Proceedings of the 1994 Winter Simulation Conference
ed. J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila

SIMULATION OF HETEROGENEOUS NETWORKS

Markus Rümekasten

University of Paderborn
Dept. of Math. & Computer Science
33095 Paderborn, Germany
E-mail: rueme@uni-paderborn.de

ABSTRACT

This paper presents a simulation tool to help the network designer to identify the optimum network design parameters for a heterogeneous network environment. The tool consists of two parts: simulation modules and a user interface. The simulation modules are created using enhancements of an asynchronous, event-driven C-function library to enable the simulation of heterogeneous scenarios. In the future, this modules will be distributed to speed-up the simulation runtime. The user interface helps the user to create the design of the network to be simulated. By this modular concept of distributed simulation modules and user interface the optimum network can be found out without much knowledge of the underlying simulation details.

1 INTRODUCTION

The optimum network design is mainly a matter of determining network parameters. Buffers have to be dimensioned, timer values have to be defined and many other parameters have to be chosen carefully not to decrease the performance of the network. For a heterogeneous scenario the choice of correct parameters becomes even more important and much more difficult because of the different mechanisms in each of the interconnected networks.

Performance analysis provides methods to determine appropriate values for the design parameters but has a disadvantage: most tools and methods for performance analysis can be used only by "experts" of a special tool or mathematical method. The most flexible method to analyze network performance is simulation: the network is modeled in a computer program that calculates the output parameters like packet loss or throughput based on input parameters like load conditions or the number of nodes.

To help the network designer using performance analysis based on simulations, a tool should simplify the network design even for a complex interconnected scenario. This tool should be as flexible as the simulation program itself but much more easy to use for a network designer who has a good knowledge of networks but only little (or even no) knowledge of simulation details and the principles of the parallel programming of the modules and their synchronization.

This paper presents such a "non-expert" performance analysis tool based on simulation currently under development at the University of Paderborn.

The tool "SimCom" (Simulator for Communication systems) consists of two main parts:

1) simulation modules for the performance analysis (described in section 2) and
2) a user interface to determine the input parameters for the simulation modules (described in section 3).

In the future, the simulation modules will be programmed in a parallel way to reduce the runtime for complex scenarios. Therefore, efficient synchronization methods have to be introduced in SimCom.

Using this modular structure of independent simulation modules, it becomes easy to create a library of simulation modules. The library includes the simulation modules for different networks and IWUs (Interworking Units). The IWUs are special nodes in the networks for the interconnection of different networks in the heterogeneous network scenario to be simulated. The interactive user interface can be used to determine easily the structure of the heterogeneous network and its parameters for the analysis of the chosen scenario. After determining the input parameters with the user interface, the interface can start the simulation modules in order to evaluate the performance of the heterogeneous scenario.

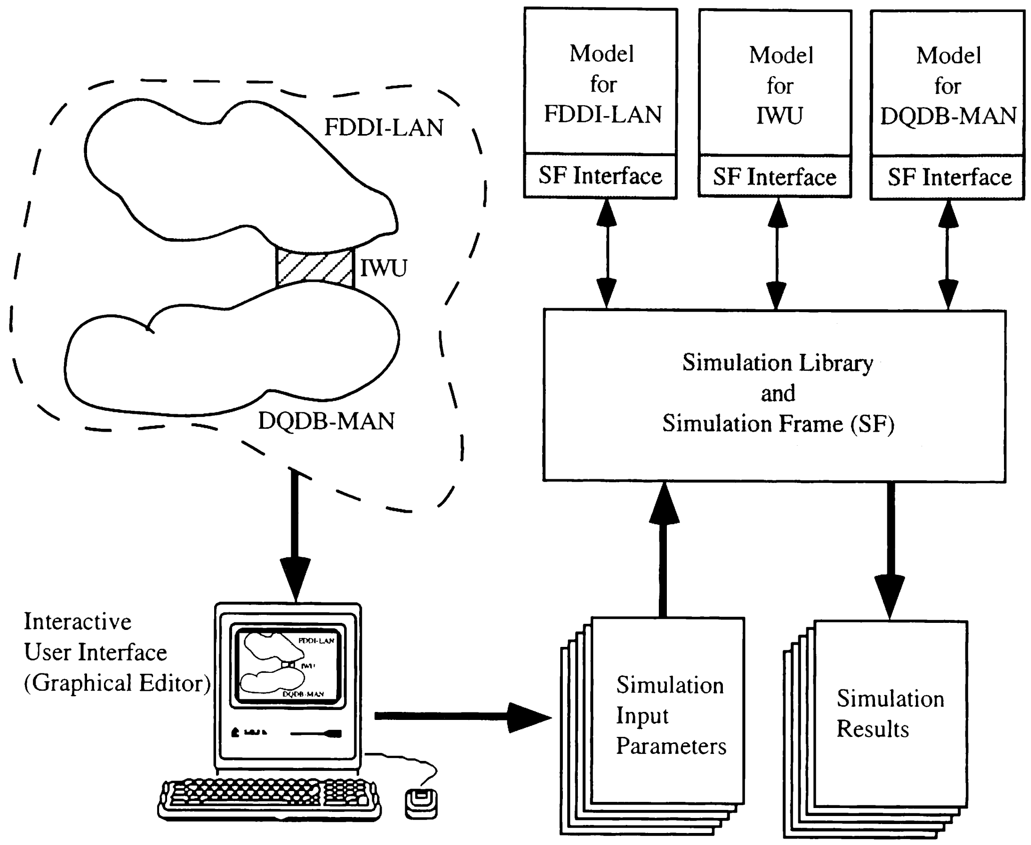

Figure 1: Simulator Architecture

2 THE SIMULATION MODULES

The architecture of the heterogeneous network simulator SimCom is shown in Figure 1. It consists of two parts: the user interface (UI) and the simulation modules.

The main function of the UI is to assist the user in defining the topology of the simulated heterogeneous network. With this information, the UI generates the input files for the simulation modules. The simulation modules and the simulation libraries are the subject of this section.

The simulation modules are written in C and include:

1) models of the networks that form part of the heterogeneous scenario considered,
2) models of interworking units (IWUs),
3) generic simulation libraries, and
4) the Simulation Frame (SF).

The networks and IWUs are represented by discrete, asynchronous, event-oriented simulation models, using the general simulation primitives provided by C-libraries. The SF is an extension of the libraries that specifically supports the simulation of heterogeneous networks. It allows the creation of several instances of the same network type, their interconnection by means of interworking units, and the routing of data units from network to network.

All network and IWU models must offer a standard interface, so that they can be integrated with each other and with the SF in a heterogeneous network simulation. Assuming this, new models can be easily added to the simulator. Currently there are models available for DQDB (Distributed Queue Dual Bus) and ATM (Asynchronous Transfer Mode) networks, and models for FDDI (Fiber Distributed Data Interface) and Ethernet are being developed. Details on the functionalities of the networks can be found in Kvols (1992).

2.1 The Simulation Library

The generic simulation functions provided by the library support the development of discrete, asynchronous, event-oriented simulation models in C. It includes general functions for (Rümekasten 1991):

- Event handling (schedule event, get next event).
- The generation of random variables with usual probability distributions, e.g. exponential, normal, etc.
- The analysis of simulation results (mean values, variances, histograms, etc).
- Input/Output functions for simulation parameters and error messages.

Additionally, the simulation functions provide predefined data structures (e.g. "packets") intended for the simulation of communication networks, in particular those structured in protocol layers.

2.2 The Simulation Frame (SF)

Looking at the reality of today's communication networks it is easy to see that heterogeneity of different network protocols and implementations is a fact. Since this will not completely change in the future, it is important to analyze the performance of heterogeneous networks. Therefore, an additional C library, the SF was developed (Ottensmeyer 1991).

The SF enhances the simulation in two important aspects.

Firstly, the SF allows the simulation of a heterogeneous system including several network types and instances of the same network. Network models which have been developed independently can be integrated in order to simulate a heterogeneous network, provided that the interface of each model (section 2.4) conforms to the rules imposed by SF.

Secondly, the SF allows a flexible definition of the simulated network structure by means of input parameter files which are read before starting the simulation. In this way, different network configurations can be simulated without recompiling. (These files are generated by the UI, so the user does not need to edit them directly.)

The SF includes the following functions (Ottensmeyer 1991):

- Initialization of the simulated system configuration as defined in the input files.
- Routing functions: these functions determine the intermediate node in the sending network and the IWU that must be used to reach the destination network.
- Input/Output functions. They coordinate the access of the different simulation modules to the command-line parameters and external files.

2.3 The Interworking Unit Module

The IWU module provides a generic interworking unit structure, which can be enhanced by the programmer to model particular interworking units. At the general level, an IWU module consists of two (or possibly more) *ports*. For each port the user specifies the network and node where the port is attached, and the networks which can be accessed through the port. This information is used by the SF routing functions.

Packets are delivered to an IWU by scheduling a predefined event. An internal event of the IWU gets each packet and passes it to the destination network. The internal structure of the IWU models and their interface to the SF are similar to those of the network modules described below.

2.4 The Network Modules

Following the event-oriented simulation paradigm, a model that is part of the heterogeneous scenario consists basically of a set of event-processing routines. Each routine deals with the events of a given type and can schedule future events of any type. Future events are stored in a list (provided by the simulation libraries) and processed in chronological order (also provided by the simulation libraries).

In this case, each module has a function, that accepts an event and calls the appropriate processing routine depending on the event type. If the event is not known, e.g. because it belongs to another component of an heterogeneous network, the function returns the event unprocessed. The SF is responsible for retrieving events from the event list, and passing each of them to each module in sequence, until one of the modules recognizes and processes the event (for more details see Vogt et al. (1992) and Rümekasten and Vazquez (1994)).

Furthermore, each network model must provide a function that accepts data coming from other networks.

Simulation models of DQDB, ATM and their IWUs are described in detail in Kvols (1992).

3 THE USER INTERFACE

The simulation modules as described up to now give a flexible and efficient possibility for the analysis of a heterogeneous network. However, there is a disadvantage that makes it hard to use the modules: each module has its own input parameter files (for more details see Rümekasten (1991) and Ottensmeyer (1991)) and the user has to integrate and connect the modules by himself to design the scenario he wants to analyze.

Therefore the user has to have some knowledge in C programming and in using the special SF input format. Since the user (the network designer) might not be a computer scientist, this knowledge can not be taken for granted for each user.

Because of this lack of programming knowledge, a network simulation tool that really wants to be used has to be as user friendly as possible to make its handling as easy as possible. In future times, when the parallelism of the simulation modules will be developed the simulation modules are much too complicated for non-specialists.

To increase the practical use of the parallel simulation library, SimCom provides an interactive UI based on the X-window system. By determing the input parameters for all the subnetworks and IWUs, the whole scenario is generated in an interactive man-machine dialogue. Based on these inputs, the simulation of the desired scenario can be started without programming the connection between the modules.

3.1 Principles in User Interface (UI) Design

Conventional user interfaces are characterized by sequential dialogues that are fixed with the sequential application. This means that there is a strong connection between UI and application. This results in badly readable programs and difficult changes in both the user interface and the application. Another disadvantage of this kind of UI design is the fact that the user interface can not be re-used for any other application and vice versa.

To make the (possibly parallel) program code more easy to understand and to enable software re-use, modern user interface design follows the principles of "dialogue independence". This results in a clear separation of user interface programming and underlying application. For a simulation environment this means that no semantics of the application (the parallel simulation program) is used in the user interface: each input value from the interface is independent from its semantics for the simulation program itself. Only the connection between interface and application maps the semantics from input values (semantics of the user interface) to network parameters (semantics for the simulation modules).

The so-called "Seeheim model" (see Pfaff (1985) and Szwillus (1991)) tries to realize dialogue independence by separation of UI and application. The model and its components is shown in figure 2.

The presentation component provides the objects of the UI. This component communicates with the user on its own using the provided objects of the interface. Each of the interface objects has a specific semantics for the interface. An object of the presentation component can be a push button that represents two values ("on-off button"). The value of that object is calculated by the presentation component and depends on the action of the user.

The shared application data model also provides a pool of objects. These objects can be part of the application objects or have a direct connection to the application objects. They map the semantics from the presentation component objects to the application objects.

By this mapping, the UI objects are connected to the underlying application.

By this concept of dialogue independence the application runs independently from the UI. For the case of a simulation environment this concept results in a remarkable advantage: the runtime of the simulation is not influenced (that means slowed down) by the UI. Since simulations are very runtime sensitive this advantage of dialogue independence is much more important for this kind of application than for other applications. Because of this, the SimCom user interface has been developed following the principles of dialogue independence using a tool called InterViews.

3.2 The InterViews Class Library

InterViews (Interactive Views) has been developed at the Stanford University, USA. It provides a library of object-oriented C++ classes of graphical objects for the X-window system environment (see Linton et al. (1991)). The user writes his own C++ interface program using the classes provided by InterViews.

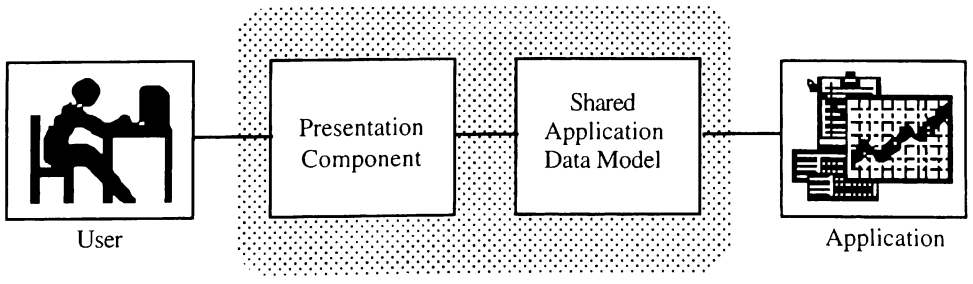

Figure 2: Dialogue independence (Seeheim model)

The hierarchical structure of InterViews makes it possible to compose different classes to new classes or to enhance or modify the existing classes of graphical objects.

In its basic version InterViews provides several classes to receive and process events like the pushing of buttons or selecting items from a menu. Other classes can be used to compose existing objects to create windows or backgrounds as new objects. Also the modification and the visualization of objects is possible using pre-defined InterViews classes. More details can be found in Linton et al. (1991).

The object-oriented structure of all the classes provides good methods to re-use classes in different parts of the interface simply by the hierarchical structure of any interface.

3.3 The SimCom User Interface

This section describes the architecture of the SimCom user interface. The UI has been created using the pre-defined InterViews classes. The UI is based on a graphical editor providing graphical objects representing different network elements for a complete definition of the input parameters. The complete scenario is defined according to the composition of the graphical objects on the interface.

The interface is structured hierarchically with general inputs at the beginning and more detailed inputs in the following. Figure 3 shows the structure of the interface. At the beginning of the specification the user has to define the general network parameters for the scenario. These are the networks to be used for the simulation and the number of those. To determine these parameters the user selects the networks and determines their parameters.

The network parameters for each of the chosen networks are determined easily by moving "stations" of the networks and by connecting the networks with IWUs that can operate in different modes. By this, the overall structure of the heterogeneous network is determined.

However, some details for each station are still to be specified. These details are the load types for each station, buffer sizes and other parameters. Those parameters are determined by selecting them from pre-defined menus.

The connection between the interface and the application (the shared application data model) is realized by creating the simulation input files after finishing all the inputs. The interface creates the input files automatically on demand and also starts the simulation if the user wants to simulate the scenario (general inputs in figure 3).

4 PARALLEL SIMULATION

Simulation of heterogeneous highspeed networks is a time consuming job: each cell-arrival (e.g. for an ATM network) has to be computed to simulate the behavior of the network correctly. Therefore lots of millions of cells have to be computed to make the program calculating correct results that can be used for performance analysis.

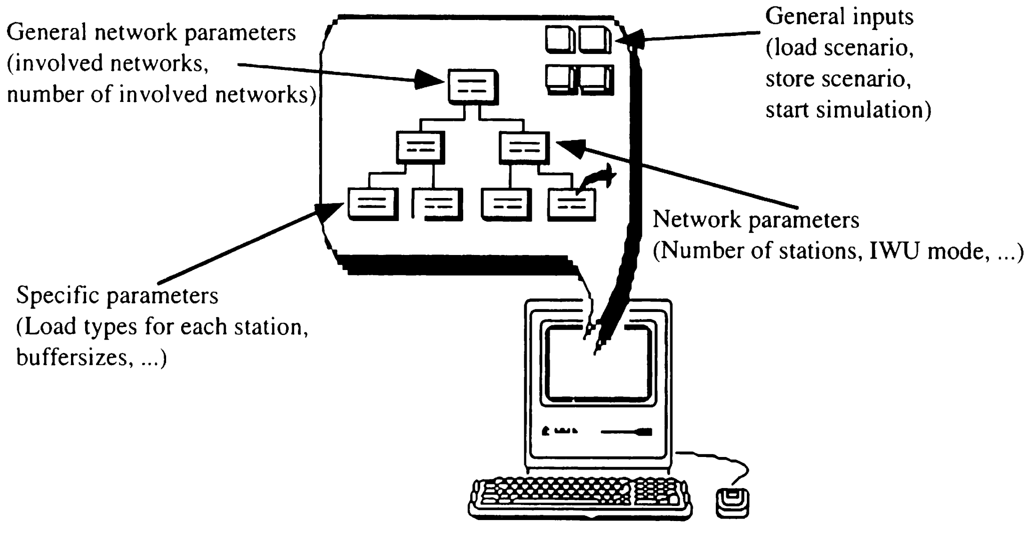

Figure 3: Structure of the SimCom user interface

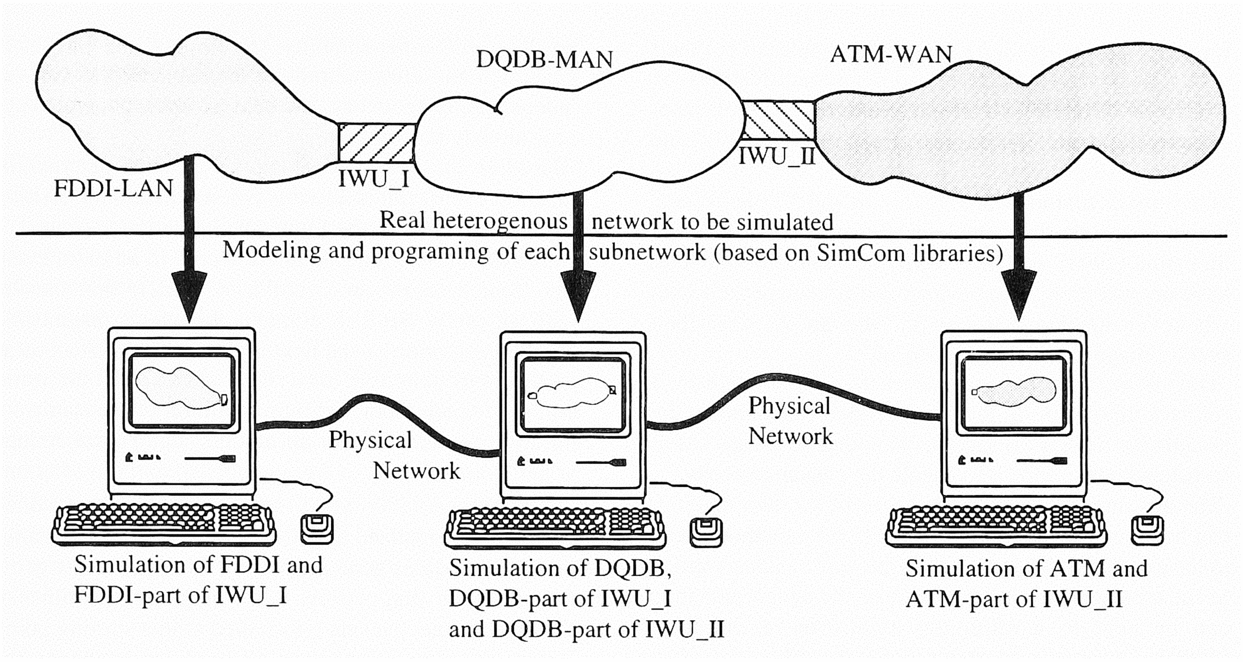
DQDB-MAN
ATM-WAN
FDDI-LAN
IWU_I
IWU_II
Real heterogenous network to be simulated
Modeling and programing of each subnetwork (based on SimCom libraries)
Physical Network
Physical Network
Simulation of FDDI and FDDI-part of IWU_I
Simulation of DQDB, DQDB-part of IWU_I and DQDB-part of IWU_II
Simulation of ATM and ATM-part of IWU_II

Figure 4: Distributed architecture of future SimCom

To speed-up such time consuming computations the distribution of the program is a promising method. But to be as runtime effective as possible one has to keep the communication overhead of the distributed program parts as low as possible. Therefore, the independently running parts of the distributed program should be as complex as possible.

In our simulator, we have a very special scenario: the simulation of heterogeneous networks. The only communication of the subnetworks is the exchange of internet-packets. In realistic environments, this event does not occur often, because most of the data is consumed locally.

Thus, for a simulation of heterogeneous networks, the independent simulation of each subnetwork on one computer in a workstation cluster (or on a multi-processor workstation) seems to be the most promising (and natural) way to speed-up the computation due to its low communication overhead.

Figure 4 shows the (future) architecture of SimCom.

4.1 A Synchronization Problem

Simulating subnetworks independently also means independently running simulation clocks with "fast" clocks for simple and "slow" clocks for complex programs.

As one easily can assume, the complexity to model networks is different for different networks: cell-based networks (DQDB, ATM) are very complex, because each cell has to be modeled. In opposite, FDDI is very easy to model: only one free token circulates on the ring. So the computation time of the networks differs very much. In fact, a FDDI program is much faster than a DQDB program. Thus, the simulation clocks of different modules run different.

So the following problem can occur: the module for network A with local clock time x sends a packet to the module for network B. When the packet arrives at the network B module, the local clock time of that module is y with y > x. Now, the network B module has received a packet that is "too late" and the computation of that packet can not be handled correctly.

Therefore the modules have to be synchronized ("synchronization event") to avoid that problem. The next section describes classical synchronization methods.

4.2 Synchronization in Distributed Simulations

In general, two synchronization methods are known: "conservative" methods and "optimistic" methods.

Conservative methods are based on knowledge of the time between two synchronization events. This means that the minimum time interval T between synchronization events is always known. If T can be determined, the simulation can proceed in cycles of T. At each time stamp $n{\bullet}T$ the simulation modules can exchange data without any risk of receiving packets "too late". Using this method it might happen that "fast" simulation modules have to stop at time stamps $n{\bullet}T$ and wait for "slower" (= more complex) modules to get their data. Conservative methods and their modifications are known as "Barrier synchronization" or "Time Bucket synchronization" (compare Fujimoto (1990) and Steinman (1993)).

Optimistic methods are based on the "Time Warp" mechanism as described in Fujimoto (1990) and Steinman (1993). In Time Warp, the simulation modules have no synchronization events and exchange their data whenever needed. Of course, the above described synchronization problem can occur.

Time Warp then has to recover the system state by a rollback algorithm. In the situation described above, the module for network B (and maybe other modules) has to be rolled back to time x to handle the packet correctly.

The advantages and disadvantages of the two methods can be compared as follows: conservative methods need to have a good knowledge when the next synchronization event takes place but the overhead for synchronization can be kept to a minimum as shown in Fujimoto (1990). Optimistic methods have the advantage that Time Warp can be used for any simulation problem without any knowledge of the synchronization events but have the big disadvantage that the rollback mechanism can be very difficult to handle and very time consuming to perform.

Thus, Fujimoto (1990) concludes that Time Warp is useful for "general purpose simulations" and conservative methods are useful for optimized simulation packages.

Based on this, the next section develops a possible synchronization method for SimCom.

4.3 A Synchronization Method for SimCom

The basis for a possible SimCom synchronization algorithm is an enhanced Time Bucket synchronization with flexible time intervals that are determined at each synchronization event (compare "Breathing Time Bucket" in Steinman (1993)).

Thus, for Time Bucket synchronization the time interval between two synchronization events has to be known. In our simulation environment the synchronization event can be the generation of an internet packet because of the well known traffic generation process as explained in the following.

Any of the internet packets can be generated as follows: the load generator creates the packet and calculates the time T when the next packet has to be created according to the underlying load generation theory. Then an event with time stamp T is generated that creates the new packet at time T. So at the creation time of one packet the creation time of the next packet of that type is always known exactly. Thus, each subnetwork can run until time T independently. At time T the packet is generated and after its transfer the subnetworks can run independently again until the next packet is generated at a now already known point of time.

Based on this good knowledge of the time of synchronization events the following synchronization algorithm can be integrated in the SimCom simulation modules:

The synchronization events are stored in an extra event list (Synchronization Event List (SEL)) that can be realized easily. So SEL contains the events responsible for the generation of the next internet packet. According to the load generator described above, SEL is never empty for each module. So the minimum time stamp for all events in all SELs for all modules can be determined at any point of time.

Then the simulation and synchronization of the subnetworks can be realized as follows: at the start of the simulation all modules generate the time stamps for the generation of their first packets. Also the generation times of the first internet packets are created at this starting point. Before the simulations start all modules exchange the time stamps of their first internet packets (stored in SEL) to all other modules. Having received all time stamps, each module determines the time T_0 when the first internet packet is generated. After that the modules start and run independently until T_0.

Only the sender of the internet packet does not stop at T_0 to be able to generate the internet packet at time T_0. After that the sender generates the time stamp for its next internet packet according to the load generator, stores the time stamp in SEL and sends it to the other modules. When the current packet has been transmitted after its media access the sender also stops.

In the time between packet generation and its transmission the following events can occur: another sender in the same module has generated another internet packet or another sender in one of the waiting modules has generated another internet packet (that can be found out by the exchanged time stamps). The first event is no problem because all the other modules are stopped and can receive the packet at the appropriate time after their re-start. The second event also is no problem because in this case the running module only runs until the time stamp for the packet generation in the other module. Then it stops and makes the other module run to generate and transmit its packet. After that the module is re-started for its own transmission. Thus, the synchronization always simulates correctly.

Having reached its destination in one of the waiting modules the transmitted packet can be handled correctly because the receiving module is still waiting.

The next minimum time stamp T_1 can be determined from the old ones except T_0 (they are still valid) and the transmitted new one. Now again all simulations can run independently until time T_I, the new packet is generated and transferred, the time stamp for the next internet packet is generated and the next minimum time stamp is determined.

This way, the simulations run independently from each generation of an internet packet to the next.

The advantages of this synchronization is its inherent exactness and its low communication overhead: the only additional data that are exchanged are the time stamps and messages that the modules have reached the synchronization event time. These messages are very short and are exchanged from the modules at each synchronization event. Since the synchronization events do not occur often (at least in realistic environments) the modules run independently most of the time.

This synchronization is only possible because we have a scenario where the time stamp of the next synchronization event is already known at the current synchronization event.

5 FURTHER WORK

Further work on the user interface part of SimCom will be the integration of additional networks and IWU models. This will include FDDI, the Ethernet and possibly new developments like the 100 Mbit/s Highspeed-Ethernet. Including all these networks, a useful library of existing networks will be provided by SimCom for simulation.

In addition to this a new graphical editor will be introduced to SimCom. This editor will provide the user with "empty" symbols and network elements to create an own UI or to enhance existing parts of the UI with new features.

More work will be done to enhance the underlying simulation structure: other synchronization techniques will be tested and the most promising (that seems to be the one presented here) will be implemented in the simulation modules in order to realize a distributed application for the performance evaluation of heterogeneous networks. The distribution of the modules will increase the performance of the tool to be able to simulate even more complex scenarios than today.

REFERENCES

Fujimoto, R. 1990, "Parallel discrete event Simulation", Communications of the ACM, Vol. 30/10

Kvols, K. 1992, University of Paderborn (internal report) "Models for Heterogeneous Networks and Their Elements", December 1992

Linton, M. A., et al. 1991: "InterViews Reference Manual Version 3.0", Board of Trustees of the Leland Stanford Junior University

Ottensmeyer, J. 1991, "The Simulation Frame for Simulation of Heterogeneous Scenarios. Programmers Manual", Techn. Report, Univ. of Paderborn

Pfaff, G. E. 1985 (editor), "User Interface Management Systems", Springer

Rümekasten, M. 1991, "ATLAS Reference Manual", Dept. of Mathematics and Computer Science, University of Paderborn

Rümekasten, M., and E. Vazquez 1994, "SimCom - A Window Based Simulator For Performance Analysis of Heterogeneous Networks", Interworking'94, IOS Press

inman, J. 1993, "Breathing Time Warp", Proc. of the 7th Workshop on Parallel and Distributed Systems (PADS93), Vol. 23, No. 1

Szwillus, G. 1991, "Support for the creation of direct manipulation interfaces", Dept. of Mathematics and Computer Science, University of Paderborn, Technical Report "Reihe Informatik", No. 90

Vogt, R., et al. 1992: "A Concept for Interconnecting DQDB MANs through ATM-based B-ISDN and Related Issues with respect to Simulation", First International Symposium on Interworking, Interworking '92, Bern

AUTHOR BIOGRAPHY

MARKUS RÜMEKASTEN studied computer science at the University of Paderborn, Germany. After receiving his M.S. degrees in late 1992 he works as a scientific assistant at the Department of Mathematics and Computer Science at the same University in the Networking Group. His research is focused on the performance analysis of heterogeneous highspeed computer networks by means of simulation. His special interest is the distributed simulation including problem based synchronization techniques in order to speed-up simulation runtimes. He is working on this topic for his Ph. D. thesis.