This paper was presented at the SIMS'95, 37'th Simulation conference at the technical university of Denmark, June 28 - 29, 1995. Editor: Niels Houbak.

EXTENDING SIMULATION NETS WITH SENSOR ARCS

Åke Gustavson and Aimo Törn
Åbo Akademi
Computer Science Department
SF-20520 ÅBO, Finland
E-mail: agustavs@ra.abo.fi

ABSTRACT

Sensor arcs are extended inhibitor arcs, which can test for more than zero. By combination with other extensions, previously introduced in Petri nets, they can express primitives which otherwise have to be modelled in the graph. This complexity reduction of the graph is essential for the descriptive clarity of the model.

INTRODUCTION

Since Petri Nets (PNs) were introduced by C.A. Petri 1962, they have been widely used for modelling in different areas [Reisig 1985]. With their simple concepts of transitions, places and tokens, they can express actions, conditions and states. However, a PN representation of a large and complicated systems will contain an unacceptable level of detail. As a result, many users of PNs developed useful extensions to fit their specific needs.

Simulation Nets (SNs) are PNs extended for convenient modelling of discrete event simulation problems [Törn 1981,1991]. Our intentions for studying PNs is to find new suitable extensions for SNs, for simplifying simulation modelling. These extensions includes both tokens and the net. The intention is to have the net to reflect the model, but to avoid technical low level constructions. In this paper we will concentrate on an extension of the arcs which we will call sensor arcs.

SENSOR ARCS

The idea is to expand the concept of an arc that performs only testing with no consuming involved. We will start with inhibitor arcs and then perform stepwise extensions of them into sensor arcs.

① An inhibitor arc is a kind of inversion of a the PN arc. The technique is also called zero-testing and that is what an inhibitor arc actually does. For reasons clear from the extension of inhibitor arcs introduced below, we will use "less than 1 testing" (<1-testing) instead of zero-testing.

② Instead of <1-testing, one can use <*k*-testing, where *k* is the minimal amount of tokens to prevent the firing of a transition. In the graph *k* is written as a multiplicity next to the arc. For *k*=1, it is equivalent to zero testing i.e. a conventional inhibitor arc. With <*k*-testing one can eliminate some low level details, for an illustration see Figure 1.

③ Assume that we could use an inverted inhibitor arc, which will test and give true for not zero. In analog with <*k*-testing we can talk about ≥*k*-testing. It can be used instead of multiple arcs (self loop), but with no consuming involved.

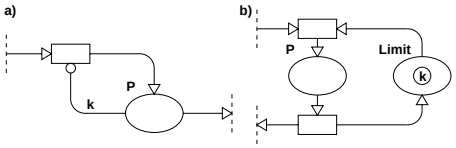

Figure 1 Using <*k*-testing for bounding a place (a). This can otherwise be done to the cost of a more complicated construction (b).

④ As a result we will introduce sensor arcs, which can test both ways. We will call them positive and negative sensor arcs. It is to be emphasized that the no consuming and the limitation to in-places finally distantiate them from conventional arcs. As a graphical symbol we will use 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. From this definition, it is clear that *k*>0.

Sensor arcs are specially useful when one or more transitions should be controlled by one single condition, often modelled by the presence of a token in a certain place. Consider the following situation, shown in Figure 2 a). There is a place *P1*, for controlling two transitions, *T1* and *T2*, so that they are mutually exclusive. For this purpose there is an inhibitor arc between *P1* and *T1*, and a double arc from *P1* for control of *T2*. *P1* sometimes contains a control token, which is deposited/consumed by an external part of the net. The place *P2* will sometimes receive a token for firing either *T*1 or *T2*.

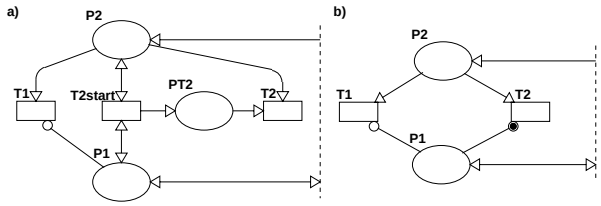

Figure 2 The example problem model in (a) is reduced by use of sensor arcs (b).

The test of *T2* is unnecessarily complicated and involves low level constructions in the net which can be avoided, see Figure 2 b). The problems in Figure 2 a) can be summarized as this:

- If the places are equipped with queue disciplines, the positive test can alter the order, which can be essential.
- Eventual statistics of arrivals in *P1* can be distorted and give misleading results.
- Consuming and depositing increase the run time.
- In case of time delay in *T2*, the place *P2* has to be one-safe. Otherwise the control mechanism would not work and we need a still more complicated construction.

When more than two transitions are involved sensor arcs become really helpful.

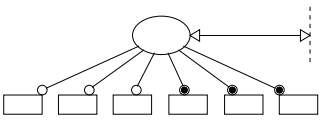

Figure 3 One single token can control a cascade of transitions.

ANALYSIS OF SENSOR ARCS

It has been showed that inhibitor arc PNs have the computational power of Turing machines [Peterson1977,1981]. In the general case, inhibitor arc PNs cannot be transformed into ordinary PNs. If the places of the inhibitor arcs are bounded, than the transformation is possible.

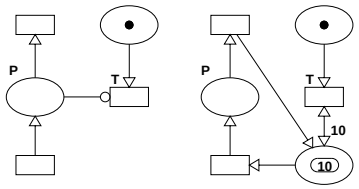

Figure 4 Example of removal of an inhibitor arc, where 10 is assumed as the bound for P

The first expansion, <*k*-testing, can be transformed in a similar way, as can be seen from an example (Figure 5).

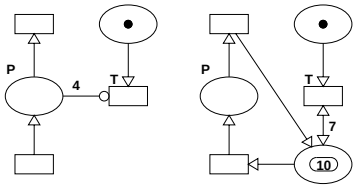

Figure 5 Example of transformation of <k-testing where k = 4 and 10 is the bound for P

The positive case, ≥*k*-testing, is analytically the same as a self loop. The none consuming property can be very useful in simulation modelling, but plays no role in analysis. For sensor- arcs we can regard two cases:

① Sensor arcs are added to a original unextended PNs. Interpreted as a generalisation of inhibitor arcs they increase the modelling power of PNs in a fundamental way.

② If the PN contains inhibitor arcs, then it is already a Turing machine and the addition of sensor arcs is a tool for getting a smaller and more readable net. Sensor arcs must then be regarded as an abbreviation property of the PN.

SENSOR ARCS AND ARC CONDITIONS

Up to now we have only used an integer to express the maximum or minimum number of tokens in case of sensor arcs. In a SN there is a possibility to name the tokens traversed by an arc. More than one token must then be given as a list. These names are local to the transition and can be used in the transition code and for routing of the token. Along with the number or local name the type of the token has to be given. This forms a condition, because if the required type is not found in the in-place, the transition is not enabled. In addition an explicit condition can be given, referring to some inner states of the token. Such a condition is local to the arc. In the same way as with ordinary arcs this can be applied to sensor arcs.

For an arc to be *valid*, *all* of its conditions must be fulfilled. An explicit arc condition will strengthen a positive sensor arc, but the weaken negative sensor arc. For sake of clearness we write out this explicitly:

- A valid sensor+ arc will *contribute to the enabling* of the firing of a transition.
- A valid sensor- arc will *inhibit* the firing of a transition.

SIMULATION NETS

As mentioned in the introduction extensions for simplifying modelling are included in SNs. One interesting point is how sensor arcs fit in this environment and how they cooperate with other extensions. Here follows a short overview:

- The tokens are expanded from typed and numerical tokens to intelligent object-oriented (OO) tokens [GustavsonTörn 1994b] . Such a token has identity and can carry even complex code. It can remember its history, make decisions based on knowledge and can learn by experience. In order to extend the knowledge base and decision making of tokens we let them communicate with each other. This is the most significant difference between our token model and other similar models. OO Tokens are permitted to duplicate or destroy themselves and have also the possibility to set themselves invisible. An invisible token cannot be consumed by a transition. This is a way of introducing time in places but can also be used for other purposes, e.g. having a token to wait for a message from another token, possibly residing somewhere else.
- Transitions play an important role in SNs, and are split into a conditional and an action part, see Figure 6. As in many other extensions of PNs, restrictions for enabling of a transition are allowed [Bandinelli et. al. 1993][Jensen 1992]. Attributes in transitions and tokens can be changed as a part of the action. Time delays can be applied to both parts. The transition enabling conditions fall in the same cathegory as the explicit conditions for arcs, but have a wider scope. It is for example possible to compare values of two tokens from different places, each connected by its own arc.
- Other extensions are stochastic or-logic arcs, transition interrupt arcs and hierarcies of subnets which could be

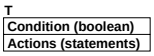

Figure 6 The graphical notation of a SN transition.

indexed.

COMBINATION WITH OTHER EXTENSIONS

Sensor Arcs can be combined with other extensions. We have mentioned the arc inscription above. A discussion will start the from an example showing some possibilities.

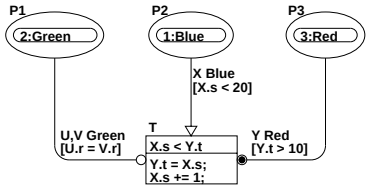

Figure 7 A collection of possibilities in combining sensor arcs with boolean inscriptions.

In order to interpret the situation in Figure 7 we shall first investigate if transition *T* is enabled. It depends on three in-places, all connected with different arcs. All tokens have local names. We shall go through them one by one.

① *P1* is connected with a negative sensor arc which inhibits *T* in case of 2 token of type *Green*. Since there are 2 *Green* tokens in *P1, T* should not be enabled if there were no arc inscription. The inscription *U.r = V.r* is however also essential for the inhibiting effect. It must true. So in this case the enabling of *T* from *P1* depends on the token attribute *r* in token type *Green*.

② *P2* is a normal in-place to *T*. One token of type *Blue* is required and resides in the place. The enabling depends on if attribute *s* in type *Blue* is smaller than 20.

③ From *P3* emanates a positive sensor arc. The required token exists in three instances, so *T* is enabled if at least one of them fulfils the inscripted requirement of attribute *t*.

④ The conditional part of *T* compares attributes of tokens from different places. Even in this case there are three possibilities in *P3*. *T* will fire only if it can find a true solution for the condition.

⑤ In the action part of *T* the attributes of tokens *X* and *Y* are changed.

When the tokens are not consumed by the transition, there are some interesting question to be answered. Can we at all treat the tokens in a similar way as they were consumed? It is clear that token manipulations can only be applied in connection with sensor+ arcs, because as with conventional arcs the token must exist. Sensor- arcs represents the opposite case. By the existence problem, it is safer to rule out the possibility of all connection with sensor- arcs and transition inscriptions.

One can however be still more restrictive: Shall we permit the transition to change token attributes although the token stays in the place all the time? Remember that the token remaining in the place is accessible from the outside world. It is not exclusively controlled by the transition and can be changed or consumed by another transition. Clearly spoken this is not a problem with immediate transitions, it is completely related to some type of time delay in the transition. But time is introduced in this way, this problem must be treated.

The situation reminds of other cases where mutual exclusiveness is essential. Of course there are some strategies for handling this. We can sum them up as follows.

① After a transition has got a reference to a certain token, it can always manipulate it wherever it is.

② The situation is treated as an error. The interpreter terminates, stops or at least print out an error message.

③ The statements for manipulation are not executed if the token is removed. Eventually a warning message is printed out.

We prefer strategy one because of the resulting extended modelling possibilities, which we will describe in the following and let the programmer be responsible for avoiding unwanted executions. It is easy to rule out such events by using the transition conditions. A typical example of the problem and its solution can be studied in Figure 8. The transition *T1* is enabled and will fire with a delay of 100 time units. Within this time it is possible that a token in *P2* will enable *T2*, so it will remove the token in *P1*. Since we by some reason do not want that to happen we do some simple programming using an attribute *check* in *X*.

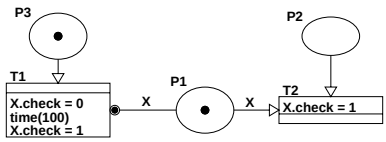

Figure 8 Preventing another transition from steeling a token.

MODELLING EXAMPLES

Modelling of Continuous PNs

In a continuous PN the marking of the places are no longer integer. Instead there is a positive real number and the firing is like a continuous flow. Models that are enabled by these nets cannot be transformed into ordinary PNs.

As we regard tokens as data carrying entities or objects, they are clearly discrete. They can however carry continuous attributes. Instead of having continuous properties in places, we can put a stationary token in that place and in this manner model non place properties of the place.

The example given in Figure 9 is taken from [DavidAlla 1994] and shows how french dressing is obtained by mixing salad oil with vinegar. A continuous place is represented by a double circle, useful when hybrid PNs are concerned. Figure 9 a) represents an initial state with 1 l of oil and 1 l of vinegar and no dressing The firing of T1 has a quantity which is not an integer, in this case it is 0.1. Since the weight of the arc *P1 → T1* is 2, then 0.2 is taken out of P1. Figure 9 b) shows the result of such fir-

ings, where 0.3 l of dressing is produced.

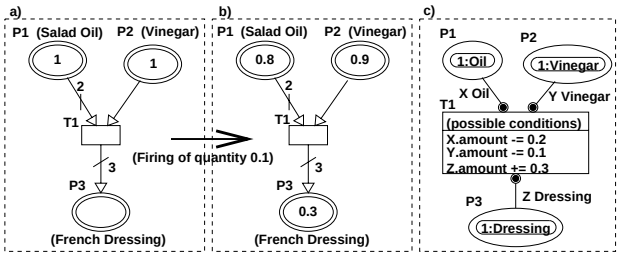

Figure 9 : Continous PN and its modelling with sensor arcs.

In Figure 9 c) we can see how the same process is modelled by numerical data tokens and sensor+ arcs. Here we explicitly program the action part of *T1* to perform the calculation of the continuous values, analogue to what is happening in the continuous net. The conditional part is omitted, but we have to ensure that the amount of the giving tokens is sufficient. The transition must also be enabled in the traditional way.

Modelling of External Events

The normal way of transition firing is by enableness. Synchronized PNs introduce the possibility of synchronizing the firing of transitions on external events. The external events corresponds to a change of state of the external world [DavidAlla 1994].

This is an extension to autonomous PNs, where the transition can fire if it is enabled, but we don't know when it will be fired. A simulation net is non-autonomous in the way that it is timed and an enabled transition will fire immediately. The only exception of this rule is when there is a conflict between two enabled transitions. In that case one of them will fire, but we don't know which one. In a synchronized PN a firing of a transition will occur *if* the transition is enabled and *when* the associated event occurs.

The diagram in Figure 10 b) explains how a motor switches between stopped and working state, trigged by external events *E1* (on) and *E2* (off). In Figure 10 a) there is an interpretation in a plain PN, where the states are modeled by places *P1* and *P2*. The transitions *T1* and *T2* are supposed to fire on the signals *E1* and *E2*, if they are enabled. As can be seen from both graphs an on-signal (off-signal) has no effect if the motor is on (off).

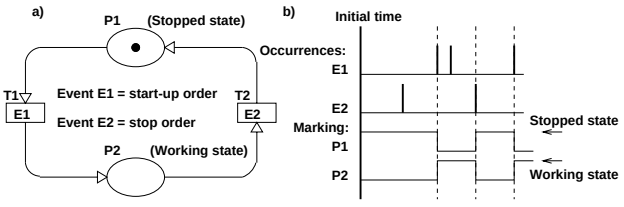

Figure 10 An example of synchronized nets from [DavidAlla 1994].

Sensor arcs combined with OO tokens gives us a possibility to include external events in the net. They also allow this to be done in a nice graphical way. Figure 11 shows how the synchronization can be expressed by an extended token and sensor+ arcs. The external events are managed by a token in *P3*. The OO token *Event* has the two attributes *E1* and *E2* that can be manipulated externally. The transitions *T1* and *T2* react on these attributes if they are enabled by *P1* or *P2*. Because the signal, i.e. the true state of *E1* or *E2*, should not remain, the transitions set them to false immediately. The transitions *T3* and *T4* will do the same with events which have no effect, as on-signals when the motor is working and the reverse. In this special case the transitions can work with only sensors as inputs, because the change of state in the token *Event* prevents them from firing more than once.

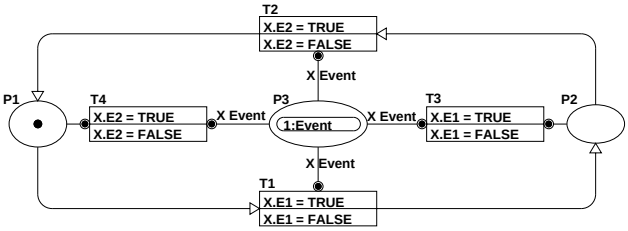

Figure 11 Modelling the example from Figure 10 with sensor arcs.

Traffic Light Control

In Figure 12 a) a traffic light example is presented for describing the modelling of a control mechanism with sensor+ arcs. Tokens representing cars are queued up in the place *Cars*. Red is represented by a token in the place *TrafficLight*. The transition *Change* symbolizes the mechanism for changing the state by consuming and deposing a token in *TrafficLight*.

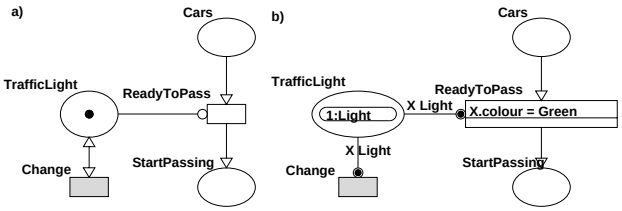

Figure 12 Traffic light model with ihibitor ((a) and sensor arc (b).

In Figure 12 b) The place *TrafficLight* contains an OO token of type *Light*. This type of token has an attribute *colour*. As can be seen in transition *ReadyToPass*, this attribute must have the value *Green* for the transition to be enabled, i.e. for the cars to start passing. The token in *TrafficLight* will never leave the place, but its attribute *colour* will change within periodically controlling the passing of the cars. Traffic lights with only one direction are however rare. If we expand the model to a more realistic one then the model with sensor+ arcs is more convenient. In Figure 13 one single token in the place *TrafficLight* controls two directions, modelled as subnets. It is easy to expand the model to more than two directions.

Figure 14 shows how to construct the control mechanism in a graphical way. *Change* consists of four timed transitions and a place, i.e. a subnet. The token in place *P* has to be consumed by each transition in *Change* to guarantee one single firing, otherwise the value of *X.colour* is enough to ensure mutual exclusiveness.

There are however a lot of alternatives for the modelling of the change control. By the communication ability of

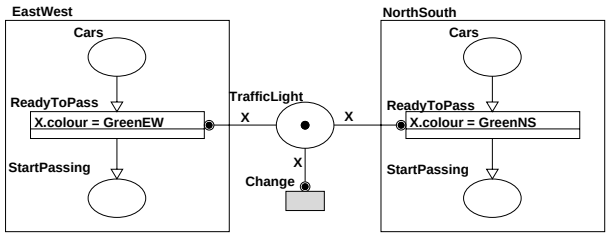

Figure 13 Two directions controlled by one token

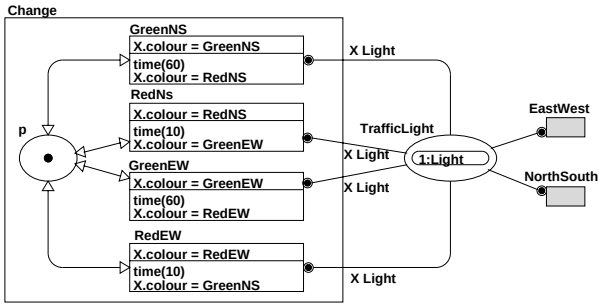

Figure 14 Graphical modelling of the change mechanism for traffic lights

the OO tokens the control can be somewhere else in the net or in another net. It can be modelled as an external event, described above. It can also be programmed non-graphically, by the built in program language of the OO tokens. It is up to the programmer to choose if the control should be shown in the graph or hidden. That corresponds to the intention of good graphical programming. The graph should describe our view of the model. It should not be overloaded with low level details.

There is however another interesting aspect of this use of sensor+ arcs, which goes beyond an ordinary simplification of the net model, namely a communication channel between two or more transitions. The token, which is never consumed, can act as a common data base for the transitions. OO tokens can have complex attributes to record a lot of data from the transitions, such as times of firing, time of last firing etc. Transitions can send messages to each other.

CONCLUSION

We have shown that testing without consuming might be a useful property of the net. The usefulness is not restricted to the negative case and zero testing, i.e. traditional inhibitor arcs. Its main advantages occurs when more than one action (transition) is controlled by one condition and mutual exclusiveness is assumed.

Sensor arcs are multi-useful and therefore adopting one of the main principles of PNs. In GPSS for example, modelling is performed with a lot (>60) of different blocks. 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.

Sensor arcs are in the first case a tool for simulation efficiency and model simplification and not an analysis extension. We have seen that sensor arcs alone have no influence on analysis. There has been no attempt to look at the effect on analysis from combinations with attribute testing. This does however not only concern sensor arcs. Also conventional arcs can be combined with inscriptions. So this problem is on the whole the question of the use of standard and user defined attributes in combination with inscriptions for controlling the net. This analysis is out of the scope of this paper.

In the next generation of our SN tool, XSimNet [GustavsonTörn 1994a], our intention is to replace the inhibitor arcs with sensor arcs. We will then use the graphical symbols and arc notation presented above.

REFERENCES

[DavidAlla 1994] David R. and H. Alla. 1994 *Petri Nets for Modeling of Dynamic Systems - A Survey.* Automatica Vol. 30, No 2, 175-202.

[Bandinelli et. al. 1993] Bandinelli S, A. Fuggetta and C. Ghezzi. *Software Process Model Evolution in the SPADE Environment.* IEEE Trans. Software Engineering (Dec.): 1128 -1144.1993.

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

[GustavsonTörn 1994b] Gustavson Å. and A. Törn. *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.

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

[Peterson 1977] Peterson, J. 1977 *Petri Nets*, ACM Computing Surveys 9, No 3 1977, 223-252..

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

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

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

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