II INTERNATIONAL CONFERENCE
TRANSPORT SYSTEMS TELEMATICS TST'02

ZESZYTY NAUKOWE
POLITECHNIKI ŚLĄSKIEJ 2002
TRANSPORT z.45, nr kol. 1570

discrete-event simulation,
transport systems,
Petri nets

Ewa OCHMAŃSKA[1]

AN IMPLEMENTATION OF TIME AND CONTROL IN SIMULATION MODELS OF TRANSPORT SYSTEMS

The paper deals with an approach to computer implementation of discrete event simulation models of transport systems. For that purpose Petri net modelling tool was applied, enriched by timing and control. Timing, based on timestamps and planning of events, has been integrated with control mechanism based on predicates. Programmatic implementation of those mechanisms in object-oriented RAD environment is described and illustrated by two categories of simulation models: with operation- or task-oriented structure. The rules for timing and control are discussed in the context of multi-thread realization of simulation models.

WPROWADZENIE ASPEKTU CZASU I STEROWANIA W MODELACH SYMULACJI SYSTEMÓW TRANSPORTOWYCH

Referat dotyczy pewnego podejścia do symulacyjnego modelowania systemów zdarzeń dyskretnych. Metoda symulacji jest oparta na sieciach Petriego, powszechnie stosowanym formalizmie sieciowego opisu modeli symulacyjnych. To narzędzie zostało wzbogacone przez rozszerzenia semantyczne pozwalające wyrazić w modelach aspekty czasu i sterowania, niezbędne dla odwzorowania procesów realizowanych przez systemy transportowe. Głównym celem referatu jest opisanie zasad komputerowej implementacji tych mechanizmów w Delphi - obiektowo zorientowanym środowisku RAD.

Przedstawione rozwiązania były użyte do definiowania dwóch kategorii modeli symulacyjnych, określonych na podstawie różnych interpretacji semantycznych modelowanych systemów - zorientowanych na operacje lub zadania. Pierwsze z tych podejść semantycznych było stosowane w modelowaniu procesów transportowych i logistycznych, drugie było użyteczne przy tworzeniu modeli rozproszonych kolejowych systemów sterowania. Krótko wyjaśniono zasady konstruowania modeli symulacyjnych tych dwóch typów.

Przedyskutowano mechanizmy czasowe i sterujące w kontekście wielowątkowego wykonania symulacji. Podział modelu symulacyjnego na szereg współbieżnie wykonywanych wątków programowych nakłada dodatkowe wymagania na procedury sterujące przebiegiem symulacji oraz wprowadza zagadnienia komunikacji i synchronizacji między procesami realizowanymi w oddzielnych wątkach.

Proponowane jest zastosowanie opisanych narzędzi symulacyjnych do tworzenia modeli interaktywnych obliczeń w transportowych systemach telematycznych.

[1] Faculty of Transport, Warsaw University of Technology, Koszykowa 75, 00-262 Warsaw, och@it.pw.edu.pl

1. INTRODUCTION

The paper deals with some approach to computer implementation of discrete-event simulation models of processes performed in transport systems. The method of simulation is based on Petri nets, commonly used net description formalism for simulation models [7], enriched by semantic extensions necessary to express timing and control aspects of modelling. Suitability of Petri nets for discrete events simulation purpose arises from the fact that their formal "behaviours" naturally represent dynamics of simulated processes, viewed as successive or concurrent occurrences of causally related events.

The implementation of timing and control mechanism, presented in the paper, derives from some known approaches [1, 2], which consist in representing time by timestamps and control by logical predicates enabling activation of dynamic net elements – transitions during simulation of modelled process.

The paper is composed as follows: In Section 2 basic Petri net (PN) definition is given and its timing and control extension is described together with an additional, supervisory control mechanism used in simulation modelling. The PN extension comprises data structures with timestamps bound to tokens, as well as enabling predicates and actions related to transitions. Token data structures, including timestamps, describe actual states of a process being executed by modelled system. Hierarchical control predicates are defined on data included in tokens.

Section 3 sketches programmatic implementation of those mechanisms in object-oriented RAD environment. The most part of the model can be mapped directly to the object classes representing transitions, places and tokens of PN. Transition classes have methods corresponding to their enabling predicates and actions. Aside from enabling predicates resolving transition local time and resource-availability conditions, two more control predicates are implemented in an object class representing general supervisor of simulated process. One of them concerns possible resolutions of conflicts between groups of transitions; another one chooses among alternative control decisions arising during simulation.

Section 4 illustrates applying of the described control rules in context of two categories of PN simulation models following different semantic conceptions: describing modelled systems in operation- or task-oriented manner. First of those semantic approaches was applied in modelling of transport and logistic processes [3, 4], the second one was useful in constructing models of distributed railway control systems [5, 6].

In section 5 presented timing and control mechanisms are discusses in the context of multi-thread realization of simulation models. Several concurrent threads of simulation program may correspond to processes performed by co-operating, distributed nodes of transport systems. This poses additional requirements on model construction and introduces issues of communication and synchronization between simulated processes.

Final remarks suggest other possible applications of described simulation tools such as models of interacting computing, and mention some problems needing further research, in particular decisive functions.

2. PETRI NET AND ITS EXTENSION

2.1. PETRI NET DEFINITION

Regular Petri net PN=(B, S_0) is defined on a bi-graph B=(P∪T, A) with two disjoint sets of nodes: a set of passive places P and a set of active transitions T, and with a set of arcs $A \subseteq (P \times T \cup T \times P)$. Initial state S_0 is a function S_0:P→N assigning non-negative numbers of tokens to places. Places contain token and transitions move tokens between places, changing actual state S of the net, S:P→N. A transition t∈T has a subset of input places IP(t)={p∈P|(p,t)∈A} and a subset of output places OP(t)={p∈P|(t,p)∈A}. A transition t∈T is enabled to perform an action consuming tokens from input places and producing tokens in output places, if $\forall_{p \in I(t)}\ S(p)>0$. An action of a transition t changes a state S of the net to a new state S' defined as follows:

$$S'(p) = \begin{cases} S(p) & \text{if } (p \notin IP(t)) \wedge (p \notin OP(t)) \vee (p \in IP(t)) \wedge (p \in OP(t)) \\ S(p) - 1 & \text{if } (p \in IP(t)) \wedge (p \notin OP(t)) \\ S(p) + 1 & \text{if } (p \notin IP(t)) \wedge (p \in OP(t)) . \end{cases} \quad (1)$$

2.2. PN EXTENSION FOR TIMING AND CONTROL

Simulation modelling consists in dynamic execution of processes in abstract models representing real word systems. Petri nets extensions well suited for that purpose should cover timing and decision-making aspects of processes simulated by system models. PN modelling tool described in the paper introduces time and control to transition enabling rules, basing on two commonly used approaches:

1. Enriching the nature of tokens by assigning to them some informative contents including semantic description of represented entities and their time parameters. The contents of all tokens present in places of a net describe current state of a process performed by the net simulation model.
2. Enriching the nature of transitions by assigning to each of them: • individual enabling rules in the form of logical functions called enabling predicates, which are defined on contents of input and output tokens; •• capability of transforming information in the form of procedures associated with actions, which compute the contents of output tokens from those of input ones.

A token may contain arbitrary data structure, always including timestamp of its creation. Enabling predicate of a transition may express any local condition specific for modeling purpose, including timing rule:

$$\text{Max}_{\,|p \in IP(t)}\ (\text{Min}_{\,|\text{tokens in } p}(\text{token timestamp value})) \leq \text{current simulation time} \quad (2)$$

An action of enabled transition produces in its output places tokens, equipped with arbitrary data structures specific for modelling purpose, but always including timestamps with values not less then current simulation time.

2.3. SUPERVISORY CONTROL OF SIMULATION MODEL

Control mechanism of described simulation modelling tool goes beyond PN extension above formulated as "enabling predicate" bound to particular transitions. Such predicate enables transition activation on the base of local data, available as contents of tokens on its input. Making decisions in situations of net conflicts needing resolution requires some more general view of current state of modelled process. On the other hand, semantics of a model may demand global view of simulated process to decide about alternative ways of performing particular actions. This can be achieved by two hierarchical levels of predicates* built in supervisory control device of simulation model:

- Operative predicate, enumerating all admissible resolutions of conflicts by means of combinatory operations on the net structure
- Decisive predicate, choosing one among all admissible resolutions, following any rule proper for simulation purpose. Such rules can be formulated taking into account data structures of all tokens describing current state of simulated process (with time horizon delimited by timestamps of tokens in PN places).

3. MAIN CONCEPTS OF IMPLEMENTATION

3.1. TIMING AND CONTROL IN OBJECT CLASSES OF PN ELEMENTS

Model elements are implermented in Delphi RAD environment as a library of general and specilized object classes. Time behaviour is common to all models constructed following presented method, hence timing is built into four general object classes written in bold on Fig.1.

Properties and methods related to control, dependent on model semantics, are redefined (extended) in descendant specialized, context-specific object classes.

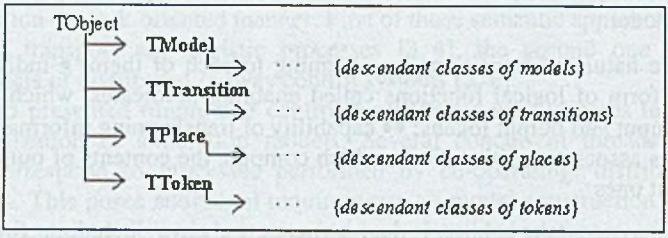

Fig.1. Hierarchy of object classes implementing simulation model

General class of tokens comprises timestamp and two methods, used by transitions by consuming token from input places and producing them on output:

* Procedures implementing logical predicates result in subsets of input tokens, for which predicates remain true.

```
TToken = class
  TimeStamp: TTime; DataStructure: Pointer; Next: TToken;
  procedure RemoveFrom(IPlace: TPlace);
  procedure InsertTo(OPlace: TPlace; NewStamp: TTime); …
end;
```

Tokens are organized in lists ordered by timestamps and assigned to places. General class of places play passive role of token container, hence it defines no methods related to timing and control. However, its properties reflect net structure of a model and are useful for manipulating tokens during simulation. Descendant classes of places redefine types of contained tokens according to model semantics.

General class of transitions defines virtual methods implementing common time aspects of their enabling predicates and actions:

```
TTransition = class
  InputPlaces, OutputPlaces: TPlaceSet;
  PreviousTransition, NextTransition: TTransition;
  function TimeFunction;
  function GetCandidateToken(P:TPlace; I:Word): TToken;
  property CandidateToken[P:TPlace; I:Word]: TToken
    read GetCandidateToken;
  procedure EnablingPredicate; virtual;
  procedure Action(I: Word); virtual;
end;
```

Descendant classes override these methods to implement context-specific behavior of active elements of a simulation model. An array read-only property of candidate tokens gives access to tuples of tokens – one per each input place, which enable the transition. These token tuples are beforehand computed by enabling predicate and indexed by `I`.

Pointer lists of type `TPlaceSet` implement net structure of a model. Moreover, for the sake of timing mechanism, general class `TTransition` is equipped with pointers to previous and next transition, used to organize active net elements in second structure: bi-directional list forming typical queue of planned events, as shown on Fig.2.

Transitions are ordered according to time expression of the left side of inequality (2), computed by a method `TimeFunction`. Keeping them in order is effectively implemented in `Action` method; only transitions with input token list changed by the action have to be considered for re-ordering.

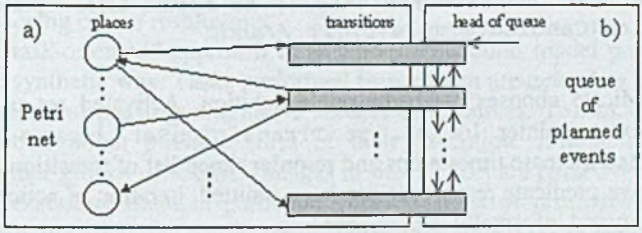

Fig.2. Dual structure of a model: a) bi-graph, b) bi-directional list

3.2. EXECUTING SIMULATION BY OBJECT CLASS TMODEL

General object class of a model supervises execution of simulation. It operates on the timed list of transitions, taking into account those with smallest values of time function. Some non-negative margin for current simulation time can be delimited, so that a broader subset of transitions (with time function values laying within that margin) may candidate for activation.

Class `TModel` implements common mechanism of model execution, basing on higher-level control predicates; it is provided with a pointer to the head of transition list and with an executive method:

```
TModel = class
  FirstTransition: TTransition;
  CurrentTime, TimeMargin: TTime;
  function GetCandidateTuple(T:TTransition; J:Word):Word;
  property CandidateTuple[T:TTransition; J:Word]: Word read
    GetCandidateTuple;
  procedure OperativePredicate;
  function DecisivePredicate(var J: Word): TTransitionSet;
    virtual; abstract;
  procedure ExecuteSimulation;
end;
```

An array read-only property `CandidateTuple` gives numbers of candidate token tuples for enabled transitions in non-conflict combinations, enumerated by operative predicate and indexed by `J`. Method for operative predicate defined in general model class is common for all models. Specialized model classes derived from `TModel` define context-specific implementations of virtual abstract function for decisive predicate. This function returns a subset of transitions to be activated along with an index of respective set of enabling token tuples, chosen according to model semantics.

General executive method has following outline:

```
procedure TModel.ExecuteSimulation;
var J: Word; T: TTransition;
begin
  T:= FirstTransition;
  while T.TimeFunction <= CurrentTime + TimeMargin do
    begin T.EnablingPredicate; T:= T.NextTransition; end;
  OperativePredicate;
//for each T in DecisivePredicate(J) do begin time update;
    T.Action(CandidateTuple(T,J)) //end;
end;
```

Decisive predicate chooses `J`-th admissible solution. Activated set of transitions is returned as an extra pointer list of type `TTransitionSet`, because actions being successively executed increase timestamps and re-order timed list of transitions. Since a strict definition of decisive predicate result type has been omitted, iteration of actions on resulting set of transitions is noted informally in comment line.

Updating current simulation time is necessary to preserve time monotonicity in the case of positive time margin. Maximal value of time function of activated transitions (before action executing) has to be considered.

Having defined a class derived from `TModel` with specialized decisive predicate, after creating an object `Model` of that class we can run simulation pass by a statement:

```
with Model do while FirstTransition.TimeFunction < TimeLimit
                do ExecuteSimulation;
```

4. SEMANTICS OF SIMULATION MODELS

Timing and control tools described in previous sections have been applied to simulate behaviour of discrete-event systems modelled following two different semantic principles. In both cases, places of PN model were treated as containers for available system resources and objects being served by the system. On the other hand, PN transitions were representing various interpretations of system behaviour.

The first approach, which can be called "operation-oriented", represents detailed view of simulated process. This approach has been used to define models of serving trains at railway freight stations [4]. Tasks performed by modelled system are decomposed to partially ordered sets of elementary operations, executed by active elements of PN model – transitions. In that case, specialized implementation of enabling predicate comprises conditions of executing operation represented by the transition action.

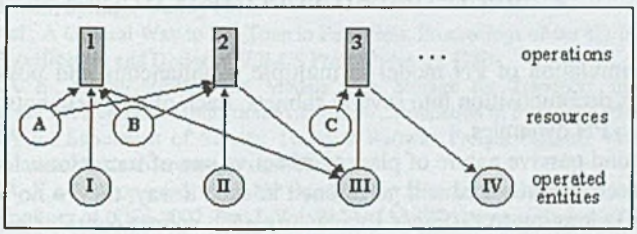

Fig.3. Operation-oriented modelling approach

In an exemplary piece of PN model shown on Fig.3, enabling predicates of operations may select appropriate resources, dependent on parameters of particular entities being operated. Operative predicate may enumerate resolutions of conflicts between operation 1 and 2 attempting to use resources A and B. Decisive predicate controls a way of process performance choosing one of resolutions.

Another, "task-oriented" approach to construct simulation model perceives simulated process in more synthetic way. Tasks performed by a system are treated as undividible units, demanding specific but possibly alternative subsets of resources. Tokens representing tasks contain information about possible ways of their execution. Availability of resources, represented by other groups of tokens, changes in time. Tasks are passed between transitions responsible for executing them in particular configurations of resources, until demanded resources are available. Enabling predicates decide of executing tasks or passing them to other configurations of resources. Such model construction gives some additional possibility of controlling execution of a stream of tasks, and therefore has been used for modelling railway control systems [6]. An illustration of task-oriented modelling principle is presented on Fig.4.

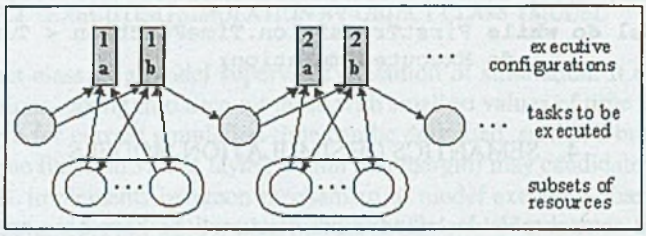

Fig.4. Task-oriented modelling approach

In this small part of a model, executive configurations are represented by pair of transitions with common inputs. Transitions of type a and b have mutually exclusive enabling predicates. Predicate of the type **a** enables actions executing tasks in case of availibility of resources. When appropriate resources are not available, predicate of the type **b** enables actions of passing tasks to some other configuration.

5. MULTI-THREAD MODEL EXECUTION

Executing simulation of PN model in multiple simultaneous and possibly distributed threads requires its decomposition into several subnets. Each of those subnets cooperates with others according to PN dynamics.

Having in mind passive nature of places and active one of transitions, let us assume that a net model has been reconfigured and partitioned in such a way, that: • no place is on input of two transitions belonging to different subnets; •• each subnet contains all input places of its transitions. Note, that the above assumptions are equivalent to the fact, that all operative predicates are defined on subsets of places belonging to common subnet.

Then co-operation of subnets consists in passing tokens by activated transitions to places in other subnets. This can be implemented by means of coordination provided by Delphi for accessing shared memory and synchronizing threads.

Such approach should be valid as long as model structure and behaviour remain those of Petri net. However, timing an control mechanisms described in the previous sections go beyond PN frame by introducing higher-level object class derived from TModel, responsible for controlling execution of simulated process by means of operational and decisive predicates working on timed queue of transitions. Some additional rules have to be stated for synchronization of those supervising levels.

In the case of executing multiple threads of simulation on one machine, one model object synchronizes behaviours of all PN subnets. When simulation is simultaneously executed on several computers, each of threads demands separate control object deriving from the class TModel. Some additional rules have to be stated for synchronization of those supervising levels to maintain monotonicity of simulation time. Different strategies for preserving time conditions in distributed environment are discussed in [8].

6. FINAL REMARKS

The described concepts of defining and implementing Petri net simulation models of discrete-event systems have been applied on the field of organising and controlling of railway transport. Gained experience permits to estimate the presented modelling tool as flexible and powerful. Further efforts on problems needing resolving are planned and new application scopes in the area of transport will be developed.

One of areas in which such tools may be used is creating simulation models of transport telematic systems basing on interacting computing. For that and other purposes, distributed mode of simulation has to be thoroughly elaborated and investigated.

Further research on specific solutions of derived methods, in particular decisive predicates, is necessary to provide templates for fast programming of problem-specific model components.

BIBLIOGRAPHY

[1] DESEL J., REISIG W., Place/Transition Petri Nets. In: Lectures on Petri Nets I; Basic Models, Vol. 1491 of LNCS, Springer-Verlag 1998

[2] GHEZZI C et al., A General Way to Put Time in Petri Nets. Proceedings of the 5th International Workshop on Software Specification and Design, IEEE-CS Press, Pittsburg 1989

[3] OCHMAŃSKA E., Object-oriented PN Models with Storage for Transport and Logistic Processes. Proceedings of the 9th European Simulation Symposium „Simulation in Industry", Passau 1997

[4] OCHMAŃSKA E., Simulation of Serving Trains at Railway Freight Stations. Proceedings of the 13th European Simulation Multiconference, Warsaw 1999

[5] OCHMAŃSKA E., Task-oriented Petri Net Models for Discrete Event Simulation. In: Computational Science – Proceedings of ICCS 2002. Part I. Vol. 2329 of LNCS, Springer-Verlag 2002

[6] OCHMAŃSKA E., WAWRZYŃSKI W., A Simulation Model of Transport Node Control System. Archives of Transport. Polish Academy of Science, Committee of Transport. Warsaw 2002 (to appear)

[7] TYSZER J., Symulacja cyfrowa. WNT, Warszawa 1990

[8] YI-BING LIN, FISHWICK P., Asynchronous Parallel Discrete Event Simulation. IEEE Transactions on Systems, Man and Cybernetics. Vol. XX, No Y, 1995

Reviewer: Ph. D. Jerzy Mikulski