785

A SIMULATION TECHNIQUE FOR DISTRIBUTED SYSTEMS BASED ON A FORMAL SPECIFICATION BY SDL

W. Fischer, K.P. Sauer, W. Denzel

University of Stuttgart
Institute of Communications Switching and Data Technics

Seidenstrasse 36, D-7000 Stuttgart 1
Federal Republic of Germany

ABSTRACT

For some basic communications mechanisms within distributed systems analytical approaches are available for calculating throughput, transfer times, waiting times and some other quantities [1,2], but for a large class of problems such methods do not exist or are not applicable due to the systems' complexity. System simulation is a common way to overcome this problem at least to a certain degree.

This paper deals with a simulation technique which works with a formal description of the system of processes to be simulated. This description is by means of SDL, the "Specification and Description Language" recommended by CCITT [3].

1. INTRODUCTION

In recent years a large amount of work has been done in developping formal description techniques for the specification of communications protocols which are the means for communication between processes in distributed systems. The need for these formal techniques especially for protocols to be standardized is due to the facts that

- o the products of different manufacturers have to interwork according to a distinct protocol.
- o a specification to be recommended has to be checked for formal correctness.

Besides the correct function of a protocol an essential item is its performance. Evaluation of performance based on a formal specification can be done by analytic investigations or by simulation. While mathematical analysis of protocol performance dealing with all features specified for example by an FSM [4] is often too complex, simulation of a completely specified protocol is a very common task. Until now most simulations are done by translating the formal specification into any programming or simulation language. The drawbacks of this approach are the long time needed for implementation and errors occuring by an inprecise translation.

The approach described here uses directly a formal specification by the "Specification and Description Language" (SDL) recommended by CCITT. It has been implemented as a tool for performance evaluation in the environment of an "SDL workbench" which is currently developped at the institute. This tool set will support specification, formal verification and performance evaluation of protocols. It is clear that by this simulation approach also a certain verification of the main features of the protocol is achieved, especially some kinds of errors like livelocks which are difficult to determine by formal verification, can be detected if the probability for entering such an error condition is not too small for it being reached within the simulation.

2. THE LANGUAGE SDL

The Specification and Description Language is based on the concept of communicating processes modelled as extended finite state machines (i.e. FSM with additional context variables). Initially, SDL has been designed for the specification of the internal logic processes in SPC switching systems. By some extensions and a formal definition of the language made in the recent study period it was made easier to use it for the specification of communications protocols.

An SDL **system** is a set of **blocks.** Blocks are connected to each other and to the environment by **channels.** Within each block there are one or more **processes.** These processes communicate with one another by **signals** and are assumed to execute concurrently.

Based on a common SDL model there are currently two concrete syntaxes SDL/GR which is a graphic form and SDL/PR which is a programming-language-like form. It is possible to map a system defined by one concrete syntax to another. The SDL syntax defines the rules for the definition of the items:

System, Block, Channel, Signal, Process, Process graph

Within the scope of this paper we shall concentrate on the definition of process graphs. A process graph is a graph whose **nodes** are connected by directed arcs. The following categories of nodes exist:

State, Input, Task, Output, Decision, Start, Stop, Create request, Procedure Call

Exchange of messages in SDL is achieved by two different approaches:

- o Shared variables (i.e. export and import of values)
- o Transfer of signals via channels

For the sake of clearness in this paper we shall use the term "message" instead of "signal" in contrast to the "continuous signal".

As the transfer via channels happens asynchronously buffers have to be provided in front of the processes. The model recommended by CCITT for the use with SDL defines that each process has only one associated buffer which is served in FIFO order. An exception from this FIFO order is allowed by the use of the save symbol. If the next message in the buffer to be served is not listed by the input nodes but by a save symbol this message will be kept in its place in the buffer and the next one will be considered. If a message is listed neither in an input node nor in a save symbol it will be absorbed.

CCITT does not recommend the use of priorities for the messages. As in real applications priorities are widely used and SDL does not offer mechanisms to deal with, we found the need to make an appropriate extension to the language. This priority mechanism is described in detail in [10].

3. THE SDL-WORKBENCH

The simulation tool is part of a software environment for the specification, formal protocol verification and performance simulation of distributed systems (Figure 1).

Based on a common internal data structure which is a mapping of the process graphs, use of both syntaxes of SDL is supported by a graphic editor and by an SDL-Parser. The internal data structure is used by a tool for the formal verification of protocols based on a reachability analysis by means of the state perturbation approach [11], and by the tool for performance simulation which is described in detail in this paper.

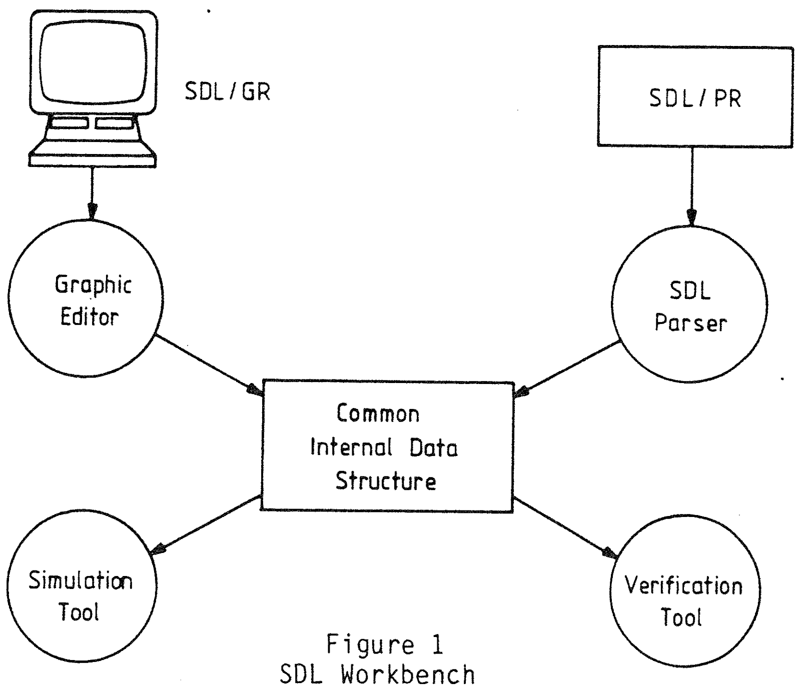

Figure 1
SDL Workbench

4. THE SIMULATION TOOL

The basic idea of this simulation technique is that processes are executed by interpretation of the SDL process graphs (Figure 2). A time-true simulation approach is used where events are initiated by tasks and output nodes of the graphs. (A related approach using a draft version of SDL has been presented by Gerrand [12] in 1976)

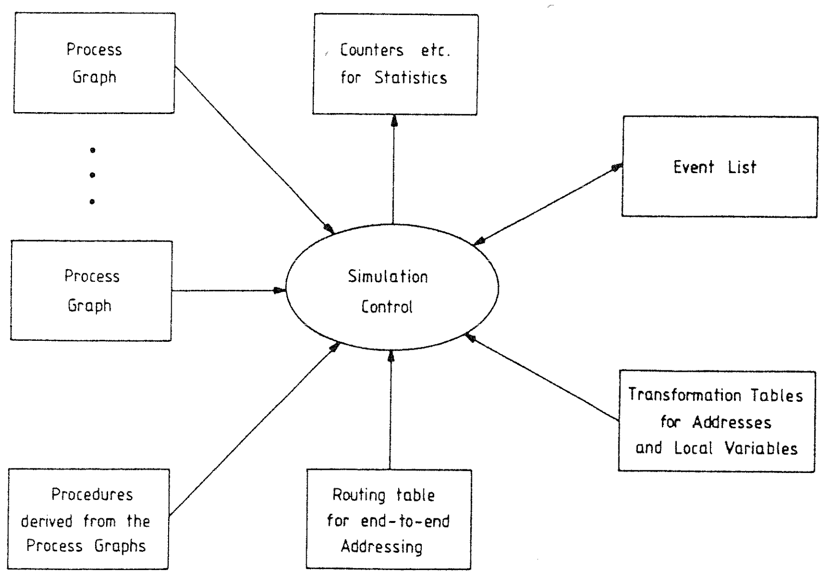
Process Graph
Counters etc. for Statistics
Event List
Process Graph
Simulation Control
Transformation Tables for Addresses and Local Variables
Procedures derived from the Process Graphs
Routing table for end-to-end Addressing

Figure 2
The Simulator

Each process with its associated input buffer is represented as a delay-loss unit. A process graph can be shared by several incarnations of a process, so that identical process graphs have to be mapped only once. The address information carried by a message is two-fold:

- o The global view uses an end-to-end addressing scheme which has to be evaluated by each process by means of a routing table.
- o The local view defines immediately the adjacent processes.

Due to the fact that a process graph can be shared, each process has a transformation table for converting the formal addresses of the graph into real addresses of the system. A similar transformation is necessary for local variables.

Processing time of a process is only represented by tasks and is expressed by its distribution function F(t). All other actions to be taken are assumed not to consume time. If processing time for any other event has to be considered a time consuming task simply can be added. Each node contains the following informations necessary for its processing:

- o <u>TASK:</u> Distribution function for processing time; arithmetic or logic expression to be evaluated; functions like "TIMER START" and "TIMER STOP" including parameters
- o <u>OUTPUT:</u> Message name; address information; variables for values to be transported.
- o <u>DECISION:</u> Logic expression which can include a branching probability; jumping distance for false-arc
- o <u>STATE:</u> State name
- o <u>INPUT:</u> Variables to be transported; enabling predicate

With these definitions we have implemented the following simulation algorithm:

- o The nodes of a process graph are executed sequentially.

- o If the actual node is a time-consuming TASK the holding time is determined, added to the actual system time and entered into the controlling event list with a pointer to the next node. Then the next event within the event list is searched.

- o If the actual node is a STATE, according to the priority mechanism the input buffers of the processes are searched for messages and the predicates representing the continuous signal are evaluated. If any message has been found it will be taken out of the buffer and the associated path of the process graph will be executed. If no message is present the process is kept in this state and the next event in the event list is searched. Similar action is taken for a "true" continuous signal.

- o If the actual node is an OUTPUT, the current process is interrupted and control of the simulation will follow the message, i.e. execution changes to the destination process until the next change in system time (The destination process can either be in a state or in a time-consuming task). Before the system time changes, the interrupted process will be resumed and executed further. For resuming the process at the correct position in the graph, suitable information is pushed onto a stack before interrupting the process by an output.

- o If the actual node is a TASK containing a "TIMER START" statement, the system time for timeout is determined and entered into the event list including the name of the timer and the destination process. If the task contains a "TIMER STOP" statement the associated event in the event list is deleted.

o A new event found in the event list can either be the end of a holding time, then execution of the associated process will continue with the node determined by the event, or it is a timeout; then a message will be put into the first place in the destination process' input buffer. In this case control will only be passed to that process if it is in a state.

There are two input sources for the simulator:

o The process graphs represented as a mapping of the graph structure used by the tool
o The configuration description defining

  - the local view of each process
  - the routing table for end-to-end addressing
  - transformation tables for mapping formal addresses into real addresses
  - the maximum length of each buffer
  - parameters like rates, holding times etc.
  - initial states, buffer contents and values of context variables

5. IMPLEMENTATION DETAILS

Application of the tool can be divided into three steps

1. System specification (Process graphs, configuration description)
2. Analysis of the system specification and synthesis of the simulator: Arithmetic and logic expressions are converted into compilable procedure definitions. (Experience showed us that interpretation of these expressions by the simulator itself will bring a large decrease in performance) These procedures are compiled and linked with the general routines to generate the simulator.
3. Simulation run

The simulation tool is implemented in PASCAL, so for the sake of simplicity the syntax of expressions in the process graphs must be PASCAL-like too (CCITT defines a CHILL-like syntax [6] for these expressions). The implementation of the event list which controls all the time dependent actions is by means of a binary searching tree which for a large number of entries is a good solution instead of a linear list [5].

6. A CASE STUDY
6.1 The Model

For the demonstration of some features of the system we use a Single-Link, Multiple-LAP Protocol (The model has some similarity with the layer 2 of the ISDN D-channel protocol [7]). While a lot of investigations have been done on performance in the data transfer phase of the HDLC-Procedures [8] and of X.25 Level 3 [9] we shall focus on the procedures for setting up and releasing a logical link.

Doing this we neglect the mechanisms for error recovery and flow control and assume a constant length of 1024 bit for data packets and of 48 bit for control packets (CONN, CONN-ACK, DISCONN and DISCONN-ACK).

The configuration of the simulated system is shown by Figure 3.

All buffers are given a maximum length of 10 places.

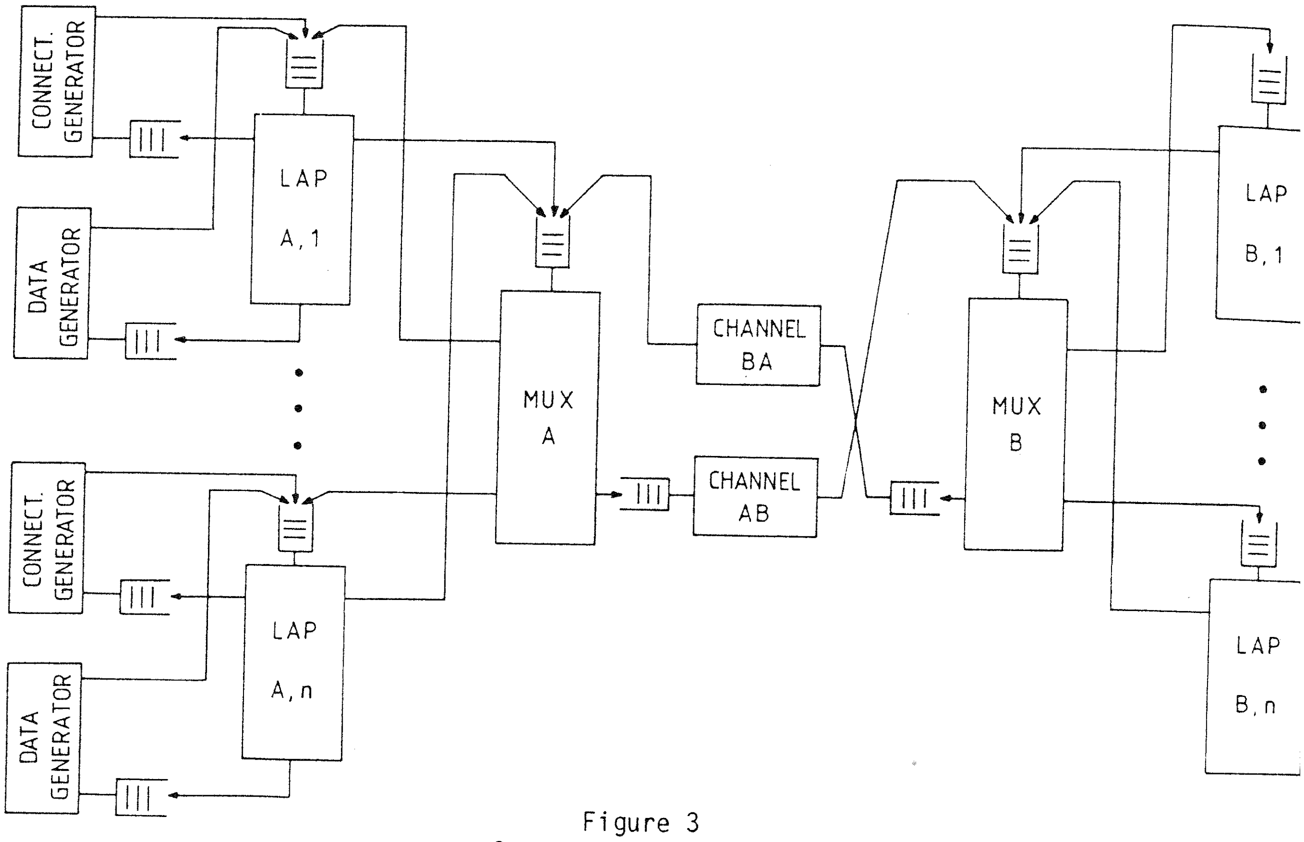

Figure 3
System configuration

The process diagrams of some processes are shown in Figures 4a,b.

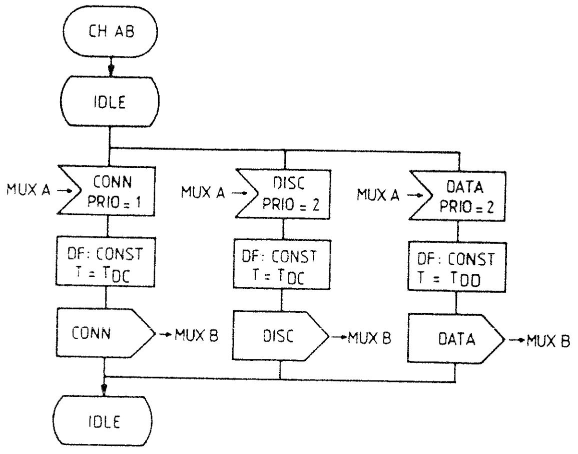

Figure 4a
Channel AB (Priority control in front of the channel)

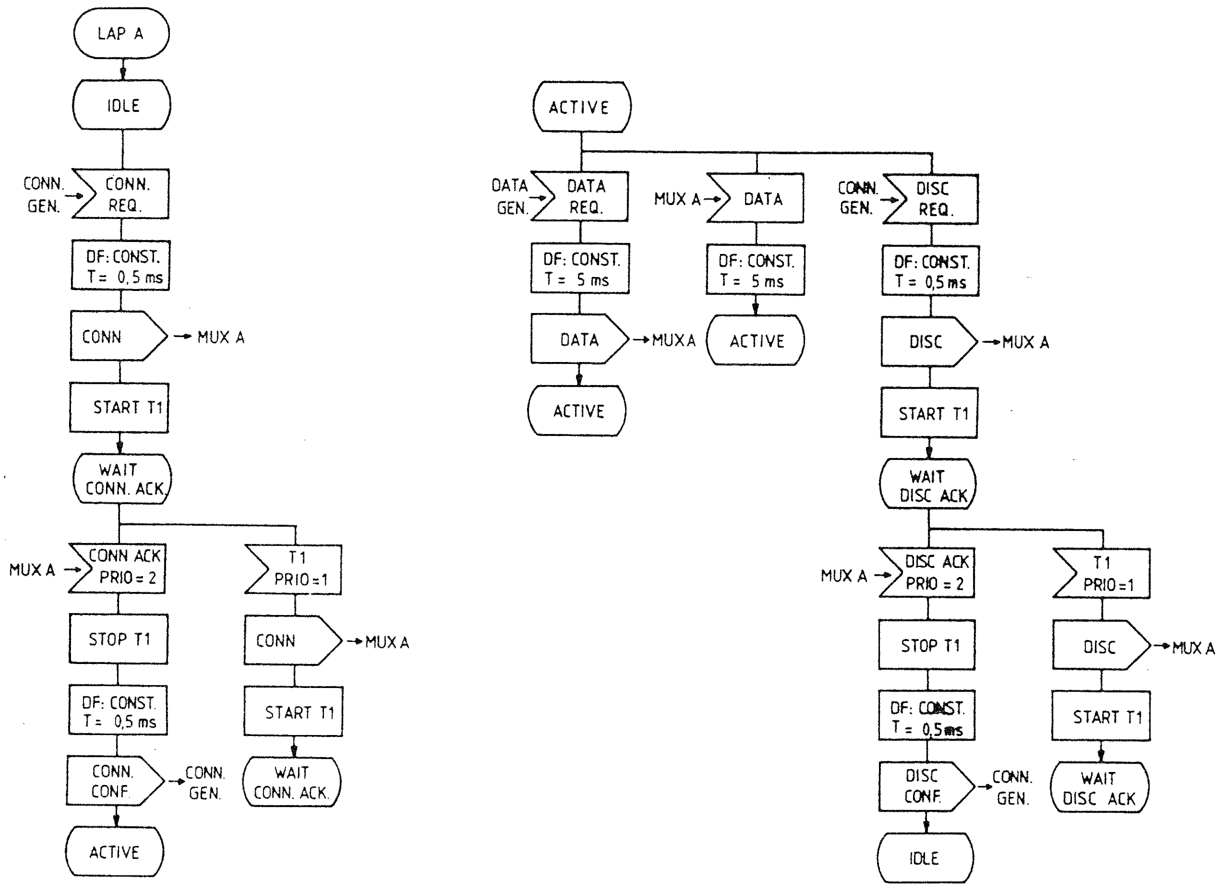

Figure 4b
LAP A

A connection between two corresponding LAPs is initiated by a connection generator associated with each LAP on the left hand side of Figure 3. The arrival process of these connections is determined by holding time T_H and idle time T_I, both negative exponentially distributed. The arrival process of the data transferred across the link is produced by a data generator associated with each LAP on each side of the link and is assumed to be Poisson with arrival rate λ_D. As for the sake of simplicity in the data transfer phase flow control mechanisms are not implemented we have to deal with blocking effects when buffers are full. So signalling packets could get lost and the system would enter a deadlock state. To overcome this problem we applied the commonly used principle of timeout and retransmission (Timer T1).

We have assumed different processing times (in LAP and MUX) for control and data packets (control: 0.5ms, data: 5ms) and have used transmission speeds of 16 and 64 kbit/s as used in the D-channel protocol of the ISDN. For obtaining minimum setup times the CONN and CONN-ACK packets are given higher priority than the DATA packets and the DISCONN and DISCONN-ACK packets. This can be achieved at two places in the system:

1. In front of the MUX
2. In front of the channel

Both possibilities have been investigated.

6.2 Results

The main results we are interested in are the call setup time and the mean buffer length in front of the channel depending on the maximum number of LAPs, their holding- and idle times and the offered data load at each LAP.

We found that for constant channel load the number of LAPs has nearly no impact on the results, what seems obvious when the relation between control load and data load is taken into account.

Under these circumstances a good measure to relate the results to is the carried load of a channel which in this case can be determined with a first order approximation as

$$Y_{CH} = n_{LAP} * \lambda_D * T_{DD}$$

n_{LAP}: Mean number of active LAPs

λ_D: Arrival rate of data packets

T_{DD}: Transmission delay for data packets in the channel

Figure 5 shows the obtained results, the mean call setup times and the mean lengths of the buffer in front of the channel for the two different transmission speeds. Included are the 95% confidence intervals.

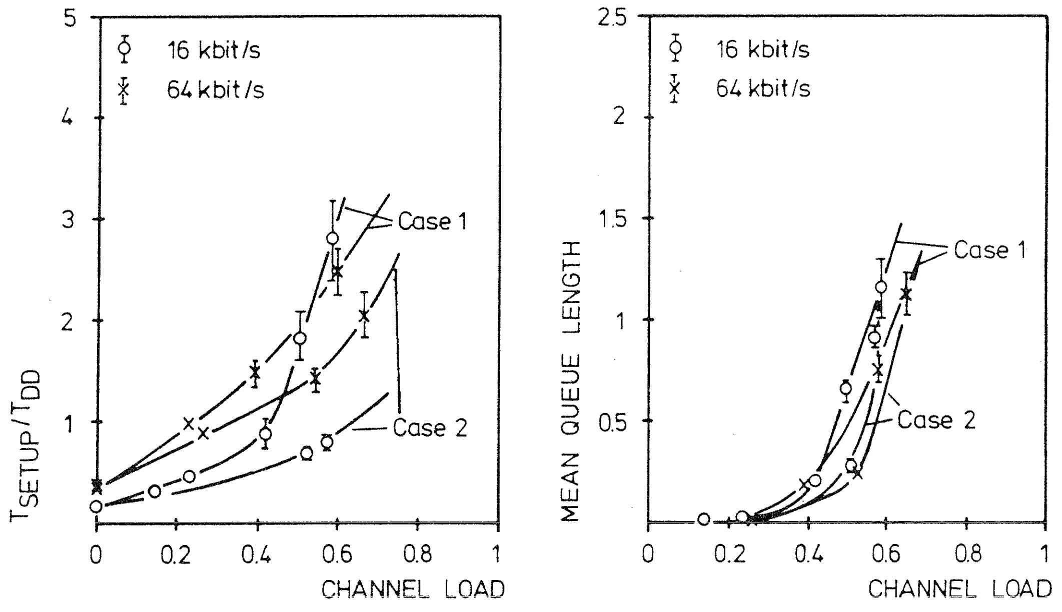

Figure 5
Results

Case 1: Priority control in front of MUX
Case 2: Priority control in front of the channel

The rapid increase of the call setup time above the channel load of 0.4 is due to the loss of signalling packets and timeout recovery.

With the parameters used within this study the bottleneck of the system is represented by the channel. Performance improvements have easily been achieved by using the applied mechanism in front of the channel.

7. SUMMARY

We have presented a simulation tool for distributed systems which uses a specification by SDL, a language widely used for the specification of logic processes in switching systems and of communications protocols by the telecommunications industry, the standardization committees, and the PTT administrations.

This tool has been developped at the University of Stuttgart, Institute of Communications Switching and Data Technics and will be imbedded into an "SDL-workbench" used for the specification, verification and performance evaluation of distributed systems.

The main advantages of this approach are the short time needed for the implementation of a model and the detailed mapping of the real system into the simulated system. Additionally, a certain amount of errors occured in the formal specification can be detected.

The simulation method of process graph interpretation has been demonstrated and the simulation of a Single-Link, Multiple-LAP Protocol has been used as a case study.

8. ACKNOWLEDGEMENTS

We wish to thank Prof. P. Kuehn for supporting this work, and A. Dettling for carefully drawing the figures.

9. REFERENCES

[1] T. Raith, H.L. Truong: Analysis of HDLC Normal Response Mode with Full Duplex Transmission - A new Calculation Method and Simulation. Proceedings 10th ITC, Montreal, Canada, paper 3.4-2 (1983)

[2] G. Fayolle, E. Gelenbe, G. Pujolle: An Analytic Evaluation of the Performance of the "Send and Wait" Protocol. IEEE Transactions on Communications, Vol. COM-26, No. 3, March 1978

[3] CCITT: Recommendations Z.101 to Z.104. Red Book, Geneva 1985

[4] H. Rudin: From Formal Protocol Specification Towards Automated Performance Prediction. in "Protocol Specification, Testing and Verification, III" (North Holland), 1983

[5] J.H. Kingston: Analysis of Tree Algorithms for the Simulation Event List. Acta Informatica, Vol. 22, Fasc. 1, 1985

[6] CCITT: Recommendation Z.200. Red Book, Geneva, 1985

[7] CCITT: Recommendations I.440 and I.441. Red Book, Geneva, 1985

[8] E.-H. Goeldner, H.L. Truong: A Simulation Study of HDLC-ABM with Selective and Nonselective Reject. Proceedings 10th ITC, Montreal, Canada, Paper 3.4-5 (1983)

[9] W. Dieterle: A Simulation Study of CCITT X.25: Throughput Classes and Window Flow Control. Proceedings 10th ITC, Montreal, Canada, Paper 3.3-6 (1983)

[10] W. Fischer, K.P. Sauer, W. Denzel: A Simulation Technique for Communications Protocols Based on a Formal Specification by SDL. in "Protocol Specification, Verification and Testing, V" (North Holland), 1985

[11] C.H. West: General Technique for Communications Protocol Validation. IBM J. Res. Develop., Vol. 22, No. 4, July 1978

[12] P. Gerrand: Applications of Processing State Transition Diagrams to Traffic Engineering. Proceedings 8th ITC, Melbourne, Australia, Paper 313 (1976)