PERFORMANCE ANALYSIS IN THE EARLY DESIGN OF SOFTWARE

Claes Wohlin* and David Rapp**

* Lund Institute of Technology, Lund, Sweden and ** Telelogic AB, Malmö, Sweden

INTRODUCTION

This paper introduces a methodology to transform descriptions of systems into models suitable for performance analysis. It also discusses the need for metrics and measurements of different system qualities throughout the life-cycle of a system, following the ideas in Rapp and Sjödin (1). Special emphasis is put on the early design phases in the sense that most of the design-decisions are tried out before implementing the system, to ensure anticipated behaviour.

The main idea of the modelling methodology is to divide the system into two model types. One model concern the use processes of the system, which are reflected in user behaviour and application software. The other model concerns the queue structures and architecture of the system.

These two models could be combined in different ways resulting in either a so called Queue-Flow-Model (1) or a Performance Prototyping Simulator. The maturity of the models will depend on where in the life-cycle they are developed, i.e. how much information and knowledge are available.

The Queue-Flow-Model is not intended to be directly analysable with mathematical or simulation methods. Therefore refined models for the analysis have to be derived from the Queue-Flow-Model.

The methodology suggested here is based on CCITT-SDL for the description of the software. One of its major benefits besides from being a standard is its formal representation, which makes modelling and parameter extraction easier.

Parts of this work is in a early stage why further research directions are pointed out.

DEVELOPMENT PROCESS

Design properties and structure preservation

Many studies show that the introduction of errors in the early phases of the development process will cost considerable efforts to fix during the operational phase. The introduction of components and changes, that are "out of concept" regarding the original system design principles, are also defined as errors from this viewpoint. Let us e.g. assume that some system was designed with poor capacity and performance properties. Also assume that this was discovered in the later phases. Any effort to improve these properties then, can result in large costs due to the fact that system principles have to be violated.

This also explains the difficulty to take a part of a system and reuse it for a completely different purpose. A system concept is namely the derivative of the purposes of a system. A purpose may e.g. be one of its services.

It appears that major system principles are established very early in the design of a system. They become the very soul or the essence of the system. Some of these "soul-principles" are very dominant throughout the whole life-time of the system and are almost impossible to change. The violation of these principles will be the most expensive to cope with. The software developed in such cases will often be complex and thus error-prone and capacity-demanding. Such principles often concern the architecture and the structure of a system, on software as well as hardware level.

The property of some principles having a dominant impact on system behaviour and characteristics throughout the system life-time, is called principle or structure preservation, described in Zeigler (2). Although this property may imply much trouble and cost, it can be turned to our favour, when using methods for analyzing the performance of the systems in the early phases. Thus, in case we, in the early phases, have established acceptable performance properties of the main principles and architectures laid down for a system, they will be preserved throughout the life-time. Performance may, of course, have to be improved, but that may then be done in a orderly and cost-controlled way.

A structure/architecture that dominates the complete behaviour is a structural bottleneck. It differs from normal bottlenecks, which are due too slow medias and/or high usage. A structural bottleneck may often lead to usage bottlenecks. In the long run a structural bottleneck has to be removed if major improvements are to be expected.

Metrics and analysis driven development processes

From the above paragraph we draw the conclusion that we need a development process that supports the collection of metrics and perform analysis of different system properties, in this case performance. Hence, this will ensure a safer way to design systems meeting the requirements. Such an approach also favours that systems are developed by using simulation and (rapid) prototyping. The development process will then be controlled by the metrics of the system itself, and not primarily by metrics of the process, e.g. manhours, the number of documents.

The ideal situation for a designer is that he may in the early phases be able to suggest different system, software and hardware architectures, evaluate each of them, find and eliminate structural bottlenecks and then choose his "favourite". When he then proceeds to the more costly design and implementation activities he will have a good idea of the systems performance properties. Precise quantitative measures might not be known, but the qualitative aspects of the system's performance are well established.

As the development process proceeds more design decisions will be taken and thus more is known about the system. This means that the models will also be more precise qualitatively and quantitatively. When the system is ready the models will very much replicate the behaviour of the real system.

SOFTWARE DEVELOPMENT ENVIRONMENTS

The development of software has become a complex and labour intensive occupation, which calls for better and more effective environments. These should include methods, both of what to produce at different stages and how it should be done, from specification to coding as well as different types of tools. These can be divided into different groups, for example, development tools and tools for analysis of the result throughout the life cycle. The tools for analysis often depend on two important aspects, extraction of parameters and collection of software metrics.

The rules for how to determine, for example, execution time for a specific path or the load on a process, are referred to as parameter extraction. While software metrics are the values of, for example the mean time to execute one line of code. Metrics ought to be collected for all projects and stored in a database, in order to work as a memory of an earlier product. Parameter extraction and software metrics will be discussed further below.

The choice of methods is based on a number of factors, for example, its applicability to the problem, the possibilities to obtain smooth transformations between phases and the availability of tools. An important step in the development is the design, since the dynamic behaviour of the software is determined in this phase. This means that the choice of the design method is crucial. Several methods exist, for example SDL (Specification and Description Language), defined in CCITT (3), Estelle, described in Budkowski and Dembinski (4), and Lotos, discussed in Bolognesi and Brinksmaa (5), which could be applied to the design phase.

The aim, here, is to show how performance analysis could be incorporated into the design phase, and in order to do this SDL is chosen as the design method. The reasons for choosing SDL are:

- standardized by the CCITT, and especially aimed at the development of telecommunication systems.
- several tools for graphical editing of SDL are available, see for example Belina and Nilsson (6), which means that it would be possible to extract parameters for performance analysis automatically.
- methods and tools for automatic translation of SDL into both Ada, Karlsson and Månsson (7), and Pascal, Karlsson and Stavenow (8), have been presented.
- tools and methods for simulation based on SDL, (8) and Sredniawa et al (9), have been developed.
- research and tool development concerning SDL and software structure metrics, often incorrectly referred to as complexity metrics, are going on, see for example Lennselius (10).
- SDL is well-suited for performance analysis, with its process and signal concept.
- SDL has a formal representation.

This list could probably be made longer, but it is quite clear that the SDL environment already offers an attractive surrounding, and by complementing it with methods and tools for performance analysis it becomes even more favourable.

SYSTEM STRUCTURES

When modelling a system two main aspects are important: The different architectures of the system and its use processes. The architectures show how the system and its resources can be used. The Use Processes show how different use needs induce different use cases in the system. We use a layering technique so that we may in some point in time disregard certain aspects from levels below the one of interest.

Architectures of a System

The system architecture from different views are the basis for deriving models for capacity analysis. The complete system is modelled with a layered architecture, in the sense of OSI. In figure 1.a the main four levels are depicted.

- Users and uses: the users may be peoples and other systems.
- Services: Categories and functions of services in the system that are provided to the users.
- System Architecture: The architecture of nodes, communication, protocols.
- Software and Hardware Platforms: Processes, busses, operating systems etc.

Each layer has its own architecture and set of layers. How many layers and their individual architectures are dependent on the system. Sometimes a layer may be so thin that it is included into another one.

The layering technique also implies that the upper level to a certain level is in fact the environment to that level. We therefore realise that the layering technique is recursive. In figure 1.a e.g. the layers users and services are the environment (i.e. the use process) of the system architecture.The environment that is created in relation to a layer is called the Embedding Environment, discussed in (1).

In an early stage of development only a few set of layers exist. For example a conceptual model of use and services.

From a designers point of view and on each level one may identify a software structure (reflecting the use of the system and its resources) and a resource architecture. The upper service layers are often referred to as application software.

Use processes

Use Processes describe how users (in the environment) of the system make use of its capabilities.They show different transaction and work flows in the architecture induced by user needs. Use Processes are of two kinds:

- To use the system for connecting a user to other resources and users in the environment to the system. The system may also support the interaction.
- To use and interact with the resources within the system.

The first use process could describe subscribers to a telecommunication system. The second use process may describe the operation and maintenance of a system.

The use of the layering technique stresses the importance to established peer level users in order to establish to what levels different users belong to. From the point of view of a use protocol a resource within a system may be regarded as being on peer level with user originally in the environment. The observation will be important further on when we try to establish sources and sinks in the system.

MODELLING CONCEPTS

Overview

In order to perform the modelling and analysis process, four concepts are defined. The concepts are shown in figure 1.b, and they are denoted

- QAM - Queue-Architecture-Model
- UPM - Use-Process- Model
- QFM - Queue-Flow-Model
- PPS - Performance Prototyping Simulator

The concepts are independent in the sense that UPM and QAM are combined so that the result either goes into the PPS or the QFM. The PPS works with great detail on the UPMs and simple QAMs. The QFM on the other hand works with a less level of detail on the UPM and put more emphasis on the QAM. One methodology would therefore to apply the QFM in the earliest studies and when the system is rather mature. The PPS is then used when designing the actual software.

The Queue-Architecture-Model

The QAM is a model of the systems architectures from a resource and a queueing aspect. The QAM includes hardware architectures as well as parts of the system software, in most cases the operating system. What is to be included in which model is something that evolves throughout the system design process.

The parameters and architectural features of major interest that has to be extracted are:

- Topology, the distribution and connection of computers, peripherals,memories etc.

- Interconnection Devices, Busses, Networks, Protocols, Routing algorithms.
- Resources/Servers, Databases, Basic System Features
- Operating System Features, Scheduling, Priorities, Resource Allocation Schemes.

For most of the parts we need to extract parameters concerning capacity. Routing, priorities and scheduling are also of importance, since these components determine the possible Flow-Paths in the QAM.

The Use-Process-Model

The Use-Process-Model reflects aspects concerning how transactions and work will flow in the system as a results of the use of the system. For example, a user in the environment initiates a transaction, this will result in a series of events inside the system. To fulfill the function implemented by the transaction various resources will be utilized.

The UPM can be defined at different system levels. At the highest users in the environment initiates flows. At a lower level, a network of resources can be seen as a environment to a specific resource causing various internal flows in the resource. The most basic parameters in a UPM are:

- User Categories, Service and Function Types
- Usage intensities
- Transactions in terms of resource use sequences per category
- Workloads in terms of execution-times for different process use.

The UPM is in a way a model of the applications of a system. The UPM can have very different level of detail dependent on use and on available information. At the highest level of detail each instruction/symbol is tagged with an execution-time, at the lowest on a mean-execution-time is given for a complete module/process.

The Queue-Flow-Model

For the analysis of performance measures of the system and its environment we must transform its architectures and use processes into analysable models. In (2) concepts for classifying models of systems are presented. There is defined the Base Model which is the maximum model, i.e. all that is known about a system/object. The Base Model could never analyzed as such. For the analysis so called Lumped or Partial Models are used. Each Partial Model (PM) reflects an aspect of the Base Model(system). PMs can be constructed recursively, i.e. one PM may be a specialization of another PM. The PMs of interest here concerns performance and capacity properties of a system.

We define a "maximum partial" Base Model which reflects all aspects concerning performance and capacity, we call it the Queue-Flow-Model (QFM). From the QFM we then derives different models (PM) that are analysable. The notation of Queue-Flow-Models has been discussed earlier in Rapp, (11), and Eklundh and Rapp, (12).

The definition of the QFM:
The QFM covers all aspects of a system on some level concerning its service- and queueing mechanisms, transactions and workflows. The QFM is built up by the two models the Queue-Architecture-Model (QAM) and the Use-Process-Model (UPM).

Note, that when we describe Use Processes on lower system levels there is a dependence between the UPM and the QAM, i.e. Use Processes on one level will generate flows (UPM) on the next level between objects in the architecture on that level.

For practical uses we will define QFMs which are at a certain level of detail. Otherwise they will be infeasible to describe. However, since the real purpose of the QFM is to describe things in such depth that all relevant mechanism on that level are understood, the models will not be feasible for analysis, except for very special cases.

The level of detail is a modelling decision and no precise rules can be prescribed for that. Expert knowledge is needed, but it should as much as possible be built into tools so that the software designer need not worry.

The method is defined so that the UPM and QAM may mature almost independently. This is very useful in the early stages of development, when not so much is known about the final system and the development is rather iterative. Often in large system development activities around the system and processor architectures as well as application software are done by separate groups. At a certain point in time one group have to try their designs on a given release of models from another group.

Performance Prototyping Simulator

The goal of the PPS is to study the software itself, i.e. to find its bottlenecks and workloads. It will use a very simple model of the QAM. Either as a ideal machine represented by a capacity to execute each instruction/symbol or a network of ideal processors, this is discussed further below. It may be implemented by adjusting a function/flow simulator so that the time of executing a instruction/symbol can be logged.

ANALYSIS PROCESS

The process of doing analysis can be divided into several aspects that have to be considered when applying analysis techniques to a problem. The aspects described are especially aimed at performance analysis, but similar aspects can be identified for most types of analysis. Four aspects have been found:

1. The point of time for analysis.
   The analysis can be carried out at any point in the project, where it is possible to derive the necessary parameters. It should also be established that the analysis process should be repeated over and over again as the project proceeds in time. This is very important, since the degree of information about the product will increase with time, and consequently the quality of the analysis will increase.
2. Analysis object (or system) and environment.
   The part that is to be analysed is referred to as the analysis object, while the surrounding is called the environment. The analysis object is often called the system, but we have to be careful so that this system definition is not mixed up with the real system being developed. Depending on, if a part of the total real system or the whole real system is to be analysed, the picture of what the analysis object and the environment are varies. In performance analysis it is very important to identify the sources to (i.e. users of) the analysis object. Hence, the layering technique, described above, is very useful to divide the system into sources (users/clients) and servers. A source is defined as something having the initiative, the purpose to use some of the capabilities (services) of a system. When the sources and servers have been identified, it is possible to define the analysis object and the environment. This means that all sources should be placed in the environment and considered as users of the analysis object, which is equivalent to that the two use processes defined above are both considered as sources in this context.
3. Modelling methods.
   The method for modelling and analysis can vary, for example when studying performance analysis we will concentrate on simulation/prototyping and analysis of Queue-Flow-Models. The latter can be divided into both simulation and analytic methods, exact and approximate.
4. Modelling and analysis levels.
   The analysis can be carried out at different abstraction levels,

depending on assumptions and the level of reality that is taken into account when performing the analysis. For example, the analysis can be done even if the actual architecture is unknown, since it is possible to make assumptions about it and formulate model from the assumptions. This aspect of the analysis process can be divided into a number of levels, but we will primarily study three levels and they are defined below.

In figure 2 the analysis process is described. At the top the system or the analysis object is shown surrounded by the environment and the use processes. It is also shown how the service demand comes into the system and how outputs, probably, are expected. The system is divided into descriptions of software and architecture, and units are identified for them. From these two descriptions parameters are extracted to a Use-Process-Model (UPM) for the software and a Queue-Architecture-Model (QAM) for the architecture. These two are combined with a distribution of the software units in the architecture, in order to obtain either a model for the Performance Prototyping Simulator or a Queue-Flow-Model. The service demand is then input to the two different models derived. Performance results can be obtained directly from the Performance Prototyping Simulator, while the Queue-Flow-Model has to be refined further in order to obtain analysis results. This refinement is discussed further below.

MODELLING OF THE ARCHITECTURE - THE QAM

In the beginning of system design the QAM could only be known to some extent. In order to be able to study some performance properties, a series of "archetypical" or ideal QAMs have to be defined. An ideal QAM contains only the most typical features of an architecture from a performance point of view. The simplest one is only represented by a processing capacity, i.e. an item in the software takes X time units to execute. A more general ideal QAM, see figure 3.a, also contains queues, priorities and simple scheduling mechanisms. Furthermore we may want to connect several ideal QAMs in a network by ideal Communication Channel, see figure 3.b. With several ideal QAMs it is e.g. possible to study the difference in performance when allocating software to one or to several nodes.

SDL-processes do themselves contain Software Queues, due to the SAVE-instruction. Although they are included in the (Application) Software they have to be modelled into the QAM at some point in time.

As a system matures the QAM becomes more complete and often also more complex. A final QAM will be a queueing network with arbitrary complex interconnection and usage schemes.

The process of modelling the architectures of a system into QAM requires experience, only in a few cases simple rules of thumb may be given. With computer graphics, AI and such, this process will be more automated in the future. However, the other part of going from queueing models to performance analysis is already today supported by several software packages on the market.

Experience from queueing modelling shows that if one manage to model the major properties of a system, see above, the outcome of mathematical analysis or simulation give good qualitative answers. The selection of a sufficient QAM, may be done by combining ideal QAMs. In cases where a very sophisticated model is required support from experienced people is however needed.

MODELLING OF THE SOFTWARE - THE UPM

Background

One major problem when developing software systems is that no method exists for doing a performance analysis from early software design. This is surprising because a lot of the dynamic behaviour of the system is described by the software already in the early design, especially this is true with SDL. The use of the system is described at this stage, which means that the Use-Process-Model (UMP) can be extracted from the SDL descriptions. This waste of valuable information can not be afforded, and therefore a methodology, for introducing performance analysis into the early design of software, has been derived. This will give an opportunity to re-design at an early stage instead of implementing a ineffective solution.

SDL is based on an extended finite-state machine model, where the machines are described by processes and their interactions with signals. This means that by studying the processes first and then their intercommunication, a complete picture of the dynamic behaviour of the description will be obtained. The first question to arise is; What information needed for performance analysis is concealed in the SDL description? This is the first step to formulate a Use-Process-Model.

Extraction of parameters from SDL

The internal behaviour of the process depends on the signal that is received and the decisions taken in the process. This means that by studying a specific input signal, state and values on variables the actual execution in the process is determined. The set of possible combinations of execution paths, based on input signals, states and values on variables, gives us the logical flow of the process. To obtain the logical flow of the system, the logical flows of the processes have to be combined into a set of execution paths for the system.

In SDL, transitions from one state to another are assumed to take zero time. This does not, however, correspond to the implementation of the system, which means that it would be favourable if each symbol in the SDL descriptions where associated with an execution time. The logical flow and the execution times are the two most important parameters, see figure 2, that are possible to extract from the SDL descriptions.

When these two parameters have been identified as essential in the performance analysis of SDL, the next is how to extract them. In order to find the logical flow of the system, the interfaces between the system and the exterior have to be studied one by one and all different execution paths based on one specific input signal have to be found. The execution of different paths depends on the state and the values of the variables for each process. This means that the number of possible execution paths through the system soon becomes cumbersome, that is without automating the process. This is, however, possible since tools exist for editing SDL graphs, for example SDT (6), and these could be complemented with a software structure analyser. It can be used for several different purposes, and especially it can be used to find the execution paths in the SDL descriptions and to locate all decisions in the processes.

This is not enough to make it usable for performance analysis, because the execution frequency of different paths is needed too. There are two factors that have to be determined and two different ways of doing it. The two factors are:

- input intensities for different sources, that is signals, coming from the two use processes defined, i.e. external and internal signals
- the probabilities for different execution paths within the processes, for example when coming to a decision.

The two ways of doing it are:

- knowledge of the current system
- values based on experience from an earlier, probably similar, product

The second way of doing it means that software metrics have to be collected and stored in a database in order to work as a corporate memory. This could become very specific as the database grows, since the criteria for the choice could be refined as the stored knowledge grows.

The execution times are easier to find than the logical flows of the system, since it is a little bit more straightforward. Each SDL symbol has to be associated with an execution time. This is primarily obtained through the use of a database, where metrics have been collected for each symbol and classified according to the content of the symbol. It means that the current SDL symbol should be compared with similar symbols, according to a number of well-defined criteria, in order to determine the execution time.

ANALYSIS

It should be observed that the first analysis could be done before the processes have been described in detail, for example the content of some tasks has not been specified. This could be seen as a way of rapid prototyping with SDL and checking the performance of the prototype. In a first analysis like this it has, likely, to be based on mean values from the database. But as the design goes on and the knowledge of the system being developed grows, the analysis could be done over and over again with more and more realistic values. Databases for software metrics will probably become an important tool in the future software development environment.

The research has been directed towards several promising areas for analysis, which also will be further investigated, as:

- Performance prototyping simulations of SDL.
  The studies aim is to do performance simulations direct from the SDL graphs, assumptions about the architecture, and the methods mentioned above concerning some of the parameter extractions and software metrics, primarily execution times and input intensities. This is referred to as Performance Prototyping Simulator in figure 2. This type of simulations should not be mixed up with functional simulations of SDL or performance simulations, where SDL is used to describe the simulation problem, see for example SDL/SIM (8) or (9).
- Implementation of the performance simulations into existing SDL tools.
  SDL/SIM has, however, been used as the basis for the performance simulations of SDL. It should be observed that SDL/SIM generates a large part of the simulation program, and that the concept has been complemented with functions in order to handle the simulation of SDL itself.
- Prototyping with SDL.
  The possibilities of describing the structure of the problem to be solved, rather than specifying the details in the tasks, and evaluate the performance of the solution are investigated. The goal of this research is to determine the quality of these type of descriptions and studies.
- Formulation of Queue-Flow-Models.
  The problems of formulating both Use-Process-Models from the software and Queue-Flow-Models of the system are studied.
- Solution of Queue-Flow-Models.
  The QFM is the basis for further modelling by simplification. In some cases it might be possible to design a simulation model directly from the QFM. What we may face as a result though, is a to complex model that takes too long time and too much memory to complete its computations. This will aggravate measurements with sufficient confidence. Mathematical modelling offers a restricted set of tools. The analysis quickly becomes fairly complex, and detailed analysis of simple system leads to equations that are numerically difficult to evaluate in a computer.

  For that reason it is important to simplify the QFM in an "intelligent" manner into analysable models. This can even be different models depending on what the aim of the model is. A whole family of analysable models could probably be derived from one Queue-Flow-Model, depending on the information wanted, the needed certainty of the result and the analysis method to apply. Different methods to analyse the obtained methods are studied, that is both exact and approximate analytic methods as well as simulations. In order to do the analysis the possibilities of generating a QNAP2-program, Véran and Potier (13), from the derived analysable models, are studied. QNAP2 is a tool for automatic analysis of queueing models, and it applies suitable methods for analysis depending on the queueing model to be analysed.

These areas are all aimed at a number of different levels of analysis of the software, which will be discussed further below.

MODELLING AND ANALYSIS LEVEL

The model obtained from the extraction process can be used for analysis on several different levels, independent on if the analysis should be made by the performance prototyping simulator or by use of Queue-Flow-Models. The discussion below will be based on the following three assumptions:

- parameters have been extracted so that the SDL-system can be modelled on at least one of the following levels; symbol, process or function.
- software metrics are available, from a database or knowledge of the current system.
- the derived model is valid, that is it models the developed SDL description and not the SDL description that ought to have been developed.

The third assumption calls for some explanation. It is obvious that with unbounded queues, as in SDL, there is always a possibility of getting into an unstable state, which makes the queues grow towards infinity. These situations could be found by analysis, if the model is valid for it, but if an erroneous situation occurs which has not been modelled then the model will fail. This means that an SDL-process becomes overloaded and this has not been found when extracting the parameters, then the analysis will fail. It is no surprise, since the model is based on the design we think we have designed, and not the actual solution. This type of errors has to be found as soon as possible, in order to avoid the error to spread into the phases to come.

This type of errors could be found by prototyping, which means that they could be found by doing the performance simulations of SDL discussed above. The difference between performance simulations of SDL and the analysis of the Queue-Flow-Model should be observed, see figure 2. The latter is based on analysis of the derived model, while the simulation of SDL is applied on the description itself. They do, however, have things in common, since the SDL simulation needs execution times and input intensities. This implies that a design error which only affect the communication between processes can be found by the performance simulations of SDL.

When it has been established that the derived model is valid, all necessary parameters have been derived and software metrics are available, then several different levels for analysis can be considered.

The analysis can be done on several different abstraction levels as discussed when introducing the QAM. The following three levels are an example on possible analysis levels:

1. Infinite number of servers with the same capacity. (Ideal processors)
   This analysis will give us a possibility to study the behaviour of the different SDL-processes. We will obtain a way of identifying the bottleneck process, i.e. the software bottleneck. This is a candidate to be the system bottleneck, but it can not be evaluated finally until we combine the software with the real architecture. It should be observed that we have three types of bottlenecks in this concept; software, architecture and system. The software bottleneck could be a way to determine if re-design of some processes are necessary or if they have to be taken care of especially when implementing the software. It will give us a hint of how the processes should be distributed in the architecture, or give an idea of how the architecture should look like.
2. The right number of servers with the same capacity.
   This analysis will give us a way to test the chosen distribution of processes among the processors in the architecture. It will

give us a possibility to see if the distribution of processes seems sensible and if the architecture at all can perform its services.

3. Taking the real architecture into consideration.
   At this level the real architecture is taken into consideration, which means that the actual capacities of the processors and the routing algorithm are taken into account.

These three levels will give us the opportunity to study different aspects of the system at different times, and by isolation concentrate on some aspects at the time. Methods and tools supporting this approach of performance analysis from SDL are being developed.

ANALYSIS METHODOLOGY

The methodology to introduce performance analysis to early software design, especially in an SDL environment is summarized below. The difference between the analysis methodology respectively process should be observed. The methodology defines a number of steps that can be taken in the order described below, while the process points out a number of aspects that have to be considered. The steps in the methodology belongs to one of the aspects discussed in the process.

1. Determine a suitable time for analysis.
2. Identify sources and servers in the system.
3. Identify analysis object (system), environment and use processes.
4. Supply metrics to the intensities of the use processes.
5. Identify decisions in the analysis object.
6. Supply metrics to the probabilities for different decisions.
7. Supply metrics, that is execution times, for the different SDL symbols.
8. Derive a Queue-Architecture-Model for the architecture.
9. Determine if the analysis should be carried out by use of the performance prototyping simulator (goto 15) or Queue-Flow-Models.
10. If the analysis should be done with Queue-Flow-Models, some more parameters have to be derived, for example blocking probabilities for different signals that are not saved in the SDL-process.
11. Derive a Use-Process-Model.
12. Combine the Use-Process-Model with the Queue-Architecture-Model into a Queue-Flow-Model, that is decide the analysis level.
13. Derive suitable analysable models from the Queue-Flow-Model.
14. Analyse the derived model with analytic methods, exact or approximate, or simulate. Goto 18.
15. Derive a Use-Process-Model.
16. Combine the Use-Process-Model with the Queue-Architecture-Model into a model for the Performance Prototyping Simulator, that is decide the analysis level.
17. Execute the derived model with the Performance Prototyping Simulator.
18. Interpret the results obtained and draw conclusions.
19. Decide suitable actions from the conclusions, for example if re-design of some SDL-processes or faster processor is needed. The analysis should, of course, be carried out after each action, otherwise it is impossible to know what the consequence of the action is.
20. Goto 1, it should, however, be observed that only steps that have been influenced by more knowledge have to be carried out when the analysis is performed at new suitable points of time.

This methodology has to be supported by tools, for example graphical editor for SDL (such as SDT), Software Structure Analyser, Performance Prototyping Simulator, program generator for QNAP2-programs and QNAP2.

A SIMPLE EXAMPLE

In order to explain the analysis methodology a little better, a simply example will be described. The example will be based on two very simple SDL-processes, see figure 4. The processes are shown both as black boxes within blocks and separate processes. These do not implement anything and should only be seen as an example to explain the methodology. The example will be gone through according to the four aspects in the analysis process described above, and we will refer to the appropriate steps in the methodology.

1. It has been decided that it is a suitable time for analysis. (Step 1)
2. The analysis object has been identified as being the two processes, in figure 4. The environment communicates with the system by the signals I1, I3 and I4, while the system sends I5 to the environment. (Step 2,3 and 5)
3. It is decided that the analysis should be done by using the performance prototyping simulator. (Step 9)
4. In figure 5 two different possibilities to distribute the processes in the architecture are shown. The analysis can be done on both of these architectures. The parameters for the two processes have to be extracted and associated with metrics. (Step 4,6,7,8 and 15)
   (a) The input intensities for I1, I3 and I4 are decided.
   (b) The decision box, D11, has to be associated with a probability for going to the right or the left.
   (c) Each box, in the graphs, has to be associated with an execution time.

   The values on the parameters extracted have to be put into the performance prototyping simulator to obtain analysis results. (Step 16,17,18,19 and 20)

Comment: If we had chosen to use the Queue-Flow-Model method, some more modelling would have been needed for the processes. This would, however, pay off later, since the analysis will become much quicker, especially this is true if the obtained Queue-Flow-Model could be turned into a model that can be solved analytically. Unfortunately, it is impossible to present the details, here, of how the processes can be turned into a Use-Process-Model and how this can be used to formulate a Queue-Flow-Model, together with a Queue-Architecture-Model. This is, however, shown in Wohlin (14).

CONCLUSIONS

A methodology for introducing performance analysis in the early design of software has been suggested. It has been discussed how the information from the software, especially SDL, and the architecture can be extracted and turned into models. These models can be combined into models for analysis, both for prototyping and queueing theory analysis.

A basis has been presented for the methodology, i.e. several processes influencing the performance analysis has been defined. A number of models has been introduced in order to formalize the analysis process. This framework was then used for describing how modelling of the architecture and software can be performed. Several research areas concerning aspects on modelling and analysis have been described. The main research so far has been concentrated on how to extract the parameters and turn them into a Use-Process-Model. The research is still going on and is to be implemented into an SDL environment. Areas for further work, on the implementation of the performance prototyping simulator and on the aspects concerning the Queue-Flow-Model approach, have been presented.

Some tools are available today, while others are on the prototype stage, and more tools will be developed as the research continues and the methodology is introduced into an industrial environment. The methodology should, in most cases, not need an expert in queueing theory to be applicable.

It has been emphasized that a methodology for performance analysis at an early stage is necessary in order to cope with the design of complex systems, i.e. in order to re-design instead of implementing an ineffective solution.

REFERENCES

1. Rapp, D., and Sjödin, G., 1983, "Capacity models: a quality and a system design tool, and an aspect of systems", Proc. IEE, 223, 128-135.
2. Zeigler, B.P., 1976, Theory of modelling and simulation, John Wiley and Sons Inc, New York.
3. CCITT, 1984 Recommendations Z.100-Z.104 and annexes.
4. Budkowski, S., and Dembinski, P., 1987, "An introduction to Estelle: A specification language for distributed systems", Computer Networks and ISDN Systems, 14, 3-23.
5. Bolognesi, T., and Brinksmaa, E., 1987, "Introduction to the ISO specification language Lotos", Computer Networks and ISDN Systems, 14, 25-59.
6. Belina, F., and Nilsson, G., 1987, "SDT: SDL design tool", Proc. Third SDL Forum, 8.1-8.9.
7. Karlsson, J., and Månsson, L., 1987, "Using SDL as specification and design language and Ada as implementation language", Proc. Third SDL Forum, 31.1-31.13.
8. Karlsson, J., and Stavenow, B., 1987, "SDL/SIM, a simulation system for discrete event simulation", Technical report, Dept. of Communication Systems, Lund Institute of Technology, Lund, Sweden.
9. Sredniawa, M., Kakol, B., and Gumulinski, G., 1987, "SDL in performance evaluation", Proc. Third SDL Forum, 21.1-21.11.
10. Lennselius, B., 1986, "Software complexity and its impact on different software handling processes", Proc. IEE, 259, 148-153.
11. Rapp, D., 1980, "Capacity analysis of SPC systems - the AXE system a case study", Proc. NTS 3.
12. Eklundh, B., and Rapp, D., 1982, "Capacity study of the AXE 10 processor system", Ericsson Review, 4 208-216.
13. Véran, M., and Potier, D., 1984, "QNAP2: A portable environment for queueing systems modelling", Technical report, Inria, France.
14. Wohlin, C., 1989, "Turning SDL-processes and architecture into a queue-flow-model", Technical report, Dept. of Communication Systems, Lund Institute of Technology, Lund, Sweden (to be written).

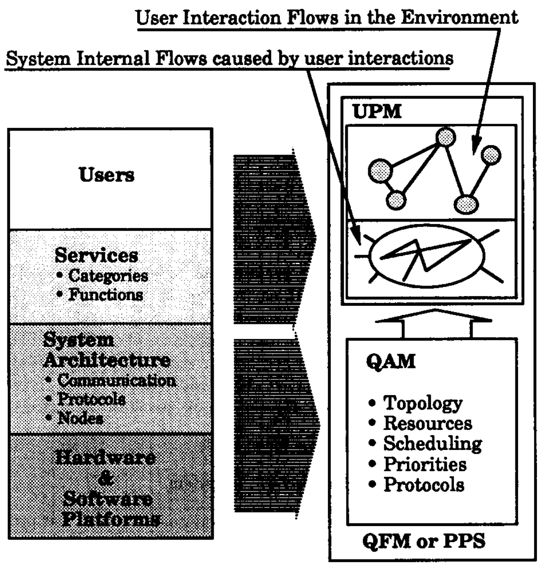

Figure 1a A layered architecture of the Users and the System
Figure 1 b The main Modelling Concepts

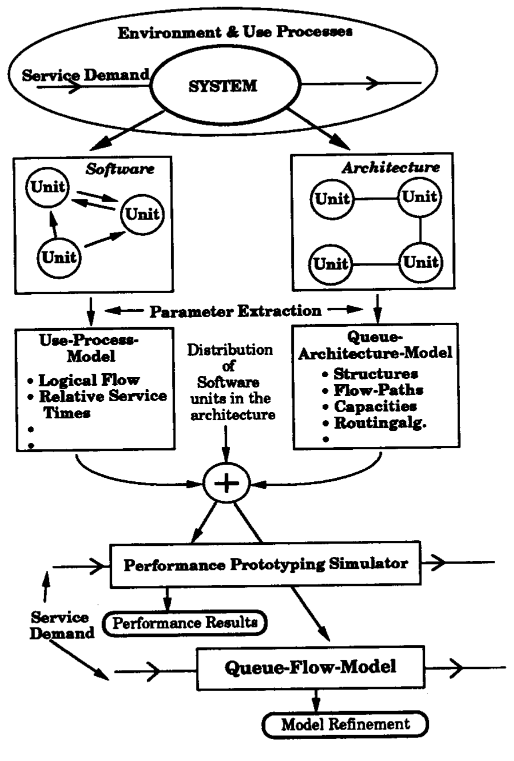

Figure 2 The Modelling Process

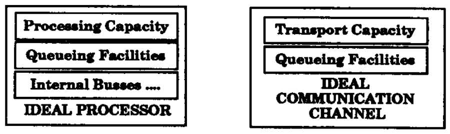

Figure 3 Ideal Components for the Queue-Architecture-Model

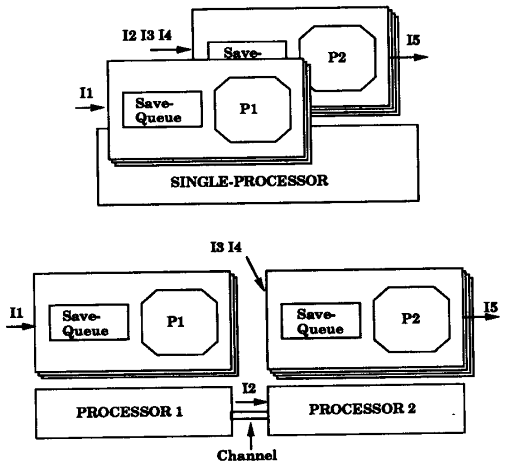

Figure 5 The allocation of the example on ideal QAMs

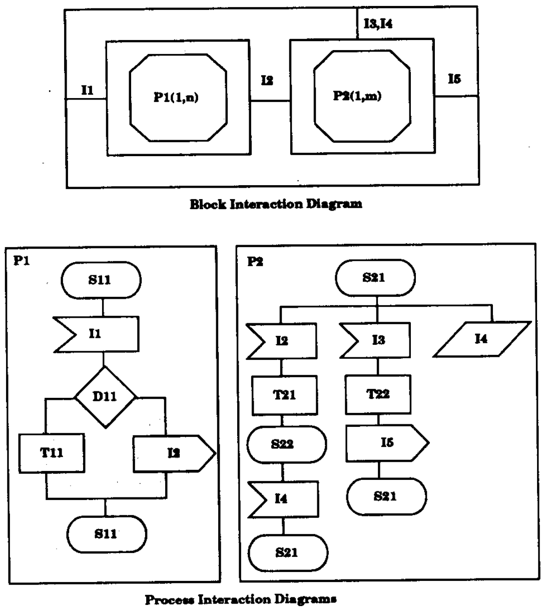

Figure 4 A simple example of two processes