Performance analysis of SDL systems from SDL descriptions

Claes Wohlin

Telesoft AB, Box 4148, S-203 12 Malmö, Sweden

Abstract

This paper gives a brief introduction to a methodology for early performance analysis based on SDL descriptions and models of the hardware architecture and the behaviour of the users. It is described how the SDL descriptions can be transformed automatically into descriptions capturing the original behaviour as well as the performance. In particular, the benefits and the opportunities with the methodology are presented by an example.

1. INTRODUCTION

A brief introduction will be given to a methodology for performance analysis of software systems based on transformations of SDL system descriptions (section 2), as well as to the transformation and generation rules of the SDL descriptions (section 3). The emphasis of the paper will be on an example showing the opportunities the methodology and in particular the transformation of SDL descriptions will provide (section 4). The rules are currently being implemented in a tool prototype, which will be described briefly (section 5). Finally, some conclusions will be drawn based on the experiences from the work (section 6).

Methods for early performance analysis of the software systems being developed are soon a necessity, [1], since it will be too expensive to develop a product and in the end realise that it will not fulfil the performance requirements. Instead it would be favourable to evaluate the performance at an early stage and be able to study different solutions from the performance perspective. For example, a poor software solution can be re-designed before implementation, different allocations of the software processes in a distributed architecture can be evaluated or the suitability of numerous different architectures can be investigated before choosing the actual architecture to run the software on. These possibilities have lead to that a methodology for performance analysis based on software descriptions, in particular SDL [2], and models of the hardware architecture and the behaviour of the surrounding environment has been developed. The methodology covers two aspects of performance analysis of SDL systems, i.e. analysis of the SDL descriptions alone and analysis of software, architecture and environment in a joint model.

The methodology is based on three modelling concepts, which are put together to form a simulation model for the software system. The modelling concept capturing the behaviour of the software can be generated automatically from the SDL descriptions, in our particular case using SDT [3, 4, 5]. This is, however, not a requirement for using or implementing the methodology.

The ideas presented regarding the methodology and its application to SDL systems are state-of-the-art research and development. Most of the research has been carried out at the Department of Communication Systems at Lund Institute of Technology, Sweden [6], while the implementation and the refinement of the ideas are being made at Telesoft. As far as known by the author no similar approach to performance analysis of software systems has been presented in the literature, not for any description language and in particular not for SDL. This also means that work remains to be done, but the results obtained so far seem promising. The methodology and the example discussed below are presented in detail in [6] and an overview of the methodology has been presented in [7]. It is also the objective to present more results and experiences as the work proceeds.

2. METHODOLOGY OVERVIEW

The methodology for performance analysis at an early stage of the development of software systems is based on that an analysis object and its environment are identified, see figure 1. The analysis object can be anything between a software process and a complete system.

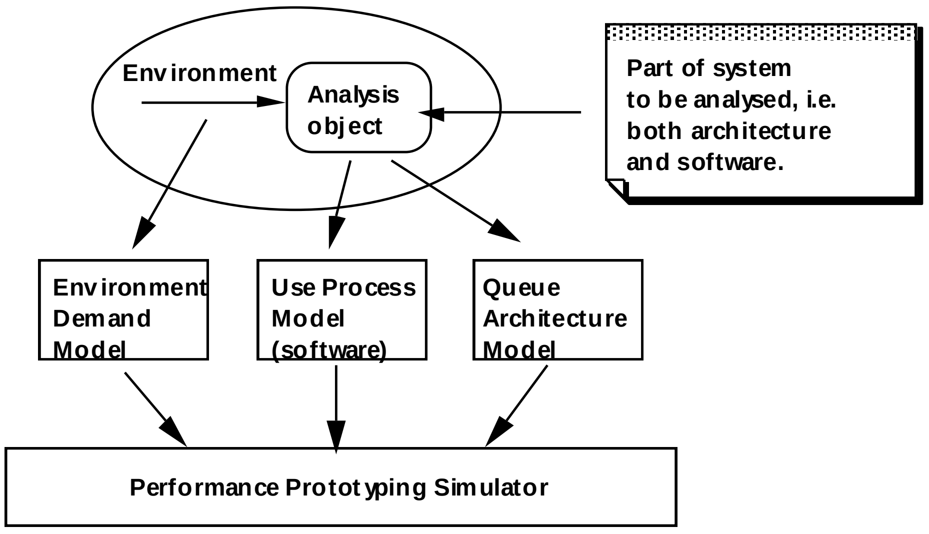
Environment
Analysis object
Part of system to be analysed, i.e. both architecture and software.
Environment Demand Model
Use Process Model (software)
Queue Architecture Model
Performance Prototyping Simulator

Figure 1. An overview of the performance analysis methodology

The basis for the methodology is that it is possible to divide the problem domain into three initially independent parts, i.e. application software, architecture (e.g. processors, busses, operating system) and the surrounding of the analysis object, hereafter denoted environment. These three aspects are modelled in one model each, i.e. application software - Use Process Model (UPM), architecture - Queue Architecture Model (QAM) and environment - Environment Demand Model (EDM). The former, i.e. UPM, is to be generated automatically from the application software descriptions (in particular SDL), while the QAM and the EDM are described with simulation models formulated in SDL. This type of models has been used during several years to do performance analysis of telecommunication systems, see for example [8, 9]. In the future it ought to be possible to generate skeletons of the QAM and the EDM respectively. Further work is, however, required.

The three models are then put together to form a simulation model of the system, i.e. the Performance Prototyping Simulator. The advantage with this concept compared to traditional performance simulations is that the actual behaviour of the software is incorporated in the analysis at an early stage, which seems utterly important since a lot of the dynamic behaviour of the system is described by the software. This means that poor solutions can be detected and that different solutions can be evaluated from a performance viewpoint before final implementation. The methodology also allows the analyst to do separate analysis of the software, i.e. to use an in some sense ideal architecture, and a complete system analysis. Both approaches ought to be applied to the software system being developed. The former provides information about bottlenecks in the software, while the latter evaluates the total behaviour of the system (or part of the system).

3. TRANSFORMATION AND GENERATION RULES

The methodology requires tool support in particular a simulator for SDL systems is needed. The simulator ought to support both functional and performance simulations. The latter can be based on the functional simulator complemented with several packages containing for example routines concerning files, random numbers, lists and information about other processes. The latter can be exemplified with that it ought to be possible to find out the process identities of all processes of a specific type. The preprocessor to the simulator must be able to transform the SDL system descriptions into new SDL descriptions capturing the performance behaviour as well as the original functional behaviour.

The transformation and generation can be grouped into two areas, i.e. the handling of hierarchical processes and the transformation of the SDL symbols. The concept of hierarchical processes is introduced to cope with that the transformed software descriptions shall execute on a simulation model of the architecture. The latter is also described with SDL processes, though it is a simulation model instead of a system description. Unfortunately, it is impossible to describe the actual transformations in any detail in a paper like this. The presentation below is only meant to give a flavour of the type of transformations that is needed. The transformation rules are presented in detail in reference [6].

The original SDL system is transformed into a partial simulation model (UPM), in which the modelling concepts from the methodology can be found. A new system and new blocks are generated. The new entities are connected together with channels. The signals from the original SDL system are transformed into a form which supports the handling of hierarchical processes, i.e. almost all signals are sent via the architecture instead of directly between the software processes as specified in the SDL system. Four types of signals in the original SDL system have to be handled, i.e. signals to the environment, between blocks, between processes and within processes. Some transformation of the processes are needed as well to support the handling of hierarchical processes. The software processes must know which architecture process they are running on. The signal sending has to be adapted to the fact that the signals shall be sent via the architecture, which includes both adding new parameters to the signals and re-directing them. Finally, the transformed processes must request execution from the architecture process on which they are executing. These transformations take care of the proposed handling of hierarchical processes, see reference [6].

The actual transformation of the original SDL system consists of several activities. The symbols have to be transformed according to the rules discussed in reference [6]. It is determined which parts of the SDL system that shall be transformed. The execution times for different symbols are modelled as delays. The decision boxes containing informal text are transformed by using random numbers, i.e. the user can decide the probability for different paths and for each execution a random path is chosen according to the probabilities given by the user.

4. AN EXAMPLE OF PERFORMANCE ANALYSIS FROM SDL

The main emphasis of the paper is on this section, where an example transformed manually and complemented with simulation models of the architecture and the environment respectively is described. The main objective with the example is to form an understanding of how the presented methodology can be put into practical use.

4.1. Introduction

The presented example is the basis for implementing the transformation and generation rules into a tool prototype, i.e. an automatic translator from SDL/PR (system descriptions) to new SDL/PR (system descriptions including the performance aspect). This will be described further in the next chapter. The example will contain both software and architecture, but the Performance Prototyping Simulator shall be able to handle analysis of software and an ideal architecture as well. This type of analysis shall be done to identify bottlenecks in the software. It is, however, felt that the combination of software descriptions with models of the architecture and the environment is more complicated than the analysis with an ideal architecture.

The objective with the example, from the implementation point of view, is to evaluate the rules for transformation and the handling of software processes that are executed on the architecture. The actual example is of minor interest. The reader is not supposed to understand what the example actually does, but rather to be able to follow the methodology as well as to understand how it can be applied and used. The presented methodology is general. It does not in itself depend on the use of SDL, which only is used as an example of a suitable description technique. The methodology does not depend on a particular tool set either, but it is of course valuable if an existing tool set can be complemented, instead of developing a tool environment from scratch. SDT has been used in this example and it has been found suitable for implementing the performance prototyping simulation concept.

4.2. General description

The example will describe the communication between two very simple telephone exchanges, which only provide the subscriber with the possibility to call a local call (within the same exchange) or long distance call (to the other exchange). The architecture is modelled with three processes, one describing an exchange, one modelling the communication channel between the exchanges and one process handling the administration of the architecture. The service provided by the exchange is specified with seven SDL processes. The environment is modelled with five processes. Some of the processes are created and terminated dynamically, and several instances of the same process type may exist simultaneously. This means that the example in total will include 15 processes. The actual content of the processes is of little interest in an example like this, and because of this it will only be described very briefly below.

The example can be summarised by:

- 5 processes modelling the behaviour of the subscribers (environment).
- 3 processes modelling the architecture.
- 7 processes describing the services provided by the software.

The environment and the architecture described with the processes above are the Environment Demand Model and the Queue Architecture Model respectively. The Use Process Model is found through transforming the SDL descriptions of the services by applying the transformation rules. The system layout of the example including the environment can be seen in figure 2.

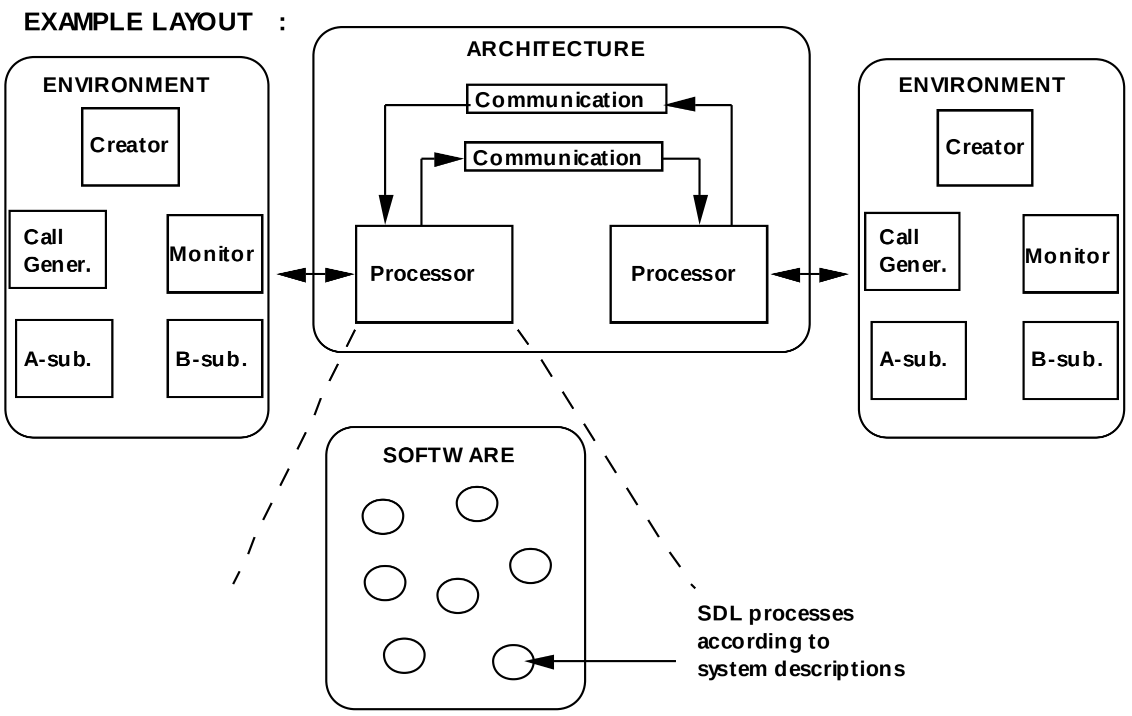

Figure 2. Layout of the example

The example will be gone through in several steps, i.e. description of the original SDL design, transformation of the descriptions, modelling of the architecture and the environment respectively and finally the execution of the obtained simulation model. These steps shall hopefully convince the doubter that the performance prototyping simulation concept will be a powerful way of doing early performance analysis of software systems.

4.3. The SDL system

Characterisation

The system can be characterised with the following:

- All "important" (in some sense basic) symbols are present in the example.
- The system consists of more than one block.
- The signalling is specified in detail, while the other symbols often contain informal text.

It has been the objective to let the SDL system contain most of the main features used when working with SDL systems, e.g. infinite number of instances, use of predefined primitives as 'sender' and 'offspring'.

Brief description of the software processes

The SDL system consists, as pointed out, of 7 processes, but before these are presented the system and block level have to be discussed. The system consists of two blocks. The first block handles all activities that concern subscribers and the second one is a block responsible for collecting the statistics of the telephone exchange.

The statistics block is simple, it only consists of one process of which one instance is created at the system start and it exists the whole life time of the system. The only thing to be noted with the process is that it calls a procedure regularly, which describes the times the statistics are put on a file.

The subscriber block consists of six processes, where one process is created by the system. This process is responsible for creating two monitors (one for each processor). The monitor process is the receiver of incoming calls and it creates other processes that handles the subscribers, both A- and B-subscribers. The B-subscriber process is created by the monitor in cases of a long distance call, otherwise the B-subscriber process is created by the A-subscriber process. The B-subscriber process is quite easy and it handles the communication with the B-subscriber in the environment. The monitor process also creates the two processes controlling the code receiving.

The process handling the A-subscriber creates a digit handler and is responsible for keeping the contact with the A-subscriber in the environment. The digit handler process checks if there is any code receivers free and if there are any it reserves a code receiver for the incoming call. The call is blocked if the code receivers all are occupied. The digit handler is responsible for releasing the code receiver when it has been used.

The code receiver process is not modelled in any detail at all. It consists mainly of signal receiving, signal sending and informal text. This is an important aspect, i.e. that the processes may be quite unspecified but still a performance analysis can be made by applying this concept.

Analysis

The SDL system has been analysed with the analyser in SDT. The analysis consists of the activities: SDL/GR to SDL/PR conversion, syntax analysis, semantic analysis and dynamic semantic analysis.

The analysis was completed without syntactic and semantic errors, while the dynamic analysis gave two warnings. The reason for the warnings were that signals were sent to offsprings, but not all the offsprings could receive that particular signal. The SDL system was studied in detail and it was found that this case never could occur. The warnings were left without action.

This does not mean that the SDL system is free from logical errors, to ensure this it is possible to do a functional simulation, but the system has to be more complete to allow this, see below.

Code generation and execution

Code was generated with the simulator to do a functional simulation of the system. The generation, compiling and linking succeeded. The execution started of as intended but failed when it came to a decision with informal text, in which case the simulator did not know which path to execute. The decision boxes with informal text shall be translated when doing the transformation into a partial simulation model.

4.4 Identify objectives with the simulation

The objective with the performance prototyping simulation shall not influence the translation technique applied, but it can influence the way the architecture and the environment are described in the model. The objectives of the simulation can lead to that a measurement process has to be included in the simulation system. In this case the objectives will be;

- Determine the total execution times of each software process type on the processors. This will give a possibility to identify the software bottleneck, since the available execution time is known through the simulated time. This shall not be mixed up with the execution time for the simulation program.
- The load of the communication links will be measured.

These objectives will affect the processor process, the process modelling the communication link and the Use Process Model processes. A complement is needed in the Use Process Model processes to be able to measure for each process type, see below. The latter measurement will require a separate measurement process. These objectives have to be modelled by the user, when describing the architecture, the environment and introducing a measurement process.

4.5. The transformed system

The original SDL system is transformed applying the rules mentioned briefly above. The transformation results in a new system level, where the original SDL system is a block. This corresponds to the Use Process Model discussed above. The new system level also contains three new blocks, i.e. one block modelling the architecture, one block modelling the environment and one general block (used to control the simulation and to make measurements). These three blocks are left without content, which means that the analysis of the transformed system gives three errors (block missing).

All block levels in the original system have been moved one step down in the hierarchical structure. The processes in the Use Process Model are the result of applying the transformation rules discussed above. It shall in particular be observed that all symbols with informal text have been removed or replaced, but they are modelled in terms of delays. It shall be noted that this provides a powerful way of making performance analysis without completely specifying the content of all SDL symbols.

4.6. Architecture and environment models

The formulation of the complete simulation model includes modelling the architecture, the environment and describing the content of the general block. The environment is modelled with five processes describing the behaviour of the subscribers, both A- and B-subscribers, as well as a model of a call generator and a monitor which is responsible for creating the B-subscriber when a call is made. The processes are all quite simple, since they only influence the actual objective of this example through their presence and not their content.

The processes in the general block are introduced to govern the simulation and to make measurements according to the (pretended) objectives with the simulation. The measurements are specified so that they shall cover some usual measurement situations.

Finally, the architecture is modelled. This part is the most difficult in the example, since it includes a general data structure which handles the connections between the different models (UPM, QAM and EDM) as well as the routing within the Queue Architecture Model. The actual content of the structure is generated by one process and the process modelling the processor then works on the generated structure. The processor process is formulated so that it can handle the structure independent of the actual generation.

4.7. The simulation system

This results in new SDL graphs describing the architecture, the environment and the general utilities. It is also necessary to alter some of the generated Use Process Model processes. These are changed due to the objective of measuring the load on the processors for each process type. The additions are only minor, i.e. the graphs are complemented with a new variable describing the process type, which shall be used to measure the load on the processors from different process types.

4.8. Simulation results

The simulation system can be executed after having been analysed, generated, compiled and linked. The simulator in SDT is further discussed in references [4, 10]. It shall only be pointed out that the simulator includes a monitor from which the simulator can be executed. The user obtains the possibility of influencing the simulation in several ways, which is very valuable when debugging the simulation program. After being convinced that the program performs as intended, it is possible to execute the simulator and obtain measurement results. It shall be noted that the obtained simulation model cannot only be executed for performance analysis. It can also be used as a real time functional simulator. This means that the methodology provides a way of doing functional simulations in cases where it in the normal case is impossible (see above), i.e. for incomplete system descriptions (that is for example descriptions containing informal text in decision boxes). The transformation and generation provides the opportunity to execute the original not completely specified SDL system from a functional perspective in a real time model environment as well. A functional error in the SDL system was found in the original SDL system during the execution, see below.

The input data to the simulation model is not based on any real figures. They are, however, chosen to work as an input set where the relative size between the different inputs are reasonable. The main objective is as pointed out earlier not the actual values, but to show that the simulation actually can be performed based on the performance prototyping simulation concept. The values on the parameters are easily changed since they are declared as external synonyms in SDL.

The following input has been used throughout the example to show the behaviour of the simulation model: mean time to dial the digit: 0.5, mean talking time: 50, mean time the phone is ringing before an answer: 5, mean transmission time on the links: 0.5, mean time to route a signal in the processors: 0.005, and time between statistical output in the exchange: 10.

The following execution times have been used for the different SDL symbols: state: 0.01, input: 0.02, output: 0.02, task: 0.05, decision: 0.03, create request: 0.05, process termination: 0.03, procedure call: 0.02, procedure return: 0.01.

Three parameters are of particular interest:

- The time when the A-subscriber thinks he has waited too long is first assigned the value 10, in which case the first output result below was obtained. The value is then changed to be equal to the simulation time.
- The simulation time is set to 1000. This may be too short to obtain real confidence in the output results, but since the actual figures are of minor interest it has been chosen short to obtain the results quickly.
- The mean time between arrival of calls is varied from 10 to 1.5.

The execution time of the simulation program becomes longer as the mean time between arrivals is lowered. The reason is of course that more happens in the same time.

The above input leads to two different results, where the first one is a functional result and the second one the expected performance results.

Functional result:

A race between two signals is discovered, i.e. the behaviour of the system becomes different depending on the order of two signals. A timer in one of the processes may be triggered (see above), which results in a signal sending and the termination of this process instance. The signal sending in its turn shall stop further sending of signals from the receiving process instance. Due to the delay in the architecture it may happen that the signal has not reached the instance before it sends a signal. This means that the signal is sent to a process instance that has terminated. This leads to a dynamic error in the simulation. The original SDL system has been specified so that under some circumstances this will occur. Specifically the problem arises for high loads. A re-design is therefore necessary to cope with this problem, which would have been difficult to find without simulation. This is, however, not done here since this only is an example, instead the problem is solved by assigning a new value to the time which triggers the timer. It is assigned a value which is equal to the simulation time and this means that the timer will never be triggered.

Performance results:

The results are shown in table 1. The table contains information about the mean time between new calls, the utilization of the two links, the total load on the most used processor, i.e. processor 1. The contribution to the load for the two processes that consumes most execution power on processor 1 is also shown in table 1. An example for processor 1, when the mean arrival time between calls is 10; the figure 45 stands for the time the process A_Handler executes and the time 176 is the total execution time used on the processor, i.e. out of 1000, which is the simulation time.

Table 1
Performance simulation results.

Mean arrival time calls	Utilization link 1	Utilization link 2	Load Pro. 1	Processor 1 A_Handler	Processor 1 Statistics
10	0.06	0.17	0.18	45(176)	51(176)
7	0.10	0.24	0.24	64(240)	61(240)
5	0.16	0.22	0.31	84(306)	72(306)
3	0.24	0.42	0.51	144(509)	106(509)
2	0.35	0.59	0.70	197(698)	140(698)
1.5	0.54	0.86	0.98	283(981)	181(981)

Some comments to the results in table 1 are worth making even if the actual result is of minor interest. It can be seen that the link from processor 2 to processor 1 is utilized more than the other, i.e. link 2. The reason is that the statistics process is only located on processor 1. It can also be seen that process A_Handler is the largest contributor to the load. It is responsible for between 26-29% of the load, which means that a re-design of the process perhaps ought to be considered. The statistics process contributes also very much and this is probably a problem, since the statistics in an exchange can be hard to motivate to the subscribers. A solution would be to distribute the statistics to all processors, and this will also cut down the utilization of the communication link between processor 2 and processor 1.

All in all it has been shown that it is possible to obtain interesting simulation results from applying the proposed simulation concept. We are able to obtain information about the performance and the functional behaviour of the SDL specification before implementing a faulty or poor solution.

4.9. Conclusions

It can be concluded that the proposed methodology can be implemented in an existing tool set for SDL, i.e. SDT. The presentation of the example has shown that rules can be formulated for transforming and generating the Use Process Model processes from the original SDL descriptions. It has also been proved that the generated processes can be complemented with descriptions of the architecture and the environment, and it has in particular been shown that the modelling concepts can be connected together. The three proposed modelling concepts (UPM, QAM and EDM) are valuable, since they let the user concentrate on one aspect at the time and then at the end connect them together. The methodology will therefore be a valuable contribution to the possibilities of doing early performance analysis and functional verification of software systems. The rules presented in reference [6] and the actual application of them in the example, is being implemented in a tool prototype within the SDT environment [3, 4, 5].

5. AUTOMATIC TRANSLATION FOR THE PERFORMANCE ANALYSIS

The work with the methodology has two main objectives; improvement and refinement of the methodology and secondly implementation of the results in a tool prototype.

The latter work constitutes of writing a translator for SDL/PR (original software descriptions) to new SDL/PR which captures the original functional behaviour as well as the performance aspects of the software. This includes changing, deleting and adding information compared to the original system design. The work is performed within the SDT environment which means that the development effort is considerable less than if developing a stand alone tool. Currently it is possible to generate most of the SDL/PR related to the handling of the software's performance. The next step is to add the features necessary to handle the hierarchical processes (see also above), i.e. the functions needed to be able to establish the contact between the software (SDL) and the simulated architecture and environment. The work has so far not reached any problems that cannot be overcome and the implementation of the rest of the rules also seems quite straightforward.

6. CONCLUSIONS

The work and the experiences from it can be summarised in the following statements;

- Early performance and functional analysis is becoming an important issue in the near future as the structure of the systems is changing and the complexity as well as the costs of the systems are continuing to grow.
- It has been shown that it is possible to develop a methodology for early performance and functional analysis based on transformation of software descriptions in SDL.
- The transformed software descriptions can then be combined with simulation models of the architecture and the environment. This shows that the methodology provides an opportunity to do analysis based on a joint model for software, architecture and environment at an early stage.
- Finally, it can be concluded that it is possible to implement the transformation and generation rules into a tool, which makes it possible to automatically generate the new SDL descriptions that both captures the original functional behaviour and the performance.

The methodology provides a basis for;

- identifying software bottlenecks at an early stage
- evaluating different distributions of software processes in an architceture
- studying the introduction of new services in an existing system (network)
- examining different architectures ability to execute a given software description
- identifying system bottlenecks

The work will continue and hopefully result in an improved methodology, a tool for automatic generation of the Use Process Model and skeletons for the Queue Architecture Model and the Environment Demand Model respectively. It is the objective that these efforts shall improve the possibilities of doing performance and functional analysis of SDL systems (containing informal text) at an early stage in the system life cycle.

7. ACKNOWLEDGEMENT

The presented work is part of a project on methods for simulation and prototyping techniques. The project is partly being funded by the Swedish Research Programme in Information Technology (number 4) and partly by the Swedish Telecom.

Many thanks to professor Ulf Körner, Department of Communication Systems, Lund Institute of Technology, Sweden for valuable discussions and comments throughout the research.

I also would like to thank the following colleagues at Telesoft for enlightening discussions regarding simulation, prototyping and performance analysis; David Rapp, Mats Löfgren, Jan Karlsson, Anders Larsson and Henric Isacsson.

Special thanks to Eva Hedman, Telesoft for introducing me to the development environment within SDT and in particular for helping me out with some of the most difficult problems encountered when implementing the tool prototype.

8. REFERENCES

1 D. Rapp and G. Sjödin, Capacity models: a quality and a system design tool and an aspect of systems, Proc. 5th Int. Conf. on Software Engineering for Telecommunication Switching Systems, Lund, Sweden, 1983, pp. 128-135.

2 CCITT Recommendation Z.100, Specification and Description Language SDL, Blue book, Volume X.1-X.5, 1988.

3 SDT 2 User's manual, Telesoft AB, 1990.

4 F. Belina, and G. Nilsson, SDT - SDL Design Tool, Proc. 3rd SDL Forum, Eindhoven, Netherlands, 1987, pp. 8.1-8.9.

5 G. Nilsson, I. Ljungdahl, and P. Madsen, SDL toolbox for support different SDL environments, In: SDL ´89: The language at work, O. Faergemand and M.M. Marques (eds.), Elsevier Science Publisher, North-Holland, 1989, pp. 87-93.

6 C. Wohlin, Software reliability and performance modelling for telecommunication systems, PhD-thesis, Dept. of Communication Systems, Lund Institute of Technology, Box 118, S-221 00 Lund, Sweden, 1991.

7 C. Wohlin and D. Rapp, Performance analysis in the early design of software, Proc. 7th Int. Conf. on Software Engineering for Telecommunication Switching Systems, Bournemouth, United Kingdom, 1989, pp. 114-121.

8 T. Karlstedt, Experience with and Results from the Usage of SDL/SIM for Performance Analysis, Proc. 5th Nordic Teletraffic Seminar, 1984, (In Swedish).

9 M. Sredniawa, B. Kakol and G. Gumulinski, SDL in performance evaluation, Proc. 3rd SDL Forum, Eindhoven, Netherlands, 1987, pp. 21.1-21.11.

10 J. Karlsson and A. Ek, SSI - an SDL simulation tool, In: SDL ´89 - The language at work, O. Faergemand and M.M. Marques (eds.), Elsevier Science Publisher, North-Holland, 1989, pp. 211-218.