first search of the same model terminates after 262 steps and exposes a problem that arises due to an unexpected user behaviour—the `SUb` process responds to the `L_SetparmInd` signal with the start of the DCL establishment (`L_EstabReq`). Both problems remain in the final corrected version of the SDL system in [3]. This verification run demonstrated the need for greater changes in the SDL system. We wanted to compare our verification results with the results from [3], hence, we decided to limit the behaviour of the protocol users in the `environment` block.

6.3. *Additional errors*

The subsequent verification runs exposed a number of additional shortcomings of the system. The system reaches a deadlock because the `dispatch` process remains in the `waitUA` state after the rejection of the user’s request for the establishment of the already active DLC.

Processes `DLC` does not finish execution in a valid end state if data transfer or DLC release stages of the protocol connection never happen. The ITU-T Recommendation V.76 deals with such execution paths with the inactivity timer T403. This timer was not included in the SDL system V76test. We decided to ignore this problem with additional valid end states in processes DLC.

If user SUa requests the exchange of the identification while DLC is established and user SUb does not respond to the request, the dispatch process in the DLCb block remains in a state where all signals from the environment result in an implicit transition. Attempt to resolve this issue with the save construct [3] is not valid because it blocks the active DLC until the reception of the response to the exchange of the identification request. Additionaly, the input queue of the dispatch process would get full, since user SUb might never respond to a request. We had to extend the SDL specification of the protocol with the stop timer T401 [6] to continue with the formal verification.

6.4. *Search for shortest trail*

The formal verification of the supplemented model terminated in the 9393rd step. The very long execution path motivated us to search for the invalid execution path with the shortest trail. Verification terminated at the 470th step after five errors in 4670 ms. Total memory usage was 1548 kB. This error occurs when user SUa requests DLC release with signal L_ReleaseReq immediately after its establishment (Fig. 2). Request is forwarded by the dispatch process to the block dataLink with command DISC. The data-link layer provides an unreliable frame transfer service. In this execution path command DISC is lost in the AtoB process. The ITU-T Recommendation V.76 handles this with the retransmission of the DISC command after the expiration of the timer T401. After N400 unsuccessful retransmissions the DLC process should inform the dispatch process and terminate. Formal verification showed that the SDL specification does not include this mechanism. Again, accordance with [3] imposed additional changes in the SDL specification—elimination of the nondeterministic decision in the processes AtoB and BtoA (Fig. 6). Now, the data-link layer provides a reliable frame transfer service.

6.5. *Additional limitations of environment*

The subsequent formal verification runs exposed additional issues in the dispatch process. We had to additionally limit the behaviour of the protocol service users to the sequence of signals that is in accordance with Fig. 2. First, exchange of the identification is performed. It is followed by the DCL establishment. Next, data transfer is conducted. Finally, DCL is released.

In spite of these changes, formal verification of the automatically generated model provides additional counterexamples. One of them demonstrates unsuitability of the default size for the associated input queues (three slots). If command SABME is retransmitted and saved three times, input queue of the process dispatch could block signal reception. We set the size of the input queues to four to resolve this issue. Up to this point all problems arose due to the invalid end states.

6.6. *Permanent probes*

When all end states in the verification model are valid, permanent probes can be used to check if the specification is in accordance with the semantic rules of SDL. A counterexample demonstrated that the range of values definition by one of the user-defined sorts could be violated by the environment process SUa.

6.7. *Predefined probes*

Next, predefined probes can be used to check for execution paths that are in accordance with the SDL semantics, but can result in an undesired behaviour, e.g., implicit transitions. Sometimes implicit transitions are used intentionally. Therefore, the *sdl2pml* tool provides mechanism for their exclusion—probes are not inserted in the model for the chosen signals. We excluded probes for the reception of signals L_SetparmResp and L_EstabResp in processes dispatch when they are

Table 1
Possible implicit transitions in SDL system

block	process	state	signal
env.	SUa	waitEstabConf	L_ReleaseReq
DLCb	dispatch	ready	L_SetparmResp
DLCb	dispatch	ready	L_EstabResp
DLCb	dispatch	waitParmResp	L_EstabResp
DLCb	dispatch	waitParmResp	L_ReleaseReq
DLCb	dispatch	waitEstabResp	L_SetparmResp
DLCb	DLC	waitUAdisc	L_ReleaseReq

in the state `ready`. Table 1 presents all possible implicit transitions in the current SDL system.

6.8. *User-defined probes*

Tool *sdl2pml* supports explicit definitions of undesired paths. An example of such definition could be the SDL decision statement where we never expect the else part to be executed. If we put probes in all such else parts, formal verification stops in the 1932nd step. The counterexample shows the execution path where process `dispatch` ignores the command `DM`. Fig. 12 shows that only command `UA` is handled. This presents an undesired behaviour. Formal verification of the corrected SDL specification exposes another similar problem in the `DLC` process.

After the correction of these errors, the SDL specification is ready for the verification of requirement specifications written with LTL formulae.

6.9. *LTL system requirements specification*

We can check if the statement "if the signal is send by the sender, it is always received at the receiver" holds with the formula $\Box(p \rightarrow \Diamond q) \wedge \Diamond p$. The p represents the probe which is inserted at the sender and the q represents the probe that is inserted at the receiver. Both probes can be composed of many subprobes if the signal can be sent or received in a number of states. Table 2 shows results of the formal verification runs for a number of signals.

Formal verification of the first claim confirms that user `SUb` always receives signal `L_SetparmInd` if user `SUa` initiates the exchange of identification with signal `L_SetparmReq`.

The counterexample from the formal verification of the second claim shows that the second part of the identification exchange is not as reliable. The same holds for the third claim, since the modified model of the user `SUa` never begins with the DCL establishment until it successfully completes the exchange of identification.

The counterexample from the formal verification of the fourth claim presents the execution path where signal `L_DataReq` is never sent. This is in accordance with the requirement specification. Therefore, the formula was changed to $\Box(p \rightarrow \Diamond q)$. Now, formal verification stops at the 3084th step and uncovers an execution path where data can be lost (Fig. 14). This happens if the `SUb` requests the DLC release right after its establishment, while the `SUa` already sent `L_DataReq`. Counterexample demonstrates the implicit transition upon the reception of the information frame (signal `I`) in the `DLC` process. The formal verification ran for almost 52 minutes and used 709742 kB of memory.

The last claim confirms our previous findings that the SDL specification does not include a mechanism for the termination of the inactive DLC. Therefore, DLC release might not be initiated after successful DLC establishment.

7. Conclusion

Our intention was to give a reader an impression of the complexity of the automated generation of a Promela model from an SDL specification and to present our research results. A detailed description of the algorithms and implementation details are outside the scope of this paper. The implementation consists of more than 100.000 lines of C code.

During the verification of the automatically generated model of system `V76test` all errors that are presented in [3] were found. We managed to find some additional limitations of the specification.

There are still a lot of unresolved issues. We would like to point out some of them:

Table 2
Formal verification of LTL-expressed system requirement specifications

p	q	result
(process, state, signal)	(process, state, signal)	$\Box(p \rightarrow \Diamond q) \wedge \Diamond p$
SUa, start, L_SetparmReq	SUb, ready, L_SetparmInd	holds
SUb, ready, L_SetparmResp	SUa, waitParmResp, L_SetparmConf	does not hold
SUa, start, L_SetparmReq	SUb, ready, L_EstabInd	does not hold
SUa, waitEstabConf, L_DataReq	SUb, ready, L_DataInd	does not hold
SUa, waitEstabConf, L_EstabConf	SUa, (waitEstabConf, SUa_endstate), L_ReleaseReq	does not hold

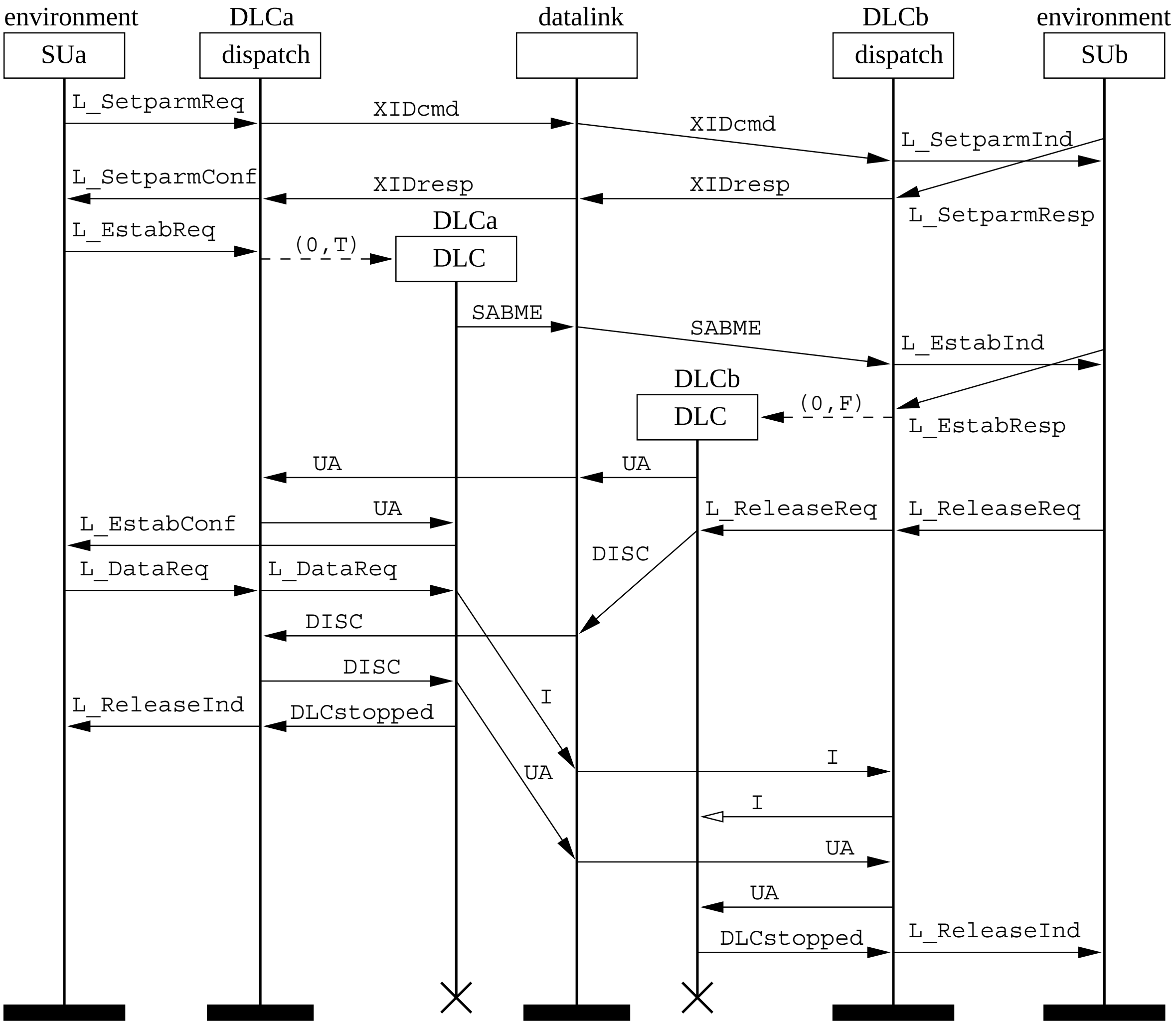


Figure 14. MSC of “data loss” counterexample

- different interpretation of the process termination in SDL and Promela,
- suboptimal use of the *unsigned* data type in the C code of the verifier — all *unsigned* variables are presented as *unsigned int*,
- suboptimal use of the *bit* and *bool* data types in the C code of the verifier — all arrays of these two types are presented as *unsigned char*,
- the data type that defines a structure with an array element cannot be used in a channel definition.

Our future research will focus on these issues. Next, we want to extend the *sdl2pml* tool with a new model of discrete time which could be used with the mainstream version of Spin. Additionally, we are working on semi-automatic inclusion of embedded C code that is used in real-life specifications with the ObjectGeode's extension of the SDL's ADT concept.

Currently we are testing the *sdl2pml* tool on a real-life industrial specification of the ISDN User Adaptation protocol. Our final goal is to promote formal verification as an equally important part of the development process and to help our industrial partners with their efforts to manufacture a reliable product.

Acknowledgements

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

References

[1] D. Bošnački, Extending Promela and Spin with Discrete Time, in: Proc. of the VIII Conference on Logic and Computer Science (1997).

[2] M. Bozga, J.C. Fernandez, L. Ghirvu, S. Graf, J.P. 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] L. Doldi, Validation of Communications Systems with SDL: The Art of Simulation and Reachability Analysis (John Wiley & Sons, Ltd, West Sussex, England, 2003).

[4] G.J. Holzmann, Practical methods for the formal validation of SDL specifications, Computer Communications 15(2)(1992) 129–134.

[5] G.J. Holzmann and J. Patti, Validating SDL specifications: An experiment, in: C. Vissers and E. Brinksma, ed., Proc. 9th Int. Conf on Protocol Specification, Testing, and Verification, INWG/IFIP (Twente, Netherland, 1989) 317–326.

[6] ITU-T, Generic multiplexer using V.42 LAPM-based procedures, Recommendation V.76, 1996.

[7] B. Vlaovič, Automatic Generation of Models with Probes from the SDL System Specification, Ph.D. Thesis (in Slovene), Faculty of Electrical Engineering and Computer Science, University of Maribor, 2004.

[8] B. Vlaovič and Z. Brezočnik, Testing of Switch Node with Call Generator Software Module, in: Proc. of the IASTED international conference, Applied informatics (IASTED/ACTA Press, Zürich, Switzerland, 2000) 568–574.

[9] B. Vlaovič and Z. Brezočnik, Analog Subscriber Call Generator, Electrotechnical Review, 69(5)(2002) 259–265.

[10] B. Vlaovič, A. Vreže, Z. Brezočnik, and T. Kapus. Verification of an SDL Specification — a Case Study, Electrotechnical Review, 72(1)(2005) 14–21.

[11] A. Vreže, Extending Automatic Modelling of SDL Specifications in Promela with Embedded C Code and a New Model of Discrete Time, Ph.D. Thesis (in Slovene), Faculty of Electrical Engineering and Computer Science, University of Maribor, 2006.

[12] A. Vreže, B. Vlaovič, Z. Brezočnik, and T. Kapus, Development of MGCP protocol stack for SI2000 digital switch node, Electrotechnical Review, 72(1)(2005) 22–29.

[13] ITU-T, Specification and Description Language (SDL), Recommendation Z.100, 1993.