Proceedings of the 1986 Winter Simulation Conference
J. Wilson, J. Henriksen, S. Roberts (eds.)

Translation of Modified Predicate Transition Nets Models of Communication Protocols Into Simulation Programs

*Chuang Lin**
Dan Cristian Marinescu

Computer Sciences Department
Purdue University
West Lafayette, IN 47907, USA

ABSTRACT

The use of modified predicate transition nets for the modeling of computer communication protocols has been investigated by the authors in reference [43]. The translation of such models into simulation programs is examined in this paper. Though protocols for all layers of a computer communication architecture can be modeled using the technique presented in this paper, we focus on the network layer protocols and we treat mechanisms like: fragmentation and reassembly, routing, store and forward buffering and congestion control.

Non-prime transitions can be embedded into the modified predicate transition nets hence a top-down modeling is possible. The present paper uses a hierarchical modeling approach to describe complex communication mechanisms in terms of simpler constructs. Models developed using the modified predicate transition nets can be automatically translated into simulation programs. The paper presents the translation of a computer communication model into a program, using a process oriented simulation language, ASPOL.

Key Words: Communication Protocols, OSI Architecture, ASPOL, modeling, predicate transition nets, Petri nets, synchronization, simulation, concurrent execution.

1. Introduction

The Open-Systems Interconnection (OSI) reference model [1] is a framework for defining standards for interconnection of heterogeneous computers. It partitions the computer communication functions into a vertical set of seven layers. Each layer provides services to the next higher layer, and relies on the next lower layer to perform more primitive functions. Layer *n* on one host carries on a conversation with layer *n* on another host. The rule and conventions used in this conversation are collectively known as a (peer-to-peer) protocol.

For computing systems and especially for systems of the complexity of a computer network, modeling is essential during system design, implementation and throughout the entire system life. In connection with computer communication protocol design, modeling is necessary in order to predict the feasibility and the performance of the protocols.

Simulation can be used to evaluate the model of a system after the model has been mapped into a computer program which describes the interesting behavior of the system. A simulation model of a computer communication architecture emphasizes a sufficiently precise representation of protocols and protocol hierarchies.

Very often a hierarchical modeling is possible, a complex system can be decomposed into subsystems. Each subsystem can be modeled separately and its model can be evaluated independently. The behavior of the entire system can be investigated by combining the previous results.

In recent years a considerable effort has been invested into modeling and analysis of computer communication protocols. Two approaches to modeling, namely the finite state automata approach [8] and the simulation language approach [9-12] have been investigated in the past. More recently queueing models have been also considered [13,41]. The Petri nets [17-22] are used mostly for the specification and validation of protocols [6,7], but they seem a natural tool for protocol performance analysis [14-16].

Predicate-transition nets [23-25], enhanced with stochastic transition time, are proposed as a suitable framework for modeling the communication protocols in a computer network. In order to illustrate this, we have considered some common mechanisms used by the network layer of the ISO reference model. In the first section of the paper a modified predicate-transition net is presented. In the second section, the computer network model based upon the modified predicate-transition nets is analyzed. Finally, the mapping of the network model into program structure is discussed.

2. Modified Predicate-Transition Nets

The Petri nets used in this paper are a modified version of the Predicate-Transition Nets and Generalized Stochastic Petri Nets. The elements of the Modified Predicate-Transition Nets are:

(1) **Token types**

In the modified nets there are three types of tokens.

Type 1 Tokens with no attribute. They are used to represent the synchronization operations among concurrent activities.

Type 2 Tokens with one attribute only. The attribute is an integer which may represent for example the value of a counter.

Type 3 Tokens with multiple attributes. The first item in the attribute record is used to identify the token subtype. Each subtype represents a different data unit exchanged by the communication protocols. We recognize the following subtypes: message, packet, etc. A token representing a message has three attributes, subtype=m, destination address a, and message length l. A token representing a packet has six attributes: subtype=p, the packet sequence number s, the destination address a (copied from the message), the packet length ll, the last packet in a message flag la, and the neighbor to neighbor packet

* On leave from State Planning Committee, Beijing, China

sequence number s1. A token conversion process occurs whenever the corresponding data unit crosses the boundary of a communication layer and it corresponds to the encapsulation and decapsulation process performed by the different protocol layers.

(2) **Places**

They are the first type of nodes in the graphs. A place is represented by a circle and has a name consisting of one or more upper-case characters. A place can contain a finite number of tokens, determined by its capacity. We distinguish two types of places: places of the first type are used to represent the queues of interest in the performance evaluation of system and they contain Type 3 tokens. Places of the second type are used to represent system environments.

(3) **Transitions**

Transitions are the second type of nodes in the graphs. A transition is represented by a bar or a rectangle and has a name consisting of one or more upper-case characters. Nonprime transitions representing model subnets are represented graphically as rectangles. This possibility enables top-down hierarchy structuring and information hiding. A similar structuring method for distributed software system design has been discussed in [34]. Predicates can be inscribed on transitions: they specify relations between variables of different tokens from the input places of the transitions. It is possible to evaluate the interesting performance aspects in the modelled system by introducing "time" into the transitions [14, 26-33]. There are two types of transitions: timed and immediate. An exponentially distributed random time (possibly marking-dependent), denoted by T( ), between enabling and firing of a transition can be associted with timed transitions only.

(4) **Arcs**

Arcs connect transitions and places together. The arcs are labelled with token variables. Operations can be inscribed on outgoing arcs of transitions. An operation performs a computation or a change of the attributes carried by the tokens from the input places, and it forms tokens to be inserted into the output places. In our modified nets multiple arcs between a place and a transition are allowed, and the number of such arcs may vary from zero to a finite value. In the followings we will show that this extension does not change the property of the predicate-transition nets, but it makes more convenient to describe some mechanisms of the protocols. The *variable arc* is represented graphically by an arc crossing a small circle, and has an associated range expression.

(5) **Firing rule**

A transition is enabled when there is at least one token in each of its input places and the predicate associated with the transition is satisfied. If there are the multiple arcs between the input place and the transition, then the transition is enabled when the number of tokens in the input place is at least equal to the number of arcs and the predicate is satisfied. An immediate transition fires immediately after being enabled. When the timed transition is enabled, it fires after a random enabling time, x. The enabling time, x, is exponentially distributed and it is specified by the T(x) associated with the transition. However, the firing of a transition is considered to be instantaneous, so the probability of two or more transitions firing simultaneously is zero. The firing of a transition consists of two steps:

a) removing the tokens (satisfying the predicate) from the input places;

b) adding tokens to the output places after performing the operations specified on the outgoing arcs.

In our model the distribution of the time x associated with a timed transition depends upon the system environment. The main difference between the nets used in this paper and the predicate-transition nets are the introduction of time and of variable arcs (a variable number of arcs may connect a place and a transition, and this number may range from zero to a finite value).

In our paper the following notations are used: $\oplus$ stands for addition modulo the size of the sequence number. [x,y] denotes the set of integers from x to y, including the lower and the upper bound while (x,y] excludes x and [x,y) excludes y. |{ }| denotes the cardinality of a set. For example, |{a, e, d}| is equal to 3.

In our nets, we often use a place representing a counter as an environment state variable in order to control the probability of firing a transition. Such values are presented in the predicates as individual token attributes. From the reference [14], we know the bounded place marking can be tested. The value of the counter or the set element in the model can also be tested and changed, including testing for zero.

The variable-arc nets are useful to model the network protocols, for instance to model the fragmentation of a message of a given length.

The correspondence between a variable-arc net and a predicate transition net will be discussed in connection with an example used to model a mechanism of the network layer protocol.

Example. The Fragmentation Mechanism . The network layer breaks up a message into several packets. Two models of this fragmentation mechanism are presented in Figure 1. In Figure 1.(a) a variable-arc net is used to describe the mechanism. X represents the arc variable. It is assumed that the maximum length of a packet is 128 bits and the maximum message length is 384 bits hence X varies in the range 1 to 3.

The following transitions occur in our model:

US is a transition representing the arrival of a message from a user. Messages arrive according to a Poisson distribution with average arrival rate λ .

FR is a transition representing the fragmentation process. This transition fires only once, when the message is received, hence the firing condition for transition FR is: $(fn=0) AND\ (0<l\leq 348)$. When it fires, a number X of packet tokens are created and deposited in place PS. X depends upon the actual message length, l .

SEN is a transition representing the sending of the next sequential packet to the output queue(place OUT).

We now focus our attention upon the places present in our variable-arc net and upon the attributes of each type of token allowed in any given place.

SQ is a place containing tokens that represent messages, and have three attributes:

m : token subtype (message) ;
a : destination address;
l : message length.

PS is a place containing tokens that represent packets. Such a token has four attributes:

p : token subtype (packet);
a : destination address (copied from the message);
ll : packet length;
la : last packet in a message flag, used in reassembly.

A variable arc connects the transition FR to the place PS. Consequently a variable number of packet tokens are deposited in place PS when the transition FR fires.

S is a place containing tokens representing the sequence number counter, s. Whenever the transition SEN fires, this counter is incremented modulo M_0.

FN is a place containing the counter fn of packets left in place PS. When the transition FR fires, the value is set to X, the number of packets created from the incoming message. Whenever the transition SEN fires, the value is decreased by one.

An equivalent model for the same mechanism is presented in Figure 1.(b) but a predicate-transition net is used instead of a variable-arc net. The complexity of this model is considerably increased. Instead of a single transition FR now we have the set of $FR_i, i=1,2,3$. The predicate associated with FR is the combination of all predicates associated with FR_i . the performance of the system will be taken into account by considering the firing time of the corresponding transitions.

3. The Network Layer

In Figure 2. we describe the network layer model using non-prime transitions. Each non-prime transition represents a mechanism of the protocol and it can be extended as a subnet.

3.1. Fragmentation and Reassembly

The Fragmentation and its corresponding Reassembly model are presented in Figure 2. The Fragmentation model has been discussed earlier, in the previous example , and we focus our attention on the Reassembly model, namely how several packets are put together into a message to be delivered to the upper layer. In case of Reassembly, transition REC corresponds to the receiving of a packet, and it fires whenever a token is placed in the input place IN and the receiver counter value r equal to s, the sequence number of the packet received. This keeps the receiving packets in sending order.

The following places are presented in our model:

R is a place related to the sequence number of the next packet expected. The current value of this counter is denoted by r. Whenever the transition REC fires, the value r is incremented modulo M_0.

PR is a place containing tokens which represent packets. It is the correspondent of the place PS in the fragmentation model. It contains tokens with four attributes which have been described in the previous example. When the last packet arrives, i.e., when a token with la = 1 arrives at the place PR, then the transition RE fires. After firing, all tokens are removed from the place PR and a token of subtype message is put in the place RQ.

RQ is a place containing tokens which represent received messages, hence it is associted with the received message queue. It is the correspondent of the place SQ, the sent message queue, from our previous Example and contains tokens with identical attributes.

3.2. Routing

A distributed adaptive routing mechanism with updated triggered by major changes, for a packet switched network operating internally as a datagram is embedded in the network model presented in Figure 2. The routing table RB is represented by a place and it is updated at random time intervals, *v*, associated with firing of the transition RDU. The routing table has N entries, one for each node of the network. Each entry consists of two items, the destination node id and the the output line or the next node to which the packet is to be delivered. Update vectors are received periodically by each node from its neighbors. Each update vector contains entries with the same format. The places RB and NRB contain tokens with two attributes: x and y as described above. The arc label N indicates that there are N arcs. Functions f_1 and f_2 represent the updates for an entry in the routing table.

The routing table is used for every packet to decide which is the optimal route, and then the token associated with the packet is passed to the proper Buffering and Congestion control mechanism.

3.3. Store and Forward Buffering and Congestion Control

A rather simple congestion control mechanism is modeled in the graph presented in Figure 3. An intermediate node maintains a *low water mark*, M_l and a *high water mark*, M_h . Whenever the amount of used buffers reaches M_h , the node informs its neighbors that it will no longer accept any messages by sending a <ref>, REFUSE control packet. When the amount of used buffer space decreases below M_l , then the node becomes friendly again, sends to its neighbors a <res>, RESUME control packet and starts accepting messages.

We focus our attention upon the congestion control mechanism sketched above. In order to simplify the graph (Figure 3), we consider only one host and one neighbor node connected to the node under examination. Some attributes of the tokens representing packets are omitted in the graph, for example the attributes ll, la, s, are not represented though the tokens contain them.

The flow of the tokens representing packets is as follows: tokens are deposited by the data link entity in the place EQ. Depending upon the type of the packet, different transitions take place. For example the arrival of data packets triggers the transition T12 and depending upon the destination address, the tokens representing packets for the local host trigger transition T13 and end up in place IN while the tokens for the other hosts trigger transition T14, are transmitted to the place OUT and go through the routing mechanism previously described.

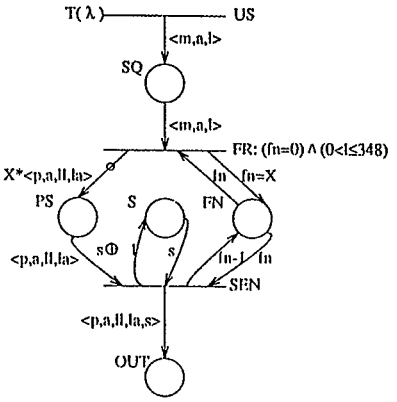
The places and attributes of the tokens they contain are:

S1 is a place representing a counter, namely the sequence number of the next packet to be sent on the outgoing link. Its current value is denoted by s1. Note that there is no need to represent the place S, the end-to-end packet sequence number.

BUF is a place representing the store and forward buffer. It contains tokens of subtype packet, with six attributes. In addition to s1 the remaining five attributes are p, a, ll, la and s are as in the case of tokens in Place OUT of the previous example.

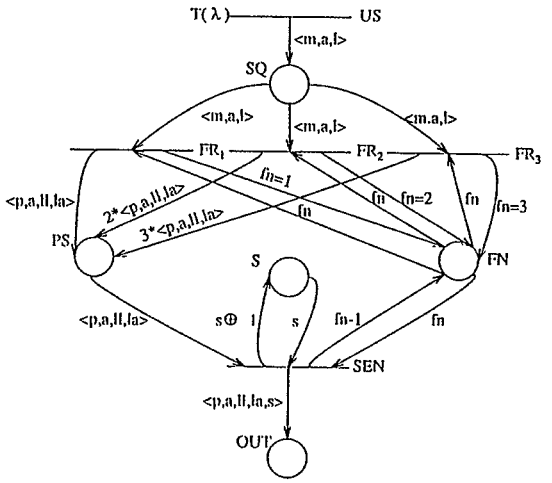
BC is a counter of the number of buffers in use, it reflects the occupancy of BUF. Its current value is *bc*. It is incremented when a new packet arrives and buffer space is allocated to it and it is decremented when the buffer space is released. It has a maximum capacity, M.

P6 is a place where tokens representing data packets are deposited.



X=INT(l/128)
where INT is a function of truncation to nearest integer not smaller than the argument

1.(a) Fragmentation modeling using a variable arc net



The following predicates associate with the transition FR_i (i=1,2,3):
FR_1: (fn=0) ∧ (0<l≤128)
FR_2: (fn=0) ∧ (128<l≤256)
FR_3: (fn=0) ∧ (256<l≤384)

1.(b) Fragmentation modeling using a predicate-transition net

Figure 1. Modeling of the fragmentation mechanism

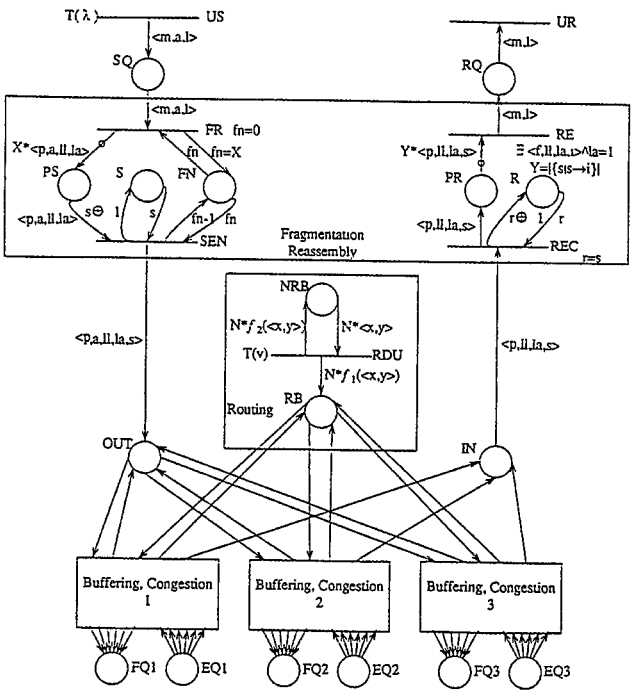


Figure 2. The network layer model with non-prime transition

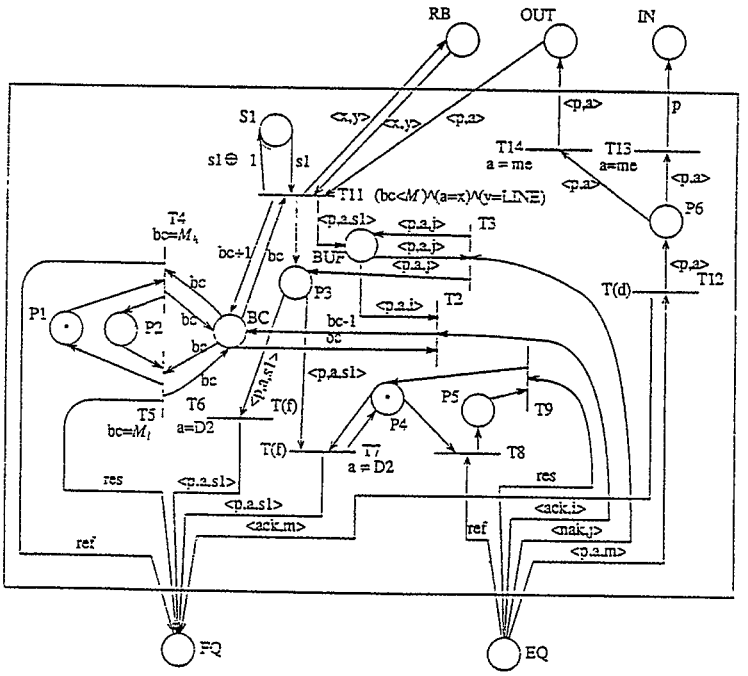


Figure 3. Store and forward buffering and congestion control model

The transitions shown in Figure 3 are described in the followings:

T2 occurs when an explicit acknowledgment for packet i, <ack,i> is received. Then the buffer holding packet i is released and a token is removed from the place BUF.

T3 occurs when a negative acknowledgment for packet j, <nak,j> is received. Then the proper token is first removed, then put back in the place BUF through the self-loop arc. In addition, when T3 fires a token is put in the place P3 which contains tokens representing packets to be retransmitted.

T4 occurs when the *high water mark*, M_h is reached and the current node sends a <ref>, REFUSE control packet to its neighbors.

T5 occurs when the *low water mark*, M_l is reached and the current node sends a <res>, RESUME control packet.

T6 if the final destination of a packet is a host connected to the immediate neighbor of the current node, the buffering of the packet is not under the control of the congestion mechanism and the transition is T6 enabled. The predicate (D2=a) is true when the immediate neighbor condition is satisfied.

T7 the enabling condition of this transition is the opposite of the enabling condition for transition T6.

T8 occurs when the current node receives a <ref>, REFUSE control packet from its neighbors. It shows the action of congestion control. In this case the transition T7 cannot fire until T9 fires (a <res> control packet is received).

T9 occurs when a <res>, RESUME control packet is received from its neighbor.

T11 occurs whenever a packet is sent on the output line. The predicate associated with this transition represents its firing condition, where LINE is related to the Buffering and Congestion address on the output line determined after checking the routing table.

4. Translation of Protocol Models into Simulation Programs

The models of the communication protocols developed in the previous section can be translated into simulation programs. The target language can be any programming language which supports concurrent processing. We have selected ASPOL since it is a process oriented simulation language which in addition to concurrent processing provides an adequate support for simulation constructs.

The mapping of Petri nets and of extended Petri nets into programming structures has been investigated in the literature [35-37]. In case of Petri nets a process can be associated with a connected subnet in which every transition has at most one input arc and one output arc from (to) any input (output) place. Other methods have been proposed recently, [38-40]. However the method used in this paper is conceptually different, and easier to use. It partitions the set of all places into two disjoint classes and associates with one class synchronization primitives and with the second one processes. We recognize different types of tokens and we associate with each of them a different type of process.

Two types of places can be distinguished in our previous models which are based upon modified predicate transition nets:

a. *Type e places*. They may contain Type 1 or Type 2 tokens which are associated with predicates that decide upon the opportunity of firing a transaction. The correspondent of such places in our simulation structures are primitives to control the synchronization between the processes executing concurrently in the simulation programs.

b. *Type m places*. They may contain only Type 3 tokens which are associated with the transmission units exchanged by the corresponding protocols. For example, in the network protocol, the basic transmission units are data packets, acknowledgment, negative acknowledgment packets, etc. In our model they are represented by different subtypes of Type 3 tokens. In this section, unless it is explicitly stated otherwise, all tokens are assumed to be Type 3 tokens. Each different token subtype will be associated with a process type. The number of concurrent processes of a given type equals the number of tokens of the corresponding subtype. A process will in fact describe the the migration of that particular token through all the places it visits in the system.

Each entity at a given network layer can be conceived as consisting of two basic agents, the sender and the receiver, and in some cases of additional auxiliary components. Each component will be mapped into a different process. The activities of the two primary agents consists of: interfacing with the immediate upper neighboring layer, the specific processing associated with the current layer and interfacing with the immediate lower layer. Both handle the same type of transmission units (packets) but they are concerned with the flow of transmission units in different directions. For example in case of the network layer we recognize a packet sending and a packet receiving process and in addition a packet retransmission process and a routing table update process which for the sake of simplicity will be omitted in our discussion.

To illustrate the translation technique from the modified predicate transition net model to the simulation program, we present in Figure 4, the basic types of processes, the packet sending, packet receiving and packet retransmission. Each structure identifies the range of the corresponding process activities and the path followed by the respective token type. A *type m place* is associated with a sequential code in the corresponding process. When a process (remember that a process is associated with a Type 3 token) visits such a place the program counter is position to execute the next sequence of instructions. The firing of a transition determines the execution of the next sequential code section. Whether the transition can fire or not, this depends upon the predicate associated with the transition, if any. If a predicate is associated with the transition then it should be checked according to the label variables of the input arcs associated with the transition. Then the sequential code is executed. Usually it will compute some environment variables and modify some attributes of tokens. These operations are shown by the labels of the output arcs associated with that transition.

In ASPOL, a process description specifies the behavior of a type of processes and defines variables and events unique to each process of that type. Each process is a particular and independent instance of a execution of a process description.

In Appendix, we give three process descriptions of packet type, which were written in ASPOL. The *sendp* (send packet) process executes first the operations associated with transition T11 from Figure 3. These operations are: routing table lookup to determine the output line and checking the availability of that line, by evaluating the predicate associated with T11. If the predicate which is also related to buffer occupancy on the selected output line is not satisfied, the process waits until the predicate becomes true. If the predicate is true, then the transition fires and it triggers the execution of subsequent operations: computation of the sequence number and the buffer counter, addition of the packet sequence number into token attribute record, creation of a *retrap* (retransmit packet) process, and decision whether to create a *sref* (sending refuse) process. The place P3 is a decision place because it has multiple output arcs. The program branchs at this place according to the attribute value of the destination address. If the predicate associated with the transition T7 is satisfied and the variable P4 is held, i.e., the refuse command is not received, the only operation associated the transition T7 (including T6) is to hold the process for a random time interval, *f*. If P4 is empty, i.e., the refuse command has been received, the process waits until the receiving resume process wakes it up. Finally, the process arrives at the place FQ, it is substituted for a sending frame process, *sendf*. The *retrap* process and the *recep* (receive packet) process are simpler and their description will be left as an exercise for the reader. The creation, termination and synchronization of processes will be discussed later.

The *Type m places* are in turn classified into three groups:

a) Places which accept as input tokens of two or more different subtypes. In the graph, such places are nodes with two or more input arcs labelled with token variables of different subtypes. For example places P3 and FQ in Figure 3 belong to this group. Tokens of two different subtypes are also in place P3 since a retransmitted packet is specially identified. Place FQ contains different types of tokens corresponding to, *ref, res, ack, nak* control packets and the data packet, *p*. This indicates that different type of processes may execute the same operations in steps to follow. At these places, we may combine such processes into a new process, since they execute the same sequence of operations and this reduces the complexity of our simulation programs.

b) Places which allow as output two or more different subtypes of tokens. In the graph, there are two or more output arcs emerging from such a node, labelled with token variables of different subtype. The place EQ in Figure 3 is an example of a decision node of the program. The tokens of different subtypes contained by the place can cause that the different transitions are executed. At these places, it is possible to substitute several types of process for the original one.

c) Places where no transformation of the process type can occur.

The transitions associated with process creation and termination are of special concern for the generation of simulation programs from modified predicate transition nets models. The following types of transitions can be recognized:

Fork — The transitions for which the number of output arcs (labelled by token variables of type 3, the following is the same denotation) is larger than that of input ones correspond to a fork operation. For example, referring to Figure 3., the transition T11 corresponds to a fork operation. When this transition fires, some new processes are created. The number of the new processes created is the difference between the number of output and input arcs. The transition T3 is another example. Whenever T3 is executed, the retransmission packet process is created while the original one still exists.

Join — The transitions with more input arcs than output arcs correspond to a join operation. When such a transition fires, some processes terminate their execution. The number of terminating processes is the difference between the number of input and output arcs. For instance, whenever the transition T2 is executed, the receiving ack process and the certain retransmitting packet process are terminated.

Synchronization among processes is represented by transitions to or from places of *Type e*. If a transition is associated with a predicate concerning with the attributes of tokens of type 2 con-

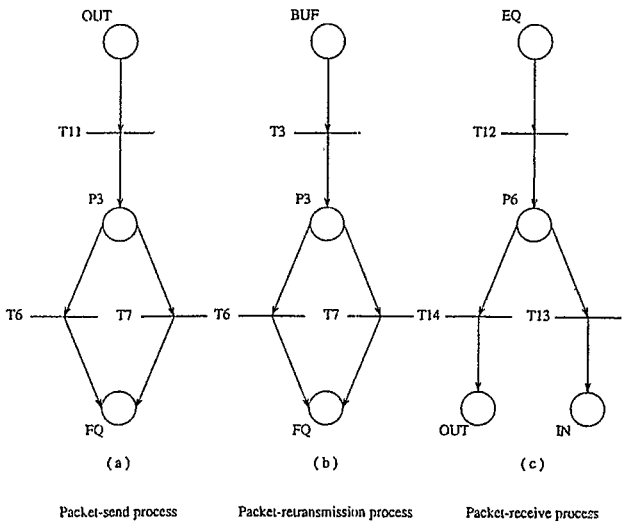


Figure 4. The primitive processes involved in packet handling

tained by its input places or if a input place of a transition can contain a token of type 1, generally we can identify a synchronization topology. For example referring to Figure 3, the input place P4 of transition T7 can contain a Type 1 token. This is not true for transition T6 since the predicate associated with it contains only Type 3 tokens. For transition T11 the predicate is concerned in the value of token bc contained by the place BC. When a process executes such a transition, it should decide whether it waits or not. If the predicate is not satisfied the process waits, else it continues.

If a transition is associated with updating Type 1 or Type 2 token variables then one may identify a synchronization topology. In Figure 3, the place BC connects the transition T11 and T2. When a process executes the transition T2, the process would like to update the token variable bc contained by the place BC. It is possible this process wakes up other processes which wait to execute the transition T11. As the same case, the place P4 connects the transition T7 and T8. There is synchronization between the sending packet process and the receiving packet process, because the receiving process would like to remove the token contained by the place P4.

Most of the synchronization problems can be treated in the framework of Petri nets or modified Petri nets. As two examples in last paragraph, the former is a classical producer-consumer synchronization, the latter is a mutual exclusion problem.

5. Conclusion

A significant part of the performance analysis of a computer communication architecture is related to the modeling of communication protocols supported by the architecture. The communication architecture investigated in this paper is based upon the OSI - Reference Model. Queuing models and analytical methods [41] have been used for this purpose.

In this paper we take a different approach. We define modified predicate transition nets and use them to model the mechanisms built into different protocols. While Petri nets have been used in the past to prove the correctness of protocols, by introducing the time concept and the variable arc concept, we are able to use modified predicate transition nets for the performance analysis of communication protocols. A two step approach is presented, first we built the graphs which model the protocol behavior and then we translate these graphs into simulation programs.

Since our models can include non-prime transitions we are capable to represent intricate communication mechanisms using simpler ones as building blocks. Using this hierarchical modeling approach we model first mechanisms like fragmentation, reassembly and congestion, etc., then we model the function of an entire layer.

We believe that the modified predicate transition nets can be successfully used as a framework for the study of communication protocols and that they can be applied to the performance evaluation, specification, validation and implementation phases.

Reference

[1]. *Data Processing-Open Systems Interconnection-Basic Reference Model,* Draft Proposal **ISO/DP 7498** International Organization for Standardization, 1982

[2]. A.A.S. Danthine, *"Petri Nets for Protocol Modelling and Verification,"* Proc. computer Networks and Teleprocess. Symp., PP. 663-685, Oct. 1977

[3]. A.A.S. Danthine, *"Protocol Representation with Finite-State Models,"* IEEE Trans Comm., Vol. **COM-28**, PP. 632-643, Apr. 1980

[4]. P. Estallier and C. Girault, *"Petri Nets Specification of a New Protocol for Controlling a Distributed System Organization,"* Third International Conference on Distributed Computing System, Miami, Florida, 1982

[5]. G. Bochmann and C. Sunshine, *"Formal Methods in Communication Protocol Design,"* IEEE Trans. Commun., Vol. **COM-28**, PP. 624-631, Apr. 1980

[6]. M. Diaz, *"Modelling and Analysis of Communication and Cooperation Protocols using Petri Net Based Models,"* Comp. Networks, Vol. **6**, PP. 419-441, Jun. 1982

[7]. G. Berthelot and R. Terrat, *"Petri Nets Theory for the Correctness of Protocols,"* IEEE Trans. Commun., Vol. **COM-30**, No. 12, PP. 2497-2505, Dec. 1982

[8]. B. Wolfinger and O. Drobnik, *"Simulation of Protocol Layers of Communication in Computer Networks,"* Computer Networks and Simulation II S. Schoemaker(ed.) North-Holland Publishing company, PP. 141-165, 1982

[9]. H. Kleine, *"A Survey of Users' Views of Discrete Simulation Languages,"* Simulation, Vol. **14**, No.5, PP. 225-229, May 1970

[10]. J.O. Henriksen, *"The Development of GPSS/85,"* Simulation Symposium 1985, PP. 61-77, 1985

[11]. E.C. Russell, *"Building Simulation Models with SIMSCRIPT II.5,"* CACI, Inc., 12011 San Vicente Blvd., Los Angeles, CA 90049, 1982

[12]. W. Fischer, K.P. Sauer and W. Denzel, *"A Simulation Technique for Distributed Systems Based on a Formal Specification by SDL"* Proc. Int. Sem. on Computer Networking and Performance Evaluation, Tokyo, PP. 11-4-1--4-10, Sep. 1985

[13]. C.H. Sauer and E.A. MacNair, *"Simulation of Computer Communication Systems,"* Prentice-Hall, INC., 1983

[14]. J.L. Peterson, *"Petri Net Theory and the Modelling of Systems,"* Prentice-Hall, INC., 1981

[15]. W. Brauer, G. Rozenberg and A. Salomaa, *"Petri Nets - An Introduction,"* Springer-Verlag, Berlin Heidelberg, New York, Tokyo, 1985

[16]. M.A. Marsan, G. Conte and G. Balbo, *"A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor systems,"* ACM Trans. Comp. Systems, Vol. **2**, No. 2, PP. 93-122, May 1984

[17]. C.A. Petri, *"Kommunikation Mit Automaten,"* (Ph. D. Thesis in German) Translation by C.F. Greene, Supplement 1 to RADC-TR-65-337, Vol. **1**, Rome Air Dev. Center, Griffiss AFB, NY, 1965

[18]. A.W. Holt, *"Final Report of the Information System Theory Project,"* RADC-TR-68-305, Rome Air Dev. Center, Griffiss AFB, NY, Sep. 1968

[19]. A.W. Holt and F. Commoner, *"Events and Conditions,"* Record of the Project MAC Conf. on Concurrent Systems and Parallel Computation, PP. 3-52, 1970

20]. J.B. Dennis, *"Modular Asynchronous Control Structures for a High Performance Processor,"* Record of the Project MAC Conf. on Concurrent Systems and Parallel Computation, PP. 55-80, 1970

[21]. W. Brauer, *"Net Theory and Applications,"* Lecture Notes in Comp. Sci., Vol. **84**, Springer-Verlag, 1980

[22]. J.L. Peterson, *"Petri Nets,"* Comp. Surveys, Vol. **9**, No. 3, Sep. 1977

[23]. K. Jensen, *"Coloured Petri Nets and the Invariant-Method,"* Theoretical Comp. Sci., Vol. **14**, PP. 317-336, 1981

[24]. J.H. Genrich and K. Lautenbach, *"The Analysis of Distributed Systems by Means of Predicate/Transition-Nets,"* Lecture Notes on Comp. Sci., Vol. **70**, PP. 123-146, 1978

[25]. J.H. Genrich and K. Lautenbach, *"System Modelling with High-Level Petri Nets,"* Theoretical Comp. Sci., Vol. **13**, PP. 109-136, 1981

[26]. J.D. Noe and G.J. Nutt, *"Macro E-Nets Representation of Parallel Systems,"* IEEE Trans. Comp. **C-22**, PP. 718-727, Aug. 1973

[27]. P.M. Merlin, *"A Methodology for the Design and Implementation of Communication Protocols,"* IEEE Trans. Commun., Vol. **COM-24**, PP. 614-621, Jun. 1976

[28]. P.M. Merlin and D.J. Farber, *"Recoverability of Communication Protocols: Implications of a Theoretical Study,"* IEEE Trans. Commun., Vol. **COM-24**, PP. 1036-1043, Sep. 1976

[29]. F.J.W. Symons, *"Introduction to Numerical Petri Nets, a General Graphical Model of Concurrent Processing Systems,"* A.T.R., 14, 1, PP. 28-33, Jan. 1980

[30]. S. Natkin, *"Reseaux de Petri Stochastiques,"* Ph.D. Dissertation, CNAM-PARIS, Jun. 1980

[31]. M.K. Molloy, *"Performance Analysis Using Stochastic Petri Nets,"* IEEE Trans. Comp. **C-31**, 9, PP. 913-917, Sep. 1982

[32]. M.K. Molloy, *"Discrete Time Stochastic Petri Nets,"* IEEE Trans. Software Eng. Vol. **SE-11**, No. 4, PP. 417-423, 1985

[33]. J. Magott, *"Performance Evaluation of Concurrent Systems Using Petri Nets,"* Information Processing Letters, Vol. **18**, No. 1, PP. 7-13, 1984

[34] S.S. Yan and M.U. Caglayan, *"Distributed Software System Design Representation Using Modified Petri Nets,"* IEEE Trans. on Software Eng. Vol. **6**, PP. 733-745, Nov. 1983

[35]. R. Janicki *"An Algebraic Structure of Petri Nets,"* Lecture Notes in Comp. Sci., Vol. **83**, PP. 177-192, 1980

[36]. U. Goltz and W. Reising, *"Processes of Place/Transition-Nets,"* Lecture Notes in Comp. Sci., Vol. **154**, PP. 264-277, 1983

[37]. R. Janicki, *"Nets, Sequential Components and Concurrency Relations"* Theoretical Comp. Sci. **29**, PP. 87-121, 1984

[38]. G. Bruno and G. Marchetto, *"Rapid Prototyping of Control Systems Using High Level Petri Nets,"* in Proc. Conf. '85 IEEE Software Engineering, PP. 230-235, 1985

[39]. P. Azema, G. Juanole, E. Sanchis and M. Montbernard, *"Specification and Verification of Distributed Systems Using PROLOG Interpreted Petri Nets,"* in Proc. Conf. '84 IEEE Software Engineering, PP. 510-518, 1984

[40]. R.A. Nelson, L.M. Haibt and P.B. Sheridan, *"Casting Petri Nets into Programs,"* IEEE Trans. Software Eng. Vol. **SE-9**, No. 5, PP. 590-602, 1983

[41]. Martin Reiser *"Communication System Models Embedded in the OSI-Reference Model, A survey"* International Seminar on Computer Networking and Performance Evaluation, Tokio, September 1985, PP 3.1.1 - 3.1.26

[42]. William Stallings *"Data and Computer Communications"* Macmillan, 1985

[43]. Chuang Lin and Dan Cristian Marinescu *"Application of Modified Predicate Transition Nets to Modeling and Simulation of Communication Protocols"* Computer Science Department, Purdue University, Technical Report CSD-TR-599, May 1986

Appendix: Examples of Program Structures

```
sim net;          comment simulation program of the network;
.
.
.
def (nod=5);      comment number of the node;
def (line=4)      comment maximum line number with a node;
def (m1=16);      comment modulo of packet sequence number;
def (m=10);       comment buffer maximum capacity;
def (mh=8);       comment buffer high water mark;
```

```
event out(line);    comment event set of output queue;
event con(line);    comment congestion semaphore;
event rq(m1);       comment event set of retransmission queue;
macro cm;           comment defining a referencing block's;
 block com;        comment defining a storage space shared;
 integer rb(nod,nod,2); comment routing table, each row contains
                    destination and line number;
 integer s1(line);      comment counter of packet sequence number;
number;
  integer bc(line);     comment buffer capacity counter;
  integer p1(line);     comment sending ref control flag;
  integer p2(line);     comment sending res control flag;
  integer p4(line);     comment receiving ref control flag;
  integer ad(nod,line); comment post neighbor table;
  integer rtq(line,m1); comment retransmission state variable;
end block;
endmacro;
 .
 .
 .

end sim;

    comment the meaning of the following arguments in processes
       a: destination address, sou: source address,
       s: sequence number of packet(end-to-end),
       s1(j) or ss1: sequence number of packet(point-to-point),
       j: the number of output line;

        comment send packet process description;

process sendp(a,s,sou);
integer i,j,a,s,sou,t1;
cm;        comment introducing the common data space;
          comment checking the routing table to decide output line;
 i=0;
 while(i.lt.nod)do
   begin
     if(rb(sou,i,1).eq.a)then
       goto g;
     i=i+1;
   end;
g:  j=rb(sou,i,2);
              comment deciding whether or not buffer is full;
if(bc(j).ge.m)then
  queue(out(j));

              comment computing the counters;
s1(j)=mod(m1,s1(j)+1);
bc(j)=bc(j)+1;
             comment deciding whether to create sref process;
if(bc(j).eq.mh)then
if(p1(j).eq.1)then
    begin
      p1(j)=0; p2(j)=1;
      initiate sref(sou,j);
    end;

             comment creating retrap process;
initiate retrap(a,s,s1(j),sou,j);

             comment deciding congestion;
if(a.ne.ad(sou,j))then
   if(p4(j).ne.1)then
     queue(con(j));
          comment delaying time of interface and connection;
hold(random(1.,5.)*.001);
         comment substituting send frame process for this one;
 t1=1;       comment setting frame type;
  initiate sendf(t1,a,s,s1(j),sou,j);

end process;
```

```
        comment retransmit packet process description;

process retrap(a,s,ss1,sou,j);
integer a,s,ss1,sou,j,t1;
cm ;     comment introducing the common data space;

      comment waiting for being woke;
queue(rq(ss1+1));
      comment deciding whether ack or nak has been received;
if(rtq(j,ss1).eq.1)then goto v;
      comment creating retransmit packet process;
initiate retrap(a,s,ss1,sou,j);
      comment the following is same with that of sendp process;
if(a.ne.ad(sou,j))then
 if(p4(j).ne.1)then
  queue(con(j));
 hold(random(1.,5.)*.001);
 t1=1;
 initiate sendp(t1,a,s,ss1,sou,j);

 v: rtq(j,ss1)=0;   comment reset the ack flag;
end process;


            comment receive packet process;

process recep(a,s,ss1,sou,j,me);
integer a,s,ss1,sou,j,me;
cm;            comment introducing the common data space;

            comment creating sending ack process;
initiate sack(ss1,sou,j);
hold(random(0.,3.)*.001);

            comment deciding whether the packet is mine;
if(a.ne.me)then
initiate sendf(a,s,sou);      comment delivering it;
else initiate recem(s,sou);    comment receiving it;

end process;
```