C. Wohlin, C. Nyberg and A. Larsson, "Reusable Simulation Models for Performance Analysis of Intelligent Networks", pp. 143-153, in "Intelligent Networks and New Technologies" edited by Villy B. Iversen and Jørgen Nørgaard, Chapman & Hall, London, UK, 1996.

Reusable Simulation Models for Performance Analysis of Intelligent Networks

Claes Wohlin[*], Christian Nyberg[*] and Anders Larsson[**]

[*] Dept. of Communication Systems
Lund Institute of Technology
Lund University
Box 118
S-221 00 Lund
Sweden
Phone: +46-46-222 0000
Fax: +46-46-145823
e-mail: (claesw, cn)@tts.lth.se

[**] Telia Research AB
Box 85
S-201 XX Malmö
Sweden
Phone: +46-40-105034
Fax: +46-40-105100
e-mail:aln@malmo.trab.se

Abstract

New services will be introduced at a much faster rate than today when Intelligent Networks are employed. It is important to control the process of introducing services and be able to forecast the consequences of a new service. This paper describes a method for performance simulation based on reusable simulation models which can be used for this purpose. The method builds on dividing the simulation model into three parts describing architecture, software of services and usage of the services respectively. Services and nodes can be described on different abstraction levels. If a new service is introduced all the old parts of the simulation program can be reused.

1.0 Introduction

The fast introduction of new services in the telecommunication networks leads to a number of problems. One of them is the problem of network performance as new services are introduced. The introduction of services must be made in a controlled manner, hence an analysis of the effect of the introduction of new services must be made prior to putting them into operation.

This paper presents a simulation method, which can evolve together with the network through reuse of simulation models. Thus allowing for performance analysis without having to formulate new models each time a new service is to be introduced.

A state of the art study of available simulation software for communication networks is presented by Law (1). There are some useful software packages available, but they do not support reusable models for performance simulation. Some tools include a software library, but they do not include support to reuse the models created by the user of the tools. This is also concluded by Saulnier (2), where they state that: “Simulation packages do not provide an effective infrastructure for reusing models …”.

2.0 Performance analysis when new services are introduced

The reusability of software, the centralization of service control and efficient design tools will make it possible to introduce new services in Intelligent Networks at a much faster rate than in traditional networks. This process must be properly controlled. A new service increases the load on the control systems of the Service Switching Points (SSP), the Service Control Points (SCP), the Signal Transfer Points (STP) and the signalling links. As a consequence of this, bottlenecks may be formed and the waiting times may get too long for both the new service and for the old services. Thus, before a new service is introduced, we ought to evaluate its consequences on the performance of the network. If we find that the load on certain nodes or links gets too high we can move services to other nodes to equalize the load, install new capacity in the network or perhaps even choose not to introduce a service.

A tool which evaluates the load on the control processors in the network and the delays experienced by the users can help us to answer questions like: where are the bottlenecks in the network and what is the highest allowed arrival rate of a new service. The input to such a tool would be the rate at which a service will be used, a description of the new service, a description of the present network and its services. The description of the new service could be on a high abstraction level to permit us to evaluate it even before it has been designed in full detail. But it should also be possible to use the detailed description. The services already present in the network may be described on a high abstraction level. It should also be possible to use different abstraction levels in the description of the nodes in the network just as well as for the services.

3.0 Performance simulation

3.1 Introduction

Simulation is a valuable tool for performance analysis, but it has some serious drawbacks, which must be overcome to make it really useful in an application which evolves rapidly, for example, with introduction of new services. Some of the major drawbacks are:

- the simulation model is based on one particular system or network, hence as the network or the available services changes the model must be changed as well. This can be hard as the model is not normally created to be changed. To overcome this problem, reuse of models must be considered to a much larger extent.
- the model used for performance simulation does not include the actual software description, hence relying on queueing models with little or no connection to the actual software. This becomes particularly crucial as new services are introduced and the delays in the network depend highly on the implementation of the service. The rapid introduction of new services does not allow for creation of good models of the services, instead the software design descriptions ought to be executed in the simulation model.

These two problems are normally not fully solved by available simulation methods and tools. Most available tools, for example, QNAP2 in Verán (3) and QNA in Whitt (4), use their own specific representation languages and they do not support inclusion of software descriptions as part of the simulation model. The objective of our approach was to include the software descriptions as part of the simulation model and that the simulation model should be formulated in the same description language as is used in software development. This objective was the major goal when developing a new approach to performance simulation, see Wohlin (5). Thus, this approach is adopted here. A summary of this work can be found in Wohlin (6). A brief summary of the concept is given subsequently.

3.2 A new approach to performance simulation

The new concept is based on the idea of dividing the simulation model into a number of models, which describe different constituents of the performance model. This objective is very similar to the one adopted in IN, i.e. a layering approach, hence it is believed that the proposed concept will be a valuable asset in performance simulation of IN.

Three models are defined to formalise the modelling and evaluation process. The mapping between reality and models is depicted in Figure 1. The models are denoted:

- Software Model
- Usage Model
- Architecture Model

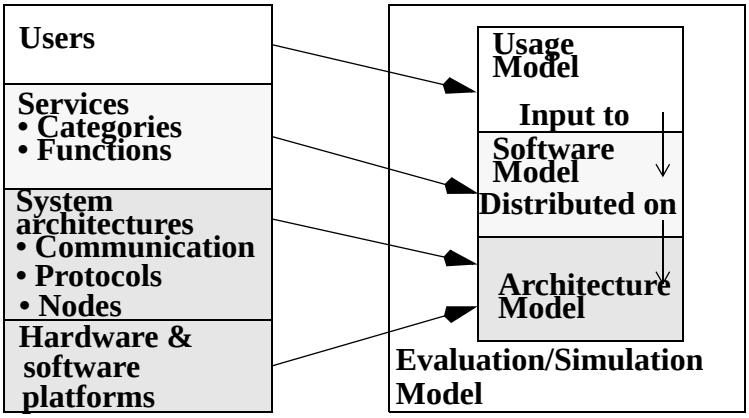

FIGURE 1. Mapping of the layers of users and system on the modelling concepts

The models, which will be explained below, are independent in the sense that the Software Model, the Usage Model and the Architecture Model are derived independently and they can be combined into a simulation model. The simulation is foremost intended to be used during the software design and hence supports re-design decisions. The Software Model and the Architecture Model are linked to each other through the distribution of the software units in the architecture. This means the Software Model is allocated to the Architecture Model in a way that describes the actual distribution of the software in the architecture. The Usage Model generates the input to the simulated system (Software Model allocated on the Architecture Model). Thus connecting the three models together into the Evaluation/Simulation Model.

3.3 Model descriptions

The models can be described as follows:

- Software Model

  The software descriptions (specification or design) are transformed to include the real time aspects of the software, which normally is not included in the software design. As part of the transformation the user is requested to add time consumption for executing different concepts of the software design. This addition of time consumption must be made based on prior knowledge or knowledge of the current system. The Software Model describes the application software, i.e. the features that the system provides to the user. The transformation can be made automatically based on a set of well-defined rules, see Wohlin (5). The Software Model includes aspects such as services and features (what is available to the external user), logical flow (the coupling between services) and relative service times (an execution time which does not refer to a particular platform).
- Usage Model

  The Usage Model describes the structural usage of the analysis object (which refers to the part of the system being evaluated) and it is complemented with usage intensities for different services. This type of model is similar to the ones used in certification of software reliability as discussed in, for example by Wohlin (7) and Whittaker (8). Therefore, it is believed that the concept originally proposed for performance simulation can be enlarged to incorporate reliability analysis as well. The Usage

Model is concerned with identification of analysis object (which part of the system, i.e. hardware and software, is going to be analysed), user categories, usage states (the external observable states that a user may be in) and usage intensities. It must also be noted that the usage model describes the usage of the analysis object for all types of users, which includes humans as well as other systems or networks.

- Architecture Model

  The architecture is described in terms of a performance simulation model. The aspects to find are those governing the performance behaviour of the architecture. The objective here is to define a performance model of the architecture in the same description technique as has been used in the software design. This is also further discussed by Wohlin (5). The main rationale behind using the same language is the opportunity to easily map the software design descriptions onto a simulation model of the architecture. The Architecture Model includes aspects such as network architecture with interconnections, servers and resources, as well as operating system features, algorithms, for example scheduling, flow-paths and capacity.

3.4 Performance simulation using SDL

SDL is primarily intended as a specification and design language, but it has been and still is successfully used in performance simulation. SDL has been used for the latter purpose in a number of years, for example, at the department of Communication Systems, Lund University in a course on performance simulation and it is also used in the research at the department.

A major benefit of using SDL is that it is standardized and tool support is available. A particular benefit in our approach to performance simulation is the use of the same description technique for describing the services and for formulating performance models of the architecture and the user behaviour, which simplifies the work considerably. The use of one language for two purposes is not a prerequisite for the proposed simulation method, but it makes the combination of service descriptions with simulation models easier.

4.0 Performance simulation of IN

4.1 Introduction

The previous section described an approach to performance simulation that uses a software, a usage and an architecture model. To be able to use this as a basis for a performance simulation method, that solves the problems when introducing new services, it is desirable that is should be possible to handle models on different abstraction levels. For example, old services can be described only by their service times on the different nodes in the network - perhaps found by measurements on the real network - and new services by a complete software description. Thus we run into several problems: what abstraction levels can be used in the software, usage and architecture models and which of these can be used at the same time. This also rises the question of interfaces. Suppose that two different abstraction levels for software are used at the same time. Is there an interface between the different parts of the complete simulation model that can handle this? Which abstraction levels are compatible with each other? And can an

interface be designed that allows us to change the abstraction level of one part of the model without having to change the rest of the model?

4.2 Abstraction levels of models

The architecture model describes the nodes and signalling links of the network. The most abstract model of a node that seems to be meaningful is a queueing system and the least abstract model is a complete emulation of the hardware and the operating system.

The most abstract level of the software model is to describe a service by its service time on the nodes in the network. This level can be used for services that are already in use or for services that are not yet completely specified. It shall also be possible to use a complete software description of a service in the simulation program.

The usage model describes the environment of the network and should reflect both the behaviour of subscribers and the parts of the network that are not incorporated in the model. A usage description must receive all signals sent to it by its corresponding service description and react to the signals in a proper way. Thus, the abstraction levels of a service description and its usage description must be on the same abstraction level.

4.3 The interface between architecture model and the other models

As pointed out previously the interfaces between the different parts of the a model must be able to handle software, usage and architecture models on different abstraction levels. It must also be possible to change the abstraction level of a node or service description without having to change the rest of the model. This has one notable exception: when a service is changed its usage model may have to be changed.

To be able to change the abstraction level of a service without having to change the architecture model, the model is designed so that it does not need to know anything about a service except to which nodes it is allocated. Thus the signals used in the software description of a service must be packed into signals that can be interpreted by the architecture model. These signals do not exist in the original software. The discussion above indicates that the following three classes of signals are needed in the simulation model:

1. Signals that contain original signals. A node which receives such a signal uses the software distribution data to check if the software is allocated on itself in which case it sends the signal to the software or usage model. Otherwise it sends the signal to another node according to the routing table.
2. Signals exchanged between the software and architecture model that are used to mark a processor as free or busy. When the software model receives a signal of class 1 from the architecture model it computes the processing time of the jobs that is triggered by the signal. The software model executes according to the processing time and informs the processor when the execution is completed.
3. Signals internal to the architecture model, for example signals between processors in distributed systems.

5.0 Example

5.1 The new service

To exemplify the methodology presented in the previous sections we have specified a new service which we intend to introduce in an Intelligent Network. The specification is a specification of the functional behaviour of the service, i.e. what is normally implemented in software. This new service is divided into two parts, SWA and SWB, corresponding to the Service Switching Function and the Service Control Function.

The service is invoked when a user sends a signal A to SWA. Upon receipt of this signal SWA sends a signal B to SWB. SWB performs some action and returns signal C to SWA. Finally a signal D is sent to the user, see Figure 2.

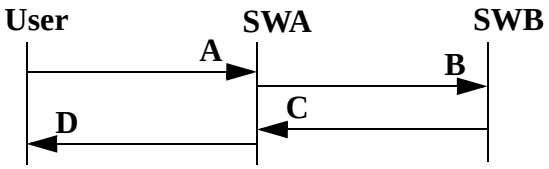

FIGURE 2. Message Sequence Chart for the new service. The signals A-D are "original signals" according to the terminology presented in the previous section.

Our new service is going to be installed in a simple Intelligent Network consisting of two SSPs and one SCP interconnected with #7-links. SWA should be installed at both SSP1 and SSP2 whereas SWB should be installed in the SCP.

5.2 The simulation model

The modelling concept states that the model should be divided into three parts: the Usage Model (UM), the Architecture Model (AM) and the Software Model (SM), see Figure 3.

- UM: Models of the users of the service.
- AM: Models of the SSPs, the SCP and the #7-links.
- SM: Model of the new service, i.e. transformed models of SWA and SWB, and models of existing services in the Intelligent Network.

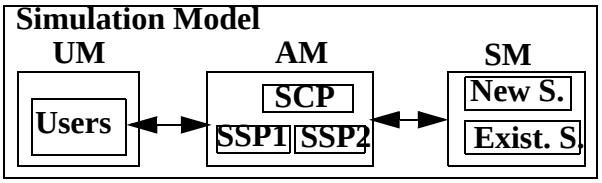

FIGURE 3. Overview of he simulation model. (Not all details are shown.)

This simulation model has been implemented in SDL, to allow for evaluation of the usefulness of the proposed modelling concepts. It was concluded that the approach is useful and that it is worth investigating further.

The transformed models of SWA and SWB are contained in the block New Service. SWA exists in two instances, one allocated to SSP1 the other to SSP2. SWB exists in

one instance only and is allocated to the SCP. Original signals that are to be sent between software instances or between user instances and software instances are packed in a general signal SWSignal and are always sent via the Architecture Model. If the sender and the receiver of an SWSignal are allocated to different processors, for example if the sender is a software instance allocated to one of the SSPs and the receiver is a software instance allocated to the SCP, it has to be sent via a #7-link. In such cases the SWSignal is packed in a signal #7Packet and sent to the model of the #7-link which forwards the #7Packet to the receiving processor.

As an example we will look at what happens in the simulation model when SWA sends the signal B to SWB, see Figure 4. The signal B is packed in an SWSignal which is then sent to the processor on which SWA is executing, for example SSP1. SSP1 realizes that the receiver of this signal is allocated to the SCP so the signal must be sent via a #7-link to the SCP. The SWSignal is packed in a #7Packet and is sent to the model of the #7-link where it is delayed a time corresponding to the transmission time for this packet. Then the #7Packet is forwarded to the SCP where the SWSignal is unpacked and sent to the correct receiver, SWB. Before starting to execute SWB must have permission from the SCP to do so, this is accomplished with the signal Execute. Then SWB starts to execute. Apart from the logic already defined in the functional specification (e.g. the creation and sending of signal C) a delay is introduced corresponding to the time it will take to execute this transition on the SCP-processor. When this time expires SWB sends the signal ExecutionCompleted to the SCP.

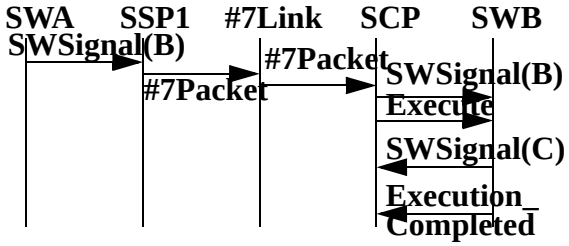

FIGURE 4. Message Sequence Chart, sending of signal B from SWA to SWB.

With this simulation model it is possible to study different performance aspects in the Intelligent Network such as the load on the SSPs, the SCP and the #7-links. The response time from a user´s point of view can also be measured.

This simulation model can be reused if we for example want to introduce a new service. Such a service can be added in the Software Model without affecting the other services or the Architecture Model.

6.0 Long-term research objective

The long-term objective is to be able to work with performance models and service descriptions in SDL in at least three abstraction levels, see bullets below. The first level is:

- Software model and usage model without taking the architecture into account

This type of analysis shall be used to identify software processes which either have long execution time or are called extremely frequently. The software processes having this characteristic ought to be further studied and perhaps even re-designed, since they probably will cause problems as they are introduced into the network.

In the long run the objective is to be able to handle different levels of abstraction of the models, i.e. several models on different abstraction levels of for example an SCP may exist. This means that it is possible to combine detailed models for some parts while others are described quite superficially. The major benefit of this is that it is possible to model, for example, critical nodes in the network in detail while other parts which are not critical for this particular simulation may be modelled on a much higher level.

Based on the above two other abstraction levels are identified, both of them include different levels of abstraction of the software and usage models. The abstraction levels of the software and usage models are: treating the software as an execution time based on measurements on existing services or taking the software specification in SDL directly. The two abstraction levels including different levels of abstraction of the architecture are:

- Simple architecture model

  The objective of the simple model is to identify the bottlenecks in the architecture, hence being able to identify parts which ought to be focused upon from a performance perspective.
- Focused architecture model

  The focused model means having a quite detailed model of some part of the architecture, primarily, of course, some part which has been identified as being problematic when analysing the simple architecture model.

The ability to handle different levels of abstraction for different model parts is essential as it allows for simulation studies focused on for example network bottlenecks or a specific problematic service.

7.0 Conclusions

A method based on reusable model components has been described. The primary objective is to formulate a method which can be used to analyse the intelligent network from a performance perspective as new services are introduced into the network. The method is based on three modelling concepts and their interaction. The models are: software model (service description transformed to incorporate execution time), architecture model and usage model.

The method provides a basis for:

- identifying software modules which will use too much execution time, due to long service times or frequent use. The early identification of these modules allows for re-design instead of identification of the problem as the service is put into operation.,
- studying the introduction of new services in an existing system (network),
- evaluating different distributions of software processes in an architecture,

- examining the ability of different architectures to execute a given software description,
- identifying system bottlenecks.

These possibilities are then further enlarged as the proposed method is supposed to cover analysis of reliability and feature interaction as well in the long run. Hence, the method supports:

- certification of reliability requirements
- identification of the most likely feature interactions

The latter two points are both based on the fact that users get more annoyed if frequently used services and features fail or interact in a way which is disturbing to the user, than if seldom used services do not work in accordance with expectation.

In summary, the method proposed will enable analysis and prediction of several critical quality attributes prior to introducing new services into a telecommunication network. Thus continuous control of the network and its services is supported.

References

1. Law, A. M. and McComas, M. G., “Simulation Software for Communication Networks: State of the Art”, IEEE Communications Magazine, March 1994, pp. 44- 50, 1994.
2. Saulnier, E. T. and Bortscheller, B. J., “Simulation Model Reusability”, IEEE Communications Magazine, March 1994, pp. 64-69, 1994.
3. Verán, M. and Poiter, D., “QNAP2: A Portable Environment for Queueing Systems Modelling”, Technical Report, INRIA, France, 1984.
4. Whitt, W., “The Queueing Network Analyzer”, The Bell System Technical Journal, November, pp. 2779-2843, 1983.
5. Wohlin, C., “Software Reliability and Performance Modelling of Telecommunication Systems”, Dept. of Communication Systems, Lund University, Report no. 106, Ph.D. thesis, 1991.
6. Wohlin, C., “Evaluation of Software Quality Attributes during Software Design”, Informatica, Vol. 18, No. 1, pp. 55-70, 1994.
7. Wohlin, C. and Runeson, P., “Certification of Software Components”, IEEE Transactions on Software Engineering, Vol. 20, No. 6, pp. 494-499, 1994.
8. Whittaker, J. A. and Poore, J. H., “Markov Analysis of Software Specifications”, ACM Transactions on Software Engineering Methodology, Vol. 2, No. 1, pp. 93–106, 1994.