20

PERFORMANCE TESTING AT EARLY DESIGN PHASES

Péter Csurgay
Department of Telematics,
Norwegian University of Science and Technology
Peter.Csurgay@eth.ericsson.se

Mazen Malek
Conformance Center, Ericsson Ltd.
Laborc u. 1., Budapest, 1037, Hungary
Mazen.Malek@eth.ericsson.se

Abstract Testing non-functional properties based on functional specifications at early design phases is still a challenge in teleservice and protocol engineering. Modelling and evaluating performance aspects along with the traditional functional verification oriented formal description techniques has to be integrated into one unified design process to avoid extra effort in maintaining consistency between multiple design models. The paper introduces PerfSDL, a semantic extension to SDL for modelling non-functional properties, and a performance evaluation framework based on PerfSDL and simulation. To verify the feasibility of the approach, three design alternatives in a small TCP-like packet transfer protocol above transport services of different quality were tested and evaluated by means of simulation and statistical analysis.

Keywords: Testing, performance evaluation, protocol engineering, formal description techniques, SDL, prototyping

1. INTRODUCTION

Standardised formal techniques for specifying and functionally verifying telecommunications protocols fail to capture non-functional aspects like performance and Quality of Service (QoS). Thus, protocol engineering is abound to use separate models for the design of functional and non-functional be-

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35567-2_25

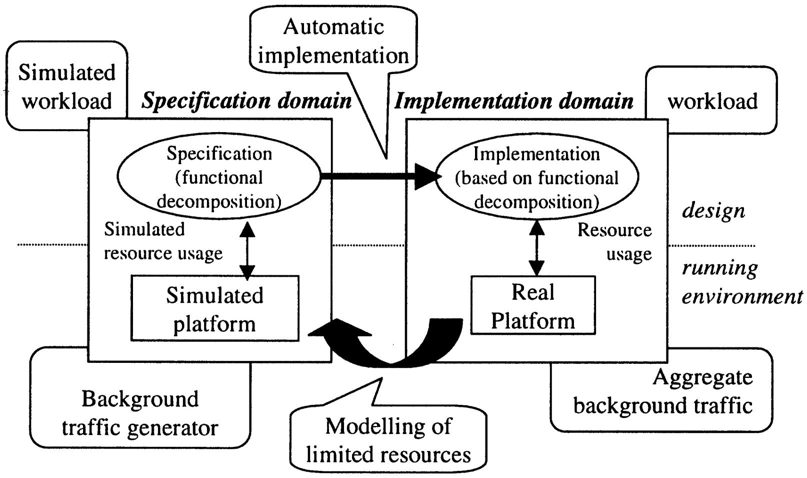


Figure 1 Integration of non-functional aspects into early design

haviour. The extra time and cost of keeping the models consistent can be saved by treating all aspects in one unified model. A feasible approach is to integrate performance issues into a well settled protocol engineering process. The ITU-T standardised formal language SDL has a growing acceptance for functional modelling and verification in practical protocol engineering [4], and ongoing research explores its applicability to performance engineering with extensions to modelling non-functional properties [10]. These efforts are summarised in Section 2.

In the background of this project lies the recognition that while originally the language was constructed for describing required and actual behaviour of reactive systems, its use spans into automated implementation of protocols based on functional specification [1]. Thus, the actual implementation of the protocol will closely reflect the functional decomposition from early design phases written in SDL. In order to be able to ask performance-related questions about the future implementation of a protocol in an earlier design phase, information about that implementation must be provided for the design model. Such information covers the scheduling algorithm of concurrent processes, limited resources of the target environment (like buffer sizes, channel capacities, etc), and resource usage of functional building blocks of the protocol. To gain statistical performance data by means of simulation, stimuli must also be taken into account in the early design models. We model stimuli as an aggregated background load on the transport infrastructure our protocol is acting upon, and the actual workload our protocol has to process. The conceptual model of our performance testing in the early design phases is illustrated in Figure 1.

According to this model, performance testing and evaluation of three design alternatives in a simple TCP-like protocol above channels of different quality were carried out by means of simulation. The new constructs of PerfSDL [9] and their use in modelling non-functional behaviour are introduced in Section 3. The simulation environment and the testing results are described in Section 4 and 5 respectively. The paper concludes and further directions are set in Section 6.

2. STATE OF THE ART

Recently, there was a large discussion on performance integration within the SDL community. The study that was initiated by the ITU-T SG10 on support of performance engineering activities in the early stages was one of them. Tools are being developed nowadays to handle such activities. SPEET is one of those tools and was introduced in [12]. It uses the C code generated from the compilation of an SDL specification, which might include some probes added by the system developer in order to address interesting performance areas. Another tool is QUEST that was introduced in [3], which defines a new extension to SDL called QSDL. This is mainly achieved through introducing two new constructs in the SDL called load and machine. A similar tool is SPECS [2]. The tool that was introduced in [9] accomplishes the integration of functional and performance aspects by introducing an interface to a simulation environment used extensively for network performance evaluation and analysis. This is a major advantage in terms of less work to be done and in terms of speed capabilities. In the literature there are a number of papers giving surveys on the additional information needed for a performance evaluation of an SDL specification, e.g. [11].

3. NEW SDL SEMANTICS IN PERFSDL

3.1 SIMULATION REQUIREMENTS

Real-time characteristics have to be added to SDL prior to any performance testing, e.g. the introduction of timed transitions, reinterpretation of concurrency, and finite input queues. In simulation-based approaches to performance testing this issue is of substantial importance. Moreover, aspects like scheduling and simulation time are closely related to those characteristics. Below we list those areas that need special treatment when simulating PerfSDL systems with performance aspects in mind.

Modeling Processing Time. Currently, in SDL, time advances when all queues of the system are empty [4]. This is generally interpreted this way

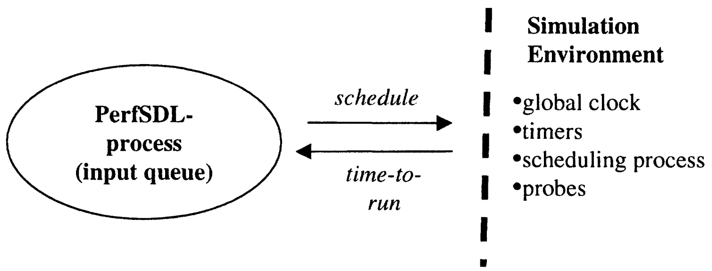


Figure 2 PerfSDL process interactions with the environment

to avoid a dramatic increase in the number of possible traces. In simulation, however, it is not adequate to advance time only when there is nothing to do, but also when there is any action to be taken.

Addition of Performance Sensors. In order to gain quantitative measures on a system's performance, different counting events of time and recurrence of execution of transitions have to be applied. Time is crucial for measuring throughput, while recurrence of transitions reflects system bottleneck.

The Concept of Scheduling. This concept is the most essential one during simulation. It specifies the time when the simulator is asked to act on certain change in the state of the model. This is not necessarily performed at the time when transitions start. When a process consumes an input signal, it should indicate to the simulator its reaction and when to handle it. Also the simulator has to consider the possible interactions that may happen after or during that reaction. In this context we consider the scheduling to be events that should occur after each progress in time, namely when consuming input or performing a transition.

Degree of parallelism. Processes that run in a completely interleaving and independent manner can schedule their actions independently on the simulation time axis. Nevertheless, when processes share the same processor, as usually, a certain priority and scheduling strategy should be worked out between them. State transitions may consist of a number of actions, like decision and assignment, and therefore can not be handled as an atomic unit of scheduling. Instead of that, actions should be that unit.

3.2 NEW EXTENSIONS TO SDL

The formal interpretation of PerfSDL, as presented in [9], is based on PerfSDL-process, which can be characterised by the following:

- set of states, e.g. s_1, s_2, s_3;
- set of internal variables;
- set of initial states, e.g. s_0;
- transition function, e.g. tr_{12}.

This is called the underlying state machine, which requires the availability of a global clock in the system. The interaction with the environment, as in Figure 2, takes place by two types of controlling events:

- **schedule** and;
- **time-to-run**.

The environment acts on process scheduling requests, through **schedule** events, by assigning a time event for performing the task requested. Additionally, it activates relevant processes when there is some task to be handled by them through **time-to-run** events. This synchronisation is performed between each running process and the environment. PerfSDL processes can communicate with each other at block level. All levels higher than leaf block level are described in the simulation environment.

"Probes" in PerfSDL. These have the ability to monitor, count and analyse the applied system. They are new elements that can be included during the design process but may be defined during testing. There are two kinds of probes - with or without time stamps. A counter counts the number of times a certain transition is visited, while stamps are used to record the time of specific events. The current status of some internal variables may be included in these stamps. In the following we will refer to probe events having time stamp requests by **stamp**(tr), where tr is the transition where we want to apply our analysis, e.g.: **stamp**(tr): **current state is** q_i **and input is** i_i

"Timed transitions" in PerfSDL. This means taking time elapsing in account in each event that uses a processor's time. A trace, in PerfSDL, is a sequence of states and timed transitions:

$$trace : \theta = s_0(t_0)s1(t_1)s_2(t_2)s_3(t_3)....$$

$$transitions : tr_{01}(t_0)tr_{12}(t_1)tr_{23}(t_2).....$$

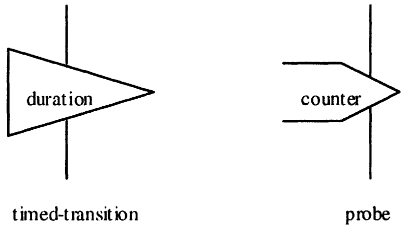


Figure 3 Introduction of new constructs for performance testing in PerfSDL

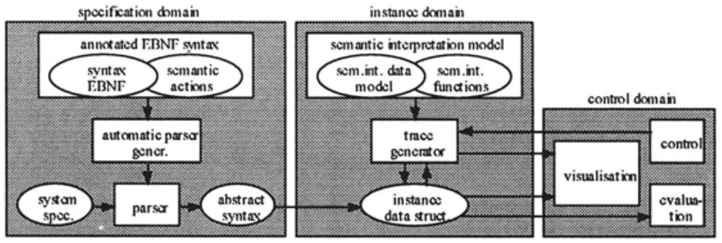


Figure 4 Behaviour interpretation of extended semantics in PerfSDL specifications

These times, e.g. t_1, include the consumption of inputs and execution of transitions. Figure 3 introduces the graphical representation of probes and timed transitions.

4. SIMULATION ENVIRONMENT

For behaviour interpretation of system specifications using new constructs and new semantics of PerfSDL in our simulations, we built a prototyping framework. The main functions and data structures of the framework are illustrated on Figure 4. Because developing PerfSDL is also a subject of our research, our framework has to support prototyping PerfSDL specifications. Meanwhile its syntax and semantic interpretation is constantly evolving.

The framework includes automatic parser generation and abstract grammar definition from extended Backus-Naur Form (EBNF) descriptions, and an easily extensible object-oriented formal definition of the semantic interpretation model. Java has been used to implement an open framework of distributed visualisation, control and evaluation components.

The ITU-T Recommendation Z.100 defines the Specification and Description Language (SDL) with (1) its concrete grammars SDL/PR and SDL/GR,

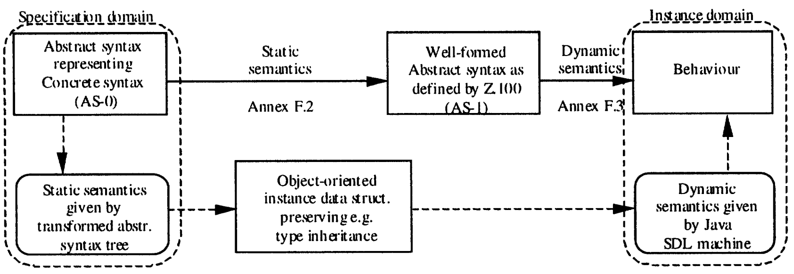

Specification domain
Abstract syntax representing Concrete syntax (AS-0)
Static semantics
Annex F.2
Well-formed Abstract syntax as defined by Z.100 (AS-1)
Dynamic semantics
Annex F.3
Instance domain
Behaviour
Static semantics given by transformed abstr. syntax tree
Object-oriented instance data struct. preserving e.g. type inheritance
Dynamic semantics given by Java SDL machine

Figure 5 Alternative to Annex F derivation of PerfSDL behaviour

(2) an abstract syntax AS_1 equivalent to the concrete grammars, (3) rules to transform a system specification into abstract syntax, and (4) the SDL-machine semantics to interpret a specification given in terms of the abstract syntax [5]. Z.100 gives formal definition of SDL in Annex F2 and F3 describing the static and dynamic semantics respectively [6]. Their relation is illustrated by the top segment of Figure 5 along with the AS_0 and AS_1 syntaxes introduced by Z.100. AS_0 is not formally defined in the recommendation, but can be derived from the BNF syntax. AS_1 is given in Z.100 as the Abstract Grammar. However, with the F.2 transformation from the abstract syntax (representing concrete textual syntax) to AS_1 (input to the dynamic behaviour definition of SDL), specification domain information is lost. Such information would regard type inheritance hierarchies or virtuality. The bottom segment of Figure 5 illustrates our alternative way to derive dynamic behaviour of a PerfSDL specification with the representation forming the basis of behaviour interpretation that preserves such information, thus being closer to the specification domain.

Time and ordering of internal events play a key role in SDL's non-determinism. Due to the vague time semantics of SDL (time may or may not be advanced by signal transactions and transitions), tool builders and users of SDL choose legal, but different, interpretations of time semantics when simulating or interpreting SDL systems. A legal interpretation for example (most probably inspired by the way some leading tools handle advancing of time) is that time passes only if there are no enabled transitions (no valid input signals in queues) of processes. This interpretation is legal, reducing the set of possible traces and simplifying validation for example, but not appropriate for performance modelling [11]. A prototyping framework should support generation of possible traces of system behaviour based on different interpretation models. This is achieved by separation of different interpretation functions in a modular object-oriented model of formal semantics.

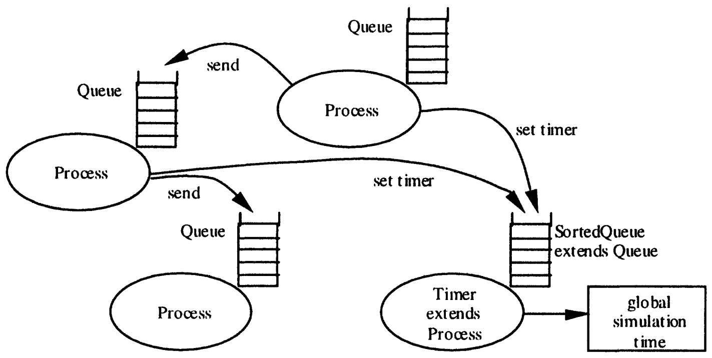


Figure 6 Time semantics interpreted with global timer process approach

Figure 6 illustrates an approach where the chosen interpretation of time semantics is orthogonal to the scheduler algorithm mentioned among the necessary implementation related aspects in Section 1. This solution is compatible with different interpretations of time semantics with respect to the generated set of possible traces. Whenever a process instance encounters setting of a timer, it is interpreted as sending a signal to a global timer process with parameters of signal name, sender id and expiration time. The timer process's queue is sorted by these expiration time values, and is handled by the scheduler the same way other processes are handled. Upon interpreting the reception of a signal by the timer process, it sets the global simulation time to the specified expiration time parameter, and sends a signal with the specified signal name parameter to the specified sender parameter. If the timer process is always scheduled last among all processes, the interpretation will generate a trace-set equivalent to the "time may not be advanced..." interpretation. If the scheduler randomly selects armed processes including this timer process, the resulting set of possible traces will be equivalent to the "may be advanced..." interpretation. Note that with this separation and orthogonal modelling of semantic interpretation functions, different concepts can be experimented with by prototyping in the modular object-oriented framework of semantic interpretation models.

5. SIMULATION RESULTS

This section illustrates the applicability of the method introduced in this paper by inspecting some design aspects of the TCP transport protocol of the Internet. We limit our investigation to the achievement of high utilisation of network resources. We attempt to address the influence of design alternative choices on the overall performance of an implementation. Figure 7 shows

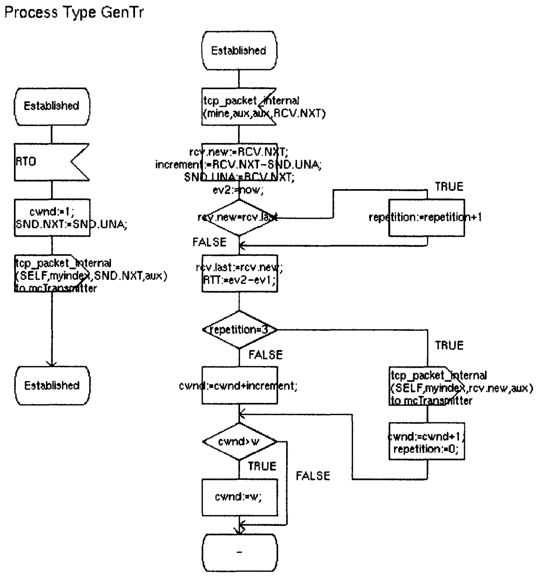


Figure 7 SDL specification with repetition set to 3 (third design alternative)

part of the SDL specification of a TCP transmitter entity. It contains the basic sliding-window aspects contained in TCP, with a lot of hidden details. Definitely, these details may have a substantial impact on performance issues in general, but they have no impact on the aspects we deal with. The model, containing transmitter, channel and receiver, gives a brief demonstration of the data transfer part of TCP with enough descriptiveness and soundness. In the diagram of Figure 7, the value of internal variable repetition is crucial, since it states when to re-inject some already sent packets. This specific part of the protocol shows how the overhead traffic will look when applied in a certain environment. There is an infinite number of design alternatives according to this aspect. They are design alternatives because they influence other parameters in the model, namely the decision of when to consider the occurance of a *Timeout* event. There are also some timers that depend on these settings. We choose three of them, in particular when repetition is equal to 1, 2 and 3. Our analysis of the performance, as mentioned earlier, will concentrate on the overhead traffic that is the major player in channel utilisation. By adding time stamps into the base specification we can obtain measurements of the repeatedly sent packets, and accordingly can compare design alternatives.

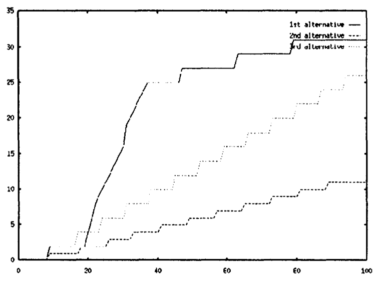
Figure 8 shows the log of four simulation measurements applied on the design alternative having repetition set to 3. These measurements are sketched to show how the choice of stamp events would help in obtaining different

Measurement1 Stamp (time,seq,ack)	Measurement2 Stamp (time,seq,0)	Measurement3 Stamp (time,0,ack)	Measurement4 Stamp (time,seq)
(0.0050,1,0)	(0.0050,1,0)	(0.0105,0,2)	(0.0155,2)
(0.0105,0,2)	(0.0155,2,0)	(0.0260,0,4)	(0.0205,3)
(0.0155,2,0)	(0.0205,3,0)	(0.0425,0,6)	(0.0310,4)
(0.0205,3,0)	(0.0310,4,0)	(0.0525,0,8)	(0.0360,5)
(0.0260,0,4)	(0.0360,5,0)	(0.0675,0,9)	(0.0410,6)
(0.0310,4,0)	(0.0410,6,0)	(0.0775,0,9)	(0.0460,7)
(0.0360,5,0)	(0.0460,7,0)	(0.0875,0,9)	(0.0510,8)
(0.0410,6,0)	(0.0510,8,0)	(0.6015,0,9)	(0.0560,9)
(0.04250,0,6)	(0.0560,9,0)	(0.6120,0,19)	(0.0610,10)
(0.0460,7,0)	(0.0610,10,0)	(0.6285,0,20)	(0.0660,11)
(0.0510,8,0)	(0.0660,11,0)	(0.6385,0,20)	(0.0710,12)
(0.0525,0,8)	(0.0710,12,0)	(0.6485,0,20)	(0.0760,13)
(0.0560,9,0)	(0.0760,13,0)	(0.6635,0,20)	(0.0810,14)
(0.0610,10,0)	(0.0810,14,0)	(0.6735,0,20)	(0.0860,15)
(0.0660,11,0)	(0.0860,15,0)	(0.6835,0,21)	(0.0910,16)
(0.0675,0,9)	(0.0910,16,0)	(0.6985,0,21)	(0.0960,17)
(0.0710,12,0)	(0.0960,17,0)	(0.7075,0,21)	(0.6065,18)
(0.0760,13,0)	(0.6015,9,0)	(0.7140,0,22)	(0.6170,19)

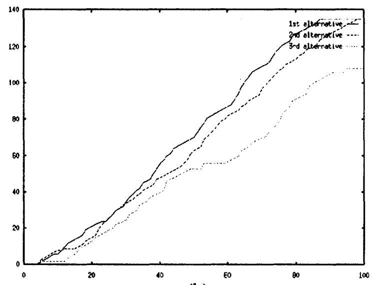
Figure 8 Measurements

statistics about system runs. They have been obtained by a simulation run of the PerfSDL specification of the TCP model. This model contains some timed-transitions and stamps applied in different places. Stamps are associated with certain transitions where inputs or outputs are in question. This way, recurrence of packets is counted and measurements related to throughput can be easily performed.

Measurement 1 for example uses two stamps for sending and receiving packets. It registers all sent packets even if they are repeated, like packet 9 at time 0.0675. Measurement 2 filters out the received packets or acknowledgement packets. That is why the ack parameter is replaced by 0. Meanwhile Measurement 3 does the opposite by showing the received packets only. Measurement 4 takes a further step and measures the packets sent with no repetition. These measurements were performed for each design alternative separately. The comparison of them is sketched in Figure 9. The comparison is done according to the amount of overhead traffic due to each design alternative. The diagrams describe the change in amount of traffic corresponding to repeated packets. On the horizontal axis there is the sequence number of packets, while on the vertical axis there is the number of repeated packets. The measurements were carried out using different models of the communication channel, and the design alternatives have different characteristics in these environments. In these comparisons we refer to the design alternatives having repetition values of 3, 2 and 1 as the 1^{st}, 2^{nd} and 3^{rd} alternative respectively. The best choice is clearly the one with least overhead traffic. It is the 2^{nd}, 3^{rd} and 1^{st} in the (a), (b) and (c) environments respectively.

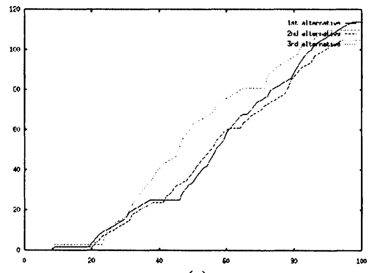


(a)



(b)

(a) relatively low loss and delay environment;
(b) high loss environment;
(c) high delay environment.



(c)

Figure 9 Amount of overhead traffic of design alternatives in different environments

6. CONCLUSIONS

A model to integrate performance engineering into the early design phases of protocol engineering was presented. New constructs in PerfSDL and their use for expressing non-functional behaviour were introduced. Performance testing and evaluation of packet transport protocol alternatives were carried out by means of statistical data gathering in a proprietary simulation environment. Simulation results show the feasibility and value of the approach in improving the integration of non-functional design into the protocol engineering process. Further efforts are being put into the specification of a PerfSDL based performance evaluation and protocol design tool environment.

References

[1] Rolv Brak, Oystein Haugen, *Engineering Real Time Systems*, Prentice Hall International, 1993

[2] M. Butow, M. Mestern, C. Schapiro, P. Kritzinger. *Performance Modelling with the Formal Specification Language SDL.* FORTE/PSTV'96. Chapman & Hall 1996.

[3] M. Diefenbruch, E. Heck, Jorg Hintelmenn, B. Clostermann. *Performance*

Evaluation of SDL-Systems Adjunct by Queuing Models. Proc. 7th SDL forum. Elsevier 1995.

[4] Jan Ellsberger, Dieter Hogrefe, Amardeo Sarma, *SDL - Formal Object-oriented Language for Communicating Systems*, Prentice Hall Europe, 1997

[5] ITU-T Recommendation Z.100, CCITT *Specification and Description Language* (SDL) 03/1993

[6] ITU-T Recommendation Z.100, *Annex F. Static and Dynamic Semantics of SDL*, 03/1993

[7] ITU-T Recommendation Z.100, Appendices I and II: SDL Methodology Guidelines, 03/1993.

[8] ITU-T Recommendation Z.100, *Supplement 1: SDL+ methodology: Use of MSC and SDL (with ASN.1)*, 05/1997.

[9] Mazen Malek, *PerfSDL: an Interface to Protocol Performance Analysis by means of Simulation*, Accepted paper, 9th SDL Forum, Montreal, June 1999

[10] Andreas Mitschele-Thiel, *Results of the discussion*, Workshop on Performance and Time in SDL and MSC, Erlangen, February 1998

[11] Andreas Mitschele-Thiel, *Performance Evaluation of SDL Systems, SAM98*, 1st SDL and MSC workshop of the SDL Forum Society, Berlin, July 1998

[12] Martin Steppler and Matthias Lott. *SPEET - SDL Performance Evaluation Tool.* SDL'97: TIME FOR TESTING- SDL, MSC and Trends. Elsevier Science 1997.