13

Performance Modelling with the Formal Specification Language SDL

M. Bütow, M. Mestern, C. Schapiro and P.S. Kritzinger
Data Network Architectures Laboratory
Computer Science Department
University of Cape Town, Private Bag
Rondebosch, 7700 South Africa.
+27-021-650-2664
email: {mmestern,psk}@cs.uct.ac.za

Abstract

Predicting the performance of a communication protocol from a formal description was first proposed about a decade ago. Such performance prediction involves two issues: the semantics to give time and the analysis method to solve the resultant model. In this paper we address the first issue: The semantics to give time and how to introduce it into the model. The approach we propose does not affect the syntax of the formal description technique and does not depend on the FDT used, but we illustrate our ideas using SDL and derive performance measures from a simulation or execution of the specification. In many cases we could equally well use Markov or queueing theory or other proposals in the literature. The problem of determining the values of the appropriate parameters in those cases remain however. We also describe a software tool which implements our proposals and an example to illustrate their application in practice.

Keywords: FDT-based system and protocol engineering, Extensions of FDTs, SDL, Automata and languages, Performance modeling and analysis.

1 INTRODUCTION

The excitement about B-ISDN and ATM, voice networks and frame-relay as an alternative to packet switching networks all point to a growing application of communicating systems and an associated increase in the complexity of the software to make these systems communicate.

That the software for such systems must be correct is understood even though no known method guarantees correctness. System developers may, however, also require quantitative measures such as throughput and response time to choose between alternative system designs. Ideally the correctness and performance analysis should be of the implementation in its target physical environment, but this may imply costly redesigns and delays. The next best option is to generate the implementation code as well as do the correctness and performance analysis using the same specification as input and to take the best possible guess at the parameter values of the target environment.

R. Gotzhein et al. (eds.), *Formal Description Techniques IX*

Standard Formal Description Techniques (FDTs) such as Estelle, SDL and LOTOS were not initially developed with these automated code generation and analyses specifically in mind, although CASE tools such as Geode (Verilog, 1995)[SA95] and SDT (Telelogic, 1995)[AB95] now use FDTs for these purposes. In this paper we assume that the system concerned has been shown to be correct with a certain degree of confidence; we address only the process of performance analysis.

Predicting the performance of a discrete dynamic system such as a communication protocol from its formal description was first proposed about a decade ago (Kritzinger, 1986)[Kri86]. Such performance predictions involve two issues: The semantics to give time in the formal description technique and second, the analysis method used to solve the resultant model.

The latter issue is a topic of this paper only to the extent that one needs to be aware what implication it has for the accuracy of the proposed analysis. The most obvious way is to obtain performance statistics from a simulation or execution of a model constructed from the specification and which reflects the target execution environment information. This is the approach we shall present.

In the literature there are various other proposals to apply analytical methods which are in general more time efficient than simulation. These proposals fall into one of two categories: The first category suggests the use of Markov chain analysis (Kritzinger, 1986)(Kritzinger Wheeler, 1993)[Kri86, KW93] or non-exhaustive and probabilistic analytical methods (Bause, Buchholz, 1990)[BB90](Rudin 1984)[Rud84] derived directly from the specification. The second category of proposals suggest that the specification should first be translated to a model such as a Petri net (Bause et al, 1995)(Kremer, 1994)[B+95, Kre94] or, most recently, that the SDL specification should be adjuncted with queueing models (Diefenbruch, 1994)[D+95]. The introduction of another abstraction detracts from the ideal of having the analyses and implementation coincide although it should be mentioned that the translation to a Petri net has the correctness *proof* (as opposed to partial verification) through invariant analysis of the system as its main goal.

Over the years the literature has suggested various ways of specifying the quantitative properties of the environment targeted for the implementation of the system described by a formal description technique. The proposals range from adding performance parameters in comments (static annotations) (Kritzinger, Wheeler, 1989)(Budkowski, 1992)[KW89, Bud92] to extending the syntax of the language (Bause, Buchholz, 1990)(Diefenbruch, 1994)[BB90, D+95]. In particular, the semantics of Estelle[ISO89] and SDL(ISO, 1989)[ITU94] do not allow the qualitative specification of all the properties necessary to construct an executable model. They provide for the developer to specify timeout values and they acknowledge that delays may occur on channels; however there is no mechanism for the developer to specify the delay period. Processes in these FDTs also have no limit on their processing resources as they are assumed to execute fast enough to handle the traffic load.

In the next section we describe a way of mapping a system specification to a target environment without changing the syntax of the specification, thus confirming that there is no need to proscribe the semantics of time in an FDT. Our proposal is similar to the annotated approach developed independently by Budkowski (Budkowski, Hendaz, 1995)[BH95], but frees the developer from the need to specify unnecessarily detailed information.

We have chosen SDL for our discussion, but what we propose applies equally well to

any other standard FDT. In Sec. 4 we focus our discussion on the performance analysis aspects of the SDL Performance Evaluation of Concurrent Systems (or SPECS) software tool and report on an example in Sec. 5 to illustrate the concepts we propose.

2 TIME IN SDL

The definition of time in the dynamic semantics of SDL is loose in the sense that it acknowledges that the system will execute in real time with delays on channels, but does not specify how the system execution is affected by this constraint. In particular, SDL assumes (ITU, 1994)[ITU94]:

- time is incremented by a clock outside the system.
- no units of time are predefined (time may be continuous).
- signal transfers over channels can take time.
- an SDL system is not limited by processing resources. This implies that processes may perform SDL actions in zero time (or negligable time compared to the duration of a signal transfer).

This interpretation of time is unsatisfactory in the following ways:

One should distinguish between signal transfers over signalroutes and signal transfers over channels. In SDL, signalroutes do not introduce delays (Belina etal, 1991)[BHS91]. Channels may introduce delays; however the ITU recommendations do not specify how the duration of this delay is determined. Moreover, channels are assumed to be perfect and to never lose signals.

In physical systems all processing takes time, therefore it is necessary to drop the assumption in SDL that individual actions can happen instantaneously. It also means that the total time taken by actions may be significant compared to signal transfer delays.

Once one accepts that executing an implementation takes time it follows that one has to account for the possibility that implementations running on different processors may execute at different speeds; something which, in combination with the channel delays, may have a profound effect on, for example, timeout settings.

We attached semantics of time to SDL to take account of all these aspects. These semantics are described in the following section.

3 ATTACHING SEMANTICS OF TIME TO SDL

In the following discussion we need the notion of an atomic set of instructions or an *action* in an FDT. In SDL we call a single statement that appears in each SDL graphical symbol in a transition an action. As an example each of the following is considered a single action in SDL: *input, output, task, timer set and reset, join, decision evaluation, nextstate.*

In our approach to the semantics of time we assume the following:

- signal transfers over channels take a user-specified time, but transfers over signalroutes are not delayed.

- processes are only allowed to execute a finite number of actions in a time unit (actions take time).
- different SDL processes may execute a different number of actions per time unit (processes can have different execution speeds).

These user-specified parameters determine how much time actions take in comparison to signal delays, and thus enable the user to simulate and analyse systems with a wide range of temporal characteristics.

3.1 Process Execution

To model the different speeds and scheduling strategies of the processes we have introduced a system of *action quotas, weights* and *processor speeds* to guide the execution of processes during the simulation.

Each processor $k, k \geq 1$ on which the final implementation will be executed is assigned a speed μ_k in units of number of actions that can be executed per time unit. This corresponds to the instructions that a real machine is able to execute per time unit.

Each block j in the specification is designated to run on a specific processor k. This may be done at the time the specification is offered for simulation and may change from one run to the next. We designate the set of blocks running on a specific processor k by B_k.

Finally, each process i belonging to a block j is assigned a weighting ω_{ij} chosen by the developer. One would expect his choice to reflect the number of instructions the process contains. Designate the set of processes in block j by P_j.

When a process i in block j is executed on processor k for the simulation, it receives its share of the processor or *action quota* q_{ij}. Each action costs one unit of action quota. Processes accumulate their action quotas until they have enough time to execute one or more actions. Once the action quota of a process is less than one, the fraction is carried over to the next time unit. This allows the developer to model processes with arbitrary speeds.

The quantity q_{ij} is computed as follows:

$$q_{ij} = \frac{\omega_{ij}}{\sum_{j \in B_k} \sum_{i \in P_j} \omega_{ij}} \mu_k.$$

This method confirms the intuitive practice of modelling machines by blocks and the tasks executed by each machine as processes. Processes in different blocks execute concurrently, while processes in the same block execute in a multitasked way. In summary:

The developer must decide which processes may execute truly concurrently by placing them in different blocks.
Each block is designated to run on a particular processor.
By changing the processor speed the developer can model fast and slow processors.
By varying the process weights and the assignment of blocks to processors, the developer can adjust relative process speeds within a block.

Note that these parameters are decided by the developer before the beginning of a

simulation, while the *effective* execution speed of a process is further determined by the number of processes currently instantiated in a block which will change as the process are dynamically created and destroyed.

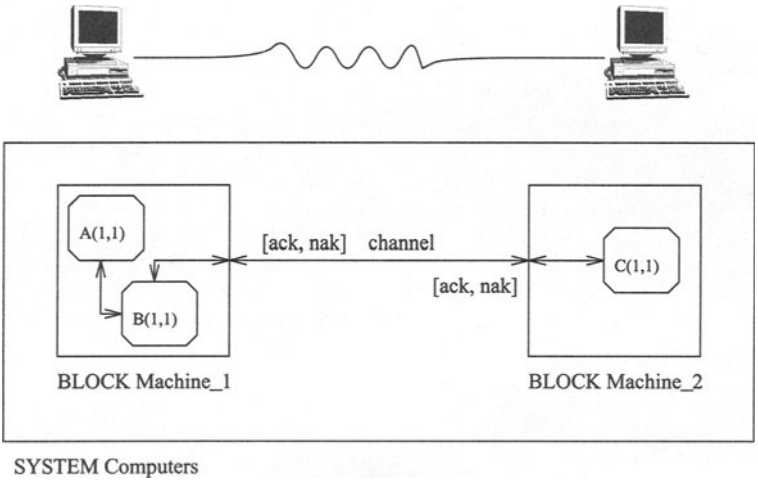

Figure 1 System used in the example.

Consider the system illustrated in Figure 1 as an example. Block `Machine_1` is assigned to a processor with speed of $\mu_1 = 12$ actions per time unit, while block `Machine_2` is executed on a machine which can only perform $\mu_2 = 6$. If process A and B are weighted equally, that is $\omega_{11} = \omega_{21} = 1$, they will each receive $q_{11} = q_{21} = 6$ actions per time unit.

Process C will get the full 6 actions per time unit of the processor on which the block `Machine_2` is executed. Alternatively, if $\omega_{11} = 1$ and $\omega_{21} = 3$, then B will execute 3 times as fast as A. Process A will receive $q_{11} = 3$ time units, while B will receive $q_{21} = 9$. If there were 10 instances of process C running at once, each one would be able to perform $q_{i2} = 0.6$, actions per time unit or each instance would execute 6 actions during a time span of 10 units.

3.2 Signal transfer

In SDL, signal transfers across signalroutes are instantaneous. A signal has to travel across a channel connecting two blocks to experience a delay. The delay may be of any particular distribution, however in SPECS we have implemented delays as deterministic or distributed according to a uniform or negative exponential distribution with a mean delay specified by the developer. Signals may also be lost with a given probability on a channel - we chose to make this extension to the semantics of SDL because it is a useful in protocol specifications.

3.3 Advancing the simulation clock

There are two conditions under which time, as kept by the global simulation clock, is advanced:

- process instances have exhausted their action quotas.
- process instances are all waiting for input.

If all the process instances have exhausted their action quotas then the clock is advanced by one unit and all the process instances receive their action quotas. If all processes are waiting for input, the clock is advanced to the time of the next *event*; a timer expiry or signal arrival is regarded as an *event*. Continuous time is approximated by rounding all *events* to the next integer time unit. Thus, each *event* is only detected and processed at the beginning of a time unit.

A system is *deadlocked* when all timers are inactive and no signal arrivals are pending.

When the simulation clock is advanced, all timers are checked for expiration and the signals that have arrived during the period of the clock advance are placed on the input queues of their destination processes.

This concludes our discussion of mapping an SDL system specification onto its target execution environment. In the next section we briefly discuss a software tool which was developed to test these notions.

4 THE SPECS TOOL

In order to test the ideas proposed in the previous section we developed a software tool which we call the SDL Performance Evaluation of Concurrent Systems (or SPECS) tool.

SPECS is a CASE tool for viewing, compiling and simulating the execution of a concurrent system specified in SDL. (Bütow etal, 1995)[BMS95] SPECS has a graphical user interface which allows an SDL/PR description of a system to be loaded and Virtual Machine (VM) code to be generated. SPECS further lets one specify the quantitative properties of the target environment and the resultant model to be executed or simulated on a Virtual Machine, either scheduling the process execution interactively if the user wishes or randomly. During the execution, traces are produced which are subsequently analysed for various performance measures. A functional overview of the tool is shown in Figure 2.

SPECS consists of five main components: a graphical user interface, a parser, a translator, a virtual machine and an analyser.

4.1 Components of SPECS

Compiler

SPECS contains a compiler which generates the code for the virtual machine. The compiler accepts a subset of SDL/PR specification. It supports the basic SDL structures such as block, processes, channels, signals, variables and macros. Inside SDL processes we support the states, inputs, saves, nextstates, timers, tasks, output, decision, and process creation.

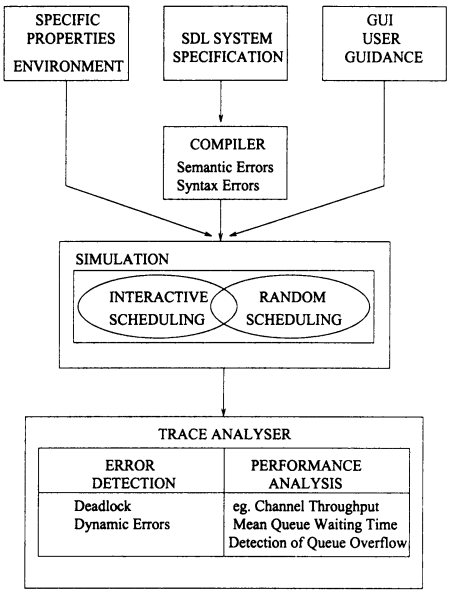

Figure 2 Functional overview of SPECS.

The excluded language structures are substructures, priority and continuous signals, packages, procedures, services, and scope unit typing mechanisms. We support the majority of the predefined data types, but do not allow the user to create new types.

The Graphical User Interface (GUI)

The GUI allows the SPECS user to interactively guide and view the simulation of an SDL specification. The GUI displays information about processes as they execute. One of the windows contains a list of the live processes in the system. By selecting the name of a process the GUI highlights the next SDL action the process will execute. The process selection window also displays the number of items in each of the process queues. A typical screen view during the specification of the environment is shown in Figure 3.

Simulation Environment

The GUI provides the SPECS user with the following features to specify and construct the environment:

- Selecting processes and blocks to be traced during the simulation and for which analysis results must be generated.
- Setting the channel attributes, i.e., the reliability and delay time distribution (deterministic, exponential or uniform distribution) as illustrated in Figure 3.
- Selecting the capacity of the input queue of a process.
- Setting the process weights and block (processor) execution speeds.

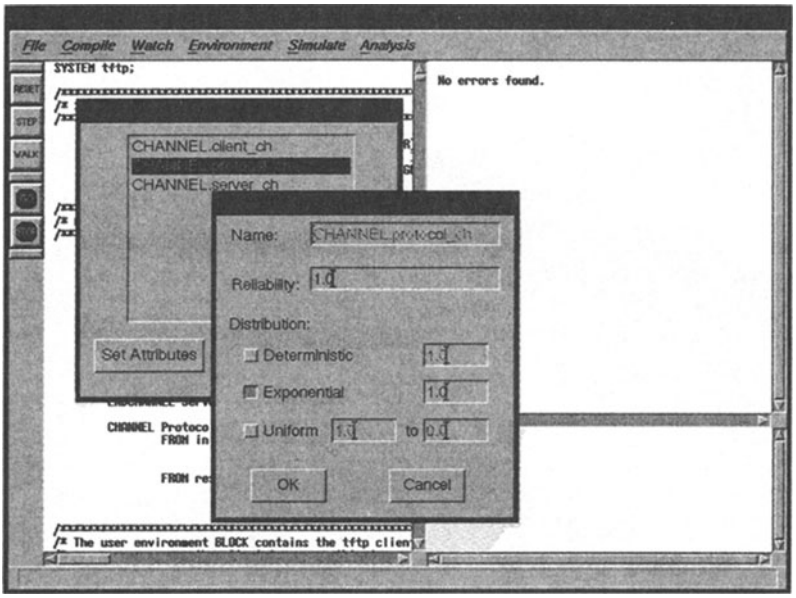

Figure 3 GUI interface during environment definition.

The execution of the simulation is controlled through the use of the following three buttons on the GUI (see Figure 3):

STEP. The selected process executes a single SDL action.
WALK. The selected process executes an entire transition. Stepping and Walking mode also allow the user to assign extra execution time to a process if the user so wishes.
RUN. The simulator runs automatically without any user intervention. The order of the process execution is chosen at random, but each process receives only the execution time allowed by the block speed and process weight. These parameters were described earlier in Sec. 3.1.

Trace analysis

After a simulation run is complete, the user is able to analyse the resulting execution trace by calling the Trace Analyser from the GUI. Trace analysis provides the following performance measures:

Mean queue waiting time for a process. The average length of time that a signal arriving at process must wait before being read. This information is useful for modelling the response time of a process.

Throughput of a channel. The rate at which signals are transferred on the channel. This is useful for determining the channel capacity needed by the system being specified.

Mean and maximum process queue length. The input buffers of a machine are often modelled by the queues of SDL processes. This result is a guide to the size requirements of these buffers.

Detection of queue overflow. If a process has bounded queues then the number of overflows is reported.

Throughput of a state. This is how often a process enters a particular state. This is used to determine the frequency of certain events. As an example, a communications protocol could have a state which resends a lost message. The throughput of this state shows the frequency of message loss.

Discarded signals. Signals may be discarded if there is no input or save specified for the signal, or the process to which the signal was sent no longer exists. These discards are likely to indicate an incorrect or incomplete specification.

Unreached states. These are parts of a process which are "dead code". Unreached states are likely to indicate logic errors in the specification. However, because the simulation may not explore all possible states of the system there is no guarantee that the states are completely unreachable.

The lifetime of a process. In SDL processes may be dynamically created and killed. This statistic measures the mean lifetime of a particular process type.

The average time spent blocked in a state waiting for a signal. This result is generated for each state in the process and indicates on average how long the process is idle in that state.

Timer information, such as expired timeouts. For each timer in the specification, SPECS will report the percentage of timer SETs that resulted in the expiry of the timer. This information is used to tune the timeout values in communication protocols.

SPECS will also detect deadlocks and dynamic errors if they are encountered during a simulation.

5 EXAMPLE

In this section we describe a set of experiments to illustrate how our proposal would be applied in practice. For our illustration we chose the Trivial File Transfer Protocol (TFTP) which is a simple protocol for the transfer of files. Although SDL provides for an environment external to the system, SPECS requires that the systems be closed. We have made the blocks Client and Server to provide the signals to the system that would normally originate in the environment. The protocol may be used by an initiator or client to read or write files from or to a remote responder or server. It is a stop-and-wait protocol and each data packet is acknowledged separately. It was designed to use the Internet User Datagram protocol (UDP or Datagram)(Postel, 1980)[Pos80] and was chosen for our example because it is typical of window flow control protocols.

The structure of the SDL specification of a communication system using the TFTP protocol is shown in Figure 4. It consists of four blocks: the `client`, `server`, `initiator` and `responder` blocks. A channel connects the `initiator` and `responder` blocks and it is on this channel that the delay is varied to observe the effect on the efficiency of the protocol. In the specification the `client` passes a write request to the `transmit` process in the `initiator` block. The `initiator` block passes the request to the `responder` block

which conveys it to the `server`. If the `server` grants the request, the response is conveyed to the `initiator` and the data transfer phase begins.

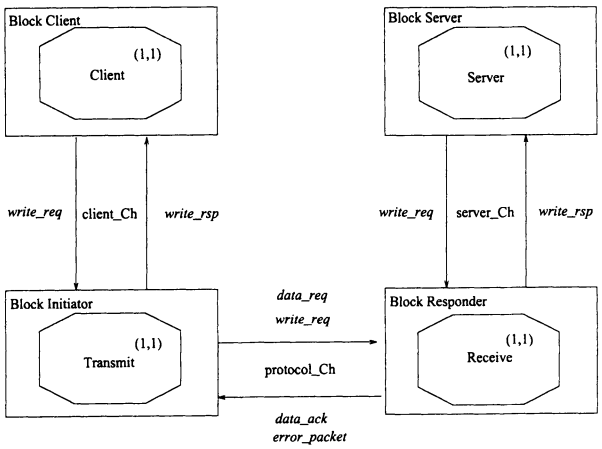

Figure 4 SDL model of a communication system using the TFTP protocol.

Data transfer takes place packet-by-packet. The initiator sends a packet, then waits for an acknowledgement from the responder. If the responder does not receive a data packet within a fixed timeout period a negative acknowledgement is sent to the initiator to cause it to resend the lost packet. All data packets and acknowledgements carry sequence numbers to prevent a packet being misinterpreted as being the previous packet or acknowledgement.

The corresponding read cycle of the client has been omitted from the specification for reasons of simplicity, but it is symmetrical to the write cycle.

5.1 Experimental results

Using the toolset SPECS, it is possible to vary parameter values of the system such as timeouts, channel delays, channel reliability, and the processor action quotas. We chose to determine the average time taken by the client to transfer a file from the server under various channel delays and processor speeds and with different timeout values. In all experiments we used a perfect channel so that no packets were lost.

We sent a file of 5 packets between the client and server. This was repeated 100 times to obtain the reported 95% confidence intervals. The total CPU time to perform the required simulations on a SUNSparc 10 workstation was 53 minutes and 22 seconds.

- *Processor speeds.* Each block in the specification was assigned its own processor in our experiment. The `client` and `server` blocks were each given a processor of speed 10 (actions per time unit) while the `initiator` block was given an a processor of speed 5. The speed of processor of the `responder` was changed from 1 to 2.
- *Channel delay.* The delay time on the channel had a negative exponential distribution with the mean varying from 2 to 8 in steps of 2 time units.

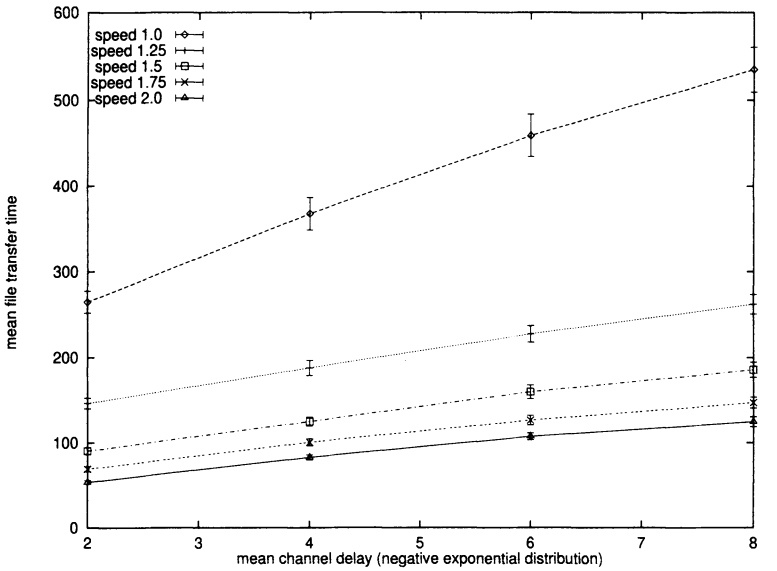

Figure 5 The effect of processor speed (of the `responder` block) and mean channel delay. 95 % confidence intervals shown by vertical bars.

- *Timeout values.* The value of the timeouts for both the `initiator` and the `responder` were set at 4 and 16 time units.

The results of the experiments illustrated in Figure 5 are of the mean file transfer time as a function of the mean channel delay for the `responder` processor speed ranging from 1 to 2. Vertical bars show the 95% confidence intervals. The effect of increased channel delays on the mean file transfer time ranging from 264.8 to 534.8 time units is clearly more significant for a slow processor while at higher speeds the channel delay has a less noticeable effect on the mean file transfer time ranging from 53.8 to 124.6 time units in that case. In all cases the value was 4 for all timeouts.

For a different view of the results we present curves in Figure 6 of the mean file transfer time as a function of the speed of the processor of the `responder` block. This experiment was performed by setting the timeout values to 4 and 16 while now keeping the mean channel delay at 6 time units with a negative exponential distribution. The confidence intervals are not shown in that figure.

The graph shows the effect of setting the timeout values to a value of 4 which is less than the mean channel delay of 6 time units. With such a low timeout value relative to the channel delay, many messages fail to reach their destination before the timeout expires resulting in many premature timeout messages. At a low speed for the responder (in the range 1.0 to 1.5), the task of discarding old negative acknowledgements and needlessly resending of acknowledgements takes up a significant proportion of its processing time and the performance of the protocol is significantly poorer. A timeout value of 16 rectifies

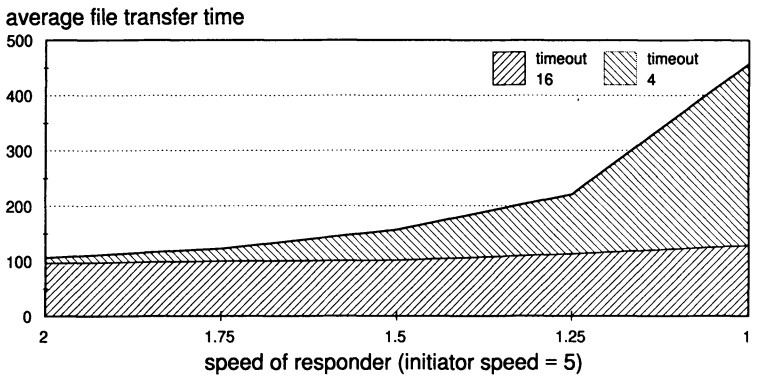

Figure 6 The effect of execution speed and various timeout values.

this. This information is typical of that which would be very useful to the developer for the design of an efficient implementation.

6 CONCLUSION

In this paper we use the concept of performance models proposed by Beilner (Beiler,1986)[Bei86] which separates the load (specification) from the machine (implementation environment) to attach time to the specification of a communication protocol (or any dynamic, discrete event system) for its simulation. Interpreted in this obvious way it becomes clear there is no need to augment the specification of a communication protocol in ways that may affect the standard syntax of the formal description technique. The example described in the previous section illustrates the feasibility of our approach applied to a simulation of the TFTP protocol described in SDL. In many cases we could equally well have used Markov or queueing theory or any of the other proposals in the literature to solve the resultant model of the system rather than simulation. The problem of determining the values of the appropriate parameters in those cases remain however.

In the SPECS SDL toolset we used in our experiments the simulation is derived from an execution of a *virtual machine* derived from the specification. In hindsight we appreciate that to have the performance predicted reflect the real situation as closely as possible, one should execute the same code for the simulation (augmented by appropriate software libraries) as that from which the implementation is derived. This observation was also made by Boehmer and Klafka (Klafa, Boehmer, 1994)[BK94]. This is being rectified in a new prototype of the software tool.

REFERENCES

[AB95] Telelogic AB. *The SDL Design Tool.* PO Box 4128, S-203 12 Malmö, Sweden, 1995.

[B+95] F. Bause et al. SDL and Petri net performance analysis of communicating systems. In P. Dembinski and M. Sredniawa, editors, *PSTV XV, 15th International Conference on Protocol Testing, Specification and Verification*, pages 269–282. Chapman and Hall, 1995.

[BB90] F. Bause and P. Buchholz. Protocol analysis using a timed version of SDL. In J. Quemada et al., editors, *FORTE'90, 3rd Int'l Conf. on Formal Description Techniques*, pages 269–285, 1990.

[Bei86] H Beilner. Workload characterisation of performance modelling tools. In G Serazzi, editor, *Workload Characterisation of Computer Systems and Computer Networks.* North Holland, 1986.

[BH95] S. Budkowski and M. Hendaz. A new approach for protocol performance evaluation using annotated Estelle specifications. In G. v Bochmann et al., editors, *FORTE'95, 8th Int'l Conf. on Formal Description Techniques*, pages 338–345. Chapman and Hall, 1995.

[BHS91] F. Belina, D. Hogrefe, and A. Sarma. *SDL with application from protocol specification.* Prentice Hall, 1991.

[BK94] S Boehmer and R Klafka. A new approach to performance evaluation of formally specified protocols. In D. Hogrefe and S. Leue, editors, *FORTE'94, 7th Int'l Conf. on Formal Description Techniques*, 1994.

[BMS95] M Bütow, M Mestern, and C Schapiro. *SPECS Users' Guide*, 1995.

[Bud92] S. Budkowski. Estelle Development Toolset (EDT). *Computer Networks and ISDN Systems*, 25:63–82, 1992.

[D+95] M. Diefenbruch et al. Performance evaluation of SDL systems adjunct by queueing models. In R. Bræk and A. Sarma, editors, *SDL'95 with MSC in CASE, Proc. of the 7th SDL Forum, Oslo, Norway, 1995*, pages 231–242. Elsevier Science BV, Holland, 1995.

[ISO89] ISO. *Estelle: A formal description technique based on an extended state transition model.* International Standards Organisation, 1989.

[ITU94] ITU-T. *Recommendation Z.100 White Book and Annexes.* International Telecommunication Union, 1994.

[Kre94] H. Kremer. Derivations of efficient implementations from formal descriptions. In D. Hogrefe and S. Leue, editors, *FORTE'94, 7th Int'l Conf. on Formal Description Techniques*, pages 431–446, 1994.

[Kri86] P.S. Kritzinger. A performance model of the ISO communication architecture. *IEEE Transactions on Computers*, COM-34(6):554–563, 1986.

[KW89] P. S. Kritzinger and G. A. Wheeler. A protocol engineering workstation. In *FORTE'89, 2nd Int'l Conf. on Formal Description Techniques*, pages 53–59. Elsevier Science Publishers B.V. (North-Holland), 1989.

[KW93] P.S. Kritzinger and G. Wheeler. Semi-Markovian analysis of protocol performance. In A. Danthine et al., editors, *PSTV XIII, 13th International Conference on Protocol Testing, Specification and Verification*, pages 159–172. Elsevier Science Publishers B.V. (North-Holland), 1993.

[Pos80] J. Postel. *User Datagram Protocol*, RFC768 edition, 1980.

[Rud84] H. Rudin. An improved algorithm for estimating protocol performance. In *PSTV IV, 4th International Conference on Protocol Testing, Specification and Verification.* Elsevier Science BV, Holland, 1984.

[SA95] Verilog SA. *GEODE Reference Manual.* Verilog SA, 52 Avenue Aristide Briand, 92220 Bagneux, France, 1995.

7 BIOGRAPHY

Mark Mestern, Michael Bütow and *Cara-lee Schapiro* completed their Honours degree is Computer Science at the University of Cape Town in 1995. The development of SPECS was project work done in partial completion of their degrees. Michael and Cara-lee are now working in industry and Mark is working on a dissertation for his Masters degree.

Pieter Kritzinger obtained an MSc in Electrical Engineering from the University of the Witwatersrand, South Africa, and his PhD in Computer Science from the University of Waterloo, Canada, where he became Assistant Professor for 2 years. This was followed by 2 years of teaching at Imperial College, University of London before he returned to become a senior lecturer at Stellenbosch University. He joined UCT in July 1985 as a full professor. Pieter spent a year or more on various study leave periods at the Fachbereich Informatik at the University of Dortmund (most recently in 1992) as well as IBM's Zürich Research Laboratory.