This paper was presented at the IMACS European Simulation Meeting, at the Szechenyi Istvan College in Györ, Hungary, August 28 - 30, 1995. Editor: Andras Javor.

OO TOKENS AND SENSOR ARCS IN THE TOOL XSIMNET

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

ABSTRACT

Sensor arcs are extended inhibitor arcs, which can test for more than zero. Object oriented tokens is a powerful extension of typed data tokens. By combining them and other more classical Petri net extensions we achieve a new level of modelling ability. It opens possibilities for a combination of discrete and continuous modelling paradigm.

INTRODUCTION

Simulation Nets (SNs) are Petri nets extended for convenient modelling of discrete event simulation problems [5] [6] [8] [9]. 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 is an extension of SimNet that can deal with coloured (typed) tokens and textual extensions (code) in the graph [1].

XSimNet is implemented in C++, on the UNIX system. It has an interface to the xwindows system, facilitating interactive use via a control panel on the screen. The tool accepts a text equivalence of a restricted set of SNs and performs the simulation implied by the SN model. In the current state the net must be described in the textual representation by the modeller.

XSimNet uses most of the classical PN extensions, like inhibitor arcs, time etc. The net can be designed as a hierarcical structure of subnets.

The time is realized as a delay in transition executions. This time is however introduced as a part of more general transition inscriptions. In these inscriptions data attributes in tokens or transitions can be initialized or changed. The system provides some built-in stochastic functions. There are no boolean statements for testing enableness in this version.

The time concept makes it possible for a transition to interrupt another transition, which is delayed in its execution. Interrupt transition-to-transition arcs are introduced and implemented in XSimNet. One transition can grab a specified resource (token) by stopping another transition.

Stochastical or-logic can be applied to arcs. A transition is enabled if one of two in-places or both together contain the required tokens. Weights can be used for preference of in-places. Another new extension is queue disciplines for assignment to a place.

SOME EXTENSIONS TO SIMULATION NETS

The experience with XSimNet have convinced us that it would be worthwhile to increase the modelling power of SNs by some new extensions. A new version is under construction, which will be more powerful. Some desirable and practical extensions which have been used in other tools are the following.

- Explicit enabling conditions in transitions and on arcs. These are explicit boolean statements for testing of attributes of the tokens and the transition. The transition is split into a conditional and an action part, which is expressed graphically with a dividing horizontal line.

- Standard attributes in places and transitions.
- Priorities for transitions.

There are however two new extensions, which we find more interesting and promising. In the following subsections we will give a deeper introduction of those.

Object Oriented Tokens

The coloured tokens in XSimNet, were implemented in a restricted way. They can have data attributes and are able to store data, but these attributes were not allowed to influence on the execution of the nets. A natural extension of typed data tokens are programmable object oriented (OO) tokens [2]. These extended tokens can participate in a model in a more sophisticated way. They can influence on their own routing in the net based on their knowledge of the system state. They can learn from experience i.e. the basis for an intelligent choice is established. They can also participate in creation and destruction of each other.

In order to extend the knowledge base and decision making of tokens we can let them communicate with each other. This will facilitate new modelling possibilities.

- Tokens can exchange information, i.e. learn from each other.
- Tokens or different parts of the net can be synchronized in time.
- The net can be split into different independent parts and be connected by token communication. This is a new way of structuring the net and is called aspect decomposition [3]. The conceptual idea is that one and the same token can appear in more than one net at the same time.
- OO tokens can take the role of intelligent objects. They can easily enable or disable a part of the net, e.g. add more servers if the queue lengths become critical by duplicating themselves or block actions by refusing to fire transitions.

Programmable tokens give another possibility to introduce time. A token can stay invisible for a time in a place. Time assigned to places is more convenient to use than transition delays in some models. Having two ways for expressing time delays in the net increases the usefulness of the tool.

Inheritance must be implemented so that one token type is a subtype of another. We don’t implement multiple inheritance and a token cannot include another token type as an attribute. Virtual tokens, i.e. token types never intended to appear in the net, acting only as superclasses for other tokens, have also a useful effect: An arc restricted to transfer a certain token type is valid also for all possible subtypes.

By introducing more powerful tokens we will increase the dynamic control of the otherwise more static net. This leads to interesting alternative ways of modelling.

Sensor arcs

Sensor arcs were introduced in [4] and is a kind of extension and inversion of inhibitor arcs. The idea is to expand the concept of an arc that performs only testing with no consuming involved and to test for more than zero. The extension can be introduced in several steps.

❶ Instead of zero-testing we have $<k$-testing, where k tokens are required to inhibit the transition from firing. For a common inhibitor arc we have $k=1$.

❷ Multiple arcs in both directions (self loops) will be replaced with $\geq k$-testing, which will be equal to an ordinary double arc, but with no token consuming involved. For simulation of the model this has practical consequences. Consuming and depositing increase the run time and eventual statistics of arrivals in a place can be distorted and give misleading results.

❸ Combining sensor arcs with other extensions, such as arc- and transition inscriptions, gives a useful tool, which may be used to model continuous Petri Nets.

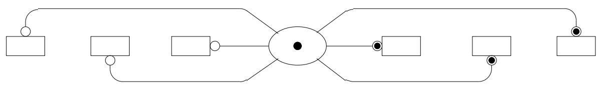
Figure 1 : One single token controls a cascade of transitions via sensor arcs.

The basic graphical symbol for a sensor arc that we will use is an inhibitor arc and for the positive sensor arc a black dot inside the circular head, symbolizing a token in the place. In case of a positive sensor arc, *k* or more tokens permits the transition to fire, and in the opposite case *k* or more tokens inhibits the transition. Sensor arcs are specially useful when several transitions are controlled by one single condition, often modelled by the presence of a token in a certain place, see Figure 1.

COMBINATIONS OF OO TOKENS AND SENSOR ARCS

In the following we will focus on some interesting modelling abilities which emanates from the combinations of sensor arcs and OO tokens.

Figure 2 shows the main principles in the combination of OO tokens and sensor arcs. To understand the net one should know that the capital *X* on an arc is a local declaration of a token. The scope of the declaration is the transition. This provides an identifier for the token which can be used for accessing its attributes. The code in the transitions is an action performed on a token attribute and a statement on an arc inside square brackets is a boolean test. In Figure 2 a) we have some anonymous places (greyed) symbolizing the connection with the rest of the net and a place *p1* with an OO token. Its relations to the surrounding transitions are the following:

① The read principle. The firing of transitions *t1* and *t3* are dependent on an evaluation of conditions using an attribute of the OO token. As one can see, depending on the attribute data, both or only one of the transitions is enabled, so the nonconsuming is essential.

② The write principle. The transitions *t4* and *t6* are not dependent on *p1*. Instead when firing they update the attribute of the OO token. We might say that the always present token “listens to” the transitions.

③ The traditional token consuming and depositing can of course be combined with sensor arcs. If the OO token is consumed by *t5* none of the operations described above can take place. The firing of *t5* can of course also be dependent on the data attribute. When a new or the same token is deposited by *t2* the process can start again. More than one token can also be present in *p1*.

For clarity we have chosen a transparent operation on a public attribute in the example. In most cases, the data of the token will be private and the operation will be performed by calling a method attribute.

Figure 2 b) shows how the communication ability of the OO tokens can be added to the possibilities described above. Operations for changing the token in *p2* can be transferred to the token in *p3* and thus influencing its neighbouring transition. The two tokens can even occur in different nets.

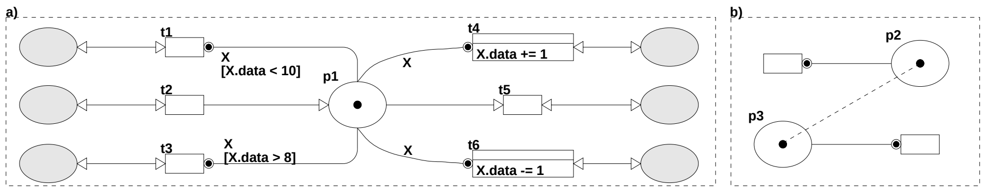

Figure 2 : Combination principle of OO tokens and sensor arcs.

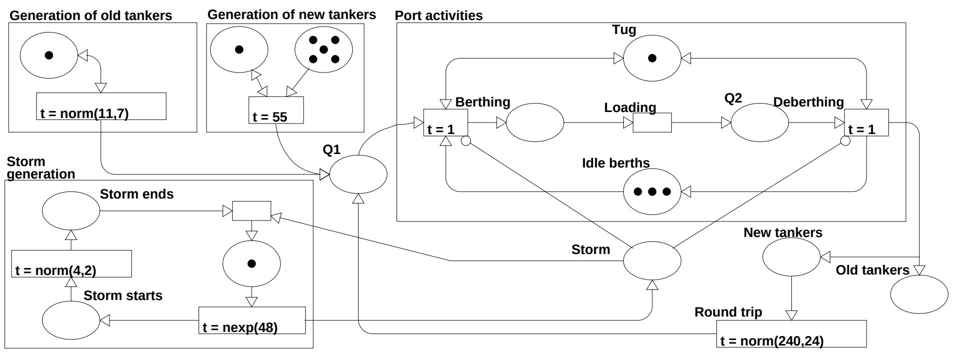

Figure 3 Model 1 of oil tanker accommodation at a port.

CASE STUDY - OIL TANKERS AT PORT

A problem that can be found in many books on simulation is about a port used for loading tankers with crude oil. The data according to Schriber [7] is the following:

- Tankers of the following three types arrive at the port every 11 ± 7 hours.

Table 1: Tanker specification

Type	Relative frequency	Loading Time, hours
1	0.25	18 ± 2
2	0.55	24 ± 3
3	0.30	36 ± 4

- There is only one tug at the port serving tankers of all types. All tanker require the tug for moving into and out of a berth. The berthing and deberthing activities require 1 hour.
- The area experiences frequent storms when no berthing or deberthing of a tanker can take place. Storms last 4 ± 2 hours and occur according to exponential distribution with a mean time of 48 hours.

A shipper is considering bidding on a contract to transport oil from the port to the UK. He has determined to use five tankers of a fourth type. These tankers would require 21 ± 3 hours to load oil at the port. After loading and deberthing, they would travel to the UK, offload their oil, return for more oil etc. The round-trip travel time, including offloading is estimated to be 240 ± 24 hours. The operation of the port under the proposed commitment is to be determined.

Old Model 1981

A SN model for this problem was first presented at the SIMS Annual Meeting in 1981 and then published in Simulation [8] and in [9]. Figure 3 shows this model in its original shape. The model is adjusted to the power of the former tool. The non coloured tokens can not express the different tanker types. Therefore the timed transition *Loading* must have a mean time or the model must be remarkably extended. These details are not specified at all in this model. At the output from the port or transition *Deberthing* the old tankers are deposited in an end place while the new tankers are circulating. For this purpose the or-arc has been introduced. Even in this case the implementation requires some specification for separating between the token types. All these problems are solved very easily in the next model with OO tokens.

The storm conditions are modelled exactly as in the problem description as a discrete state occurring according to distributed time intervals. The place Storm has inhibitor arcs for prevent-

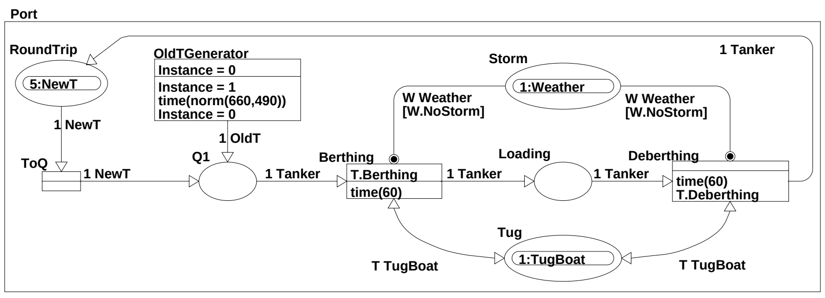

Figure 4 : The Port part of the new model of tanker accommodation at port.

ing tankers to arrive or leave. The token in the place is deposited and removed by the storm generation subnet.

Extended model for the new tool

By modelling the same problem using the new extensions we want to point out their advantages.

Despite its simplicity the definition of the storm generation is quite complicated in the old model. This is because time can be applied only to transitions. With OO tokens we can build the same model using one transition plus the *Storm* place. The token stays in the place when storm conditions apply and otherwise in the transition. To make the model more interesting we shall assume a more complicated storm condition. We will have a partly continuous model for the storm generation. In our case it means the updating of a continuous variable with very short time intervals.

For the new model we shall use the ability of decomposition and data transmitting sensor arcs. We will have one model similar to the old one and a separate model for the storm generation. A token of type *Weather* is simultaneously present in both of them. We will also choose minutes and not hours as the time discretisation.

The *Port* model is shown in Figure 4. It is simplified by the programming ability of the tokens.

- The *Tugboat* counts the tankers that arrives and leaves so they will never exceed three. The place for idle berths can then be omitted. The transition *Berthing* has a conditional part where the *Berthing* attribute in the *Tugboat* is called. This gives the *Tugboat* a possibility to prevent the transition from firing.
- The *NewT* tokens are initially deposited in *RoundTrip* and will individually calculate the start time at which they will set themselves available.
- The time for loading is determined by the ability for tokens to stay invisible in the place *Loading*. The tokens calculate the time according to tanker type.
- The storm condition is determined by a function *NoStorm* of the token in the place *Storm*. The outcome of the function is dependent on several variables of the token, corresponding to case ① above. In this part of the model we do not know or bother about the realisation of the storm condition. The *Weather* token is updated in another net.
- All tanker tokens have a procedure which is invoked in the place *RoundTrip*. The old tankers destroy themselves and the new tankers remains in the place for a calculated round trip time. The types *OldT* and *NewT* have a superclass *Tankers*. The superclass contains all common attributes, e.g. a *Loading* routine. From the arc inscriptions one can see their common route (*1 Tanker*) and where they differ (*1 NewT*).

The use of local user defined attributes in transitions reduces the generation of old tankers to a single transition, *OldTGenerator.* The place in the older model is used to limit the ongoing transi-

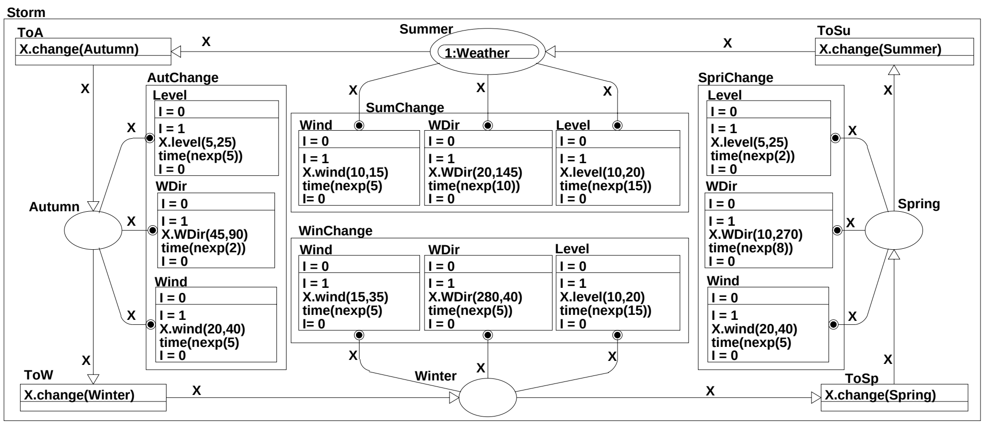

Figure 5 : The changes of seasons and its influence on weather conditions.

tions to one at a time. The attribute *Instance*, used in the conditional and action part of the transition, performs the same job.

Figure 5 shows the change cycle of the storm conditions. A *Weather* token moves around in a circle, modelling the seasons of the year. This represents a discretisation of the time, which if needed could be made finer. For each season there is a corresponding subnet, containing three transitions. For each transition there is a sensor arc for updating values in the token. This represents the case ② above. Each transition calls an updating procedure in the token with some parameters. The transitions are the same for each season in this case but the updating parameters differ, representing different conditions for different seasons. Of course this is not a strictly correct metherological model, rather a fictive example to express the modelling ability of the new SN. The weather is a good example in that it is independent on the port activities. A more realistic model of metherological conditions will be much more complicated.

The function *NoStorm* can have the definition which is shown graphically in Figure 6. For the real implementation of such a function in the tokens, a special language will be introduced which supports such constructions. The function has three variables as inputs, the same ones that are updated in the *Storm* net. The variables are evaluated according to fuzzy logical functions, multiplied by a weight, added and transferred through a threshold function. The weights express the importance of each variable and the interpretation of them is the following.

- *Wind* means the strength of the wind and is naturally causing storm when its value increases.
- *Direction* is the direction the wind blows and is measured in degrees. The harbour is assumed to be situated so that some directions are more critical than others.
- The *Water-level* is also important in that when its low the requirements for the other variables decrease.

The changes which takes place in the *Storm* net occurs within random but small time intervals.

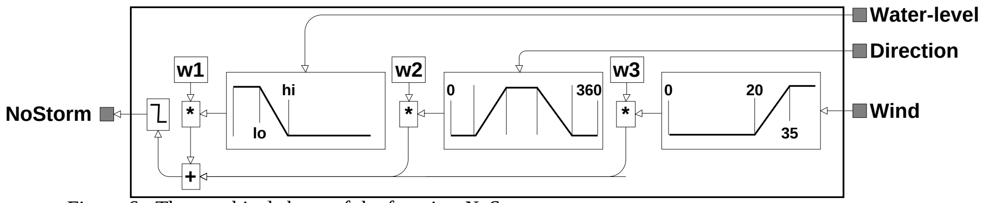

Figure 6 : The graphical shape of the function *NoStorm*

The changes in one step is also assumed to be small. The model is no more treated as a discrete one. The *Port* net however is still the traditional SN model. We have in this manner introduced continuous parts in an otherwise discrete event simulation.

CONCLUSION AND FUTURE PLANS

A PN model is built up from a very small amount of primitives. If we add more to a PN, we must assure that the additions are simple, consistent and possible to combine with other primitives. In that sense we adopt to one of the main principles of Petri nets. Our intentions for adding new features to the net must follow these principles and one should not introduce too many extensions. Therefore multiusefulness is a very important property of the extensions. The multiusefulness is truly fulfilled by both OO tokens and sensor arcs and the combination of them has also turned out to be useful.

In the next generation of XSimNet all new extensions mentioned above will be included. We intend to have a first version of the new system running in 1996. This will include a graphical editor and an interpreter for nets containing the extensions and properties described above. The editor will have a designer for token programming and construction of the nets. For the moment a beta version of the design editor exists. The interpreter will have extended animation possibilities and will also include an interpreter for the token programming language. In all, this represents a large step forward from the current version of XSimNet.

REFERENCES

[1] Gustavson Å. and A. Törn. 1994. *XSimNet, a Tool in C++ for executing Simulation Nets*. In Proceedings of 1994 European Simulation Multiconference (Barcelona, Spain, June 1-3 1994), 146 - 150.

[2] Gustavson Å. and A. Törn. 1994, *Object-Oriented Tokens, A Way of Increasing the Modeling Power of Simulation Nets*, In Proceedings of 1994 European Simulation Symposium (Istanbul, Turkey, Oct. 9 - 12 1994), 97-101.

[3] Gustavson Å. and A. Törn. 1995. *Decomposing Simulation Nets by Token Communication.* In Proceedings of 1995 European Simulation Multiconference (Prag, Chech, June 5-7 1995).

[4] Gustavson Å. and A. Törn. 1995. *Extending Simulation Nets with Sensor Arcs*. In Proceedings of SIMS'95 Simulation Conference (Lyngby, Denmark, June 28-30 1995).

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

[6] Reisig W. 1985. *Petri Nets, An Introduction*. Springer-Verlag. Berlin Heidelberg.

[7] Schriber T.J. 1974. Simulation using GPSS. Whiley, New York.

[8] Törn A. 1981. *Simulation Graphs: a general tool for modelling simulation designs*. SIMULATION 37:6: 187 - 194.

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