Qualitative and Quantitative Analysis of Timed SDL Specifications

Falko Bause, Peter Buchholz

Informatik IV
Universität Dortmund
Postfach 50 05 00
4600 Dortmund 50
Germany

Abstract. Timed SDL (TSDL) is a modified version of SDL designed for the examination of qualitative and quantitative system aspects within one model description. Because realistic communication protocols tend to be very large a program package was developed, which transforms a TSDL model into an internal representation of an equivalent Finite State Machine (FSM). This FSM representation can be efficiently analyzed by algorithms for qualitative and quantitative protocol analysis. In particular algorithms for partial state space exploration have shown to be very suitable. One of these algorithms is described in detail and is slightly improved. It is shown, that this kind of analysis leads to reasonable results, even for large models.

1 Introduction

For the description and specification of communication protocols formal description techniques (FDTs) are widely used. Several FDTs have been standardized by the ISO or CCITT (see [7, 9, 10, 11]) among them SDL, which is the FDT we will focus on in this paper. FDTs were originally developed to allow the application of (partially) automated methods for protocol analysis. Typically FDTs describe only the logical behavior, quantitative aspects related to the performance of a protocol are not considered. However, protocol analysis ideally covers both, qualitative and quantitative aspects. Qualitative analysis has to ensure the correct behavior of the protocol. Quantitative analysis provides results on the performance and indicates if the expected quality of service can be obtained.

Quantitative analysis of a formal description requires timing information. This motivates the recent extension of SDL to Timed SDL in [4]. Timing information can be included in different ways, however, the most common one is the assumption of exponentially distributed times, such that the protocol describes a Markov chain. This assumption allows to use efficient analysis techniques from the area of Markov chains for protocol analysis. Since many temporal aspects are rather constant than exponentially distributed, results of a protocol analysis based on Markov chains have to be carefully interpreted. Nevertheless, Markov chains have shown to be suitable for protocol analysis. A classical problem is consistency between the implementation of a protocol and its low level Markov model. Starting from a SDL specification the confidence in consistency increases significantly, if tools support at least a semi-automatic transformation into a low level Markov chain and executable implementation. Furthermore the importance of model based protocol analysis increases, since qualitative and quantitative aspects are analyzed with the same model.

Although the idea of integrating timing information in FDTs and analyzing the resulting low level Markov chain sounds rather simple, there are still several practical and methodological problems. The integration of timing information in FDTs has to be done carefully to be consistent with the standardization of the FDT. Software support of the whole analysis process is absolutely necessary, since this is the only way to ensure consistency in all steps and allow a protocol designer to use the techniques without being an expert in protocol analysis. This is also important, because most analysis algorithms are developed from a rather low level point of view. Furthermore a principle problem of protocol analysis is complexity. The Markov chain resulting from a realistic protocol often exceeds the limits of modern supercomputers, let alone standard workstations. Therefore an exhaustive analysis is often not possible, and one has to think about alternative solution techniques yielding not exact, but reasonably accurate results.

In this paper we present an approach for the integrated qualitative and quantitative analysis of a formally specified protocol. We introduce a timed version of SDL, an improved algorithm for protocol analysis and a tool based on these concepts. The usability of the approach is shown by means of an example.

The structure of the paper is as follows. In Sect. 2 TSDL and a tool for the analysis of TSDL models are briefly introduced. Section 3 describes an improved algorithm for protocol analysis in detail. In Sect. 4 an example protocol specification is analyzed.

2 Timed SDL (TSDL) and Tool Support

Timed SDL (TSDL) [4] enriches SDL [7] with timing aspects, which is the basis for performance evaluation. The SDL version of CCITT is a real subset of TSDL and only slight modifications are introduced, so that TSDL remains close to the standard. This simplifies the expansion of existing SDL models to their timed version in TSDL.

Quantitative aspects are introduced by extending the specification of SDL processes. An SDL process is an Extended Finite State Machine (EFSM) [5, 9], and timing aspects are added with respect to transitions. At a particular point in time an SDL process is in a certain state. A state of the entire SDL model comprises values of global variables, states of all processes, values of their local variables and messages of their input queues. For integration of quantitative information to the dynamic behavior of an SDL model, it is necessary to define the amount of time the system will remain in a particular state and/or the probabilities of transitions to other states. Adding such information to an SDL model appropriately leads to a stochastic process (SP). Typically two types of states occur in such a SP (cf. [1, 2]):

- tangible states in which the SP remains for a particular amount of time and
- vanishing states which the SP leaves immediately.

For every transition of a process either a time or a probability has to be specified. This is achieved by the following modification to SDL syntax.

```
<transition> ::= { <transition string> [<terminator statement>] }  |
                             <terminator statement>
<terminator statement> ::= [<label>] [<valuation>] <terminator> <end>
<valuation>            ::= TRATE <expression> <end> |
                           TTIME <expression> <end> |
                           TPROB <expression> <end>
```

The *valuation*-expression contains the modified part of SDL's syntax. <*expression*> is a valid SDL expression (cf. [7]), which should be of type real. *TPROB* <*expression*> is the probability for this transition. *TTIME* <*expression*> and *TRATE* <*expression*> are specifying the duration of the transition. These constructs can be used alternatively, because *TTIME a* is equivalent to *TRATE 1/a*. The following TSDL code illustrates the use of the <*valuation*>-expression.

```
STATE wait_event;
      INPUT message(frame_nr, ack, address);
            TPROB prob_of_error;
            NEXTSTATE -;
      INPUT message(frame_nr, ack, address);
            DECISION (frame_nr = frameexpected);
              (TRUE):   OUTPUT mess(frame_nr) via channel TO host;
                        TPROB 1.0 - prob_of_error;
                        NEXTSTATE prove_ack;
              (FALSE):  TPROB 1.0 - prob_of_error;
                        NEXTSTATE prove_ack;
            ENDDECISION;
      INPUT timer_signal;
            TASK nextframetosend := ackexpected;
            TPROB 1.0;
            NEXTSTATE retransmit_messages;
ENDSTATE wait_event;
```

TSDL additionally offers certain constructs (*QUEUELEN,INSTATE,INQUEUE*) for inspecting the input queue and the state of a process. This was done for two reasons:

modelling convenience SDL offers no other possibility of inspecting the messages in a queue than reading them. These constructs allow a straightforward description of collisions and improve readability ([4]).

evaluation of performance figures E.g. the throughput of a protocol can be calculated by determining the flow out of a process state, e.g. the sender's "sending state".

The above mentioned constructs can be used in expressions and are described below:

– *QUEUELEN* returns the number of elements in a process' queue.
– *INSTATE(<state>)* is true if the specified process is in state *<state>*.
– *INQUEUE(<signal name>)* is true if the queue contains the signal *<signal name>*.

Example:

```
STATE collision_on_net;
   PROVIDED (queuelen > 1);
      NEXTSTATE empty_queue;
   PROVIDED (queuelen = 1);
      NEXTSTATE read_mess;
ENDSTATE collision_on_net;

STATE empty_queue;
   PROVIDED (queuelen = 0);
     NEXTSTATE wait_event;
   INPUT *;
     NEXTSTATE -;
ENDSTATE empty_queue;
```

Validation and performance evaluation of protocols is only feasible with suitable tool support. Parallel to the development of TSDL we have also implemented a tool (cf. Fig. 1) for the analysis of TSDL models, which is based on the TSDL's corresponding FSM description. This form of description was chosen because most of the analysis algorithms are described for FSMs ([8, 15, 16]).

The most important module of the TSDL-Tool is the *TSDL-Parser and State Generator*. Given a TSDL description in textual form (like SDL/PR) and a Control File (mainly containing additional information necessary for performance evaluation), the TSDL-Parser creates an internal representation of an equivalent FSM. This module supports the analysis modules by exporting some functions especially for calculating the initial state and the successor states of a given state.

The *TSDL-Parser and State Generator* provides analysis modules with further information on the nature of a state. A state description also incorporates information whether the state is vanishing or tangible. Furthermore it gives relative probabilities, resp. times, of all transitions leaving this state. A different set of functions deals with the translation of warnings and error messages into a TSDL-like form. The TSDL-Parser encodes all generated TSDL states into an integer vector, assigns an integer number to all transitions and exports this information to the analysis modules. In case of an error or a warning the analysis modules can inform the protocol analyst in a form according to the TSDL description using these functions. Presently the TSDL-Tool offers three non-exhaustive analysis algorithms. An improved algorithm based on *Probabilistic Validation and Performance Evaluation* [8] (as described in Sect. 3), an algorithm for *Non-Exhaustive Validation* by West [16], and *Non-Exhaustive Performance Evaluation* as published by Rudin [15]. Furthermore a prototype version of a simulation module has been implemented. The integration of exhaustive analysis based on the numerical analysis of a finite Markov chain is underway. Appropriate analysis algorithms are described in [12]. Additional

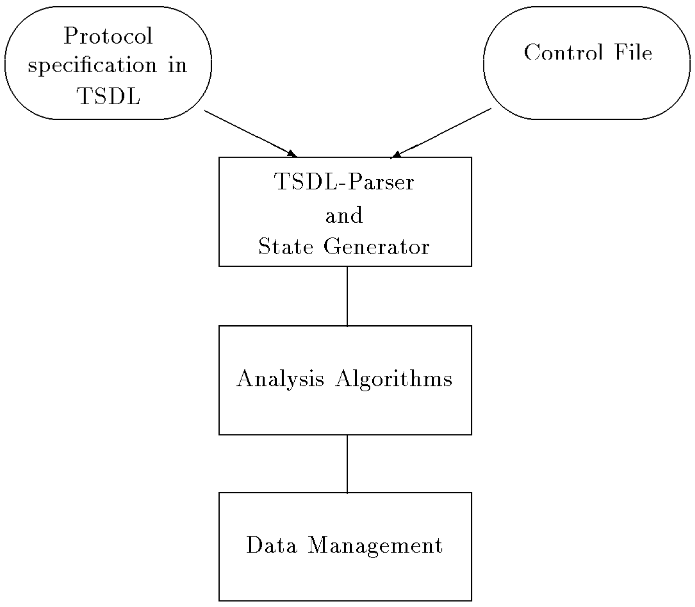
Protocol specification in TSDL
Control File
TSDL-Parser and State Generator
Analysis Algorithms
Data Management

Fig. 1. Main Modules of the TSDL-Tool

analysis algorithms can be integrated without much effort, since the interfaces of the *State Generator* and the *Data Management* are well defined and easy to handle.

The *Data Management Module* offers several operations for manipulation of the reachability tree. The efficiency of these operations have an impact on the efficiency of the overall analysis process. Thus tuning reasons justify a separate module.

To complete the description of an analysis experiment the user must also specify additional parameters, e.g. initial values for global variables and some parameters for analysis algorithms. This additional information is specified by a second input file, the *Control File*. A typical Control File looks like the following.

```
SDL-FILE      token_ring.sdl;
RESULTFILE    token_ring.result;
QUEUE-LENGTH  10;
EXPERIMENT    experiment1;          /* name of the experiment */
VARLIST;
REAL          timeout 0.125;        /* parameters of */
REAL          prob_of_error 0.0001;  /* the experiment */
MASK          block station1 / process sender
                               instate (message_transmitted);
              ...
              /* specification of analysis dependent parameters */
```

The entry *QUEUE-LENGTH* specifies an upper bound for the queue length of any TSDL-process. This restriction is essential for most TSDL models to keep state space finite, which is a precondition of exhaustive analysis algorithms. If a message is sent to a saturated queue, it is discarded, and a warning is printed.

The entry *MASK* describes the mask for specifying the class of marked states (cf. Sect. 3). The expression for describing such a mask must be an extended SDL-expression, which evaluates to a boolean value. States in which the boolean expression is *TRUE* are marked by the *State Generator*. Performance results on state level are computed with respect to marked states.

3 TSDL Model Analysis by non-exhaustive State Space Exploration

Although qualitative and quantitative analysis of Markov chains is conceptually simple, realistic protocols yield chains with an enormous number of states often far beyond the capacity of contemporary computers. Therefore an exact analysis is often hopeless, which causes a need for alternatives. One common approach is simulation, which can be used regardless of the size of the state space. However, simulation has some serious drawbacks. Results are statistical in nature and a protocol normally includes states which are reached very rarely, e.g. the loss or duplication of a message. Nevertheless, these rare events are often responsible for an abnormal behavior. Performing a straightforward simulation the number of events to be simulated including also rare events is very large. It is not transparent which part of the state space has been visited during simulation and which states have never been entered. The problem of rare events is not new and also not restricted to the area of protocols. Especially from reliability analysis several papers have been published dealing with this problem. However, all approaches use knowledge about process behavior and consider only quantitative aspects, the qualitative behavior is assumed to be known. In order to detect an error in the protocol specification by simulation, on the one hand the error state has to be entered. On the other hand it must be recognized. The latter is not trivial in general.

An alternative to simulation is numerical analysis restricted to a subset of the state space. Obviously analyzing only a subset of all states can give results only about the states in the subset. Behavior beyond this subset remains unknown. But in contrast to a simulative analysis the detection of errors is much easier. The number of states which have been considered is known and, most important, the results are more detailed. Appropriate techniques yield lower bounds for the mean time to failure (MTF) and for the time (T_0) between two visits of the initial state s_0. Such results are of practical importance, since they show that the protocol will be error-free on the average, at least for a time equal to the lower bound of MTF. If the lower bound is large enough, i.e., in the range of the mean time between other failures like hardware errors, this result is sufficiently secure. The common approach of Markov chain based analysis is to generate the state space or a part of the state space and afterwards analyze the behavior on this generated part. If only a subset of states is analyzed, the major problem of the two-step approach is to decide which states are important and have to be included in the subset and which states can be neglected. Such decisions normally need a lot of user experience and are therefore not suitable for integration in a tool for semi-automatic protocol analysis. Even more promising is the integration of state space generation and analysis in one step, which is done in [8]. A similar idea is used in [13] for the analysis of steady state availability in repairable computer system models. The former approach has been used for the analysis of TSDL models in [4]. In this section we present a modified version, which, is based on continuous time Markov chains instead of discrete time chains and is more efficient in terms of memory and time requirements, if we focus on specific results.

Although the protocol is analyzed on the level of a Markov chain, errors in the SDL specification can be detected during the analysis (e.g., if a state is generated which describes a strange situation at the SDL level). Performance results can also be directly interpreted on the SDL specification of the protocol. These are additional advantages of an approach that integrates high level specifications with efficient analysis algorithms instead of analyzing an isolated low level model.

Using TSDL for protocol specification, two different kinds of transitions exist: immediate and timed. Immediate transitions describe probabilistic choices, needing a negligible amount of time. Timed transitions need an exponentially distributed amount of time for the transition to occur. We assume that immediate transitions have priority over timed transitions and that the choice among activated transitions of the same priority level (immediate/timed) is done in a probabilistic way, i.e., each immediate transition is valued with a probability and each timed transition with a rate. The probability of firing a specific transition is given by the probability/rate of this transition, divided by the sum of probabilities/rates of all activated transitions of the same priority level. Thus the state space of the stochastic process described by the TSDL model contains to types of states: vanishing states and tangible states (cf. Sect. 2). The analysis of TSDL models is similar to the analysis of Generalized Stochastic Petri Nets [2]: during calculation of the steady state distribution of the underlying Markov chain, vanishing states are eliminated. This can cause a sometimes significant reduction in the size of the state space. In [4] an algorithm is presented to delete vanishing states during state space generation. This yields to a model with only tangible states, i.e., to each state s_i a value μ_{ij} describing the transition rate to state s_j can be directly generated. The state generator of the TSDL tool provides a function which generates all successor states s_j of a state s_i and the transition rates μ_{ij}. Vanishing states are implicitly eliminated. If the elimination of vanishing states yields an error, this indicates a so called "timeless trap", which is an error in the model specification [3]. The following informal description of this algorithm abstracts from any details of generating successor states with the help function *Succ*.

The main idea of the algorithm is to partition the set of all states S of the protocol into three subsets:

1. S_e the set of "explored states", including all states which have been completely analyzed by generating all successor states.
2. The set of unexplored states S_u, containing all states which have been detected as successors of explored states, but which have not been further analyzed.
3. $\bar{S}$ including all states, which have not even been detected. $\bar{S}$ can be finite or infinite.

During analysis, states are transferred from S_u to S_e by generating all successor states and inserting them into S_u, if necessary, and by updating the analysis on the set S_e.

If we interpret all states from $S_u \cup \bar{S}$ as an absorbing set and the set S_e as a transient set of states, then analysis of the system can be performed with techniques based on absorbing Markov chain analysis. Using this approach, we can detect deadlocks on S_e described by a closed subset of S_e with no possibility of leaving this subset. Performance results can be computed by determining the mean time the process stays in S_e after starting in s_0. If we assume the worst case scenario that each state outside is an error state, then the sojourn time in S_e is the mean time to failure, in other scenarios it is a lower bound to the mean time to failure. The time between two visits of s_0 can also be computed. This value describes the turnaround time of s_0 if we assume the best case scenario, that the process returns immediately to s_0 after leaving S_e. In all other scenarios this value is a lower bound for the turnaround time. An upper bound for the turnaround time also would be important to know, but is unfortunately infinite until $S_e = S$, since one of the non-explored states might yield a deadlock with an infinite sojourn time. More general performance results can be computed by marking states using the MASK in the Control File (e.g., all states after a successful transmission of a message can be marked). Considering the explored state space $S_e \subseteq S$, we can calculate the time the process stays in marked states and the throughput of these states. Both quantities are approximations or in some special cases bounds for the results on the complete state space.

3.1 The Analysis Algorithm

After the informal introduction of our algorithm we give a formal specification and start with the definition of some necessary notations. Let $|S|$ be the number of states in the set S. Let $\underline{M}$ be a matrix of order $|S_e| \times |S_e|$ and $\underline{W}$ a matrix of order $|S_e| \times |S_u|$. Element $M(i,j)$ later includes the mean time the process will stay in state j before leaving S_e, if the actual state is i. $W(i,k)$ is the probability that the process will enter k as the first state after leaving S_e, if the actual state is i. The values in the matrices are updated during analysis. We use two notations for states here, s_i describes the state as given from the state generator, including the decoded specification of all states of the TSDL processes. Inside the sets S_e and S_u each state gets a unique number ranging from 1 to $|S_e|$ or $|S_u|$ resp. States are numbered consecutively, i.e. a state added to a set receives the last number. If a state j is deleted from S_u, then all state numbers larger than j are decremented by 1. Nevertheless, the state description s_i has to be stored to compare newly generated states with the states from the sets S_e and S_u. We refer to a state either by its state description s_i or by the unique number i in the sets S_e or S_u depending on the context. The algorithm, as given in [4, 8] for discrete time chains, consists of the steps described below. We do not give the proofs here, since they are similar to the original work.

Algorithm 1 *Partial state space exploration*

```
I) Initialization:
    $S_u := \{s_0\}, S_e := \emptyset$.
II) Choose next state to be explored:
    If $| S_u | > 1$ then choose $s_i \in S_u$ such that $W(0,i) \geq W(0,j), \forall s_j \in S_u$.
```

```
III) State exploration for state $s_i$:
  Define $Succ(s_i)$ as the set of successor states of $s_i$. $Succ(s_i)$ and the transition
  rates $\mu_{ij}$ are generated by the State Generator. Define $\mu_i := \sum_{s_j \in Succ(s_i)} \mu_{ij}$.
  a) $S_u := S_u - \{s_i\}, S_e := S_e + \{s_i\}$.
     Add a new row and column $i$ to $\underline{M}$ with
     $M(i,i) := \frac{1}{\mu_i}, M(i,j) := 0, M(j,i) := W(j,i) * M(i,i), \forall s_j \in S_e, i \neq j$.
     If $|S_e| > 0$ then
         delete column $i$ from $\underline{W}$ and add row $i$ initialized with 0.
  b) $\forall s_j \in Succ(s_i)$ :
         if $s_j \in \bar{S}$ then
             $S_u := S_u + \{s_j\}$.
             Add a new column $j$ to matrix $\underline{W}$ by setting
                 $W(k,j) := M(k,i) * \mu_{ij}, \forall s_k \in S_e.$        (1)
         else if $s_j \in S_u$ then
                 $W(k,j) := W(k,j) + M(k,i) * \mu_{ij}, \forall s_k \in S_e.$        (2)
         else if $s_j \in S_e$ then
             if $1 - M(j,i) * \mu_{ij} \neq 0$ then
                 $M(k,l) = M(k,l) + \frac{M(k,i)\mu_{ij}}{1-M(j,i)\mu_{ij}} M(j,l),$
                 $W(k,t) = W(k,t) + \frac{M(k,i)\mu_{ij}}{1-M(j,i)\mu_{ij}} W(j,t)$        (3)
                 $\forall s_k, s_l \in S_e, s_t \in S_u.$
             else $s_j$ was the last unexplored state and the
                 whole state space has been explored (exhaustive analysis).
IV) Calculate Results
                 $MTF = \sum_{s_i \in S_e} M(0,i)$        (4)
                 $VAR(MTF) = 2.0 \sum_{s_i \in S_e} (M(0,i) \sum_{s_j \in S_e} M(i,j)) - MTF$        (5)
                 $V_i = M(0,i)\mu_i \qquad T_i = \frac{1}{M(i,i)\mu_i} * \sum_{s_j \in S_e, s_j \neq s_i} M(i,j)$        (6)
         Stop, if termination condition is satisfied, see below.
  Goto step II.
```

Matrix $\underline{W}$ is stochastic after the complete exploration of a state, which can be verified using results from the theory of absorbing Markov chains. If in (3) the denominator of the fraction equals zero, this indicates a closed set of states which can never be left. Therefore this situation occurs if j is the last unexplored state and the analysis becomes exhaustive. This case can be easily handled to get exact results by defining a vector $\underline{\pi}$ on S_e with $\pi(i) = \sum \mu_{jk}/\mu_j * M(k,i)$ for $i \neq j$ and

$\pi(j) = 1.0/\mu_j$. $\underline{\pi}$ normalized to 1.0 is the stationary vector of the Markov chain underlying the whole protocol. $\underline{\pi}$ is the basis for determining performance quantities like throughputs and sojourn time. Since the complete Markov chain is irreducible, this shows a correct protocol at the level of the state space. On the other hand, if a closed set of states is detected and the analysis has not completely explored the state space, this indicates a specification error because the system will never return to the initial state s_0. Since the state description includes the states of the TSDL processes, error states can be indicated to the user at the level of his protocol specification, rather than on the level of some Markov chain underlying the protocol.

Results of protocol analysis can be computed after each newly added explored state. MTF (eq. (4)) is a lower bound for the mean time to failure and equals the mean time before the first exit from S_e after starting in s_0. The variance of the time before the first exit from S_e ($= VAR(MTF)$) (eq. (5)) provides some additional information about the protocol behavior on the set of explored states. A large variance indicates that there are still several situations where the process leaves the set S_e quite rapidly compared with the mean value and it might be necessary to explore some more states. A small variance shows that the time of the first exit from S_e is likely to be close to the mean.

For performance analysis T_i, a lower bound for the time between two visits to a state $s_i \in S_e$, and the mean number of visits V_i in s_i between the start in s_0 and leaving S_e are also important (eq. (6)) results. Using appropriate states (often s_0 is appropriate) the duration of a message transmission or the message throughput can be bounded.

Termination of the algorithm is ensured by user-given limits to the number of explored states (rather unsatisfactory but possibly necessary due to space or time restrictions), the values of MTF, $VAR(MTF)$, V_i or T_i and any combination of these values.

3.2 Example

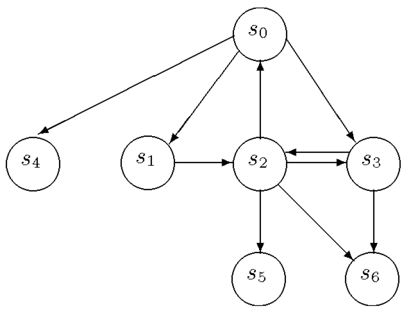

Fig. 2. State transition diagram

The following example illustrates two steps of the algorithm for a state space whose first seven states are depicted in Fig. 2.
Let μ_{ij} be the transition rate from state s_i to state s_j.

Initialization: $S_u := \{s_0\}, S_e := \emptyset$.
II) Choose s_0. It is the only possible state.
III) $Succ(s_0) = \{s_1, s_3, s_4\}, \mu_0 = \mu_{01} + \mu_{03} + \mu_{04}$
  a) $S_u = \emptyset, S_e = \{s_0\}, \underline{M} = \left(\frac{1}{\mu_0} \right)$
  b) $\forall s_j \in Succ(s_0)$:
    $s_j = s_1 \in \bar{S} : S_u = \{s_1\}, \underline{W} = \left(\frac{\mu_{01}}{\mu_0} \right)$
    $s_j = s_3 \in \bar{S} : S_u = \{s_1, s_3\}, \underline{W} = \left(\frac{\mu_{01}}{\mu_0} \; \frac{\mu_{03}}{\mu_0} \right)$
    $s_j = s_4 \in \bar{S} : S_u = \{s_1, s_3, s_4\}, \underline{W} = \left(\frac{\mu_{01}}{\mu_0} \; \frac{\mu_{03}}{\mu_0} \; \frac{\mu_{04}}{\mu_0} \right)$
IV) $MTF = \frac{1}{\mu_0}$
Next iteration:
II) Assume $\mu_{01} > \mu_{03}$ and $\mu_{01} > \mu_{04}$ thus choosing state s_1.
III) $Succ(s_1) = \{s_2\}, \mu_1 = \mu_{12}$
  a) $S_u = \{s_3, s_4\}, S_e = \{s_0, s_1\}, |S_u| = 2, |S_e| = 2,$
  $$\underline{M} = \begin{pmatrix} \frac{1}{\mu_0} & \frac{\mu_{01}}{\mu_0\mu_1} \\ 0 & \frac{1}{\mu_1} \end{pmatrix}, \underline{W} = \begin{pmatrix} \frac{\mu_{03}}{\mu_0} & \frac{\mu_{04}}{\mu_0} \\ 0 & 0 \end{pmatrix}$$
  b) $\forall s_j \in Succ(s_1)$:
  $s_j = s_2 \in \bar{S} : S_u = \{s_2, s_3, s_4\}, |S_u| = 3,$
  $$\underline{W} = \begin{pmatrix} \frac{\mu_{01}\mu_{12}}{\mu_0\mu_1} & \frac{\mu_{03}}{\mu_0} & \frac{\mu_{04}}{\mu_0} \\ \frac{\mu_{12}}{\mu_1} & 0 & 0 \end{pmatrix} = \begin{pmatrix} \frac{\mu_{01}}{\mu_0} & \frac{\mu_{03}}{\mu_0} & \frac{\mu_{04}}{\mu_0} \\ 1 & 0 & 0 \end{pmatrix}$$
IV) $MTF = M(0,0) + M(0,1) = \frac{1}{\mu_0} + \frac{\mu_{01}}{\mu_0\mu_1}$.
  Note that only the entries $M(0,i)$ for newly explored states have to be added to the MTF-value of the previous iteration.

If s_2 is the next state to be explored, states s_5 and s_6 will be detected yielding the state transition diagram depicted in Fig. 2. Notice that transitions originated in non-explored states are not detected yet.

3.3 Time and Space Requirements

Obviously it is necessary to store two matrices of order $|S_e| \times |S_e|$ and $|S_e| \times |S_u|$, a data structure including the state descriptions s_i and for each state of $S_e \cup S_u$ the information in which set it is located. If we assume $|S_e|$ to be not much smaller than $|S_u|$, which is a realistic assumption, then space requirements are in $O(|S_e|^2)$.

Time requirements for the introduction of a new transition depend on the location of the destination state (i.e., which of the formulas (1) - (3) are used). (1) and (2) require $|S_e|$ operations, whereas (3) needs $|S_e|^2 + |S_e||S_u|$ operations. The whole number of operations can be approximated by $\sum_{n=1}^{|S_e|} O(n^2) = O(|S_e|^3)$. Since the number of transitions originated in a fixed state depends on the number of TSDL processes, rather than on the number of states, the analysis effort lies in $O(|S_e|^3)$.

3.4 Modifications and Improvements

Normally it is not necessary to compute all values V_i and T_i. The computation of MTF, V_0 and T_0 is often sufficient for protocol analysis. In this case the time and space complexity of the algorithm can be decreased significantly. We define two vectors $\underline{m_\Sigma}$ and $\underline{m_0}$ each of order $|S_e|$ by

$m_\Sigma(i) := \sum_{s_k \in S_e} M(i,k)$ and $m_0(i) := M(i,0)$. The required results can be expressed using these vectors.

$$MTF = m_\Sigma(0) \qquad V_0 = m_0(0)\mu_0 \qquad T_0 = (m_\Sigma(0) - m_0(0))/V_0 \qquad (7)$$

For the exploration of a new state s_i and the update of the vectors and matrix $\underline{W}$ we temporarily need a third vector $\underline{m_i}$ including the i-th column of $\underline{M}$. Step II) of the algorithm proceeds as previously, only the matrix elements $M(k,i)$ are substituted by $m_i(k)$. The same is done with equations (1) and (2). In equation (3) the different vectors and matrix $\underline{W}$ are used. The new equation is given below, the situation is the same as in (3), s_i is the state to be explored and a transition to $s_j \in S_e$ is newly introduced.

$$m_\Sigma(k) = m_\Sigma(k) + \tfrac{m_i(k)\mu_{ij}}{1-m_i(j)\mu_{ij}} m_\Sigma(j),\; m_0(k) = m_0(k) + \tfrac{m_i(k)\mu_{ij}}{1-m_i(j)\mu_{ij}} m_0(j)$$

$$m_i(k) = m_i(k) + \tfrac{m_i(k)\mu_{ij}}{1-m_i(j)\mu_{ij}} m_i(j), \qquad W(k,t) = W(k,t) + \tfrac{m_i(k)\mu_{ij}}{1-m_i(j)\mu_{ij}} W(j,t), \qquad (8)$$

$$\forall s_k \in S_e, s_t \in S_u$$

The proofs for the formula above are based on a straightforward summation in (3). With the above transformation the storage requirement for matrix $\underline{M}$ is reduced from $|S_e|^2$ to $3|S_e|$. The same holds for the number of operations. If results V_i and T_i have to be computed for other states than s_0 we can proceed in exactly the same way. For each state a new vector $\underline{m_i}$ is needed.

The algorithm can be further improved, if we can assure that for a state $s_i \in S_e$ ($s_i \neq s_0$) no state $s_j \in S_u \cup \bar{S}$ exists with a transition from s_j to s_i. In this case s_i can be simply eliminated. However, to verify this condition, state generation has to be run in reverse order.

4 Analysis of an Example Protocol

For illustrating the suitability of the algorithm presented in Sect. 3, we choose a token ring protocol described in [6, 14].

In our model the token ring consists of three stations communicating via the ring and a monitor station for error detection. Typical errors are loss of a token, multiple circulation of a busy token and duplication of a free token. For our example we only consider loss of tokens. The monitor can detect a lost-token error, if it has not seen a free or busy token for a certain period of time. It will then transmit a special shut-off signal for cleaning the ring. When this special signal returns to the monitor, which implies that all stations have received it, the monitor generates a new free token.

Messages to transmit arrive at each station according to a Poisson distribution specified by parameter λ. The length of each message is described by an exponential distribution determining its transmission time. For simplicity we consider only one class (type) of messages. Transmission of a message is completed in 43 time units in the average, where a time unit is the signal delay between two neighboring stations. Each station is assumed to have an unlimited buffer capacity for arriving messages. Thus the state space of the model is not limited and an exhaustive state space analysis is not feasible in principle.

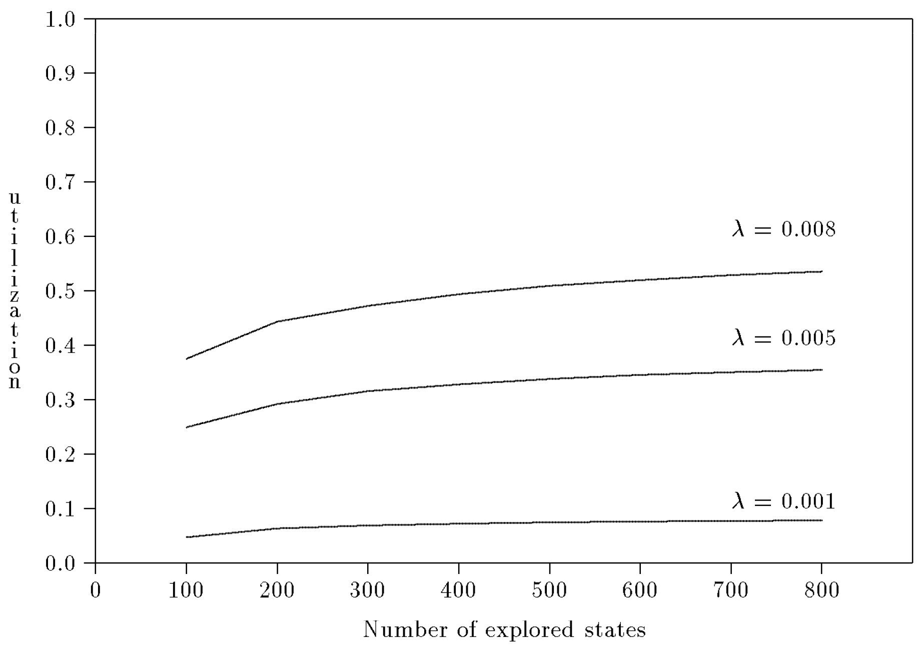

Fig. 3. Utilization of one station of the token ring

	Simulation			Numerical analysis		
Parameters	CPU time	simulated model time	#visited states	CPU time	MTF	#explored states
$\lambda = 0.001$	519.8	1.34631×10^5	86	400.2	3.30623×10^5	800
$\lambda = 0.005$	526.1	1.59401×10^5	190	546.7	1.09079×10^5	800
$\lambda = 0.008$	519.6	1.98084×10^5	248	698.5	0.67446×10^5	800

Table 1. Comparison of both analysis methods

Figure 3 shows the utilization at a certain station of the token ring for different arrival rates of messages at this station and for different number of explored states. The arrival rate of the two other stations is kept fixed to 0.005. Utilization only

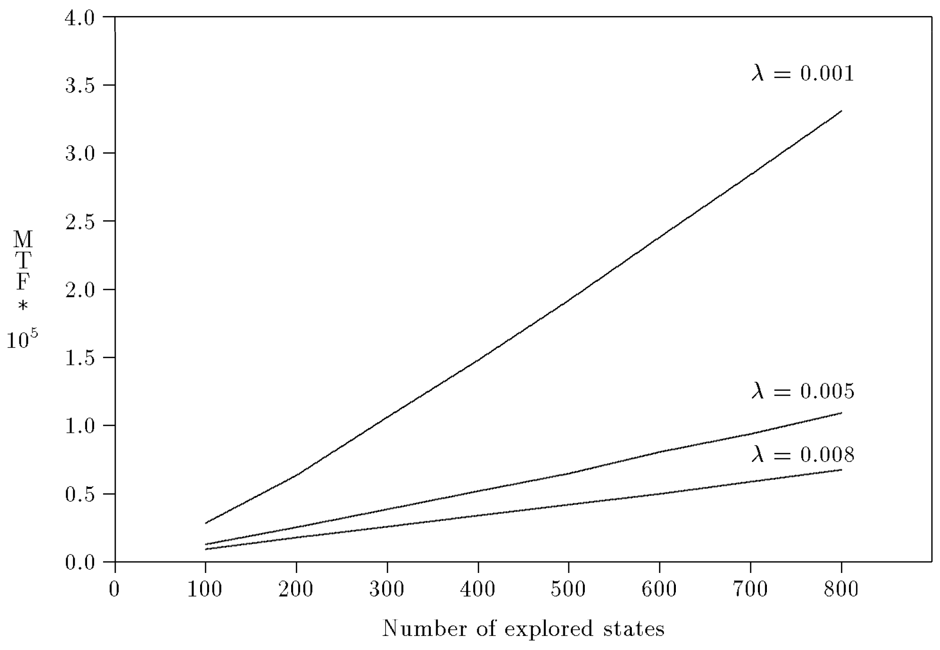

Fig. 4. Lower bound for Mean Time to Failure

slightly changes after exploration of 600 states, so a further increase of the number of explored states will not lead to significant different results.

Figure 4 shows the lower bound of the Mean Time to Failure (MTF) for the experimental series shown in Fig. 3. For a low utilization of the chosen station, MTF increases significantly with the number of explored states, because error states will be explored more frequently. Increasing the arrival rate of one station implies that the corresponding events are more probable, thus decreasing the slope of MTF.

Table 1 illustrates the advantages of the non-exhaustive state space exploration algorithm compared with a simulation of the Markov chain. Although CPU time is approximately the same for both analysis techniques, there is a significant difference in the number of explored and visited states resp. Furthermore the reader should note, that the simulated model time cannot be directly compared with the MTF, because the simulation might be in an error state not yet detected, whereas in the algorithm of Sect. 3 an error, e.g. a livelock, has been surely detected.

5 Conclusions

We have presented an integrated approach for protocol specification, qualitative and quantitative analysis based on an extended version of SDL (TSDL). The approach has motivated the implementation of a prototype version of a tool which allows the formal specification of communication protocols and their subsequent analysis using techniques which originally have been described on a rather low level. One of these techniques, the *Probabilistic Validation and Performance Evaluation* has been improved in terms of space and time requirements and is presented in some more detail.

The environment introduced so far, is extendable towards other analysis algorithms and allows the use of all these algorithms from a common interface even for an inexperienced user. In particular non-exhaustive analysis techniques seem to be a suitable approach for protocol analysis, since they can be used for models with a very large or infinite state space, which cannot be analyzed by exhaustive techniques. Furthermore they provide more accurate results than simulation.

References

1. M. Ajmone-Marsan. *Stochastic Petri Nets: an Elementary Introduction.* Advances in Petri Nets. G. Rozenberg (ed.), Lecture Notes in Computer Science 424, 1989, pp. 1-29.
2. M. Ajmone-Marsan, G. Balbo, G. Conte. *A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems.* ACM Transactions on Computer Systems, Vol. 2, No. 2, Mai 1984, pp. 93-122.
3. F. Bause: *No Way Out ∞ The Timeless Trap.* Petri Nets Newsletters 37, December 1990, pp. 4-8.
4. F. Bause, P. Buchholz: *Protocol Analysis Using a Timed Version of SDL*, Proceedings of the IFIP TC/WG 6.1 Third International Conference on Formal Description Techniques for Distributed Systems and Communications Protocols, FORTE'90, November 1990, pp. 239-254.
5. F. Belina, D. Hogrefe: *The CCITT-Specification and Description Language SDL*, Computer Networks and ISDN Systems, September 1989.
6. W. Bux et al: *A Local-Area Network Based on a Reliable Token-Ring System*, Proc. Int. Symposium on Local Computer Networks, (ed.) P. Ravasio, G. Hopkins, N. Naffah, North Holland 1982.
7. CCITT Z.100: *CCITT SPECIFICATION AND DESCRIPTION LANGUAGE SDL*, COM X - R 15 - E, Genf, March 1988.
8. D. D. Dimitrijevic, M.-S. Chen: *An Integrated Algorithm for Probabilistic Protocol Verification and Evaluation*, IBM Research Report RC 13901, August 1988.
9. D. Hogrefe: *ESTELLE, LOTOS and SDL*, Springer-Verlag, 1989.
10. ISO DIS9074 *Estelle: A Formal Description Technique based on an Extended State Transition Model*, 1987.
11. ISO DIS8807: *Lotos: A Formal Description Technique*, 1987.
12. U. Krieger, M. Sczittnick, B. Müller-Clostermann: *Modeling and Analysis of Modern Telecommunication Networks by Markovian Techniques: Foundations, Algorithms and an Example*, IEEE Trans. on Comm., special issue on Computer-aided Modeling, 1990.
13. J. C. S. Lui, R. R. Muntz: *Evaluating Bounds on Steady State Availability of Repairable Systems from Markov Models*, Proc. of the First Int. Workshop on the Numerical Solution of Markov Chains, Raleigh, 1990, pp. 469-489.
14. H. Rudin: *Validation of a Token-Ring Protocol*, Proc. Int. Symposium on Local Computer Networks, (ed.) P. Ravasio, G. Hopkins, N. Naffah, North Holland 1982.
15. H. Rudin: *An Improved Algorithm for Estimating Protocol Performance*, IBM Research Report, RZ 1314, April 1984.
16. C. H. West: *Protocol Validation by Random State Exploration*, IBM Research Reports, RZ 1482, Zürich, 1986 (also in: Proceedings of the 6th Int. Workshop on Protocol Specification, Testing and Verification, (ed.) B. Sarikaya, G. V. Bochmann).

This article was processed using the LaTeX macro package with LLNCS style