C. Wohlin, "Performance and Functional Prototyping from Software Descriptions",
Proceedings 8th International Conference on Software Engineering for
Telecommunication Services and Systems, pp. 102-106, Florence, Italy, 1992.

PERFORMANCE AND FUNCTIONAL PROTOTYPING FROM SOFTWARE DESCRIPTIONS

Claes Wohlin

E-P Telecom Q-Labs, Lund, Sweden

INTRODUCTION

The use of formal description techniques, in particular standardised, are a prerequisite for transformation of the descriptions into other representations. These can be either a step in the development process or a way to analyse or dimensioning the qualities of the system at an early stage.

The objective with this work is to formulate a general (independent of description technique) methodology for performance analysis at an early stage. It shall, however, be noted that as a side effect it is possible to do functional analysis in a real time model as well. A thorough presentation of the methodology is given in Wohlin (1), while a general introduction in terms of modelling concepts is presented in Wohlin and Rapp (2) and the implementation of an example is discussed in more detail in Wohlin (3). The main concern in this paper will be to present some results in transforming SDL system specifications into SDL descriptions describing the original SDL system from a performance viewpoint. SDL (Specification and Description Language) has been standardised by CCITT (4) as a suitable description technique for telecommunication systems. An introduction to SDL can be found in for example Belina et al (5). It shall, however, be observed that the methodology is not dependent on SDL. The methodology in itself is independent of description technique, and SDL has only been chosen as a suitable example based on experience and available tools.

This type of methodology for prototyping of software systems is new, i.e. no similar method has been found for any description technique and in particular not for SDL. A lot of work is going on concerning prototyping in general and some of it is related to SDL. The combination of software metrics and SDL for performance prototyping simulations seems, however, to be unique. The main research concerning the methodology has been performed at the Department of Communication Systems at Lund Institute of Technology, Sweden and it is presented in (1). As part of a project within the Swedish Research Programme in Information Technology (No. 4) the methodology has been refined and adapted to SDL and a tool prototype supporting the methodology has been developed. The prototype is based on the existing tool environment SDT (SDL Design Tool). SDT is described in Belina and Nilsson (6) and TeleLogic (7), and the simulator is described in more detail in Karlsson and Ek (8). The project is also partly supported by the Swedish Telecom.

This is the basis for the Performance Prototyping Simulator, whose objective is to provide a possibility to execute the system descriptions written in SDL from a performance point of view (as well as a real time functional perspective) at an early stage during the development. The methodology can be used both to verify qualities of software and systems at an early stage as well as for capacity dimensioning of telecommunication systems and services.

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.

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 our particular case SDL), while the QAM and the EDM are described with simulation models formulated in SDL. This type of simulation models have been used succesfully during several years for performance analysis. In the future it ought to be possible to generate skeletons of the QAM and the EDM respectively.

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 as well as functional before final implementation. The methodology makes it possible to do separate analysis of the software, as well as doing a system analysis, see analysis object above.

The introduction of this new concept for early system verification both from a functional viewpoint, but in particular from a performance perspective which is the main concern in this paper, requires additions to existing tool environments or a new view when developing tools. An overview of the steps and the actions when implementing this new approach to system verification is presented in figure 2.

It shall be noted that the numbers in the figure refers to different steps to go through. The steps, however, are not completely related to the order, i.e. number 1 describes the way things are done today while number 2 through 6 describes the new approach. In the latter case the numbers are in time order. The important thing to remember is that there is no horizontal time axis, i.e. the new approach 2-6 can be performed long before the current way is possible to go.

The numbers in the figure will be gone through in some more detail below. The meaning of the numbers can be summarised by;

1. Ordinary (traditional) way. This is the "normal" way in system development and it will not be discussed any further.
2. Applying transformation rules, section 3.
3. Generation of Use Process Model (UPM) from SDL, section 4.
4. Generation of skeletons for the Queue Architecture Model (QAM) and the Environment Demand Model (EDM), section 5.
5. Completion of the simulation model, UPM (optional) respectively QAM and EDM (mandatory), section 6.
6. Performance (and functional) simulations based on the new concept, section 7.

These steps will hopefully show the opportunities and the advantages with the proposed methodology for early verification of performance and functionality.

TRANSFORMATION RULES

One of the main objectives is to use the system description, in for example SDL, to do performance analysis. It is also the objective that the system description has not to be completely specified, for example tasks and decision boxes may contain informal text. The main structure of the system has to be specified and the communication between different entities, for example processes in SDL. It is also possible to formulate a model based on for example message sequence charts (MSC). Independent of the chosen level to apply this concept on it will require a transformation of the original description and metrics on for example execution times for different symbols or parts, probabilities for different paths if informal text is used in a decision box and on input intensities for different signals (i.e. the use of different services). The transformations can be divided into two separate areas; description technique and handling of the hierarchical structure (software executing on architecture). 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 (1).

SDL descriptions

The actual transformation of the original SDL system consists of several activities. 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. Both the execution times and the probabilities have to be known or estimated from the system being developed or experiences from earlier projects. As an example of a transformation we will consider the transformation of a task in an SDL process, see figure 3.

The task will remain after the transformation if it contains a functional behaviour, i.e. does not contain informal text. Independent of the content of the task a procedure call is added before the execution of the task. This procedure is denoted the delay procedure and one parameter is passed to the procedure, i.e. the delay for the task. The length of the delay has to be determined by the user of the methodology, as a first approach every symbol of the same type is assigned the same delay. The procedure is also shown in figure 3. The procedure delays the execution for the specified delay by use of the timer concept in SDL. It shall be noted that all signals are saved within the procedure, and the reason is of course that the transformation may not alter the original functional behaviour.

Hierarchical processes (software - architecture - environment)

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.

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.

At each start of a new transition an execution request procedure is called. The procedure sends an execution request to the architecture process modelling the processor that the actual SDL process shall execute on. The connection between the architecture and the SDL processes is described with a general data structure which content has to be specified by the user at the start of the simulation. This is further discussed in (1). At the end of each transition a release signal is sent to the architecture process.

GENERATION OF UPM

The transformation rules are being implemented in a tool prototype within an existing tool environment, i.e. SDT. The work constitutes of writing a translator from original SDL descriptions to new SDL descriptions, which captures the original behaviour as well as the performance aspect. The rules are implemented through additions, changes and deletions in the abstract syntax tree from the original description or through adding comments to the symbols in abstract syntax tree or trough adding SDL/PR at the end of the generated file. These three approaches will be described briefly.

The abstract syntax tree

This is perhaps the most natural way of changing the original description. It is consequently the most common used approach. In a typical case a search routine is used to find the first occurrence of a particular concept, another routine is used to add for example the procedure calls discussed above, after the addition to the tree the next occurrence of the symbol is located until all occurrences have been found.

Comments to the abstract syntax tree

The transformation of SDL into new SDL makes it possible to use the existing environment in a way that is impossible when translating to, for example, C or Ada. It is possible to benefit from the fact that the generated description can be put into the same environment as the original description, i.e. an existing analyser or code generator can be used on the second "lap", see figure 2, number 6. In our particular case this means that we can benefit from that the generated SDL is analysed when doing the second lap, i.e. we can add "comments" (in SDL/PR notation) to a node (without the comment notation in SDL) in the abstract syntax tree and then unparse the tree. This gives us a file with SDL/PR and when putting this into the analyser the text added as "comments" in the tree is interpreted as SDL/PR code, which means that we can add code in a much simpler way than creating the corresponding nodes in the abstract syntax tree. This approach is particular useful when adding several lines of code after a specific symbol in the abstract syntax tree.

Adding SDL/PR

A third possible way to do transformations is to add SDL/PR code directly to the unparsed file, i.e. the file that is generated from the complemented abstract syntax tree. This approach is particularly useful when code shall be added at the end of the file, for example the two procedures discussed above can be added in this way. The generation of skeletons for the Queue Architecture Model and the Environment Demand Model, which will be discussed next, will also be quite simple by using this method.

GENERATION OF SKELETONS FOR QAM AND EDM

First, it shall be observed, as pointed out above as well, that the Queue Architecture Model and the Environment Demand Model are described with SDL. These are, however, simulation models in another sense than the Use Process Model.

Currently no skeletons are generated for the QAM and the EDM. It will, however, be possible to generate parts of these models, in particular those parts of the models that models the interfaces between the different modelling concepts. For example, the data structure describing how the software processes are distributed in the architecture, as well as the coupling between the environment and the system (architecture and software).

COMPLETION OF THE SIMULATION MODEL

The completion of the generated model covers two main aspects, i.e. additions or changes to the generated Use Process Model and completion of the simulation models of the architecture and the environment respectively.

Changing the UPM

It shall not be necessary to change the generated UPM, but the user shall be able to do so if desired. It may happen that it is necessary to change the generated UPM due to the measurements. As an example it can be mentioned that in the example below one of the objectives was to measure the load of the different process types on the processors. This meant that it was necessary to introduce a new variable to keep track of the type of the different process instances that executed on the processor to be able to sum up the total execution time of a specific process type. Several other situations can probably occur, but the important thing is that the user is allowed to alter the generated UPM. The user is, however, not advised to change in the UPM if it is a change in the system description, on the contrary the user shall alter in the specification and the re-generate the UPM

Adding information to the simulation model of the QAM and the EDM

Today this "addition" means formulating all of the simulation models, but in the future it ought to be possible to generate the skeletons as discussed above. The formulation of the QAM and the EDM means consequently that the models have to be formulated from scratch. The user is supposed to describe the architecture and the behaviour of the environment with SDL. It is not possible to model all details in the architecture and the environment, but the important thing is to capture the main behaviour or the soul principles.

PERFORMANCE PROTOTYPING SIMULATION

The performance prototyping simulation is then performed through using the existing tool environment on the generated code, see figure 2. The usefulness of the methodology is best seen through an example. The example has been presented in more detail in (1) as well as in (3). In this paper we will only try to point out the main results from the execution of the obtained simulation model. The perhaps most important conclusion from the example is not the actual results, but the fact that it is possible to generate, complement and execute a simulation model based on this concept.

The execution of the simulation model gives two interesting results; 1) functional simulation results from a real time model and 2) performance analysis 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. Due to the delay in the architecture it may happen that a terminate signal has not reached the receiving instance before it sends a signal to the process 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.

Performance results. The results from the performance part of the simulation depend on what is specified by the user. The measurements are specified by the user of the system when complementing the generated simulation model by the Queue Architecture Model, the Environment Demand Model and some general processes that governs the simulation. The latter has not been discussed earlier, but it is necessary to have some general processes for starting the simulation and perhaps for making measurement. These type of processes are quite simple to formulate and they will not be any problem for the user.

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.

These early warnings and possibilities of changing instead of being surprised when the project fails during the test phase will soon become a necessity. The formal description techniques form the basis for methods for early verification as the one presented. It can from the example be concluded that the proposed methodology can be implemented in an existing tool set for SDL. The presentation 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 shown 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, independent of the chosen

description technique, i.e. SDL has only been used as an example.

CONCLUDING REMARKS

Formal and standardised description techniques, as SDL, provide an excellent opportunity for automatic transformations to other representations. The other representation can either be a step in the development life cycle or a special representation for evaluating one or several qualities of the system. The qualities of the systems of today is becoming a critical issue as the systems are getting larger and more complex. This means that techniques and methods for analysis of system qualities are needed to stay in control of the software system being developed.

This paper has considered how the SDL system specifications can be used for evaluating the performance of the system at an early stage. It has been discussed which parameters that have to be extracted from the descriptions and how the system specifications can be transformed into a model describing the system from a performance perspective. The method of automatic transformation of SDL into performance models for simulation is being implemented into a tool prototype.

The methodology provides a basis for;

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

These issues will become important aspects as the demands on new services and systems grow in the same time as the requirements on short lead times and higher productivity continue to grow. Part of the solution to these problems is most certainly to put more emphasis on the early phases of the system life cycle through introduction of formal techniques and methods that support different aspects of the development process. It is believed that methods for automatic translations of formal specifications into other representations will be one of the key issues to cope with the productivity and quality problems of software systems. The presented methodology provides an opportunity to tackle the problem of early verification of both performance and functionality, as well as for doing capacity dimensioning of a network.

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.

I would like to thank the following persons for valuable comments and discussions, as well as some practical help from time to time; thanks to professor Ulf Körner, Department of Communication Systems, Lund Institute of Technology, Sweden and David Rapp, Mats Löfgren, Anders Larsson and Eva Hedman at Telia Research, as well as Jan Karlsson at TeleLogic.

REFERENCES

1. Wohlin, C., 1991, "Software reliability and performance modelling for telecommunication systems", Dept. of Communication Systems, Lund Institute of Technology, Lund, Sweden, PhD thesis.
2. Wohlin, C., and Rapp, D., 1989, "Performance analysis in the early design of software", Proceedings 7th Int. Conf. on Software Engineering for Telecommunication Switching Systems, pp 114-121, Bournemouth, England.

3. Wohlin, C., 1991, "Performance analysis of SDL systems from SDL descriptions", In Ove Færgemand and Rick Reed, editors, SDL ´91: Evolving Methods, pp 353-364, Elsevier Science Publisher B.V., North Holland, Amsterdam, The Netherlands.
4. CCITT Recommendation Z.100, 1988, "Specification and Description Language SDL", Blue book, Volume X.1-X.5, 1988.
5. Belina, F., Hogrefe, D., and Sarma, A., 1991, "SDL with applications from protocol specification", Prentice Hall, United Kingdom.
6. Belina, F., and Nilsson, G., 1987, "SDT - SDL design tool", Proceedings 3rd SDL Forum, pp 8.1-8.9, Hague, The Netherlands.
7. TeleLogic, SDT2 - user manual, TeleLogic AB, Malmö, Sweden, 1991.
8. Karlsson, J., and Ek, A., 'SSI - an SDL simulation tool', In Ove Færgemand and Maria Manuela Marques, editors, SDL'89: The language at work, pp 211-218, Elsevier Science Publisher B.V., North Holland, 1989.