SDL and Petri Net Performance Analysis of Communicating Systems

Falko Bause* Heinz Kabutz† Peter Kemper Pieter Kritzinger

Abstract

The automated functional and performance analysis of communicating systems specified with some Formal Description Technique has long been the goal of protocol engineers. In this paper we give a description of a Petri net enhanced with queued places which enables one to automatically translate an SDL specification to a net for its direct functional and performance analysis.

In particular, the new type of Petri net allows one to describe the process queues and SAVE construct, the FIFO channel queue, as well as timeouts which previously evaded temporal analysis of systems specified in SDL. The new net is called SDL-net. We analyze the performance of a modified Inres protocol using SDL-net as the modelling paradigm.

Keywords. Specification language, performance analysis, correctness analysis, Petri net, Markov process, protocol

1 Motivation

The automated functional and performance analysis of communication systems specified with some Formal Description Technique has long been the goal of protocol engineers as one can tell from the literature[2, 3, 9, 10, 13]. Although progress has been made in the functional analysis, notably through the work of Holzmann[7] and West[14], automated performance analysis has been less successful.

In recent years various researchers[4, 5, 11] have translated SDL specifications to Petri nets. The purpose of these translations was to test the correctness of the SDL specification; performance analysis, which imply that the both the specification and the Petri net model involved time, was not possible. The

*Falko Bause and Peter Kemper are from Fachbereich Informatik IV, Universität Dortmund, D44221 DORTMUND, Germany

†Heinz Kabutz and Pieter Kritzinger are from the Data Network Architectures Laboratory, Computer Science Department, University of Cape Town, Private Bag, RONDEBOSCH 7700, South Africa. All correspondence should in the first instance be directed to the last author: Telephone +27.21.650.2664, facsimile +27.21.650.3918, e-mail, psk@cs.uct.ac.za

translation by Taubert[11] was the only one that used timed Petri nets, but it used time only to derive a functionally more accurate model of the SDL model and not for performance measurement.

The translation of an SDL specification to a Petri net has the objective of using the well known theory of Place- and Transition-invariants and their relation to functional properties such as liveness and boundedness of the corresponding net. Two techniques are used to determine the functional correctness of a Petri net model: *Invariant analysis* which is a structurally-based technique, and *Coverability Graph analysis* which explores all reachable states of the system.

Performance analysis is done by deriving a Markov chain from the timed Petri net model of the specification. Time-augmented Petri nets which allow such a Markovian analysis are amongst others, Stochastic Petri nets, Generalized Stochastic Petri nets and Queueing Petri nets[1].

This document also describes a translation from SDL to a Petri net which we call an SDL-net. It differs from the earlier work cited, in that the primary objective is to analyse the performance of the system so described in SDL, while realising that the model needs to be functionally correct to be able to do the analysis. Correctness analysis is a natural consequence of the proposed method.

In the next section we discuss the aspects of an SDL specification of a distributed system which need to be accounted for in particular, if a model is to reflect the performance of the system in any degree of accuracy. Subsequent sections describe the elements of the SDL-net and the document concludes with an example of modelling the Inres protocol[6] in this way.

2 Time in SDL

The most important message delays in any communication system are the times spent waiting in a channel or signal buffer before processing. These times are orders of magnitude larger than the time taken by the processor to process a message or to place a message in, or remove a message from a buffer. Concomitant with these delays are the timeouts, which may result in retransmissions which cause further delays due to congestion. A realistic model for performance analysis must therefore model these effects as best as possible.

In the following we assume that the reader is familiar with SDL and describe only those features which are essential for a performance analysis. In SDL, time delays can occur at two points:

- Channels may delay signals in order to model transmission times. They act as FIFO queues and "serve" signals in order of their arrival.
- SDL specifications clearly allow for timeouts. On sending a signal, a timer is set to expire at some future time unless some other specified event occurs. When it expires, it inserts a timeout signal at the end of the

input queue, cf. Sect. 3, where it is processed like an ordinary signal. The statement *reset* is used to stop a timer. In this case a possibly pending, timer signal is removed from the input queue.

An example of how the timer construct may be used is shown in Fig. 1. In the transition from the start state to State_A, we set the timer to expire in 5 time units. While process ONE is in State_A, it waits until it receives either signal s_1 or s_2, or until the timer expires, in which case it would receive T. If either s_1 or s_2 arrives before the timer expires, the timer is reset before going to State_B or State_D respectively.

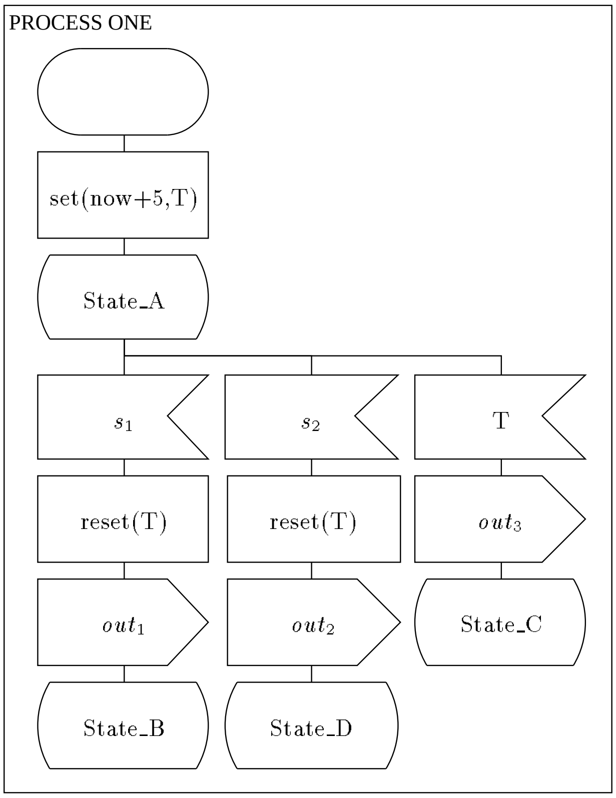

Figure 1: Timeout structure in SDL

Timeout durations are considered to be deterministic, although in practice timers may not be of an exact fixed length, depending, for instance, on the interrupt mechanism of the system. In general, channel delays are however stochastic, since congestion along a channel may involve other processes and events outside the control of the SDL process itself. In the theory to follow, all time distributions are assumed to be Coxian, which allows one to approximate any distribution arbitrarily closely by a series of negative exponential stages.

Apart from modelling time, input queues in SDL, and their associated SAVE mechanism, are difficult to represent by conventional Petri net formalisms. The next section describes a novel way to model these.

3 The input queue in SDL

Each SDL process has a dedicated input queue where arriving signals are queued in order of their arrival. The SAVE construct in SDL causes an exception to this FIFO scheduling strategy in that the SAVE construct allows the consumption of a signal to be delayed until one or more other signals, which arrive subsequently, have been consumed. In general an input queue operates as follows.

At every state, every signal is treated in one of the following ways:

- it is shown as an explicit input followed by a state transition,
- it is shown as a SAVE,
- it is covered by an implicit input followed by a null state transition.

An SDL process reviews signals in its input queue in the order of their arrival. An explicit, or implicit input-signal is removed from the queue and the corresponding transition executed. A signal shown in a SAVE construct is not removed and remains in its current position in the queue and the next signal in the queue is reviewed.

A single SAVE symbol can save a signal only while the process is at the state where the symbol appears, and saves it for the duration of the transition to the next state. At the next state, the symbol will be consumed via an explicit or implicit input, unless either the SAVE symbol with the signal name is repeated, or there happens to be another signal saved ahead of it in the implicit queue and available for consumption.

Several signals can be saved at each state. This can be represented by either having one SAVE construct for each signal, or by writing all the signal names in the SAVE construct. If several signals are to be saved, the semantics of the SAVE symbol implies that the order of their arrival is preserved.

An example of how the SAVE construct can be used is given in Fig. 2.

Assume that we have signals s_3, s_2, s_2, s_1 in that order of arrival. Assuming that we are in State_A, s_3 would be saved, s_2 would be explicitly consumed and the process would enter State_B. Now s_3 would be considered again, and explicitly consumed, leading to State_A. Next the second s_2 would be consumed explicitly, leading to State_B. In State_B, signal s_1 is not expected, so it is consumed implicitly, and the process remains in State_B.

In the following we describe a time-augmented Petri net for the analysis of SDL models. We call this new type of net an SDL-net.

4 SDL-net

SDL-net is a subclass of Queueing Petri nets (QPNs) [1] which are themselves Coloured Generalised Stochastic Petri nets (CGSPNs) with a new type of place,

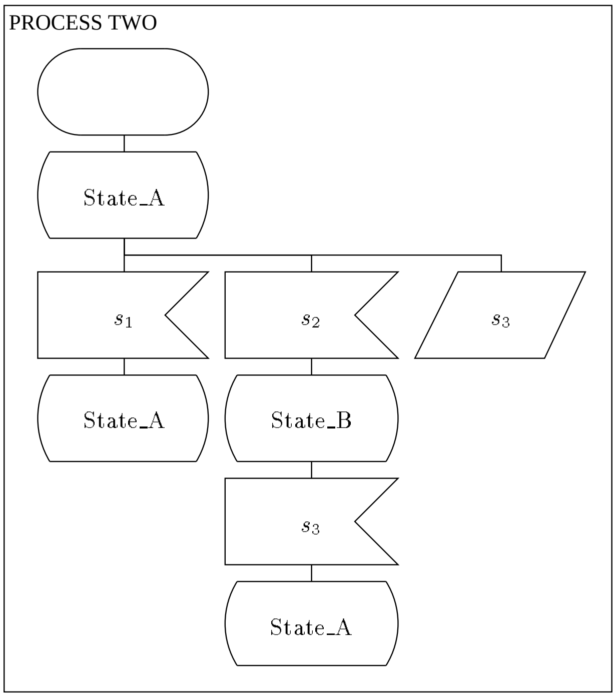

Figure 2: SAVE construct in SDL

called a *queued place*. A queued place, illustrated in Fig. 3, consists of a *queue* and a *depository* for tokens which have completed their service at the queue. Tokens, when fired onto a queued place by any of its input transitions, join the queue depending upon the scheduling strategy of the queue. Tokens in a queue are not available for the transitions. After completion of its service, the token is placed onto the depository. Tokens on the depository are available to all output transitions of the queued place as in the case of an ordinary place. As in GSPNs, individual timed transitions which are enabled will race to fire after an exponentially distributed delay time. The firing of immediate transitions has priority over those of timed transitions.

The QPN formalism distinguishes between two types of queues: timed and immediate [1]. Timed queued places act as described above. In immediate queued places the service in a queue takes place immediately. Immediate queued places make sense for specific scheduling strategies, like FIFO with blocking, where the queue only "serves" tokens when the corresponding depository is empty. We will use immediate queued places to model input queues of processes.

The general principle of queued places can be simplified to accurately model the SDL constructs described in Sects. 2 and 3. An immediate queued place called a *Save-place* is defined to model input queues and the associated SAVE construct in SDL, while a *Timeout-place* is defined to model the timeout mechanism in SDL – or any specification language, for that matter. Surely a Timeout-place is a timed queued place and allows for Coxian distributions in order to approximate deterministic timeouts.

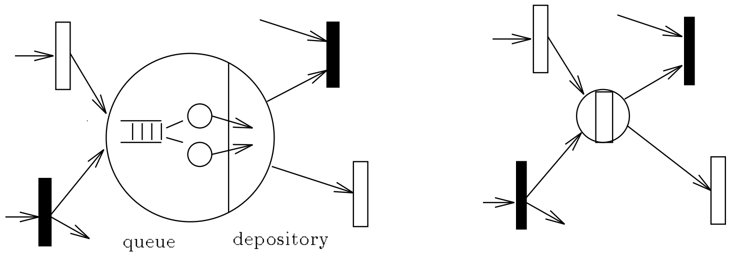

Figure 3: A (timed) queued place and its shorthand notation

Furthermore timed queued places with FIFO scheduling strategy and Coxian service distribution are used to model SDL's channel queues accurately.

4.1 Save-place

A Save-place is modelled by an immediate queued place with a modified FIFO strategy taking care of the SAVE construct.

A Save-place p has a colour set which consists of two complementary subsets $S(p)$ and $@S(p)$ such that any signal/colour $s \in S(p)$ which needs to be SAVED has a complementary colour $@s \in @S(p)$. In the sequel we drop the argument (p) in the notation since only a single place p is ever under consideration.

- A signal $s \in S$ joins a FIFO queue when fired onto the place.
- A signal $@s \in @S$ is however reserved when fired onto the place. These reserved signals only influence the scheduling of the FIFO-queue and are immediately available to the output transition set of the place. While a signal $@s \in @S$ is present, all signals of its complementary colour $s \in S$ are not served, i.e., they are held back in the FIFO queue according to the semantics of the SAVE construct.

The use of this construct is described by the example illustrated in Figs. 2 and 4 respectively.

In SDL-specifications signals in an input queue can be consumed explicitly, implicitly consumed or saved. Note from those figures that, in order to retain the advantage of Petri net analysis algorithms, we define a queueing specification to describe an input queue only to consider explicit consumption and save-operations. Consequently, tokens inside such a queued place are neither created nor destroyed; creation and destruction of tokens must be expressed by appropriate transition firings.

4.2 Timeout-place

The reader is referred to Sect. 2 and Figs. 1 and 5 for the following description of the Timeout-place. Each timer in the specification is represented by a Timeout-place (place P_T in Fig. 5). The colour set of P_T consists of the timer signal

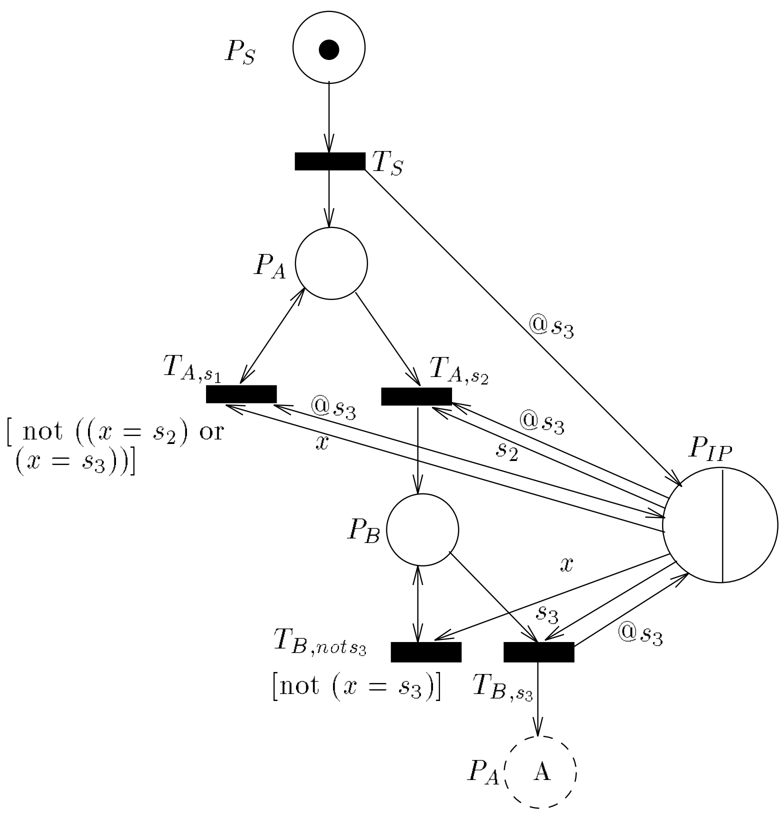

Figure 4: Equivalent representation in SDL-net using the Save-place

T and colour $\#T$ representing a complementary signal. The colour set of P_{IP} consists all possible input signals to process P. The colour set for all other places in this example is ω, meaning a token of arbitrary colour. It is not necessary that each timer should be represented by a unique Timeout-place. Here we will assume it to be the case, however, to simplify the discussion. The extension to the case where several timers are represented by a single Timeout-place with an appropriate colour set, is not very difficult.

- When the timer is set, a signal of colour T joins an Infinite Server queue where it is delayed for a colour dependent, randomly distributed time before being put on the depository. The distribution is Coxian with a number of exponentially distributed phases approximating a constant timeout length arbitrarily closely. Simultaneously a signal of colour ω is fired onto the place State_A.

- Should a signal of colour s_1 or s_2 arrive before the timer expires (i.e., before the signal T in place P_T is put in the depository) this signal, together with the signal ω will enable the corresponding immediate transition T_{s_1} or T_{s_2}. This transition now fires a signal $\#T$ onto the Timeout-place. This resets the timer. If a token of a complementary colour $\#T$ is fired onto the Timeout-place the queue in this specific queued place immediately aborts the "service" of the signal T, and both T and $\#T$ are placed onto the depository. If there is no signal T in the queue only the token of colour $\#T$ is placed onto the depository.

– Should a signal of unexpected colour arrive, e.g. s_4 or s_5, it is consumed by the transition with guard [not $((x = s_1)$ or $(x = s_2)$ or $(x = s_3))$]. The Timeout-place is not affected.

– Alternatively, should the timer expire first, the signal T, together with signal ω at place State_A enables immediate transition T_{s_3} which fires, thus disabling transitions T_{s_1} and T_{s_2}.

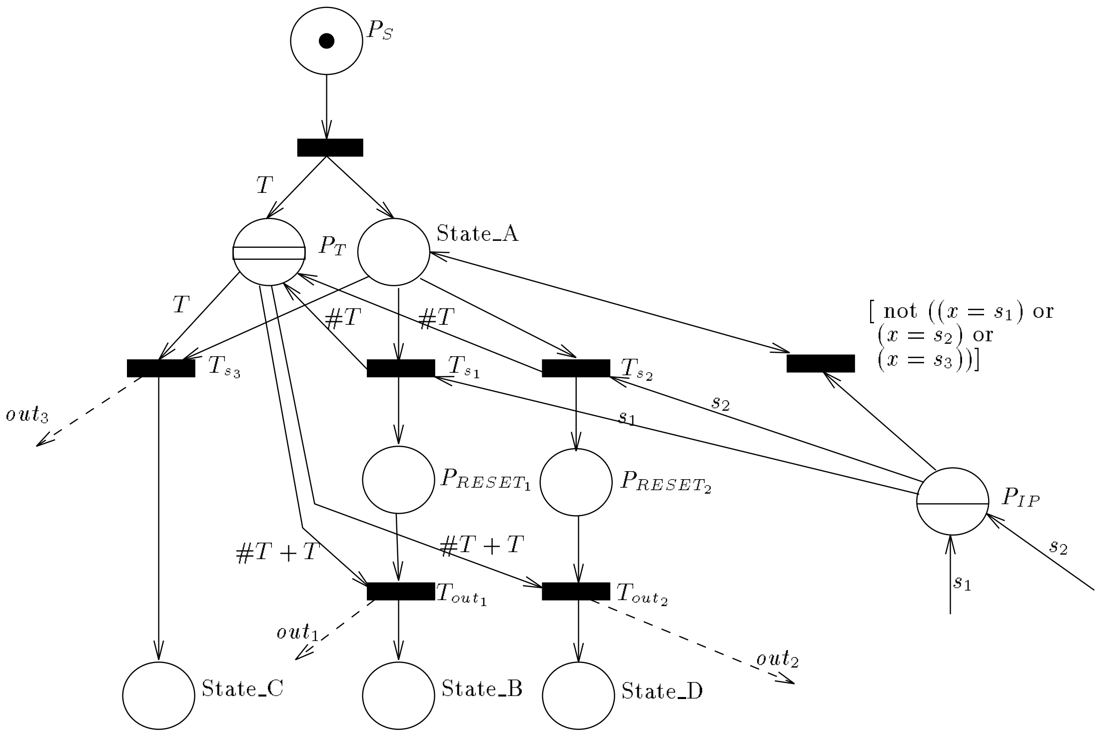

Figure 5: Equivalent representation in SDL-net using Timeout-place

In Fig. 5 the timer signal T has been handled separately and not been inserted into the input queue P_{IP} for reasons of simplicity. Otherwise further transitions have to be inserted to consume a single $\#T$ signal at the timed queued place P_T.

Note that a Timeout-place does not create or destroy tokens either, for the same reasons as those given in Sect. 4.1.

5 Analysis of SDL-nets

SDL is used to design concurrent systems and SDL-nets, in turn, are used to model SDL specifications of such systems. The main motivation for using an SDL-net, rather than analyse the SDL specification itself, is the variety of analysis algorithms which exist for SDL-nets. Such analyses are of two kinds: Checking the functional correctness of the SDL-net in the first instance and, secondly, its performance.

5.1 Correctness analysis

Correctness analysis of an SDL specification of a system corresponds to determining the functional properties in the Place-transition net determined by unfolding the SDL-net model of the system. There are two main techniques used to determine the functional correctness of a Place-transition net: *Invariant analysis* which is a structurally-based technique, and *coverability graph analysis* which explores all reachable states of the system and which suffers from the familiar state explosion problem. Functional properties of the net such as deadlock, liveness or boundedness translate directly to equivalent properties of the communication system.

Invariant analysis uses linear algebraic techniques to determine invariant assertions which are valid for all initial markings of the net. In particular, place invariants specify a weighted sum of tokens which remains constant for all possible reachable markings of the net, while transition invariants specify order-independent transition firing sequences which leave the marking of the net unchanged. In general a Petri net which is covered by place invariants is bounded, i.e., its state space is finite, and if it is also live then it is covered by transition invariants as well. The associated theory is well-known and will not be discussed here.

Coverability graph analysis, in turn, involves the enumeration of all reachable markings, followed by an analysis of the strongly connected components in the graph. If the net is bounded, the markings correspond to the states of the embedded Markov chain.

SDL-nets have been carefully defined to ensure that Save-places and Timeout-places conserve tokens. Consequently, invariant analysis can be performed for a SDL-net just as for a coloured Petri net by considering Save-places and Timeout-places as ordinary places marked with the colour set of the place, ignoring the queueing order of the signals.

For example, the mapping of SDL specifications to SDL-nets maps any SDL process to an identifiable subnet in the SDL-net. In such a subnet a single token circulates which denotes the state of the corresponding SDL-process. Consequently an expected result of the invariant analysis of the corresponding SDL-net is a place invariant with all weights equal to 1 for each such subnet.

It is furthermore very simple to test whether a system behaviour which is expected to be invariant, really is invariant. E.g., in a protocol specification a complete message transfer should end in the start state and thus should correspond to a transition invariant.

A more interesting result are the place invariants which cover Save-places since they give finite, upper limits for the number of tokens on such an Save-place. These limits translate directly to limits for the number of signals in an input buffer of an SDL process. Such places which are not covered by place invariants therefore suggest unbounded buffer occupancy. The main advantage of invariant analysis is that its complexity is independent of the size of the state space.

5.2 Performance analysis

There are two techniques for performance analysis of the communication system corresponding to the SDL-net model: Calculating the steady state probability distribution of the *embedded Markov Chain (MC)*, an exact analytical technique, and *simulation* which provides estimates by monitoring an animation of the SDL-net. In this section we focus on the Markov chain analysis of the SDL-net.

A state of the Markov chain is given by the markings of all ordinary places plus the state of Timeout-places and Save-places. A state of a Timeout-place or Save-place is given by the marking of the depository and the state of the queue. Transitions from one state to another can be caused either by firing of a transition or service completion in a Timeout-place, or Save-place. If the reachability graph of a given SDL-net is finite and contains a single final strongly connected component, the corresponding Markov chain has a unique steady state distribution. Since SDL-nets are special cases of Queueing Petri nets the generation and numerical solution of the Markov chain is known[1].

As an example of how performance measures of the original SDL specification can be obtained directly from the steady state distribution of an SDL-net, we consider the following:

Let S_i be the set of states of an SDL-process i in the SDL-specification. S_i corresponds to a set of places P_i of the SDL-net, where the tokens over P_i over all reachable markings of the SDL-net sum to 1. This can be determined by finding a place invariant as mentioned in Sect. 5.1. The probability distribution over S_i can be then obtained by summing the distribution of individual states in the set. Let f_p be a function on the reachability set RS of a given SDL-net with steady state distribution π. Let $f_p(m) = 1$ iff m is a marking such that a token is on a place $p \in P_i$ which corresponds to a state $s \in S_i$ and $f_p(m) = 0$ otherwise. Then the probability of SDL-process i to be in state s is given by

$$P_i[s] = \sum_{m \in RS} \pi_m * f_p(m)$$

Performed for all states of all SDL-processes it can be seen on which signals processes wait most for and which part of the system slows down the rest. If several places correspond to a state, function f_p is modified appropriately.

In the same manner a performance measure for the mean, $E[X]$ of the number of signals X in the input queue of SDL-process i can be defined. Let $q_i : RS \longrightarrow \mathbb{N}_0$ be the number of tokens(signals) in the queue of the Save-place of S_i, then

$$E[X] = \sum_{m \in RS} \pi_m * q_i(m)$$

Alternatively, let EN_j be the subset of RS in which a transition t_j can occur. Then, the probability $\overline{r_j}$ that t_j occurs in unit time is given by

$$\overline{r_j} = \sum_{m \in EN_j} \pi_m \left(\frac{\lambda_j}{(-q_{mm})} \right)$$

where $(-q_{mm})$ is the sum of transition rates out of marking m.

Several other measures, such as the mean number of retransmissions, is given by the ratio of probabilities for service completion on the corresponding Timeout-place and for service completion of a final acknowledgement signal on the Save-place, can be derived.

6 Example

In this section we describe how an SDL-net can be applied by typically modelling the Inres[6] protocol described briefly in the next section.

6.1 The Inres Protocol

The Inres protocol is a simple communication protocol that describes the hald-duplex communication between an *Intiator* and a *Responder*. The service is connection-orientated. During the connection setup phase, the *Initiator* attempts to establish a connection. If this takes too long, the *Initiator* reverts to the start state and may repeat the connection request, or it may disconnect.

Upon receipt of a connection request signal, the *Responder* sends a connection-indication signal to its user process. This may either send a response to the connection-indication, or it may send a disconnect-request. In the full Inres protocol, the user of the *Intiator* may now transmit data, which is acknowledged using a single bit window.

This continues until either the *Intiator* has too many timeouts and disconnects, or until the user of the *Responder* replies to a data packet with a disconnect-request.

Based on work done by[12], a user process for the *Initiator* and a user process for the *Responder* were introduced to exercise the protocol for our analysis.

The full description of the SDL-net corresponding to the Inres protocol is given in[8]. In order to speed up the computation, the authors chose to consider only the connection-setup phase for performance analysis, as it contains both a timeout and a SAVE construct, similar to that in the data-transfer phase.

6.2 Correctness Analysis

Reachability tree analysis on the SDL-net of the full Inres model revealed a deadlock in the protocol. The deadlock state in our model corresponded with the state described in [12] as being one in which deadlock occurs and was fixed using the suggestions of that author.

In order to speed up the computation without detracting from the value of the experiments for the purpose of illustrating the feasibility of the paradigm, the authors reduced the model by considering only the connection-setup phase for performance analysis. The reduced model was found to be live and bounded using reachability tree analysis.

After capacities were added to the process input queues, place invariant analysis revealed that the model was bounded. It also confirmed that each process was a state machine, in that it is in one state only at any one time.

6.3 Performance Analysis

Performance analysis of the reduced Inres protocol was done by solving the Markov chain of the corresponding SDL-net model. The model contained approximately 208000 states.

Although many experiments are possible, the intention here is to illustrate some results obtained with the use of the model which are difficult to obtain using conventional methods.

Fig. 6 illustrates performance data which would be difficult to obtain without the use of our method. It describes the relationship between the *Initiator* queue length and the channel loss rate.

In the *Initiator* process used for the analysis, `IDATreq` signals are saved in each of two system states arbitrarily called `disconnected` and `wait`. In state `connected` of the *Initiator* they are consumed explicitly, since the data-transfer phase was not modelled in its full complexity. At low channel loss the system reaches this state more frequently for a short timeout setting than for a longer one. At a channel loss rate of 10 percent, this is no longer so for a short timeout (for the same reason as that explained in connection with Fig. **??**) and a medium timeout of 25 time units is more efficient.

These examples serve to illustrate typical performance measures that one may derive from an analysis of a SDL-net model of a protocol, thereby enabling the protocol engineer to design his implementation for maximum efficiency.

7 Conclusions

The report describes an approach to the combined functional and quantitative analysis (performance evaluation) of protocols and concurrent communication systems specified in SDL with the use of a new Petri net called SDL-net which offers the following advantages:

- A close correspondence of SDL states and the markings of the SDL-net. Thus the parameters for the SDL-net can be directly extracted from the

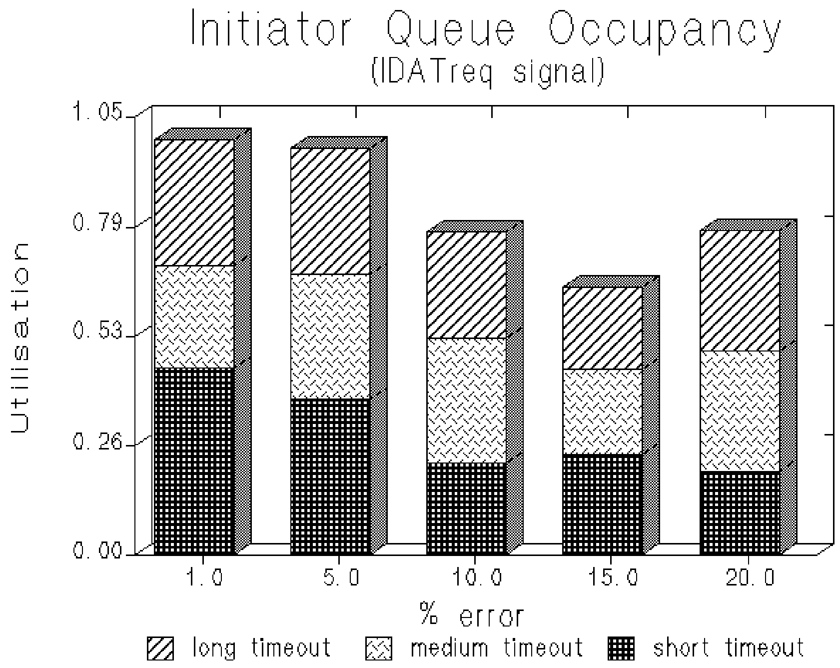

Figure 6: Channel buffer occupancy as a function of loss rate and timeout setting

parameters of the SDL model. E.g., signal delay times and timeout intervals determine the service parameters of the channel queues and Timeout-places on a one-to-one basis. Furthermore this also simplifies the interpretation of the results obtained during the analysis of the SDL-net in terms of the original SDL model.

- The structure of the SDL-net is such that it guarantees the applicability of Petri net analysis algorithms. Signals are not destroyed or created in Save-places, Timeout-places, and FIFO channel queues, so that very efficient invariant techniques can be used.

- Any SDL specification can be mechanically translated to an SDL-net, thereby allowing one to automate the performance analysis of any communication system so described.

It would seem from the experience of applying the methodology to the analysis of the Inres protocol that it provides for realistic models to assist the protocol system engineer in his task of designing efficient and correct systems.

References

[1] F. Bause. Queueing Petri nets: a formalism for the combined qualitative and quantitative analysis of systems. In *Proceedings of the 5th International Workshop on Petri Nets and Performance Models*. IEEE, October 1993.

[2] F. Bause and P. Buchholz. Protocol analysis using a timed version of SDL. In *FORTE'90, 3rd Int'l Conf. on Formal Description Techniques*, pages 239–255. North Holland Elsevier, 1990.

[3] P. Dembinski and S. Budkowski. Simulating Estelle specifications with time parameters. In H. Rudin and C. West, editors, *7th Int'l Conf. on Protocol Specification, Testing and Verification, Zurich, Switzerland*, pages 265–279. North Holland Elsevier, 1987.

[4] M. Lindqvist E. Kettunen. Towards Practicality of Predicate/Transition Petri Net Reachability Analysis of SDL. In P.A.J. Tilanus R. Saracco, editor, *SDL '87 - State of the Art and Future Trends*, 1987.

[5] J. Grabowski. Statische und dynamische Analysen für SDL-Spezifikationen auf der Basis von Petri-Netzen und Sequence-Charts. Master's thesis, University of Bern, 1990.

[6] D. Hogrefe. *Estelle, LOTOS und SDL.* Springer Berlin, 1989.

[7] G.J. Holzmann. *Design and validation of Computer Protocols.* Prentice-Hall International, 1991.

[8] H. Kabutz et al. SDL and Petri net performance analysis of communicating systems. Technical Report CS94-07-00, Department of Computer Science, University of Cape Town, 1994.

[9] P.S. Kritzinger. A performance model of the ISO communication architecture. *IEEE Transactions on Computers*, COM-34(6):554–563, 1986.

[10] P.S. Kritzinger and G. Wheeler. Semi-markovian analysis of protocol performance. In Andrè Danthine et al., editors, *13th Int'l Conf. on Protocol Specification, Testing and Verification, Liège, Belgium*, pages C3–1 – C3–14. North Holland Elsevier, 1992.

[11] U. Taubert. Entwicklung eines Analysewerkzeuges für SDL'92 Spezifikationen unter Nutzung erweiterter Petri-Netze. Master's thesis, Humboldt University of Berlin, 1993.

[12] J. Fisher U. Taubert. Vergleich zweier Methoden zur Analyse von SDL-Spezifikationen mit Hilfe von Petri-Netzen am Beispiel des Inres-Protokolls. Technical report, Fachbereich Informatik, Humboldt-Universität zu Berlin, 1993.

[13] G. von Bochmann and J. Vaucher. Adding performance aspects to specification languages. In *8th Int'l Conf. on Protocol Specification, Testing and Verification, Atlantic City, USA*, pages 19–31. North Holland Elsevier, 1988.

[14] C.H. West. Protocol validation in complex systems. In *Proc. 8th ACM Symposium on Principles of Distributed Computing, Austin, Texas*, 1989.