ELSEVIER Simulation Practice and Theory 6 (1998) 397–411

SIMULATION PRACTICE AND THEORY

Modeling and quantitative analysis of discrete event systems: A statecharts based approach

P. Gruer *, A. Koukam, B. Mazigh

Laboratoire de Recherche en Informatique de Sévenans, Institut Polytechnique de Sévenans, 90010-Belfort Cedex, France

Received 28 December 1996; revised 1 July 1997

Abstract

Development and implementation of new computer networks as well as modification of existing ones involve designers with problems of complexity and cost. Modeling and quantitative analysis, which allow performance evaluation of such systems before tackling their implementation, can alleviate these problems. Toward this end, we propose a modular approach based on dynamic behavior modeling and performance evaluation. It advocates the use of an architectural description of the network as a set of interrelated components from which the dynamic behavior is investigated in a bottom up fashion. The dynamic behavior is expressed in the statecharts formalism; an extension of finite state automata. In addition to the usual definition of statecharts, we introduce the time delays associated with actions which we specify probabilistically to model the system under uncertainty. This approach is developed through the modeling and quantitative analysis of a wide area network.© 1998 Elsevier Science B.V. All rights reserved.

Keywords: Object oriented modeling; Statechart; Performance evaluation; Simulation

1. Introduction

Over the past few years, computer networks have received an increasing amount of attention owing to their use in the development of applications such as distributed systems. As they become more complex, it is increasingly clear that modeling approaches for quantitative and qualitative analysis of network systems are needed before their effective development. Qualitative analysis provides the system designer with information about whether the system satisfies a set of properties such as reachability, usability and liveness. On the other hand, quantitative analysis focuses on the evaluation of system's performance, safety and reliability.

Many efforts in modeling for qualitative and quantitative analysis have concen-

* Corresponding author. Fax.: +33-3-84583065; e-mail: pablo.gruer@utbm.fr.

trated on the behavioral model specification, using formalisms such as the Petri nets [7–9]. Methodological efforts such as global models [3], and use of T-invariants [12], have focused on how to avoid the state space explosion. The purpose of this paper is to present an approach in which the modeling of the structural and behavioral aspects [2], are combined to allow the quantitative analysis of computer networks. The structural model describes the system as a set of inter-connected components and acts as a useful guide to model the dynamic behavior in a bottom-up fashion. This allows us to build progressively the dynamic behavior model of the overall network by combining the behaviors of their components. The dynamic behavior description is achieved through the use of the statecharts [4]; an extension of finite state automata which provides hierarchical structuring of states, explicit representation of concurrency and broadcast communication. This paper shows how to apply this approach to the modeling and quantitative analysis of a wide area network. We are primarily interested in using quantitative analysis to evaluate the performance of this network. Message delay, load and throughput are used as performance indices and simulation is used in the evaluation phase. Though statecharts have been widely used for qualitative analysis, they lack features to support the performance and dependability aspects, namely quantitative analysis. We show how to combine some statechart constructs to define in natural way these aspects and to model the stochastic behavior of discrete event systems.

This paper is organized as follows. Section 2 presents an overview of statecharts and describes how to adapt them for quantitative analysis. Section 3 focuses on the modeling of the wide area network. Section 4 is devoted to performance analysis, by showing how performance indices are defined using the behavioral model, and then evaluated using the Statemate® environment [5]. The last section is devoted to a summary and concluding remarks.

2. Adapting statecharts for quantitative analysis

Statecharts provide high level constructions to model the dynamic behavior of Discrete event Systems. This section reviews some of these constructions and shows how to combine them to allow quantitative analysis. In particular, it introduces the time delays associated with actions of the system and a solution to resolve structural conflict.

2.1. An overview of statecharts

Statecharts are an extension of finite state automata which provides hierarchical decomposition of states, explicit representation of concurrency and broadcast communication. The hierarchy results from the Or/And decompositions which allow a single state at a higher level of abstraction to be refined on a set of states. Fig. 1 gives an illustration of these features. The superstate A describes a parallel composition of two concurrent processes defined by the states B and C. This is represented by a broken line which specifies an And decomposition of the state A. Hence being

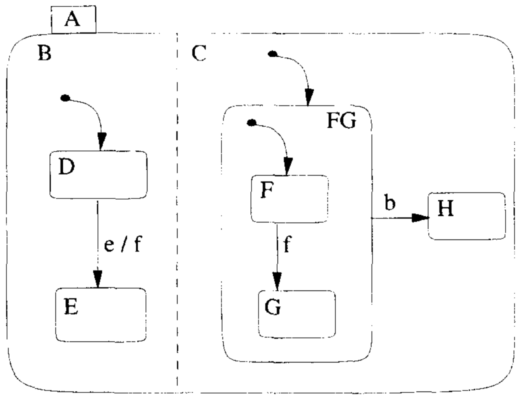

Fig. 1. A simple statechart.

in the state A entails being in both B and C states. The state B is refined to consist of two states, namely D and E. It is called an Or-state: to be in B one must be in either D or E but not both.

The states D and FG are annotated by a small arrow with no source state. The pair (D, FG) represents the default or start state for the statechart A. Transitions between states are represented by arrows and are caused by events. The general syntax of a transition is: e[c]/a, where e is an event, c is a condition and a is an action. This means that the transition fires when the event e occurs and the condition c is true. The action a is carried out when the transition is taken. Notice that the transition b which leaves the Or-state FG applies to both F and G. This shows the ability to have transitions leave and enter states on any level and then reduces the number of transitions. Actions can also be associated with the entrance to or exit from a state. For example, if we associate the action St(a) with the entrance to state S, activity a will be started whenever S is entered. The communication is based on an instantaneous broadcast mechanism. For example, if the statechart A is in the configuration (D, F) and the event e occurs, then the action f is immediately activated as a new event, triggering the transition from F to G. The new configuration will be (E, G). If necessary, instead of being activated immediately, an action can be activated after a time delay. To this end, function sc!, a shorthand for schedule, is used. For example, a transition e/sc!(f, 5) specifies that event f is activated 5 time units after the triggering of the transition by event e.

Note that statecharts provide a set of predefined events, conditions, actions and other features which we do not consider here. Only those used in this paper are described.

2.2. *Modeling of timed actions*

Statecharts provide means of dealing with qualitative analysis of discrete event systems. They use a logical model of time based on a partial order on event occurrences, subject to restrictions such as causality. However, for quantitative

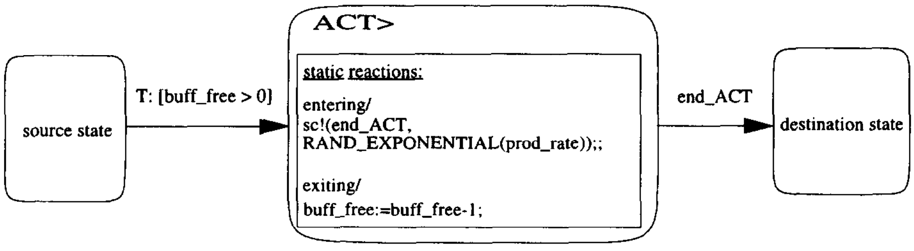

Fig. 2. A statechart model for a timed action.

analysis, it is necessary and useful to introduce time delays associated with actions to be performed by the system. Additionally, we are interested in specifying the delays randomly to model the system under uncertainty.

Models such as those proposed by stochastic Petri nets [1], introduce timed transitions to represent the delay associated with actions. A timed transition has an exponentially distributed firing time that expresses the delay from the enabling to the firing of the transition. This solution is not feasible here as it is in conflict with the formal semantic of statecharts [6]. Indeed, a transition in statecharts should be instantaneous. We overcome such a problem by associating the time delay to states or, in other words, we represent actions as states. In addition, we define modeling capabilities to represent two optional important features: first, associating a precondition to the modeled action and second, defining the effect of the action on the value of the state variables of the model.

We illustrate the proposed approach to the modeling of timed actions through an example: consider an action ACT that consists of producing data items and storing them in a data buffer. A new data item can be produced only if there is a free location in the buffer. Producing a new data item consumes an amount of time that is uncertain. Nevertheless, the production rate *prod_rate*, i.e., the average number of data items produced per unit of time by action ACT, is known. Fig. 2 illustrates the proposed construct, based on static reactions associated with state ACT>, which represents the action execution. In statechart notation, the symbol '>' following the state identifier indicates that static reactions have been associated with the state. We introduce state variable *buff_free* to represent the number of free locations in the buffer. Transition *T* activates state ACT>, if precondition [*buff_free*>0] is verified. A static reaction, executed upon entering ACT>, consists in delaying an occurrence of event *end_ACT*. To determine the delay amount, the predefined rand_exponential function is used. Consequently, state ACT remains active for a duration determined randomly, with an exponential law. The effect of the action ACT on state variable *buff_free* is represented by the static reaction executed upon exiting the state: as a new data item has been stored in the buffer, the number of free locations decreases.

2.3. Structural conflict

Structural conflicts represent non deterministic situations where many mutually exclusive actions can be started. One of those actions is randomly selected to be

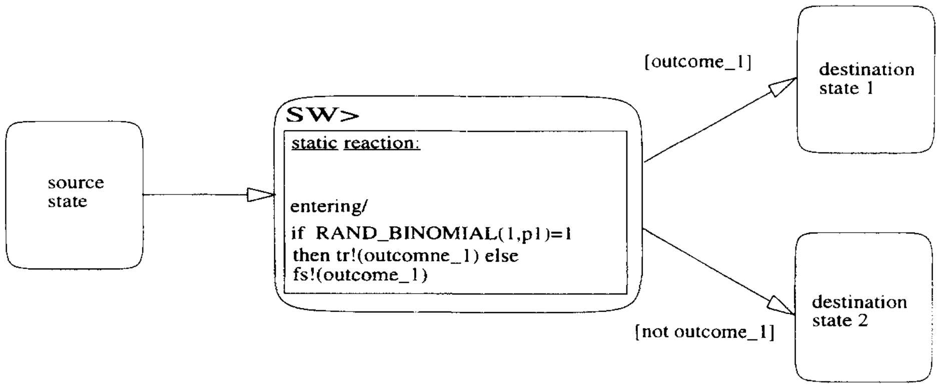

Fig. 3. A statechart model for a two-way random switch.

executed. The modeling formalism should offer means to assign probabilities to each one of the possible evolutions. Those modeling constructs are frequently called *n-way random switches*, with n equal to the number of mutually exclusive issues. Generally, an n-way random switch is considered to be instantaneous, as the probabilistic choice is supposed to take no time.

Statecharts can resolve a structural conflict with two possible issues, (i.e., a two-way random switch) in the way illustrated by Fig. 3. Upon entering state SW, a static reaction is executed. It consists in randomly assigning value *true* or *false* to a boolean variable *outcome_1*. The standard function rand_binomial with parameters $n=1$ and $p=p_1$ is used to this end. The value of boolean variable *outcome_1* is then used in conditions [*outcome_1*] and [not *outcome_1*] to select one of two possible outgoing transitions. Consequently, p_1 is the probability of *outcome_1* being assigned value *true*. The proposed statechart model of a two-way random switch can be considered to be instantaneous, provided that the asynchronous time model is adopted during simulation. If, on the contrary, the synchronous time model is adopted, the random switch operation will consume one unit of simulated time.

3. The wide area network model

The aim is to build a model for quantitative analysis of a wide area network (i.e, WAN), made up of two local area networks (i.e, LAN). Each LAN has a number of nodes or computers which communicate through a token ring; a communication network with a loop topology. Transmission contention within the loop is resolved by a token, a particular message circulating from node to node. The possession of the token gives access to the medium. Nodes located in different LANs communicate by a remote link. Before the elaboration of the WAN dynamic model that we will use as a basis for quantitative analysis, we first define the WAN components and their interaction.

3.1. The WAN components

One of the main factors which contribute to the complexity of network systems is the increasing number of their components and the complexity of the relationships among them. A useful modeling approach must provide the means to identify the system components and how they interact. This will enhance the features and abilities to capture the dynamic model. Object oriented analysis which allows one to model systems as objects and their interrelationships provides a better support for modeling the system structure. A number of object oriented approaches have been proposed. They describe conceptually similar definitions, although they adopt alternate terminologies [8]. We have chosen to follow the approach proposed by Shlaer and Mellor [10] to model the structure of the WAN. This approach describes the structure of the system by providing two models, namely, the information and object communication models. The former specify the system in terms of objects, attributes and relationships, while the latter shows objects and messages exchanged between them. We define the structural model of the WAN system by combining the information and object communication models as shown in Fig. 4.

We have represented the WAN system as an aggregation of LAN and Link objects.

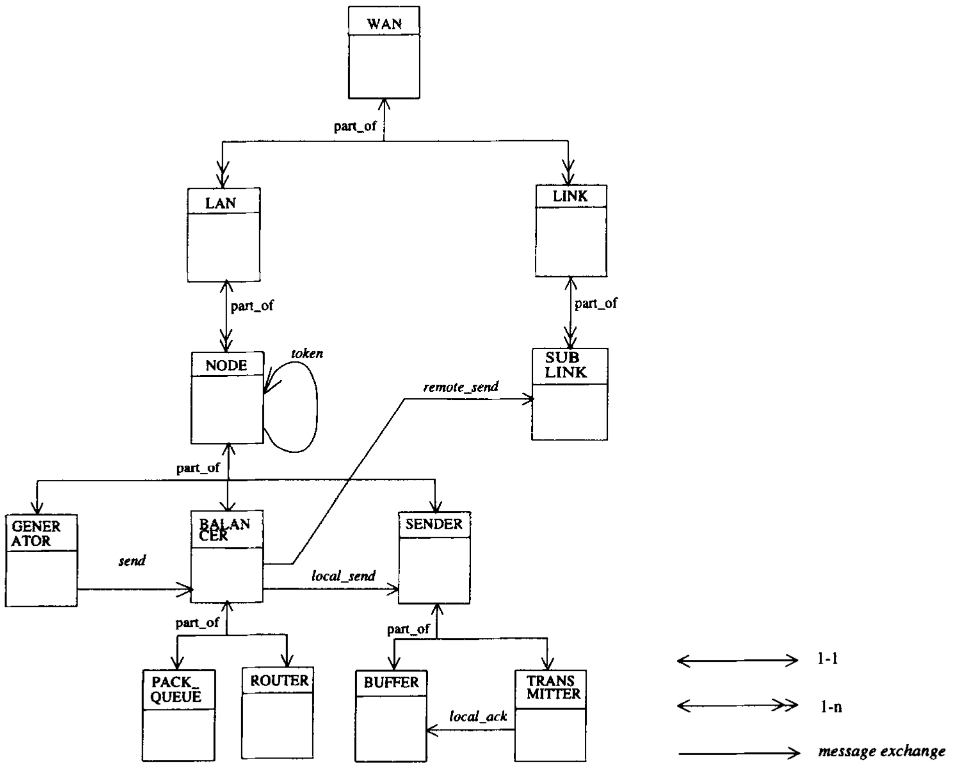

Fig. 4. The WAN structural model.

A LAN object is made up of a number of nodes. Each node is an aggregation of three objects, namely, the Generator, the Load Balancer (or more briefly, the Balancer) and the Sender. The Generator is a packet generating entity that takes data and formats it into packets ready to be sent. The Balancer is an entity that distributes the load (i.e., the produced packets) over the WAN. In particular, the Balancer determines whether a packet, once produced, is to be sent through the LAN or through the Link. Two objects are parts of the Balancer: the Packet Queue and the Router. The later applies the load balancing policy, to choose between delivery through the LAN or through the Link. The Sender is the local communication object and involves two entities: the Buffer object and the Transmitter object. The Buffer preserves packets which are waiting to be sent. The Transmitter takes packets from the Buffer and sends them to a destination node in the local LAN. The Link object takes packets in a LAN and sends them to a destination node in the remote LAN. The Link is a bidirectional communication object that results from the aggregation of two unidirectional Sublink objects.

Notice that some arrows are labeled by names in italics. They represent the message exchanges between objects. As an example, after the Generator generates a packet, it requests the Balancer to route it towards the LAN or the Link. This is done by the intermediate of the *send* message. Depending on the routing policy and on the state of the system, the Balancer uses the *local_send* message to request the sender for a local delivery of the packet, or else the *remote_send* message to request the outgoing Sublink for the remote delivery of the packet. The Transmitter uses the *local_ack* message to signal the Buffer that a packet has been successfully sent and that it can be removed or replaced by another packet.

Once the structural model has been produced, attention turns to the behavioral model elaboration which describes the dynamic behavior of the WAN according to timing information. This task follows a bottom-up approach. We first model the behavior of individual components and then combine them to build progressively the behavior of the overall system using the aggregation relationship.

3.2. The node behavioral model

As a LAN includes a number of identical nodes, we describe first the behavioral model of a single node and then transform it to obtain the global model of a LAN.

Fig. 5 presents the behavior of node i. The nodes are numbered from 0 to $N-1$. The behavior of a node is defined by the parallel composition of the Generator. Balancer and Sender behaviors. A state that represents an action is identified by the '>' symbol following the state identifier. The packet generating process (i.e. Generator) waits for the packet queue to be free (precondition [not queue(i)]) to generate a new packet (action Generating). When the packet is ready the *end_gen(i)* event is generated and the Generator uses the *send* message to requests the Balancer to route the packet to a destination. The *send* message is modeled as an event emission, as indicated in the transition from action state Generating to state Wait_on_q.

The Balancer applies the load balancing policy: a packet that waits in the packet

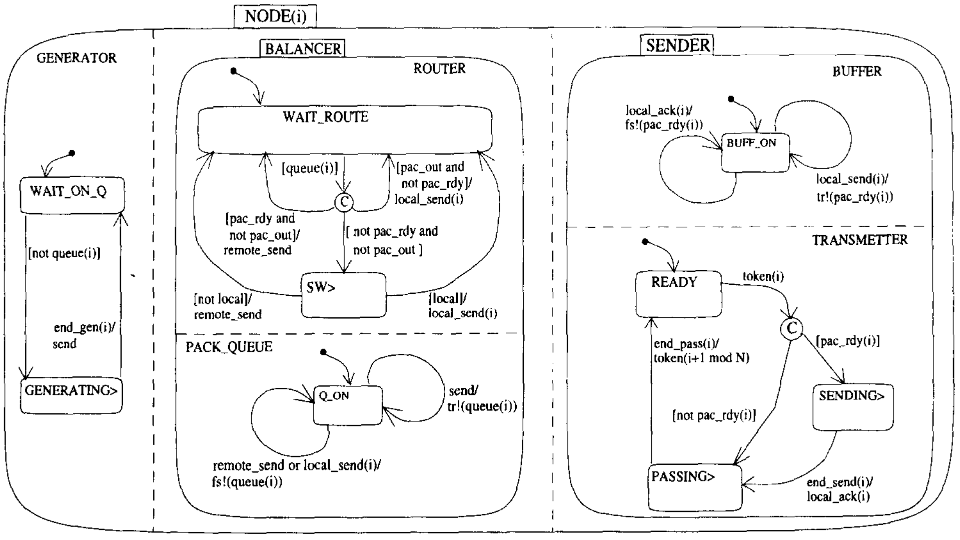

Fig. 5. Behavioral model of a node.

queue must be sent as soon as the local Sender is ready (i.e. pac_rdy = false) or the out-going Sublink is free (i.e. pac_out = false). In the first case, the packet is delivered to another node of the local LAN. Otherwise, it is sent to the remote LAN through the Link. If the local Sender and the Sublink are simultaneously free, the Router decides randomly whether the packet is sent to a local or to a remote destination. This is modeled by state SW that represents a two way random switch. Fig. 6 presents the static reactions associated to the states that model actions, and to the state SW.

As an example of static reactions, we consider the Generating state. Upon entering the state, an occurrence of the *end_gen(i)* event is scheduled using the rand_exponential function, in order to calculate randomly the duration of the action. Parameter

```
GENERATING:
entering/sc!(end_gen(i), RAND_EXPONENTIAL(gen_rate));;
PASSING:
entering/sc!(end_pass(i), RAND_EXPONENTIAL(pass_rate));;
SENDING:
entering/sc!(end_send(i), RAND_EXPONENTIAL(send_rate));;
SW:
entering/if RAND_BINOMIAL(1, p_local)=1 then
        tr!(local) else fs!(local) endif ;;
```

Fig. 6. Static reactions of active states of the node model.

gen_rate represents the mean generation rate. The occurrence of the *end_gen(i)* event indicates the completion of the action and, consequently, causes the Generator process to exit the Generating state.

3.3. The LAN behavioral model

The LAN behavioral model results from the combination of the behaviors of their nodes. To determine how to combine the node behaviors, we ascertain that nodes in a LAN constitute concurrent entities. From this, the straightforward approach to obtain the LAN's model is by concurrent composition of N instances of the node's behavior, as shown in Fig. 7 (for $N=3$). The main drawback in doing so, is that the resulting model has a rapidly growing number of global states, as the number of nodes increases.

In this paper we prefer to define an approximate model of the LAN behavior, called the global model: under the assumption that individual nodes are statistically identical and independent, a unique modified node model is used to represent all the nodes in the LAN. This consists of introducing a new state variable in the behavioral node model. More precisely, we introduce the following state variable:

– integer variable ST_ON represents the number of node stations that are ready to produce a packet. Static reactions associated to the active states Generating and Sending are modified as shown in Fig. 8, in order to manage state variable ST_ON: a node station is not ready from the instant when it begins to generate the packet, up to the end of the packet sending.

– the boolean array queue(i) is replaced by integer variable Q i.e, the total number of packets waiting in the packet queues. Action fs!(queue(i)) is replaced by $Q:=Q-1$ and tr!(queue(i)) by $Q:=Q+1$. Precondition [not queue(i)] is replaced by [ST_ON>0] to indicate that the new precondition for packet generation is that at least one node is ready.

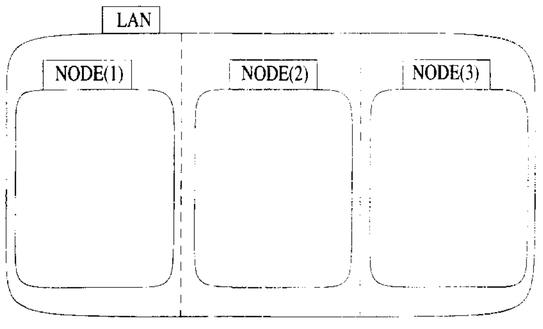

Fig. 7. Model of a token ring.

```
GENERATING:
entering/
sc!(end_gen(i), RAND_EXPONENTIAL(gen_rate));
ST_ON:=ST_ON-1;;


SENDING:
entering/sc!(end_send(i), RAND_EXPONENTIAL(send_rate));;
exiting/ST_ON:=ST_ON+1;
```

Fig. 8. Static reactions of actions generating and sending.

3.4. The WAN behavioral model

The WAN behavioral model is defined by the concurrent composition of the behaviors of two LANs (LAN1 and LAN2) and the behavior of a Link that models the communication between the two LANs. Fig. 9 presents the behavioral model of the WAN.

The Link's behavior is composed of two processes. Sublink1 models communication from LAN1 to LAN2 and Sublink2 models communication in the other sense. The Link has two buffers for incoming packets (modeled by state variables pac1_in and pac2_in) and two buffers for outgoing packets (variables pac1_out and pac2_out). As an example, consider the Sublink1 process: if it is idle (state SL1W) and there is a packet in its output buffer (pac1_out = *true*) and its input buffer is empty (pac1_in = *false*) then the packet can be sent from LAN1 to LAN2. The action that consists in sending a packet is represented by state SL12S.

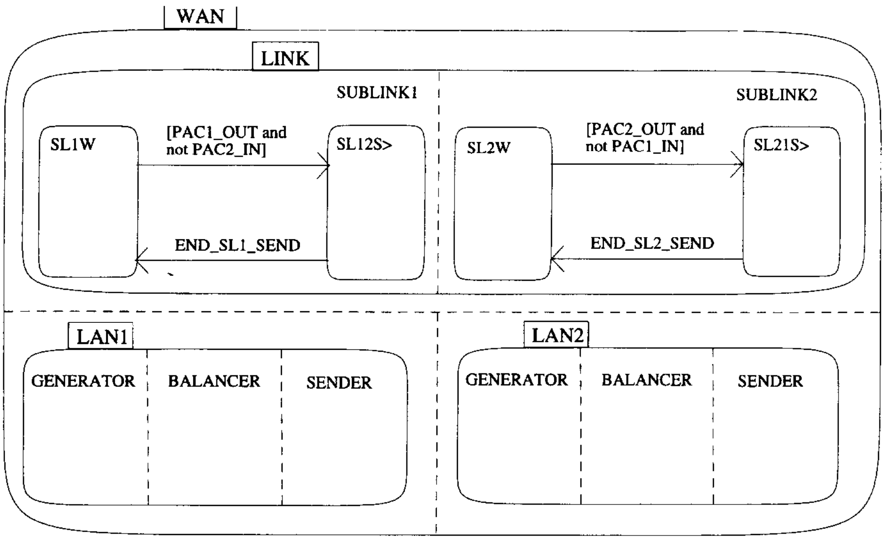

Fig. 9. Behavioral model of the WAN system.

4. Simulation of the distributed system

Based on the behavioral model described previously, we perform quantitative analysis of the WAN system. Quantitative analysis consists in calculating the performance indices, to give useful information about the operation of the WAN. Values of performance indices are calculated by simulating the behavioral model, under different operating conditions that will be presented hereafter. To this end, we use Statemate [11], an industrial environment of i-Logix, which allows the graphical representation and the simulation of the statechart model. In Section 4.1 we define some performance indices and we determine how to calculate them from the behavioral model. In Section 4.2 we show how to control the simulation in order to evaluate the performance indices.

4.1. Performance indices definitions

The basis for the calculation of the performance indices from the statecharts model, is the computation of event rates. Generally, a good estimation of the mean rate of an event is obtained by counting the number N of occurrences of the event over a reasonably long period of time T. The mean rate estimate is obtained by dividing N by T.

We define performance indices of load and throughput, that are related to event rates. In what follows, exit(S) represents the event 'state S was exited'. If *ev* identifies an event, $O(ev)$ represents the number of occurrences of *ev* during simulation.

Offered load L_i of LAN i ($i=1, 2$) is defined as the average number of packets generated in LAN i, per unit time. Load L_i is defined as the total number P_i of generated packets divided by the total simulation time T_s. The number P_i is determined by counting, during simulation, the occurrences of the event 'state GENERATING of LAN i has just been exited':

Offered load of LAN i ($i=1, 2$):

$$L_i = \frac{P_i}{T_s} = \frac{O(\text{exit}(\text{GENERATING}))}{T_s}$$

Local throughput τ_i of LAN i ($i=1, 2$) is defined as the average number of packets delivered locally in LAN i, per unit time. Load L_i is defined as the total number D_i of delivered packets divided by the total simulation time T_s. The number D_i is determined by counting, during simulation, the occurrences of the event 'state SENDING of LAN i has just been exited':

Local throughput of LAN i ($i=1, 2$):

$$\tau_i = \frac{D_i}{T_s} = \frac{O(\text{exit}(\text{SENDING}))}{T_s}$$

Remote throughput $\rho_{i,j}$ from LAN i to LAN j ($i,j=1, 2$ or $2, 1$) is defined as the

```
PROGRAM  offered_load;

-- Constant declarations
CONSTANT
INTEGER sim_duration := 50000;

-- Variable declarations
VARIABLE
INTEGER exit_gen_count := 0;

-- The next section includes  breakpoint definitions: SCL statements
-- to be executed conditionally, at each simulation step, if the trigger
-- is true.
SET BREAKPOINT ex(GENERATING1) DO
    exit_gen_count := exit_gen_count + 1;
END BREAKPOINT;

-- Next comes the main section, in which to place any SCL statements, to
-- be executed sequentially.
BEGIN
   SET INFINITE LOOP 100000;
   GO STEP;
   WHILE cur_clock < sim_duration LOOP
        GO EXTENDED
   END LOOP;
   load1:=exit_gen_count / sim_duration
END;

END.
```

Fig. 10. Simulation control program to calculate offered load L_i

average number of packets delivered by LAN i to LAN j, per unit time. Throughout $\rho_{i,j}$ is defined as the total number $D_{i,j}$ of packets delivered from LAN i to LAN j, divided by the total simulation time T_s. The number $D_{i,j}$ is determined by counting, during simulation, the occurrences of the event ‘state SLijS has just been exited’:

Remote throughput of LAN i to LAN j ($i,j=1$, 2 or 2, 1):

$$\rho_{i,j}=\frac{D_{i,j}}{T_s}=\frac{O(\text{exit(SLijS)})}{T_s}$$

Once the performance indices have been defined, we indicate how to determine their value by simulating the operation of the system model.

4.2. Evaluating performance indices by simulation

The Statemate simulator operates under the control of a simulation control program (SCP); a sequence of simulation control language (SCL) commands [11] grouped in sections. We use SCP to calculate the performance indices during simulation.

Fig. 10 illustrates how to calculate the offered load of LAN1, according to the definition given in Section 4.1. To count the occurrences of event exit (GENERATING1), we use the breakpoint mechanism to increment variable exit_-gen_count. Breakpoints are SCP sections including two elements: a trigger and a list of SCL commands. At each simulation step, breakpoint triggers are evaluated and if true, the command list of the breakpoint is executed. The breakpoint definition in Fig. 10 has ex(GENERATING1) as its trigger. The later evaluates to true if state Generating1 is exited during the current simulation step. In that case, variable exit_gen_count is incremented.

The main section contains some general-purpose SCL command. The *set infinite loop* avoids the simulation halting because of cycle detection by the simulator. The GO commands allow simulation to progress. Identifier *cur_clock* designates a predefined simulator variable containing the current simulation clock value (current time) at any instant.

Our aim is that quantitative analysis results provide guidelines to a better implementation of the network application. To this end, we investigated the sensitivity of the performance indices relatively to changes of value of certain system parameters. Particularly, the effect of the following factors on local and remote throughput have been investigated: the probability p for a message to be addressed via the link, to the remote LAN and the number N of active stations per LAN. The following numerical values for the average delays of timed actions have been used: generation delay = 0.01 time units (tu), local delay = 0.001 tu, token passing delay = 0.001 tu, LAN to LAN transfer delay = 0.01 tu. Figs. 11 and

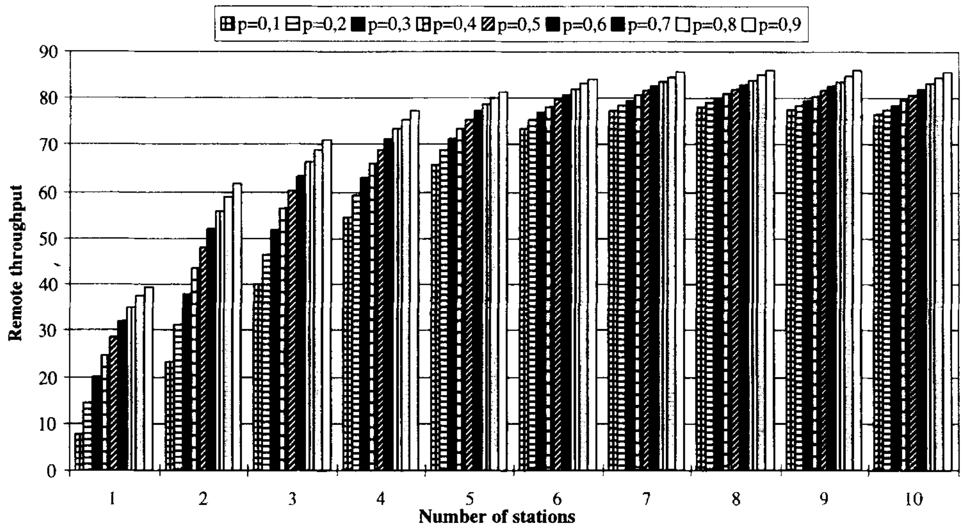

Fig. 11. Remote throughput (LAN_i to LAN_j) versus the number of stations per LAN.

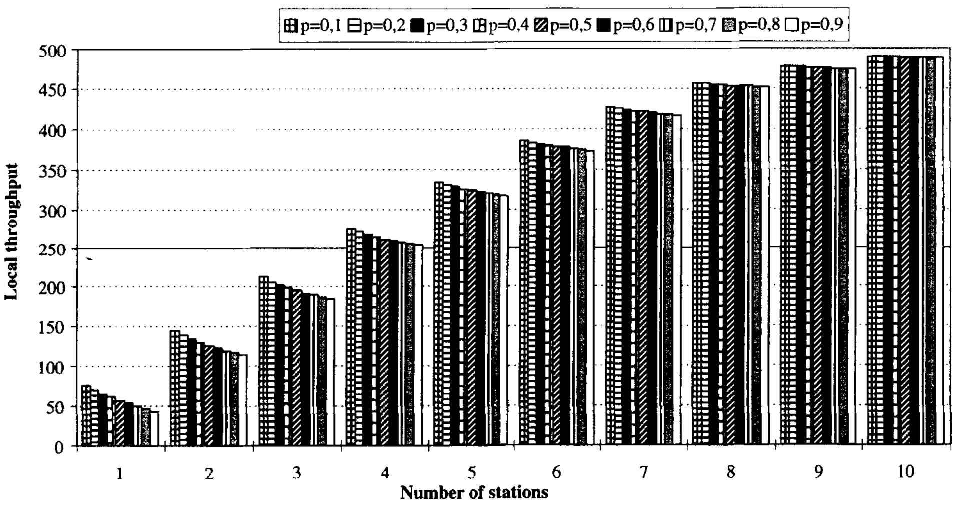

Fig. 12. Local throughput of LAN_i versus the number of stations per LAN.

12 presents the results of simulation, for N ranging from 1 station to 10 stations and p ranging from 0.1 to 0.9. As expected, local throughput decreases as the probability for a message to be remotely delivered increase. The opposite is verified by the remote throughput. Throughput also increases with the number of stations per LAN, but remote throughput seems to reach a ceiling at about 5 stations per LAN, while local throughput continues to grow beyond this number of stations.

5. Conclusion

Through the WAN system, we have considered how object oriented paradigms and statecharts can be combined to provide an approach for modeling and quantitative analysis of discrete event systems. Object oriented paradigms are used to model the system components and their interrelationships. Based on this model, statecharts are then used to model the behavior of the system in bottom-up fashion. We have proposed a solution to specify probabilistically the time delays associated with actions and to resolve the structural conflict. To perform the performance evaluation of the WAN, we have used the Statemate environment.

This work must be extended to define some rules and well defined steps leading the user to progressively model the given system.

References

[1] M. Ajmone-Marsan, G. Balbo, G. Conte, A class of generalized stochastic Petri nets for the performance analysis of multiprocessor systems, ACM Trans. Comput. Syst. 2 (2) (1984) 93-122.

[2] P. Gruer, A. Koukam, Hierarchical and Formal Modeling of Object Behavior, Proceedings of the S.C.S. Conference on Object Oriented Simulation, Las Vegas, 1995.
[3] R.K. Guha, M. Bassiouni, D. Roy, Modeling and Evaluation of Distributed Simulation with a Wide Area Network Architecture, Proceedings of the SinTec'92, 1992.
[4] D. Harel, Statecharts: A visual formalism for complex systems, Sci. Comput. Program. 8(3) (1987).
[5] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtul-Traurig, STATEMATE: A working environment for the development of complex reactive systems, IEEE Trans. Software Eng. 16(4) (1990).
[6] D. Harel, A. Pnueli, J.P. Schmidt, R. Sherman, On the formal semantics of statecharts, Proceedings of the first IEEE Symposium on Logic in Computer Science, 1986, pp. 54-64.
[7] M.K. Molloy, Performance analysis using stochastic Petri nets, IEEE Trans. Comput. 41(9) (1982).
[8] D.E. Monarchi, G.I. Puhr, A research typology for object-Oriented analysis and design, Commun. ACM 35(9) (1992).
[9] T. Murata, Petri nets: Properties, analysis and applications, Proc. IEEE 77(4) (1989).
[10] S. Shaler, S. Mellor, Object-Oriented System Analysis: Modeling the World in Data, Prentice Hall, New York, 1991.
[11] STATEMATE Version 6.0 Analyser Reference Manual, i-Logix, 1995.
[12] F.-Y. Wang, H. Jungnitz, K. Gildea, Performance Analysis of MMS using GSPN in: Proceedings of the 1991 IEEE Conference on Robotics and Automation, Sacramento, CA, 1991.