Modelling and Analysing the SDL Description of the ISDN-DSS1 Protocol *

Nisse Husberg, Teemu Tynjälä, and Kimmo Varpaaniemi

Helsinki University of Technology,
Laboratory for Theoretical Computer Science,
P.O. Box 9700, FIN-02015 HUT, Espoo, Finland
`Nisse.Husberg@hut.fi, Teemu.Tynjala@hut.fi, Kimmo.Varpaaniemi@hut.fi`

Abstract. The modelling of a telecommunication protocol, ISDN-DSS1, defined using SDL, is described and the problems and methods used are discussed. The formalism used for the model is a high-level Petri net, the PROD input language which is close to a predicate/transition net. The influence of the model on the reachability analysis is also discussed with a special attention on the use of priorities and reduction methods. Finally, further development of tools which aid in modelling and analysis of this class of systems is discussed.
Keywords: ISDN, DSS1, SDL, protocol verification, reachability analysis, high-level Petri nets

1 Introduction

We present modelling and analysis of the ISDN-DSS1 [4] protocol. The project originated in problems with the implementation of DSS1 in telephone exchanges used by the telephone operator Helsinki Telephone Corporation. The task was to check whether the DSS1 protocol allowed false connections, and, if the protocol itself was correct, to test the implementations, i.e. the concrete switches, in order to find possible loopholes in the implementations of the protocol.

DSS1 is the network level protocol used in the ISDN network. It is responsible for connecting and releasing traffic channels as well as reserving and releasing the Call Reference (a unique reference tag used to identify a call in progress across the ISDN network). The ISDN network has differentiated the signalling channels and traffic channels from each other. Traffic channels are referred to as *B channels* and the signalling channels as *D channels*. The DSS1 protocol signalling is carried out only on the D channel.

A block diagram of the DSS1 protocol and the other protocol blocks to which it is related is given in Figure 1 . The operation of DSS1 is tightly related to the *Call Control Block*. The Call Control Block enables and disables accounting and initiates call setup and teardown. DSS1 offers a Service Access Point (SAP) to the Call Control Block.

* This work has been supported by the ETX program of The National Technology Agency of Finland and by Helsinki Telephone Corporation.

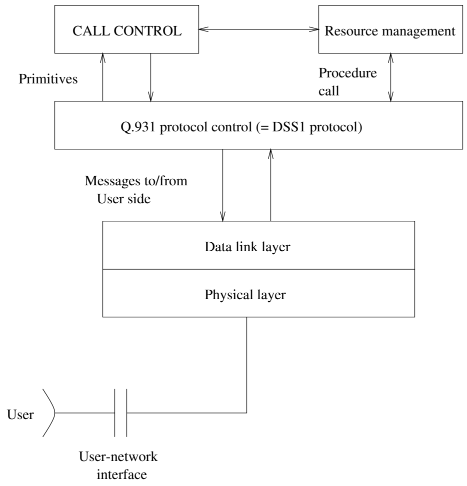

Fig. 1. DSS1 and related protocols ([4], p. 11).

The SDL [13] diagrams in the standard [4] of DSS1 were used as the source material in the project. PROD [19], a reachability analysis tool for Pr/T-nets, was used in the analysis. Modelling SDL is rather easy using high-level Petri nets, but there are a few problems, like the input queues and the timers. However, even if a system is described in a clear and formal way, it is necessary to *prepare* the model for analysis, usually abstract from a lot of implementation features. Although the DSS1 protocol specification is in itself fairly abstract, there are details which might be irrelevant for the specific analysis.

2 SDL and High-Level Nets

There are two different forms of SDL, the Graphical Representation (SDL/GR) and the (textual) Phrase Representation (SDL/PR). Here we are referring to the *graphical* form of SDL because the DSS1 description [4] is given using SDL diagrams.

2.1 The Graphical Representation of SDL

If we forget about the block structure of SDL, a system can be thought of as a collection of *SDL processes*. An SDL process can be seen as an *extended finite state machine*. It has an *input queue*, and the communication between processes is *asynchronous* and can only happen by putting signals (messages) into the queue. A process also has *timers* which have a fairly complex semantics. When a timer expires it sends a signal to the input queue of the process. However, if a timer which has expired is reset, the signal it has put into the queue should be removed.

An SDL process has a number of *SDL states* in which the process is waiting for an input from the queue. After the input (or possibly a SAVE statement), the process makes an *SDL transition* which consists of a number of *SDL statements* like OUTPUT, TASK, CREATE (a process), DECISION etc. The SDL transition always ends in a NEXTSTATE or other terminator statement.

In Figure 2, the first "box" is an SDL state (DSS1 *call state* 1: CALL INITIATED), the second an INPUT statement for the signal PROCEEDING REQUEST, the third an OUTPUT statement for signal CALL PROCEEDING with *parameter* B-CHNL, the fourth a TASK statement CONNECT B-CHNL and the last one a NEXTSTATE statement moving the control to DSS1 call state 3: OUTGOING CALL PROCEEDING.

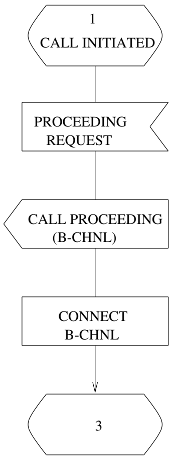

Fig. 2. An SDL transition.

2.2 Tools for SDL Analysis

There are commercially available tools for SDL, like SDT (Telelogic Tau SDL Suite) [20] and ObjectGEODE [17], but these mainly use *simulation* for the analysis of the systems. We are interested in applying *reachability analysis* to SDL systems (but not only to these) and especially in developing methods to handle such analysis efficiently.

We also have an analyser, EMMA [7–10], which is a front-end for our reachability analyser PROD. EMMA is, however, made for TNSDL (TeleNokia SDL) which is a special dialect of SDL-88. It is furthermore a *programming language* with data types and control structures. Thus it is quite different from the SDL diagrams used in the DSS1 standard. The use of EMMA in this work would have required the rewriting of the SDL diagrams as TNSDL code. On the other hand, the model generator of EMMA is somewhat inflexible and often produces strictly unoptimal nets w.r.t. both analysis and readability. Therefore, a net model was made for the PROD analyser manually [14].

3 Modelling DSS1 Using Pr/T-Nets

For pragmatic reasons, it was decided to treat the primitives sent by the Call Control Block as "spontaneous". (The only so far obtained public description of the block is not very informative. Effectively, it only tells us that the state of the block can be fully calculated from the information that has been in the high-level net models all along.) An arbitrarily behaving Call Control Block can force the User side and the Network side to make their output queues full w.r.t. the chosen capacities and then immediately ask both sides to write again. In this way, a deadlock is reached. The phenomenon is independent of the capacities of the queues as long as all the capacities are finite.

In the chosen way to model DSS1, in order to avoid deadlocks of the above kind, it was decided that a request from the Call Control Block is taken into account when and only when the request can be realized immediately. For the same reason, it was decided that if a reading of a timeout signal has an associated write operation, the reading is synchronised with the writing. In principle, the latter decision damages the timeout mechanism, but if the timer abstraction described below is accepted, the decision becomes acceptable, too.

For the purposes in our analysis the few messages received from the Data Link layer were deemed extraneous.

The SDL description includes two resources which our Pr/T-net model must adopt. The first is the B channel state which is always exactly one of the following: *connected*, *committed* or *free*. In our net, there is a place for each of the three states on both the Network and the User side. The other resource to be modelled is the call reference. A call reference is used as an identifier for a single call in an ISDN network. It has two possible states, *free* and *committed*. Our model includes a place for both states on the Network and the User side. As for transitions related to the resources, a *select* transition moves a B channel from free to committed, and *connect* moves a B channel from committed to connected. A *disconnect* transition does the inverse of *connect* and *release* does the opposite of *select*. For call references the situation is similar. A *select* transition changes the state of call reference from free to committed and *release* does the opposite action.

The use of "don't care" symbols is one of the most popular abstraction techniques and an obvious way to alleviate state space explosion. At certain states of the models of DSS1, the status of a B channel was known to remain unused until any potential next setting of the status and unused in any such setting as well. At such states, a fixed "don't care" value was used for representing the status.

The SDL diagrams defining the DSS1 protocol have divided the flow of control into 17 states on the Network and 16 states on the User side. Each of these *call states* is represented by one net place. When a token moves from one such place to another it indicates a change in the call state. The mapping of SDL states to net places is very intuitive.

The modelling of the SDL diagrams was quite straightforward except for some problems with timers, queues, atomic and non-atomic SDL transitions which are discussed below.

3.1 Converting SDL to Pr/T-Nets

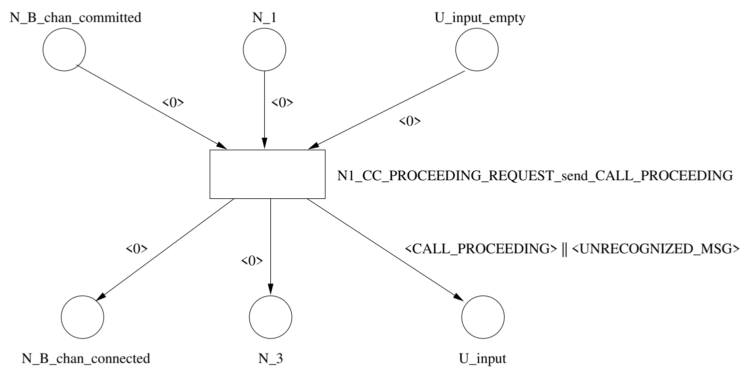

Fig. 3. A Pr/T-net transition corresponding to Figure 2.

Let us see how one of the SDL transitions in the DSS1 standard [4] can be modelled. The SDL transition in Figure 2 can be modelled by the net transition in Figure 3. The places N_1 and N_3 are modelling the control points in the SDL diagrams. The places N_B_chan_committed and N_B_chan_connected are Network side resources (variables). Finally, the places U_input_empty and U_input model a fixed writing slot of the User side input queue. (One might think that we should always keep the contents of the queue in some "normal form". However, as becomes explained in Section 4.3, we have a good excuse for not doing that.) The output arc to place U_input contains either CALL_PROCEEDING or UNRECOGNIZED_MSG as the message to be transmitted. The model will generate an execution path for both possibilities. The latter represents an erroneous message transmission or message corruption. This is discussed further in Section 3.4. Figure 3 translated into PROD code is shown in Figure 4.

It is justifiable to model an SDL transition by one net transition if the SDL transition is *atomary*, i.e. has no interference with the other SDL processes or global resources in the protocol, and if the SDL statements in the SDL transition are independent of each other. (See [10] for a discussion about merging of SDL statements in general.) A non-atomary SDL transition needs several net transitions. Then the protocol control can be modelled by *internal call state places*. According to the semantics of SDL no signals can be read by the process from its input queue unless the control is in the correct (call) state. Thus the internal call

```
#trans N1_CC_PROCEEDING_REQUEST_send_CALL_PROCEEDING
    in      { N_1: <.0.>;  N_B_chan_committed: <.0.>;
              U_input_empty: <.0.>;  }
    out     { N_3: <.0.>;  N_B_chan_connected: <.0.>;
              U_input: <.msg.>;  }
    comp    { msg = CALL_PROCEEDING;    Accept();
              msg = UNRECOGNIZED_MSG;  Accept();  }
#endtr
```

Fig. 4. A PROD representation of Figure 3.

state places only allow interleaving of the SDL transition with net transitions generated from other SDL processes.

3.2 Modelling Communication in the DSS1 Model

In SDL the input queues are infinite, but in real systems the queue length is, of course, finite. The analysis of long queues is usually impossible due to the huge state space which they generate. Restricting queue lengths, however, brings up a new problem: what to do if a process wants to write to a queue that is full? There are at least three ways of overcoming the problem. One is that the message is modelled as being lost. In some systems this may be a wise modelling choice. The second way is to overwrite the last element in the queue. This way is analogous to the first way since the only difference is in the choice of the lost message. The third way is to prevent the process from writing to the queue until there is room. Our DSS1 model follows the third way. The reason for this is that in reality very few messages are lost in a wire line telephone network.

3.3 Modelling SDL Timers

There is no time concept in the net class used in PROD. In EMMA, a timer is modelled by a net transition with a usual control place and another Lock place. The timer is started by the arrival of a control token in the control place, but the Lock place stops the timer from expiring. EMMA defines a *timer window* where the timer can expire, by using other transitions in the model to *open* and *close* the Lock. The expiry window can be moved to “interesting” SDL statements to see what happens if the timer expires concurrently to those statements. The net transitions in the window will be interleaved with the timer expiry transition in all possible ways in the analysis.

The expiry of a timer sends a timeout signal to the input queue of the process. This approach is justified by the fact that the communication between processes is completely asynchronous and the time concept has a meaning only within the same process.

In our model we have adopted a variant of this approach. Each timer is represented by two net places: one place indicates that the timer is ON and the other indicates that it is OFF. A *start* transition takes a token from timer_on and puts it to timer_off. A *timeout* transition does the reverse. The timer window in which a timeout can occur is normally small: one call state. DSS1 has been designed in such a way that a maximum of two timers can be on simultaneously in any call state.

Our model does *not* strictly follow the SDL semantics of time. Setting and resetting of timers as well as reading of timeout signals were modelled, but time itself was not modelled, partially because the timers were essentially not "competing" with each other, and partially because any serious inclusion of time would have seriously complicated all analysis. Reading of timeout signals was modelled without modelling the signals explicitly and without modelling expiries explicitly. It was assumed that whenever the latest explicitly modelled operation concerning a timer is a set-it-on operation, a timeout signal from the timer is readable. At worst this means that a timeout signal may become read before some"normal" signal that has been in the input queue for a longer time.

Even if we had decided to follow the SDL semantics, we would have had the problem that the SDL description in [4] is incomplete w.r.t. timers. For example, the timer T307 of the Network side is such that the SDL description in [4] refers to it only in a footnote on page 19: "The expiry of timer T307 is not shown in these SDLs as it runs in the Call Control Block." The project has tried to obtain an SDL description of the Call Control Block, but according to [11], the block does not have any public SDL description. However, a detailed textual description of T307 was found from [3]. The description is shown in Figure 5.

Timer number	Default time-out value	State of call	Cause for start	Normal stop	At the first expiry	At the second expiry	Cross reference
T307	3 min	Null	SUSPEND ACKNOW-LEDGE sent.	RESUME received.	Clear the network connection. Release call identity.	Timer is not restarted.	Mandatory

Fig. 5. An excerpt from "Table 9.1 (2 of 3): Timers in the network side" ([3], p. 153).

3.4 Representing Errors in the DSS1 Model

One of the purposes of the DSS1 model is to verify the correctness of the protocol in various error situations. The DSS1 model considers two erroneous behaviours of a protocol, firstly the loss of a message in the transmission channel, secondly the distortion of data in the transmission. Both errors are simulated by the transmission of a special signal UNRECOGNIZED_MSG. Each time a message is sent there are two possibilities. Either the intended message or the erroneous UNRECOGNIZED_MSG is transmitted over the line.

We used the capability of PROD to model non-deterministic behaviour in net transitions. In the analysis, this means that in addition to correct behaviour, all possible error paths related to message loss and corruption are included in the reachability graph.

3.5 Modelling Accounting in DSS1

As an end-to-end communication protocol, DSS1 plays an integral role in accounting [11]. The Call Control Block is the entity carrying out the actual accounting. Accounting starts when the remote party answers the phone and can end in two ways. One is that the Network side initiates call teardown (this can be thought of as the remote party replacing the receiver). In this case the Call Control Block (in the local exchange) disables accounting as it sends the message DISCONNECT_REQUEST to DSS1. The second manner of call teardown is initiated by the User side of the DSS1 protocol (analogous to the initiator party replacing the receiver). The local exchange of the call initiator will then receive call teardown messages and its DSS1 block shall send messages DISCONNECT_IND or RELEASE_IND to the Call Control Block. The reception of either of these messages will disable accounting.

The procedure described above is included in the DSS1 model. The addition of accounting means that the model is extended with two new places, Call_type and Accounting. Call_type describes whether the call is incoming or outgoing. Call charging is applicable only to outgoing calls since our model does not consider collect calls. The Accounting place contains a token indicating whether accounting is active or not.

The transitions handling the connection, disconnection and release of B channels must be augmented with the above places. Transitions that model the activation (connection) of B channels also enable accounting whereas transitions modelling disconnection or release of B channels must make active accounting inactive.

3.6 About the Sizes of the Nets

The modelling covered 145 pages of [4], all these pages consisting of SDL diagrams only. Due to parameters, options and abstractions, there are several models which are variants of each other. For example, the capacity of a queue is a parameter, whereas the inclusion/exclusion of UNRECOGNIZED_MSG is an option. Moreover, some analysis tasks imply inclusion of transitions that would otherwise be excluded.

The number of high-level places and high-level transitions strongly depends on the degree of folding. Each of the nets described in [14] has about 600 high-level places and about 1100 high-level transitions. By carrying out systematic folding, more compact nets have later been obtained, one of them having precisely 10 high-level places and precisely 70 high-level transitions. It would theoretically be possible to continue folding until there would be only one high-level place and only one high-level transition.

If we really want to measure the amount of information in a high-level net, we are more or less forced to talk about a low-level net that corresponds to the high-level net "in a reasonable way". For each analysed model of DSS1, the low-level net computed by PROD has more than 1000 places, more than 10 000 transitions, and more than 90 000 arcs.

4 Analysis of the DSS1 Model

4.1 About the Space Consumed in the Analysis

The amount of information in the full reachability graph depends very strongly on the parameters, options and abstractions. In order to measure the amount of information in a (full or reduced) reachability graph, we should take into account not only the number of vertices and edges but also the contents of the vertices and edges. In the case of PROD, the amount of hard disk space consumed by the graph can be used as a measure. When, as usual, the operating system uses 32-bit addresses for addressing a file, 2 gigabytes is an upper bound for the size of a single file that PROD can handle. In the case of the DSS1 models, this upper bound has been encountered several times. It would of course have been possible to choose an operating system that uses 64-bit addresses instead. However, hard disks of the day still have considerably limited capacities. On the other hand, mere availability of physical resources will hardly ever solve the state space explosion problem.

Due to the unbounded queues of SDL, the complete state space of an SDL system is in principle infinite. In the project being presented, the queues were abstracted to have the capacity 1 or 2, and consequently, the full reachability graphs of the net models were guaranteed to be finite. For a certain model where every queue had the capacity 2, the full reachability graph was still known to be too large to be generated in an operating system that had the above mentioned 32-bit problem. Fortunately, the stubborn set method [15, 16] in PROD successfully constructed a reduced reachability graph that sufficed for showing that the full reachability graph had no terminal state. However, even that construction took more than two days. Dropping all the queue capacities from 2 to 1 changed the situation dramatically: it took less than one hour from PROD to construct the full reachability graph and to conclude that it was strongly connected.

4.2 Prioritised Transition Instances for Fast Debugging

Constructing a model of a large system by hand is inevitably error-prone. Detection of modelling errors needs practically as much automatic support as the search for the actual errors of the system. When there are many modelling errors, it is often the case that almost every branch in the reachability graph contains an indication of a modelling error. The state space explosion problem still complicates reachability analysis so much that some way to reduce the reachability graph is needed. Assigning *priorities* to transition instances is an intuitive way to reduce the reachability graph, with the risk that the intuition does not fully grasp the reality.

If an instance a has a priority over an instance b, at most a is allowed to be executed when a and b are simultaneously enabled. (We said "at most" since there can be an instance c which has a priority over a.) The user of PROD is allowed to assign the priorities quite arbitrarily. The cost of such a freedom is that the user is responsible for the quality of the reduced reachability graph. However, the quality is not a problem as long as errors are found from the reduced reachability graph. Intentionally restrictive priorities can be used for reducing the time needed for detecting some of the errors. (Randomly incomplete error detection is a research field of its own. See e.g. [6].)

It is possible to use priorities in such a way that also the absence of errors can be concluded from the reduced reachability graph. As shown by [1], such a controlled priority method is effectively the stubborn set method with strong guidance from the user. In PROD, the priority method has not been integrated with the stubborn set method.

4.3 Supporting the Stubborn Set Method

In low-level Petri net terms, a stubborn set consists of transitions. The stubborn set method computes a stubborn set in every state of the reduced reachability graph which in turn is determined by the computed stubborn sets. The reduced reachability graph is guaranteed to contain all terminal states of the full reachability graph. Additional constraints can be used in order to preserve more properties, such as the validity/invalidity of an LTL formula without a next-time operator. The reduced reachability graph can be finite even when the full reachability graph is infinite.

The stubborn set method in PROD works on P/T-nets (place/transition nets). The needed unfolding from a Pr/T-net into a P/T-net is done automatically, but the user must understand e.g. the correlation between the number of high-level transition instances and the number of low-level transitions. We now consider two difficulties which have been encountered in the DSS1 project. The first difficulty is related to the modelling of queues, whereas the second difficulty is related to redundant places. Both of the difficulties are such that a modeller can circumvent them, and they were circumvented in the DSS1 project in particular. (The difficulties could to some extent be eliminated automatically, but then there should be tools which would really do that. We shall return to this subject in Section 5.3.)

One thing the modeller should learn is that the stubborn set method does not like "simultaneity by atomicity". For example, think of a high-level transition which models reading of a single element from a queue by taking the whole contents of the queue as an input and by returning the new contents as an output. If the capacity is low enough, unfolding succeeds, but the low-level transitions corresponding to the read operations tend not to commute with the low-level transitions that correspond to the write operations on the same queue. The reduced reachability graph then becomes much larger than necessary. What the modeller should do is to make the elements in the queue move one element per time and one slot per time. With minimal additional guidance from the user, the stubborn set method in PROD is clever enough to avoid introducing redundant interleavings that would be due to the change in atomicity. PROD also has an option which eliminates "intermediate states" [16], such as the states where the queue is not "in the normal form".

The stubborn set method does not easily recognise redundant places, and so it is best to avoid such places. For example, imagine a high-level place that keeps count of the number of tokens in another high-level place while the only use for the value of the counter is to compare the value to a known theoretical maximum. If the value of the counter is represented by a high-level token, transition instances incrementing the counter do not commute with the instances that decrement it. If the value of the counter is represented by a multiplicity of low-level tokens, the stubborn set method falsely assumes that these tokens form a critical resource. (If the counter place also has a classic complement place, the method analogously assumes that the tokens in the complement place form a critical resource, too. Such a pair of critical resource assumptions has a concrete effect, no matter which stubborn set computation algorithm we choose from the literature.) Then again, the reduced state can become much larger than necessary.

4.4 Fairness and Diagnosis of Counterexamples

Liveness properties to be verified were written as LTL formulas, without including any fairness assumption in the formulas. Since PROD has no built-in support for fairness assumptions, unfair counterexamples were obtained. Looking at the counterexamples, it was observed that the unfairness was always caused by messages interchanged during a fully established phone call. Such messages were redundant w.r.t. the properties being verified, and it was easy to abstract such messages out from the model. After sufficiently many redundant messages had been abstracted away, the liveness properties were seen to hold.

The LTL counterexamples reported by PROD are paths in the product of a reachability graph and a certain Büchi automaton. A counterexample which is found is reported in the form of a "characteristic prefix" that contains exactly one cycle, the cycle being at the end of the prefix. (Each terminal state in a reachability graph is transformed into an artificially infinite path by adding a dummy edge from the terminal state to the state itself.) In the analysis of the models of DSS1, a single characteristic prefix reported by PROD typically had hundreds of states. By carrying out separate semiautomatic searches, it was found out that considerably shorter prefixes would have been obtained if PROD had chosen some other counterexamples to the same formulas.

4.5 Potential Errors Found during the Analysis Procedure

The analysis of DSS1 protocol uncovered three problems with the protocol specification. Two are of minor importance but the third one encompasses almost every call state. Fortunately none of the three problematic definitions is disastrous to the correct operation of the protocol. Each one is introduced briefly in the subsequent paragraphs.

The first problem is on the User side of the protocol. Consider Figure 6 which shows two branches from different call states on the User side. In the call state 6 (CALL PRESENT), it is possible to receive a PROCEEDING_REQUEST from the Call Control Block. As the branch shows, the control moves to the call state 9 (INCOMING CALL PROCEEDING), without any timers starting. In the call state 9, it is possible to receive a DISCONNECT message. The branch on the right hand side of the figure shows the actions performed as a result. The first action is to turn off the timer T313. However, that timer is not on in the execution path described. By investigating all the other diagrams that refer to T313, it can be shown that T313 is never on in the call state 9. So, either the resetting of T313 in Figure 6 is a redundant operation, or a few things are missing from the graphical representation of the standard. The project proceeded by assuming that the resetting is redundant.

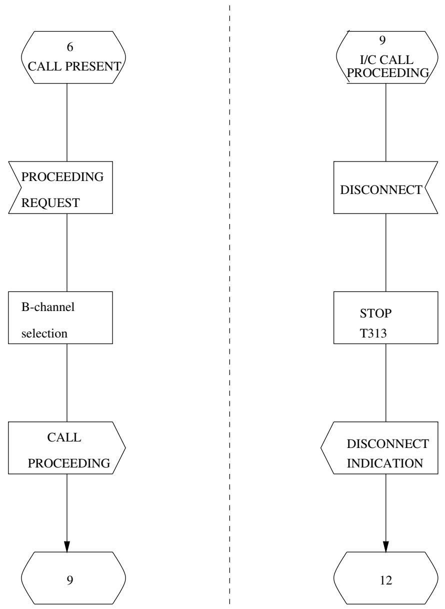

Fig. 6. A potential error on the User side.

The second problem (or inaccuracy) of the specification can be found on the Network side. In the call state 15 (SUSPEND REQUEST), the execution after Call Control message SUSPEND_RESPONSE is given in Figure 7. The control moves to the call state 0 (NULL) after the Call Reference is released. However, the B channel is left in an active state. This is a contradiction with the

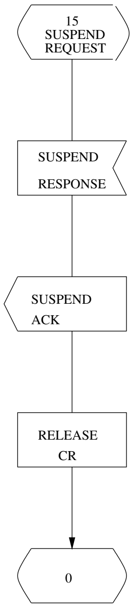

Fig. 7. A potential error on the Network side.

knowledge that the project had about accounting, i.e. accounting was supposed to be inactive during call suspension. The project decided to formally keep the B channel in a "suspended" state when the call is suspended.

The third problem, which is more serious, in the protocol description concerns timers that are possibly left on as the control moves to Call State 0. Every call state has transitions for the reception of RELEASE and RELEASE_COMPLETE. These transitions do not turn off timers before control moves to call state 0. If a timer is active in call state 0, it may remain active in some subsequent call states too and expire at a moment it should not expire. In order to make further analysis free from the effects of this error, it was eliminated from all variants of the net model by making every reception of messages RELEASE or RELEASE_COMPLETE to reset all timers.

In principle, the same observations could have been made by using static analysis only. However, we have the opinion that some of the positively verified safety and liveness properties would never have become verified without a true reachability analysis tool. It should also be pointed out that the potential problems in the implementations of DSS1 are not necessarily caused by the standard.

4.6 About the Positively Verified Properties

Section 4.1 already mentioned two positive verification results. More generally, whenever PROD succeeded in generating a full reachability graph for a correct model of DSS1, each queue had the capacity 1 and the graph was strongly connected. On the other hand, whenever the stubborn set method in PROD succeeded in generating a reduced reachability graph for a correct model of DSS1, the graph had no terminal state. (The "largest" of these correct models is the one mentioned in Section 4.1, i.e. a model where each queue has the capacity 2.) Provided that the method has been correctly implemented in PROD, the stubborn set theory [15, 16] implies that the full reachability graph of the same model has no terminal state either. Though the absence of terminal states is not necessarily any actual requirement, terminal states typically indicate poor design. Analogously, if a full reachability graph is not strongly connected, a good explanation is typically needed.

Though the stubborn set theory and its implementation in PROD supports e.g. verification of LTL formulas, the following results are limited to the cases where the generation of the full reachability graph is no problem.

The positively verified safety properties were related to the correctness of the status of the B channel and to the correctness of accounting, up to the abstraction used in the modelling of the accounting. Each of these properties can be verified by using an additional high-level transition that is enabled precisely at those states which violate the property. For example, it was shown that the accounting is correct w.r.t. those states where the input queue of the User side is empty. (The project did not find a meaningful "correctness of accounting" criterion that would have covered all states.)

The positively verified liveness properties concerned the repeatability of the call state 0. As said in Section 4.4, the liveness properties were formulated as LTL formulas. For example, it was shown that if the Network side is infinitely often in the call state 0, the User side is infinitely often in the call state 0, too.

5 Future Work

5.1 DSS1 Testing System

The modelling of the DSS1 protocol was the first stage in a subproject of the MARIA [18] analyser project. The task is to check the operability of the implementations of the DSS1 protocol using formal methods. We have, however, no access to the documentation of all these implementations. It is thus necessary to design a test system where the implementation is treated as a black box.

The test problem is not one of conformance analysis. These exchanges have been thoroughly tested already and probably work very well according to the conformance tests. The problem is that in real life, there are some users which are behaving very badly. It is thus necessary to create tests which follow the protocols to some extent only.

The problem is that the possibilities are enormous, not only in the sequences of messages but also in the timing. The strategies for such testing are not within the range of this paper but a testing system has been created which can connect this model to the ISDN line of an exchange. The model can be used to create correct sequences of communication and then distorted a little in different ways. Some heuristics can be used to create sequences which have a fair chance of fooling the implementation.

So far the project has succeeded in building a system which attempts to realize any given sequence of send/receive actions. Though the built system has not yet found errors in any exchange, it has detected a modelling error: a certain passed sequence indicated that the behaviour of a certain exchange conformed to an unmodelled branch in a certain SDL diagram.

5.2 MARIA: a Modular Reachability Analyser

During the EMMA project, some deficiencies of the implementation of PROD became obvious. There were problems with the expression power of the input language, especially missing data types. On the other hand, adding new algorithms to PROD tended to be more and more complicated in each addition. Therefore, it was decided that a new modular analyser should be designed.

The MARIA analyser is to have all the capabilities of PROD, but it is *modular* in order to make it easy to develop it further and to add new front-ends and algorithms. We hope to give a large group of users the possibility to alter the code to suit some particular problem.

One of the main tasks in the analyser project is to build a front-end for standard SDL. We also plan to add interfaces to MARIA to allow it to read the output of some largely used SDL tools. Thus the manual error-prone and time-consuming modelling work could be avoided or at least supported by efficient tools.

5.3 The Effect of the DSS1 Project on MARIA

The Partial-Order Package [5] solves the queue problem of Section 4.3 without disturbing the modeller. The solution is to make the queue a black box which can only be accessed by built-in operations of the protocol description language. In PROD, implementing the idea would mean constructing a "black box low-level subnet" which would be handled exceptionally. We plan to implement this idea in the MARIA tool.

An interesting way to solve the problem of redundant places, also discussed in Section 4.3, would be to refine the definition of concrete stubbornness towards the definition of *dynamic stubbornness* [16]. However, a more straightforward way to solve problems related to recognition of redundancy is to carry out a *net reduction* [2] before constructing any reachability graph. Section 4.3 suggests that it may be best to avoid promoting "simultaneity by atomicity" in a net reduction. If some algorithms later need the original net, a bidirectional translation between the original net and the reduced version suffices for that purpose. Unfortunately, PROD has no automatic support for net reductions. It would thus be nice if MARIA had at least an interface to some net reduction tool. (The laboratory where MARIA is being developed actually has a net reduction tool, called NRED [12], but NRED does not very well support a modern analyser.)

Though justifiable in special cases, categorical elimination of certain messages from a model, as was done in Section 4.4, is not a good general purpose technique for elimination of unfair counterexamples. The MARIA tool is to have explicit support for verification under fairness assumptions.

6 Conclusions

In this work, we have modelled and analysed the ISDN-DSS1 protocol using high-level Petri nets. Several experiments with the stubborn set method have been performed, and the use of priorities has been critically investigated. Like many others, we have learnt something about the benefits and risks of abstraction.

References

1. Falko Bause. Analysis of Petri Nets with a Dynamic Priority Method. In P. Azéma and G. Balbo, editors, *Application and Theory of Petri Nets 1997*, volume 1248 of *LNCS*, pages 215–234. Springer-Verlag, 1997.
2. Gérard Berthelot. Transformations and Decompositions of Nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Central Models and Their Properties (Advances in Petri Nets 1986, Part I)*, volume 254 of *LNCS*, pages 359–376. Springer-Verlag, 1987.
3. European Telecommunications Standards Institute. *Integrated Services Digital Network (ISDN); User-Network Interface Layer 3; Specifications for Basic Call Control; Part 1, ETS 300 102-1*, December 1990.
4. European Telecommunications Standards Institute. *Integrated Services Digital Network (ISDN); Digital Subscriber Signalling System No. One (DSS1) Protocol; Signalling Network Layer for Circuit-Mode Basic Call Control; Part 2: Specification and Description Language (SDL) Diagrams, ETS 300 403-2*, November 1995.
5. Patrice Godefroid and Didier Pirottin. Refining Dependencies Improves Partial-Order Verification Methods. In C. Courcoubetis, editor, *Computer Aided Verification (CAV '93)*, volume 697 of *LNCS*, pages 438–449. Springer-Verlag, 1993.
6. Gerard J. Holzmann. An Analysis of Bitstate Hashing. *Formal Methods in System Design*, 13(3):289–307, November 1998.
7. Nisse Husberg. Verifying SDL Programs Using Petri Nets. In *1998 IEEE International Conference on Systems, Man, and Cybernetics (Volume 1)*, pages 208–213. IEEE, 1998.
8. Nisse Husberg and Tapio Manner. Emma: Developing an Industrial Reachability Analyser for SDL. In J.M. Wing, J. Woodcock, and J. Davies, editors, *FM'99 — Formal Methods (Volume I)*, volume 1708 of *LNCS*, pages 642–661. Springer-Verlag, 1999.
9. Tero Jyrinki. Dynamical Analysis of SDL Programs with Predicate/Transition Nets. Report B17, Digital Systems Laboratory, Helsinki University of Technology, 1997.
10. Markus Malmqvist. Methodology of Dynamical Analysis of SDL Programs Using Predicate/Transition Nets. Report B16, Digital Systems Laboratory, Helsinki University of Technology, 1997.
11. Sakari Pernu (Nokia Telecommunications), private communication, 1998 and 2000.

12. Marko Rauhamaa. Design and Implementation of a Reduction Tool for PrT-Nets. Report B6, Digital Systems Laboratory, Helsinki University of Technology, 1988.
13. Specification and Description Language (SDL), ITU-T Z.100, 1988.
14. Teemu Tynjälä. Reachability-Based Verification of DSS1 Protocol. Master's thesis, Department of Computer Science and Engineering, Helsinki University of Technology, 1998.
15. Antti Valmari. State Space Generation: Efficiency and Practicality. Doctoral thesis, Publications 55, Tampere University of Technology, 1988.
16. Kimmo Varpaaniemi. On the Stubborn Set Method in Reduced State Space Generation. Doctoral thesis, Report A51, Digital Systems Laboratory, Helsinki University of Technology, 1998.
17. Worldwide web, page http://www.csverilog.com/products/geode.htm.
18. Worldwide web, page http://www.tcs.hut.fi/Software/maria/index.html.
19. Worldwide web, page http://www.tcs.hut.fi/Software/prod/index.html.
20. Worldwide web, page http://www.telelogic.se/solution/tools/sdl.asp.