Proceedings of the Conference on Modelling and Simulation 1994
eds. Guasch and Huber

XSIMNET, A TOOL IN C++ FOR EXECUTING SIMULATION NETS

Åke Gustavson and Aimo Törn
Åbo Akademi
Department of Computer Science
SF-20520 ÅBO, Finland

ABSTRACT

Simulation Nets (SNs) are Petri Nets (PNs) extended for convenient modelling of discrete event simulation problems. The extensions included are zero testing, firing time for transitions, colored tokens, or-logic and interrupts. The tool XSimNet is an interpreter which accepts a text equivalence, SNL, of SNs and performs the simulation implied by the SN model. Validation is facilitated by providing several forms of trace and statistics. The automatic statistics collection concerning places and transitions can be extended by user defined formulas.

INTRODUCTION

Few modelling tools for simulation stand on a firm theoretical base. Most are “ad hoc” and are tied to a specific language.

We propose a general modelling tool, SNs, based on the well known PNs. The concept behind SNs is similar to that of GPSS, with transactions (tokens) traversing a block diagram (net).

The theoretical base of SNs allows consistent modelling and therefore most of the different blocks (> 60) needed in GPSS can be modelled by using half a dozen constructs only. Also in addition to simulating a SN model, it is possible to theoretically analyze the model itself.

A SN uses most of the classical extensions like inhibitor arcs, time, colored tokens etc. Some new extensions are added, such as interrupt arcs, stochastic or-logic and hierarcies of nets. What might be characteristic for SN is also the way of modifying the classical extensions. The modifications are done for simplifying simulation modelling.

SN have been a research interest at the department since 1981 (Törn 1981). Two tools have been developed. SimNet, which is written in Simula and available in VMS and MS-DOS, has automatic statistics collection, animation (MS-DOS) and most of the SN properties described below. XSimNet written in C++ and described in a later chapter, is a further development of SimNet. Both accepts a text equivalence of a restricted set of SNs and performs the simulation implied by the SN model.

SIMULATION ENVIRONMENT

Structure of the environment

We aims at a simulation system, based on suitable PN extensions, where we can draw the net in a graph editor, and then execute it. Instead of building a huge application, with both graph editing and interpretation of the net, we preferred to split the system into several parts. The system development can thus be divided into three main steps.

- Making up the rules and choose what extensions should be added to PNs and define them. This is an ongoing process.
- Definition of a data format for storing the nets, which will be the interface between the applications. We prefer a readable ascii file format. This means constructing a language, SNL, including syntax and semantics.
- Construction and implementation of the applications. The first steps are an interpreter and a graphical design tool for SNs. The graph editor shall export the created graph to SNL.

The structure above gives the opportunity to realize the parts separated from each other. It also makes it easy to add other modules i.e. an automatic analyzing tool or applications transferring statistics to graphs and animation.

Contemporary state

The graph editor is under construction. The interpreter and SNL definition are finished up to modifications and refinements. The SNL language describes the net with respect to simulation. It does not include graphical information i.e. coordinates for the graphical symbols. This has to be added if the graph editor should be able to recreate the graphical picture from the textual description of the net.

For this first version the net must be described in SNL by the modeler. Here one can see the advantage of dividing the application into separate parts and having a readable data file. Smaller modifications could also be made directly in the SNL representation of the problem, even in the future, after the whole tool is implemented.

It is difficult to give a precise definition of SNs because of the ongoing development. In the following we will give a review of the SNs at their current state and also discuss some future amendments.

SIMULATION NETS

In this section we will concentrate on the main difference between SNs and PNs with respect to what is important for simulation. Not every extension presented here is unique for SNs, some of them are standard in other variants of PNs. We assume some fundamental knowledge about PNs.

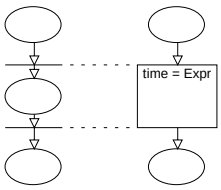
Time

Time is of central interest in simulation, the extension to PNs is called Timed PNs. There are several ways of introducing time in PNs. Four major strategies can be found.

1 When a transition is enabled there is a delay before the firing can take place. Under this time the token to be consumed stays in it’s place.
2 When a transition fires, it will consume all required tokens from it’s input places, keep the tokens unreachable for some time and deposit them in the output places after the time has passed.
3 Tokens have a time stamp, which means that despite of residing in the in-place, they are kept ‘invisible’ for the particular time. This implies that tokens are colored in some way.

4 Time is associated with places. Each token that arrives to a place is delayed before they become available.

We choose to connect time with transitions according to strategy 2 above. The main reason is the convenience for simulation modelling. This time concept has however been criticized from a theoretical viewpoint for violating the main PN concepts of states (places) and events (transitions). A transition, timed in this way, represents both of these concepts. It is equivalent to a small net.



Colored tokens

In a PN all tokens are identical. Introduction of individuality to tokens has been made in different ways, and by tradition these tokens are named colored and the resulting net is sometimes referred to as a CPN, colored PN.

In a SN tokens are colored in the following way. All tokens are of a certain type and has priority. There is a predefined, "uncolored" tokentype called *any* with a fixed priority. All other types are user defined and must be declared. Such tokens differ from the predefined type in that they can have an unlimited number of numerical attributes. The values of the attributes and the priority of a declared token type are user defined and may be changed dynamically in the execution of the model.

Or logic

Or logic means that only one of two or more places are involved when firing a transition. The extension means that there is an inclusive or with a built-in random choice, based on probabilities.

Introduction of probabilities makes this extension very useful for simulation. The probabilities are implied by integer numbers written on the branches of the arc. Based on given probabilities, the interpreter will make a random choice of the branches. For input places the random choice is made among those having tokens. The probabilities can also be equal, which results in a pure inclusive or logic.

The same type of arcs can be applied to output places, but the meaning is different. In this case there is a probabilistic routing of the token(s).

Stochastic functions

In simulation, time, like time between arrivals, is distributed according to some probability distribution. To support this SNs contain built in stochastic functions.

Three such functions can be used as operands in formulas (see Code in transitions). The functions include the negative exponential, the normal and the uniform distribution.

Modification of the firing rule

The firing rule for a PN says that if a transition is enabled it may fire. In simulation we don't want such an uncertainty. If a transition is enabled it must fire if this does not conflict with the firing of another transition. In the latter case, the new rule says that one of them must fire. The choice is then deterministic but arbitrary because there is no priority between transitions, unless this is explicitly modelled by using inhibitor arcs.

Queue discipline in places

With uncolored tokens, a place and it's markings are like a counter, which is decremented and incremented with the firings of transitions. A consequence of having colored tokens is that a place contains entities. With entities, it makes sense to introduce order of entities. A place with ordered entities is more like a queue than a counter.

Each place will treat the tokens according to one of the following queue disciplines. The two classical queue disciplines, *fifo* and *lifo*, may be used. The individual differences between tokens gives a third queue discipline, *priority.* The tokens are sorted by their priority. There is a fourth queue discipline, i.e. unordered. We will call this discipline *random*. It differs from the others in that when a token is to be consumed any token may be removed. In the other cases only the first to leave can be removed.

Local naming of tokens

There is sometimes a need for identifying a specific token when it is passing a transition. Reasons for this could be routing or accessing an attribute of the token.

The way to name a token is to write the name (identifier) on the arc, instead of just giving the number of tokens required. If more than one token is required, all of them are named in a list, e.g. X,Y:*type*. The scope of the declaration is the corresponding transition. This means that the same token can pass another transition with another name.

Interrupts

A transition may interrupt an executing transition in order to make use of a shared resource. The interrupted transition will continue it's execution after the termination of the interrupting transition. This feature is different from the traditional behavior of a PN in many ways. It's main characteristics can be summed up as follows.

- It is a connection and a token transfer between two transitions, with no place in between.
- It requires time, introduced in the manner described above.
- There must be a concurrence of a shared resource, modelled by a place, between the two transitions.

The interrupting transition will try to start normally, i.e., by consuming the token from the shared input place. If the input place is empty, it will look for an ongoing transition of it's concurrent neighbor, stop it and use the resource (token) for it's own execution.

Code in transitions

With the introduction of time in the manner described above, the role of the transition has changed. It has become the dominating, or active node in the net. In a traditional PN states are represented by places and their marking and the transitions are actions which change the current state. In a SN we can have ongoing transitions, which also express a state.

Considering a transition as the executing node, makes it natural to extend a transition to include code which prepares and collects data that could be of interest for the simulation results. A transition can have local variables, for computing and storing of data, and updating attributes of the tokens. This points to a simple formula-like collection of statements, basically consisting of assignments. The time delay is also included in the code with the predefined variable *time*.

Hierarchy

A modelling tool must support hierarchical design. Mainly one wants to gradually split the problem into smaller pieces. In PNs there is a rule that says that every node can be extended to a become net, which is quite easy to understand. The rule for SNs says that for any transition (not place), a subnet may be substituted. This rule is stronger and creates a strict tree structure. As a consequence, a subnet can only be connected to the outer world by places in the supernet.

A net or subnet must have a name (identifier) and is in the graph inscribed in a rectangle. The node names inside a net are local and can be identical to names outside the net. Every object is referred to by dot notation starting from the highest level (e.g. RootNetName.{SubnetName.}ObjectName).

This hierarchy does not affect the analysis of the net. It is very easy to remove the hierarcies, i.e. to flatten the net. The only thing to take care of is to avoid name clashes. This is easily done by applying the dot notation explained above.

Analysis of Simulation Nets

A PN has a well defined theory as a base for analysis of nets. With the extensions much of the analysis ability is lost.

To obtain the same ability for a SN, we can try to extend the rules for a PN. This leads to problems with the complexity of the net. To overcome the problems we will try a solution in two steps.

- Perform transformations of the net, by replacing extensions with PN constructs. The new net must be equivalent to the original SN with regard to analysis, not to simulation.
- Use extended analysis functions for the new hybrid net.

As a complement to the simulation tool, there will be an application for automatic analysis of nets. This project is however only in it's initial phase.

A MODELLING EXAMPLE

Spare Part Case

The problem description is taken from (Törn 1991).

A certain machine uses a type of part which is subject to periodic failure. On failure the machine is turned off, the part is removed and, if available, a good part is installed. A repairman is responsible for repairing failed parts. His duties also include repair of other items, with a higher repair priority.

The objective is to investigate the fractional utilization of the machine as a function of the number of spare parts. The details are the following:

- Lifetime for parts: normal distribution (350,70), unit: hours.
- Repair time: normal distribution (8,0.5), unit: hours.
- Time to remove: 4 hours.
- Time to replace: 6 hours.
- Other items arrives with a mean interval time of 9 hours according to the poisson distribution.
- Service time for other items: uniform 4 - 12 hours.

Figure 1 shows a SN representing one interpretation of the problem. We want to emphasize some details of the model, as representative for SN properties:

- Since the goal is to obtain the optimal number of spare parts in *SpareParts*, this number is given by a variable, noted with a dollar sign. The interpreter will discover that, and open a window permitting the user to assign and reassign the value at run time.

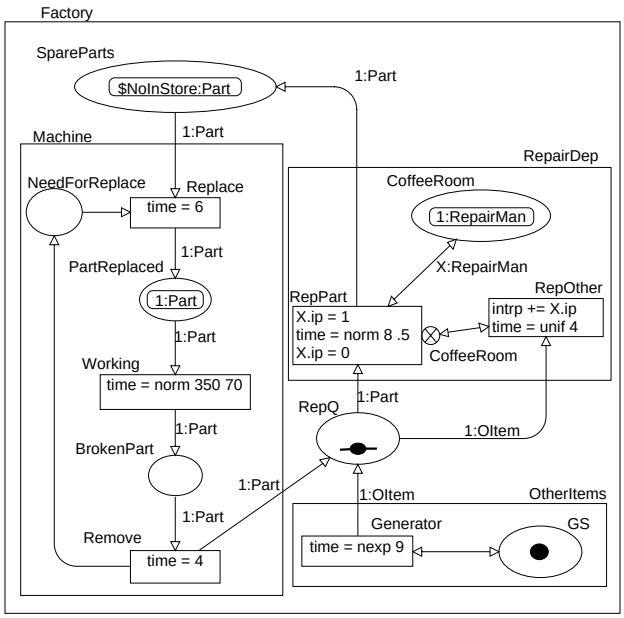
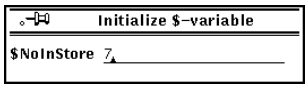


Fig 1. Spare Part Case modelled in SN.



- In a PN priority can only be expressed by an inhibitor arc. This is replaced by the *priority* queue discipline of *RepQ* (expressed by the dot/bar symbol). A spare part will automatically be placed behind the other items if any. The priority is strengthened by the interrupt arc between *RepPart* and *RepOther.*
- The transition formulas contain abbreviation operators such as += in *RepOther* (A += B ≡ A = A + B).
- Token attribute *ip* is used for interrupt checking. It is 1 when X is obtained by interrupt, 0 otherwise. The *intrp* attribute in *RepOther* counts how many times interrupt occurs.
- The different departments are reflected in the hierarchical structure.

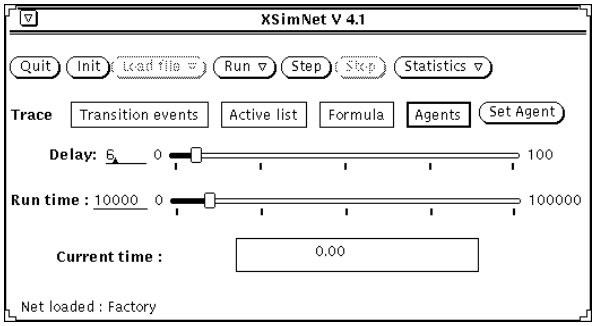
The graph will have the following SNL code:

```
Netname              Factory
     Subnet          Machine
     Subnet          RepairDep
     Subnet          OtherItems
     Place           SpareParts random
          init       $NoInStore Part
     Place           RepQ prior
Netname              Factory.OtherItems
     Transition      Generator
          in-out     GS 1 any
          out        RepQ 1 OItem
          code       time = nexp 9
     Place           GS
          init       1 any
Netname              Factory.RepairDep
     Transition      RepPart
          in         RepQ 1 Part
          in-out     CoffeRoom X RepairMan
          out        Storage 1 Part
          code       X.ip = 1
                     time = norm 8 .5
                     X.ip = 0
```

```
      Transition       RepOther
            in         RepQ 1 OItem
            code       intrp += X.ip
                       time = unif 4 12
            interrupt  RepPart CoffeeRoom
      Place            CoffeeRoom
            init       1 RepairMan
Netname                Factory.Machine
      Transition       Replace
            in         Storage 1 Part
                       NeedForReplace 1 any
            out        PartReplaced 1 Part
            code       time = 6
      Transition       Working
            code       time = norm 350 70
            in         PartReplaced 1 Part
            out        BrokenPart 1 Part
      Transition       Remove
            code       time = 4
            in         BrokenPart 1 Part
            out        RepQ 1 Part
                       NeedForReplace 1 any
      Place            NeedForReplace
      Place            BrokenPart
      Place            PartReplaced
            init       1 Part
Token                  Part 4
Token                  RepairMan 4
      attrib           ip
Token                  OItem 8
```

THE SIMULATION TOOL XSIMNET

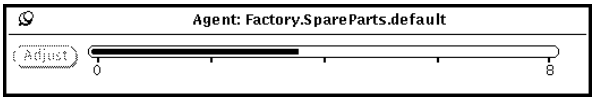
XSimNet is based on the Xview interface to UNIX, which is a high level user interface with no need of writing commands. The main command window looks like this.



The command window allows the user to run the model using predefined options. The SNL code does not include any description how to run the model. It is only a statical description of the net and the execution is performed according to the rules and the user's requests. Statistics and trace are displayed in an output window, which have all the properties of a text editor, except for loading files.

Some properties of the tool

The interpreter is constructed for simulation and is emphasizing the finite result more than step by step animation. At this stage we don't have the form of animation that allows following the tokens traversing the net. For a longer simulation, the advantage of such an animation is questionable. Other types of animation, using bars and curves, might be more informative. In XSimNet there is a possibility to have animated horizontal bars which shows the value of a chosen attribute at run time. This animation is connected to so called agents. An agent is a special reporting object, which can be set to watch a special attribute. Each time the attribute changes it's value, the agent will report the new value, by text output or by changing the state of the graphical bar representation.

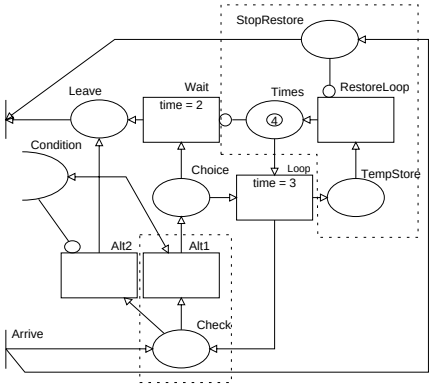


For final statistics there are two fixed output formats and one programmable. In the latter case the interpreter will compute the result according to a script file with a special syntax.

FUTURE PLANS

Drawbacks of the current model

The qualities of PNs in modeling of systems has been documented in several papers and research reports. When the model is simple and small the graph is also very readable and gives an immediate understanding of the problem. With growing complexity however this clarity might be lost. There can be so many constructions of more or less trivial primitives in the net that they cover the overview. The construction below, taken from a traffic light simulation, exemplifies this.



A token appears in the system from the *Arrive* transition and disappears at *Leave*. The graph expresses the following:

```
for i = 1 to 4
     if not Condition then Leave
     else wait 3
wait 2
Leave
```

All nodes inside the dotted lines is just a construction for the loop and choice mechanism. They could all be removed if we had some numerical attribute which was decremented and tested.

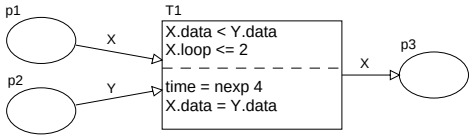
As the intention with SN is to advance simulation modeling, one may ask if the current modeling power is sufficient. The answer is clearly no, more power is needed. In the following we will cover some interesting modifications for extending the modeling capability.

Testing token attributes

As mentioned above the main principles for the transition formulas was just data collection. No influence on the execution of the net is possible. When firing a transition we can test for the token types in the in-places, but there is no possibility to check the token attributes. As a consequence a lot of dynamics is lost.

The planned solution to this problem is quite similar to that of ER-nets (Ghezzi et al. 1991). A boolean predicate is set up,

which must be satisfied for the transition to fire. We will include this in the transition box, placed before the action code and separate it with a dotted line. The following figure shows an example graph and the corresponding SNL code.

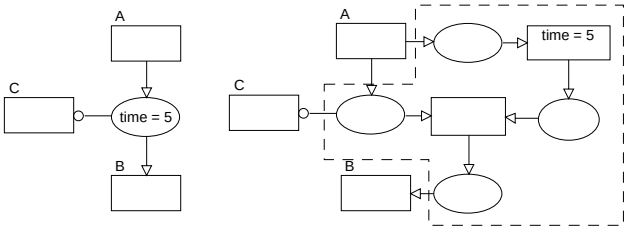



```
Transition T1
     in    P1 X type
           p2 Y type
     out   p3 X type
     cond  X.data < Y.data
           X.loop <= 2
     code  time = nexp 4
           X.data = Y.data
```

Expanding the time concept

The introduction of the time concept in PNs, has been discussed in several papers. The time concept preferred is oftenly dependent of which stand point one has or what the intentions with the model are. Time connected to transitions can in some situation be efficient, but in other cases not preferable.

We may ask: Why shall a net, and especially a SN, which is intended for simulation, have just one time concept? The clarity of the model will not suffer from using more and different time concepts, if these are carefully defined. Instead this will lead to reduced net complexity. A good example is the use of inhibitor arcs for preventing some actions for a certain time. In the graph below transition C is inhibited for 5 time units from the end of transitions A to the start of transition B.



If we can't assign time to a place, we have to make a disturbingly complicated construction. The single place is expanded to the net inside the dotted line in figure to the right. The new model is not only smaller, it is also closer to our intensions.

A different way of assigning time to transitions was already mentioned in the chapter above as case 1. An enabled transition remains waiting before it can fire. This construction allows another transition to "steal" the enabling tokens. It is a useful mechanism for modeling time-outs and could easily and naturally be included in the header for condition testing introduced in the proceeding section.

Expanding the tokens

As well as assigning time directly to places we can connect time to a token. The time stamped token (Jensen 1992) will have exactly the same function, and is a more natural concept. The main reason is the passive character of a place. The solution, if we want the time to be place dependent, is to allow the token to have several time stamps, each one connected to a specific place.

This leads to adding code to the tokens. The next step is the exciting idea to go from colored to object oriented tokens. As the dynamic part of the net, tokens are well suited for carrying code and the objectoriented concepts public and private variables and methods, will be the best way to introduce the code.

Related to the implementary parts of the project, a design tool for tokens is the needed. Only the statical net has up to now been subject to graphical design. Later on an object language must be introduced. This idea is for the moment only in it's first stages and only a very simplified prototype of a token designer exists.

Other expansions

Future plans contain the design and implementation of:

- Libraries for often used and parameterized graph modules.
- Interpreter extensions, so it can also show the execution of the net.
- Analysis techniques for the simulation net.

REFERENCES

Evans J. B. 1993. "The Net Semantics of Demian." Technical report TR-93-11. Department of Computer Science, Faculty of Engineering, University of Hongkong. (Dec.)

Ghezzi, C., D. Mandrioli, S. Morasca and M. Pezz'e. 1991. "A Unified High-Level Petri Net Formalism for Time-Critical Systems". IEEE Trans. Software Engineering (Feb): 160-172.

Javor A. 1993. "Petri Nets in Simulation". EUROSIM - Simulation News Europe. no. 9 (Nov): 6-7.

Jensen, K. 1992. Coloured Petri Nets. Volume 1. Spinger-Verlag, Berlin, Heidelberg.

Papelis Y. E., and T. L. Casavant. 1992. "Specification and Analysis of Parallel/Distributed Software and Systems by Petri Nets with Transition Enabling Functions". IEEE Trans. Software Engineering (Mar): 252-261

Peterson, J. 1981. Petri Net Theory and the Modeling of Systems. Englewood Cliffs, N.J. 07632: Prentice-Hall.

Reisig W. 1985. Petri Nets, An Introduction. Springer-Verlag. Berlin Heidelberg.

Törn, A. 1981. "Simulation Graphs: a general tool for modelling simulation designs". SIMULATION 37:6: 187 - 194.

Törn, A. 1991. Simulation Modelling. Åbo Akademi University, Reports on Computer Science & Mathematics, Ser. B, No 12, 140 pp.

IWong C.Y., T. S. Dillon and A. E. Forward. 1985. "Timed Places Petri Nets with Stochastic Representation of Place Time." In proceedings of International Workshop on Timed Petri Nets (Torino,, Italy, July 1-3). IEEE, Computer Society Press, Silver Spring, 96-103.