CODEN: LUTFD2/(TFRT-5452)/1–79/(1991)

Implementation of Petri-Net and Grafcet Primitives in Omola and Modelling of Markov-Processes

Håkan Nilsson

Department of Automatic Control
Lund Institute of Technology
August 1991

Department of Automatic Control
Lund Institute of Technology
P.O. Box 118
S-221 00 Lund Sweden

Document name
MASTER THESIS

Date of issue
August 1991

Document Number
CODEN: LUTFD2/(TFRT-5452)/1-79/(1991)

Author(s)
Håkan Nilsson

Supervisor
Mats Andersson and Rolf Johansson, LTH

Sponsoring organisation

Title and subtitle
Implementation of Petri-Net and Grafcet Primitives in Omola and Modelling of Markov-Processes.

Abstract

The automation of complex sequential systems calls for modelling and simulation tools which are able to describe the reality in correct way and at the same time give a structural overview to facilitate analysis. The thesis discusses the basic requirements on a modelling and simulation tool for Petri-Nets and Grafcet and it presents the fundamental concepts for the design in the object-oriented modelling language Omola. The introduction of stochastic properties is discussed and constraints are made on the structure of Petri-nets to model Markov-processes. A tool for the analysis of Petri-nets modelling Markov-processes which is based on probability theory is presented.

Finally, as an example, the simulation of a Petri-net representation of a Kanban production system is presented together withan analysis of the same system using probability theory.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
English

Number of pages
79

Recipient's notes

Security classification

Implementation of Petri-Net and Grafcet Primitives in Omola and Modelling of Markov-Processes

Master of Science Thesis
by
Håkan Nilsson

August 29 1991

Lund Institute of Technology
Department of Automatic Control

Acknowledgements

I would like to thank my advisors Mats Andersson and Rolf Johansson who have been a most valuable help during my work on this thesis.

I would also like to thank everybody at the department of Automatic Control for helping me with practical matters.

Lund, August 1991

Håkan Nilsson

Table of Contents

1.Introduction 1

2. Implementation of Petri-Nets and Grafcet in Omola 2

2.1 Differencies between Petri-Nets and Grafcet 2

2.2 Ideas behind the Modelling Tool 3

2.3 Eventhandling Facilities in Omola 4
2.3.1 Conditional Events 4
2.3.2 Event Actions 5
2.3.3 Scheduled Events 5
2.3.4 Simulating Algorithms 5

2.4 Implementation 6
2.4.1 Basic Objects 6
2.4.2 Conflict Resolution 10
2.4.3 Various Objects 12
2.4.4 Simple I/O 12

3. Discrete Markov-Processes in Continuous Time 15

3.1 Stochastic Petri-Nets 15

3.2 Implementation in Omola 15

3.3 Analysis of Markov-Processes Modelled by Petri-Nets 17
3.3.1 Basic Properties from Probability Theory 17
3.3.2 Analysis Using Probability Theory 19
3.3.3 Monte Carlo Simulations 26

4. Modelling Example: Kanban Production System 29

4.1 Petri-Net Modelling 29

4.2 Performance Analysis 31

5. Conclusions 34

5.1 Simulation 34

5.2 Markovian State Space Generation 34

5.3 Performance Analysis 35

6. References 36

A. Petri-Net Primitives 37

B. Functions Generating an Intensity Matrix 51

C. Statistical functions 64

D. Production System Primitives 70

1. Introduction

When a machine or a process is to be automated it is of great importance to communicate detailed information about the machine/process to the designer of the control system. This is normally done in form of a written text which is to be interpreted together with a drawing. As the tasks become more complex it is often more difficult to give a clear and distinct textual description of a problem. A simple example is a manufacturing cell where robots, mills, lathes and conveyers are to cooperate adequately to complete a certain task.

Today's development towards automation of more complex systems calls for modelling languages which are able to describe the reality in a correct way and at the same time give a structured overview to facilitate analysis. A graphical modelling tool has been proposed in the international norm IEC848 which has been approved and is available in various European languages including English and French [6]. This norm is based on the principle of Grafcet which has been developed by the French organization AFCET (Association Francaise pour la Cybernetique Economique et Technique). Grafcet is a graphical method based on the theory of Petri nets (PN), [5], [11], [12] and finite state automata [12] with elements such as steps, transitions, actions and conditions to descibe processes. The function of sequential control system can be defined in Grafcet in a clearcut way with sequences of low complexity. However, the methods of analysis provided by PN theory have not been exploited in the control system design. The PN theory has mainly been developed in the field of theoretical computer science and until now there has been less attention in the field of control theory. A contributory cause is probably that the PN theory has been developed within the European Community and much less in the English-speaking world.

A PN is a mathematical representation of a system and can provide important information about the system's structure and dynamics. This information can be used for evaluating the systems to suggest improvements or control action. The practical application of PN has been successful for communication protocols, programmable logic controllers (PLC), operating systems, parallel computation and software engineering.In this thesis the PN theory will be used to make a simple performance analysis of a production system.

The efficiency and development Japanese industry has shown in the recent 30 years has lead to an increasing interest in Japanese production philosophy. The Japanese companies have by reducing inventory costs and lead time reduced production costs and increased the flow of capital. Production systems can generally be divided into two different groups:

1. The Push-system

The system forecasts the demand of inventory parts or material in process at each stage of the production, considering the flow-time up to the final stage. Based on this forecast, the whole production process is controlled by justifying inventory or final products in each part of the process.

2. The Pull-system

In this system there is a certain amount of inventory at each stage. A succeeding process orders and withdraws parts from the storage of the preceding process only at the rate and at the time it has consumed items.

Many western production systems belong to the first group while many Japanese systems are members of the second group.

In this thesis PN will be used to model Pull-systems together with Markov theory to make a simple example of performance analysis. The main scope has been to test and implement a simulation tool for PN and an analysing tool for PN-representation of discrete Markov-processes in continuous time. The reader is assumed to have basic knowledge of PN-theory and to be familiar with the simulation language Omola [2] as well as the matrix handling program Matlab [10].

2. Implementation of Petri-Nets and Grafcet in Omola

2.1 Differencies between Petri-Nets and Grafcet

Before presenting the ideas behind the design of the modelling tool which has been implemented some differencies between PN and Grafcet should be noticed.

The structure of a standard PN is a bipatite graph which syntax consists of objects such as places, transitions and directed arcs. The semantics is represented by tokens and simple firing rules and one of the major advantages with PN is the capability to model parallelism and synchronisation, introducing the concepts of conflict and concurrency. Many extensions have been made from the standard PN. In this thesis objects such as multiple- and inhibitor arcs will be implemented as well as stochastic properties.

Grafcet differs from PN in three major ways[1]:

1. In Grafcet the state of a step is a boolean variable, in PN it is an integer with value ≥0. This implies that a step in Grafcet can contain one token at the most.

2. A standard PN is autonomous while in Grafcet the behaviour of a net is governed by external conditions. In a standard PN a transition is fired if it is enabled (provided that there is no conflict, see further down). In Grafcet an external condition is associated with each transition leading to extended semantics. To be able to fire the transition must be enabled and the associated logical condition must also be true. There are two principle ways in which an enabled transition can respond when the condition changes its state.

   1. The transition will fire repeatedly while it is enabled and the condition is true.

   2. The transition will fire exactly once and only if it was enabled when the condition became true. The condition must become false again before the transition can fire another time.

   The result of firing transitions according to the two different principles is illustrated in fig1. The same condition e has been associated with the two transitions.

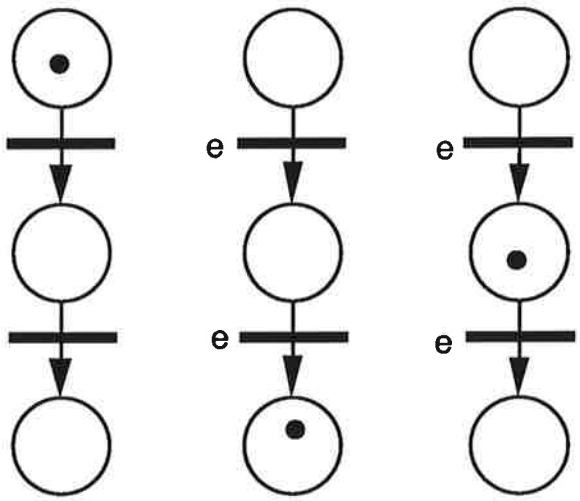


Fig1 The left marking is the initial. The result of firing transitions according to principle 1 is shown in the middle and the result of principle 2 is shown to the right. E is a common condition.

[1] The comparisons have been made with the Grafcet standard in [6]. Another standard is given in [5].

3. The situation in fig2a is an example of what is called a conflict.

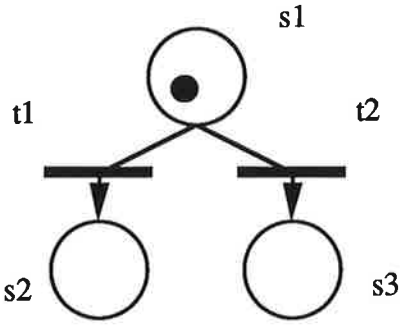


Fig2a Conflict in a PN.

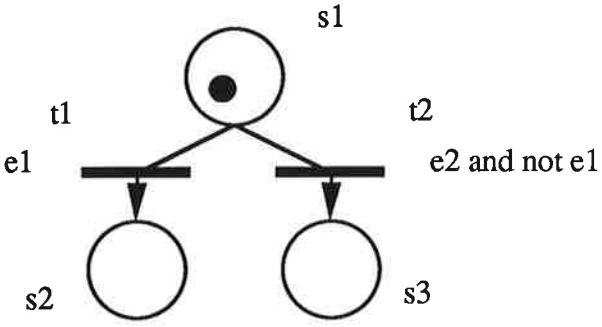


Fig2b Example of how conflict can be avoided in Grafcet

Both t1 and t2 are enabled according to the rules of standard PN but only one transition can fire at the most. Conflicts are not allowed in Grafcet and the situation can be avoided by adding constraints on the external conditions as in fig2b. Notice that the external conditions associated with t1 and t2 never are true at the same time and thus conflict is avoided.

2.2 Ideas behind the Modelling Tool

The complexity of a modelling tool is governed by the complexity of the systems that are to be modelled. The ambition with this work has been to implement a tool which is capable of handling typical PN properties such as conflicts and multiple states of a place as well as typical Grafcet issues such as external logical conditions. The final goal has been to model and simulate non-trivial systems with stochastic properties such as the example in chapter 4.

The syntax and semantics of the standard PN have served as a base. To increase modelling power the syntax has been extended with multiple arcs and conditional transitions with the same semantics as in Grafcet. Finally stochastic properties have been added. The modelling tool consists of:

1. Syntax

1. Places and Steps[2]

2. Transitions
   Three different types of non-stochastic transitions will be used:

   I. Transitions
      Autonomous transitions with no condition, that is the standard PN transition.

   To separate the behaviour of the transitions which are provided with a condition two types of conditional transitions are introduced.

   II. ConditionTransitions
      This type will fire as long as it is enabled and the condition is true.

[2] The designation is places in PN litterature and steps in Grafcet standard. In this thesis the designation steps has been used.

III. PulseTransitions
This type will fire once if and only if the transition was enabled when the condition became true. The condition must become false again before this type of transition can fire again.

3. Arcs

4. Stochastic objects

2. Semantics

1. Conflicts will be resolved for all three types of transitions by the introduction of priorities. It should be noted that this is consistent with the prohibition of conflict in Grafcet. It is left to the user to prevent conflict but should it occur the result will be repeatable.

2. The PN primitives shall be able to handle the Condition- and PulseTransitions so that Grafcet can be modelled within the PN-implementation.

If a system is to be modelled as a strictly standard PN then only Places and Transitions should be used (i. e. not Pulse or ConditionTransitions). If a system is to be modelled as a Grafcet then only Steps and Condition- or PulseTransitions should be used[3].

The rest of this thesis will be dedicated to the extended PN-model above. In chapter 2.3 the implementation of the basic primitives is described and in 3.2 the stochastic objects are presented.

2.3 Eventhandling Facilities in Omola

The implementation of the PN and Grafcet primitives has been made in Omola, an object oriented modelling language which is under development at the Department of Automatic Control in Lund. The language contains facilities for simulating discrete events in continuous time and these facilities form the base which the primitives have been built upon. A short summary of the event handling concept in Omola is presented. A thorough description of the language is given in [2] and in a special note on event handling [3].

2.3.1 Conditional Events

Definition: An event is a discontinuous change of a state (not specified) at a specific moment in simulation time.

The event occur due to a change in some logical condition or due to some other event. An event may cause one or many state variables to change value and other events to occur. There are two synthatic forms of event clauses:

1.
```
ONEVENT <logical condition> DO
    action body
END;
```

In this case the actionbody will be executed in some order exactly once every time the logical condition changes from false to true.

2.
```
WHEN <logical condition> DO
    action body
END;
```

---

[3] See 5.1

In this case the action body executes in some order repeatedly as long as the condition evaluates to true.
To faciliate event propagation, special event objects can be defined. For example:

```
E1,E2 ISAN Event;

ONEVENT <logical condition> CAUSE E1,E2;

ONEVENT E1 OR E2 DO
    action body;
END;
```

The effect of the event propagation is that all members of the propagation chain (the list of events which follow the keyword CAUSE) will be treated as one single event, that is their associated actions will be executed, sorted in a proper order.

2.3.2 Event Action

A set of actions in a `DO..END;` actionbody are usually not performed in a sequence but are sorted and executed in an appropriate order depending on the variables used. In order to define relations between the value of state variables before and after an event, the variable operator `NEW(x)` is introduced. `NEW(x)` in an action body refers to the value of x immediately after the event. The normal use of this operator is:

```
NEW(x) := expression;
```

All assignments associated with an event will be sorted such that an assignment to a particular variable will be executed before other assignments referring to `NEW(x)` in the right hand expression.

2.3.3 Scheduled Events

If the exact time when an event will occur is known the event could be scheduled,i. e., it can be put in a queue of time ordered events. A special procedure call `SCHEDULE(E,delay)`,where E is an event, may be used in event action bodies. The effect is that the event E will be scheduled to occur in delay time units from now.

2.3.4 Simulating Algorithms

Omola contains a simulator which handles models that are a combination of continuous elements, events that are excited by continuous and discrete variables and scheduled events. The simulation is handled by the algorithms below which partly have been developed to suit PN-simulation. Especially the CEF-algorithm is of importance when conflicts in a PN is resolved. A further explanation of the conflict resolution is given in 2.4.2.

Conditional Events Firing Algorithm (CEF-algorithm)

There are two types of conditional events:

1. ONEVENT <logical condition> DO
2. WHEN <logical condition> DO

In the following algorithm an event is said to be fireable if it is a type 2 event and the condition evaluates to true, or it is a type 1 event and the condition evaluates to true and the last evaluation was false.

1. Evaluate the condition for every conditional event.
2. Make a list L of all fireable events. Let the ordering of L be determined by the order in which the events are defined.
3. If L is empty then break.
4. Fire the first event in L.
5. For e being the second to the last event in L:
   a. Evaluate the condition of e.
   b. If e is still fireable then fire it.
6. Save the last evaluated condition state of every event.
7. Go back to 1.

Scheduled Events Firing Algorithm (SEF-algorithm)

Scheduled events are events where the firing time is determined in advance. These events are saved in an event queue until the predetermined firing time is reached. The queue is ordered in firing time order with the smallest time first. If more than one event is scheduled for the same time they are put in a last-in-first-out order.

1. Let t_c be the current simulation time.
2. Let t_s be the time of the first event in the event queue.
3. If $t_s<t_c$ then break.
4. Remove the first event from the event queue and fire it.
5. Goto step 2.

The two algorithms above interact with the

Main Simulating Algorithm

1. Set current simulation time to the user defined simulation start time.
2. Run the SEF-algorithm.
3. Run the CEF-algorithm.
4. If any events was fired in the previous step then
   a. Run the SEF-algorithm.
   b. If any event was fired go to step 3.
5. Run the continuous simulation integration routine until the user defined stop time, or to the time of the next scheduled event , or until some continuous time event condition becomes true, whichever comes first.
6. If the user defined stop time is reached then stop, else goto step2.

2.4 Implementation

In this paragraph the implementation of the objects 1-3 in 2.2 is described together with primitives for simple I/O-communication. The implementation of the stochastic objects is descibed in 3.2. Appendix A contains the final Omola code.

2.4.1 Basic Objects

The arcs are represented as Omola terminals on the step- and transition objects. There are two pairs of Step-Transition terminals:

```
StepLTerminal ISA DiscIntTerminal WITH
    State ISA DiscIntTerminal;
    Trigg ISAN EventInput;
    Weight ISA DiscIntTerminal;
END;
```

```
TransitionUTerminal ISA RecordTerminal WITH
    State ISA DiscIntTerminal;
    Trigg ISAN EventOutput;
    Weight ISA DiscIntTerminal;
END;

StepUTerminal ISA RecordTerminal WITH
    Trigg ISAN EventInput;
    Weight ISA DiscIntTerminal;
END;

TransitionLTerminal ISA RecordTerminal WITH
    Trigg ISAN EventOutput;
    Weight ISA DiscIntTerminal;
END;
```

(DiscIntTerminal=Discrete Integer Terminal)

A StepLTerminal on a Step-object should be connected to a TransitionUTerminal on a Transition-object and a StepUTerminal on a Step-object should be connected to a TransitionLTerminal on a Transition-object. Since the two pairs of terminals contain different numbers of variables an error message will be given at the instantiation if the terminals are connected in an inappropriate way. A correct connection between two terminals represents an arc in a PN.

A Step and a ConditionTransition have the following principle definitions.

```
Step ISA GrafObject WITH
attribute:
    State TYPE DISCRETE Integer;
parameter:
    InitState ISA IntParameter;
terminals:
    Upper ISA StepUTerminal;
    Lower ISA StepLTerminal;
event:
    Init ISAN Event;

behaviour:
    ONEVENT Init DO
        NEW(State):=InitState;
    END;

    ONEVENT Upper.Trigg DO
        NEW(State):=State+Upper.Weight;
    END;
    Lower.State:=State;
END;


ConditionTransition ISA GrafObject WITH
parameters:
    Produce ISAN IntParameter WITH default:=1; END;
    Consume ISAN IntParameter WITH default:=1; END;
terminals:
    Upper ISA TransitionUTerminal;
    Lower ISA TransitionLTerminal;
```

```
behaviour:
     ONEVENT Init DO
          NEW(Lower.Weight):=Produce;
          NEW(Upper.Weight):=Consume;
     END;
     WHEN Condition>0.0 AND Upper.State>=Upper.Weight
     CAUSE Upper.Trigg, Lower.Trigg;
END;
```

The interaction between the objects is described in fig3 which shows a simple system consisting of two steps and one transition.

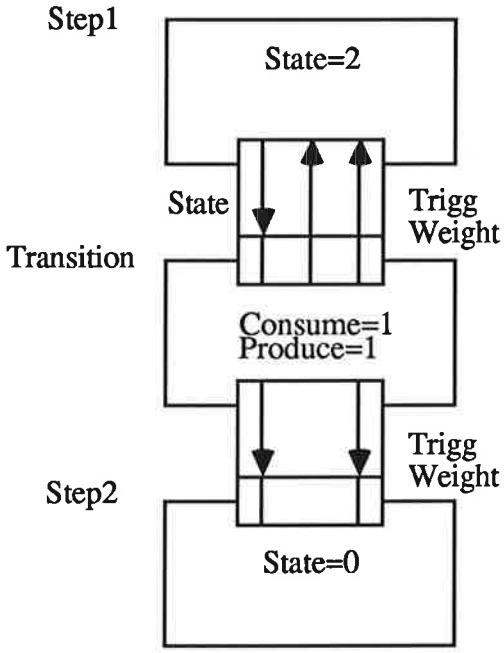


Fig3 Basic principle of implementation.

The system has the following principle declaration in Omola:

```
Example ISA Model WITH
submodels:
     Step1, Step2 ISA Step;
     Transition ISA ConditionTransition;
connections:
          Step.Lower AT Transition.Upper;
          Transition.Lower AT Step2.Upper;
END;
```

When the simulation is started the states representing the number of tokens are given their initial values. Further the weight of the ingoing (to the transition) arc is given by the value of Consume and the weight of the outgoing arc is given the value of Produce. Assuming that no scheduled events exist, the main algorithm will just integrate the simulation time. The state of Step 1 is propagated to the transition. Since 2=Upper.State>=1=Upper.Weight the transition is enabled but will not fire assuming that the condition is false. When the condition becomes true the conditional event fires as a consequence of step 1-6 in the CEF-algorithm. This event is propagated to the proceding and following steps by the events named Trigg. This in turn leads to that the state of the proceding step is decreased with one unit (token) and the state of the following step is increased with one unit. The proceding step now contains one token and is still enabled. The CEF-algorithm

will execute with an additional firing as a result (step 7-1-7) giving zero tokens in the proceding step and two in the following.

A transition followed by more than one step means the start of a parallel activity. If the weights on the arcs from the transition to the following steps are the same, no special objects are needed to represent a fork, it is sufficient to make one connection from the transition to each following step.

A number of steps followed by a transition means a synchronisation of parallel activies. This construction needs a special object which models the case where the arcs between the steps and the transition have the same weight. The basic principle for synchronisation is:

```
Sync_2 ISA GrafObject WITH
terminals:
    Upper_1, Upper_2 ISA TransitionUTerminal;
    Lower ISA StepLTerminal;

behaviour:
    Lower.State:=min(Upper_1.State,Upper2.State);
    Upper_1.Weight:=Lower.Weight;
    Upper_2.Weight:=Lower.Weight;
    ONEVENT Lower.Trigg CAUSE Upper_1.Trigg, Upper_2.Trigg;
END;
```

To keep the graphical symmetry a dummy fork object has been implemented.

In extended PN multiple arcs with different weights are commonly used. It is very easy to extend the objects above to handle to these and here only a simple example is shown. When this thesis is written Omola has no array handling facilities which leads to tedious repetitive code. In the final implementation each object has three upper terminals and three lower terminals making it possible to connect six arcs with different weights at the most. To each of these connections of a Transition-object a Sync_3 or a Forc_3 object can be connected so that each transition can be connected to 18 other objects at the most.

```
ConditionTransition ISA GrafObject WITH
parameters:
    Produce1 ISAN IntParameter WITH default:=1; END;
    Produce2 ISAN IntParameter WITH default:=1; END;
    Consume1 ISAN IntParameter WITH default:=1; END;
    Consume2 ISAN IntParameter WITH default:=1; END;
terminals:
    Upper1, Upper2 ISA TransitionUTerminal;
    Lower1, Lower2 ISA TransitionLTerminal;

behaviour:
    ONEVENT Init DO
        NEW(Lower1.Weight):=Produce1;
        NEW(Lower2.Weight):=Produce2;
        NEW(Upper1.Weight):=Consume1;
        NEW(Upper2.Weight):=Consume2;
    END;
    WHEN Condition>0.0 AND Upper1.State>=Upper1.Weight AND
    Upper2.State>=Upper2.Weight
    CAUSE Upper1.Trigg, Upper2.Trigg, Lower1.Trigg, Lower2.Trigg;
END;
```

2.4.2 Conflict Resolution

The situation in fig4 is an example of a conflict.

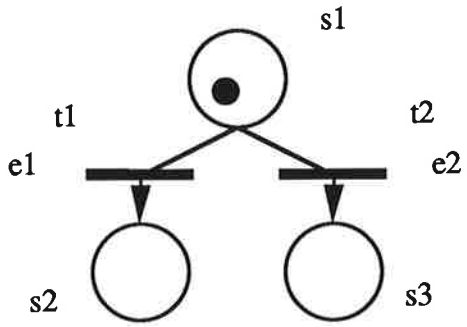


Fig4 Conflict involving two ConditionTransitions

T1 and t2 are assumed to be ConditionTransitions which are affected by the same condition e. Both t1 and t2 are enabled but when e becomes true only one transition can fire at the most. The implementation resolves this by letting the transition which has been declared first in the code fire. As e becomes true the conditional events in t1 and t2 become fireable. The CEF-algorithm makes a list of the conditional events in the same order as t1 and t2 have been declared and fires the first event in the event queue thus firing the corresponding transition. The algorithm re-evaluates the condition of the second conditional event but since the token has been removed from step s1 as the first transition was fired the condition will evaluate to false and the algorithm will break.

If the structure which can cause conflict is preceded by a transition like in fig5 this transition must be declared after the structure, which can be understood by the following example.

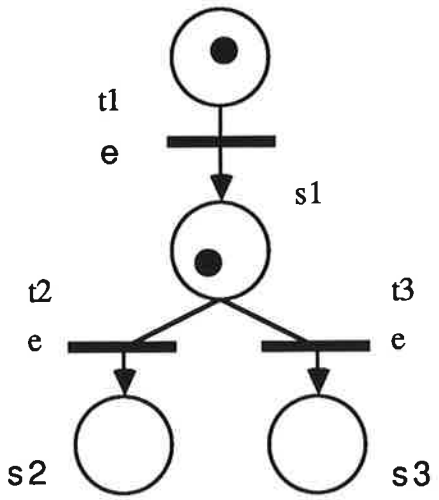


Fig5 A situation which requires a declaration rule.

Assume that the ConditionTransitions are declared in the order t1, t2, t3 and that the net has the marking in the figure. The net contains a conflict and the CEF-algorithm will make a list of the conditional events in the transition sorted in the order L:Et_1, E_{t2}, E_{t3}. As the algorithm continues with step 5a and b transition t1 will fire adding a token in step s1. Now the CEF-algorithm will allow both t2 and t3 to fire since both of the conditional events of these transitions will be fireable.This leads to an incorrect marking with one token in s2 and one in s3. If the transitions are declared in the order t2, t3, t1, then t2 will fire removing the token from s1. Now the conditional

event of t3 will not be fireable and t3 will not fire. The conditional event of t1 is still fireable and t1 will fire. The CEF-algorithm will now start from step1 again, firing t2 another time and giving the correct marking of two tokens in s2.

Thus far only the ConditionTransition has been dealt with. The original PN transition is defined by:

```
Transition ISA ConditionTransition WITH
    Condition:=1.0;
END;
```

The principle definition of PulseTransition is given by:

```
PulseTransition ISA GrafObject WITH
parameters:
    Produce ISA IntParameter WITH default :=1; END;
    Consume ISA IntParameter WITH default :=1; END;
terminals:
    Upper ISA TransitionUTerminal;
    Lower ISA TransitionLTerminal;

variable:
    Fireable TYPE DISCRETE Integer;

behaviour:
    ONEVENT Condition>0.0 DO
        NEW(Fireable):=Upper.State>Upper.Weight;
    END;

    ONEVENT Fireable AND Upper.State>=Upper.Weight
    CAUSE Upper.Trigg,Lower.Trigg;

    ONEVENT Fireable>0 DO NEW(Fireable):=0; END;
END;
```

The construction

```
ONEVENT Condition>0.0 AND Upper.State>=Upper.Weight
CAUSE Upper.Trigg,Lower.Trigg;
```

is inappropriate.The only time when the PulseTransition can be fired is when the condition goes from false to true. The construction above will fire if the condition is greater then zero and Upper.State>=Upper.Weight goes from false to true and this explains the solution in the code above using two conditional events.

It is easy to see that the CEF-algorithm will be able to handle conflict between PulseTransitions in the same way as in the case of ConditionTransitions. The declaration rule presented in connection with fig4 above is also valid for PulseTransitions. In this case the final marking will be one token in s1 and one in s2.The case of a mix of Pulse- and ConditionTransitions in conflict, however, needs some further consideration. PulseTransitions with their two conditional events construction are fired by running the CEF-algorithm twice. (first a list of PulseTransitions with condition>0 is considered. The second time a list of PulseTransitions with Fireable>0 and Upper.State>= Upper.Weight are considered leading to the actual firing of the transitions.) To obtain the right marking the ConditionTransitions (and Transitions) must execute in the same phase as the PulseTransitions which lead to the following principle modification:

```
ConditionTransition ISA GrafObject WITH
...
    WHEN Condition>0.0 AND Upper.State>=Upper.Weight DO
        NEW(Fireable):=Upper.State>=Upper.Weight;
    END;
    ONEVENT Fireable>0 AND Upper.State>=Upper.Weight CAUSE
    Upper.Trigg, Lower.Trigg;
...
END;
```

2.4.3 Various Objects

Here three objects are presented which increase the modelling power.

When a step is to be connected to an ingoing and one outgoing arc of the same transition the SingleStep should be used. In fig6 a typical application of this object is shown.

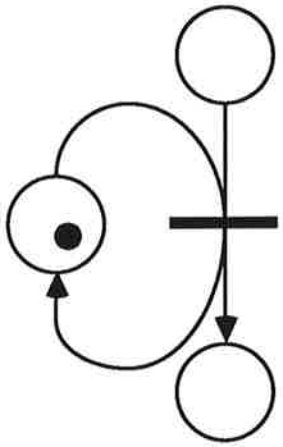


Fig6 Example of how the SingleStep should be used

In extended PN inhibition arcs are sometimes used extending the enabling rule of the following transition:

A transition is enabled if and only if it is enabled with respect to its normal ingoing arcs and if the step(s) that are connected with the transition with an inhibition arc do not contain any tokens at all.

The inhibition arc radically increase the modelling power but makes analysis impossible since the net becomes a Turing machine.

InitStep is a step which has an additional event InsertNew. Every time the InsertNew is activated a token is added to the state of the step.

2.4.4 Simple I/O

Omola facilitates saving and plotting trajectories of arbitrary variables without special arrangements in a model. However, when simulating PN-models it is convenient to make printouts only when the state of a net is changed. This can be done by adding the Omola command PRINTLOG() to the model. This command prints the value of a variable or an array of characters on the special ASCII-file printlog.t. To keep the modularity of the implementation a number of printer objects have been implemented.The printer communication between the step objects and the printer object is made through separate terminals:

```
PrinterInTerminal ISA RecordTerminal WITH
    State ISA DiscIntTerminal;
    Print ISAN EventInput;
END;

PrinterOutTerminal ISA RecordTerminal WITH
    State ISA DiscIntTerminal;
    Print ISAN EventOutput;
END;

PrinterInConn ISA RecordTerminal WITH
    In1, In2 ISA PrinterInTerminal;
END;

PrinterOutConn ISA RecordTerminal WITH
    Out1, Out2 ISA PrinterOutTerminal;
END;
```

Each step has the follwing communication code:

```
terminal:
    Printer ISA PrinterOutTerminal;

behaviour:
    ONEVENT Lower.Trigg OR Upper.Trigg CAUSE Printer.Print;
```

To document the marking of a PN, two different approaches have been used. The first is to log the state of the net every time that a state of a step in the net has been changed. This is done by printing the state of each step in the net after the change, together with the simulation time and the time which has past since the last alteration. In standard PN there is no concept of time and the last two entities will be zero. Insted the time is used in the stochastic models in chapter3. Principle:

```
Printer1 ISA Model WITH
terminal:
    Comm ISA PrinterInConn;
event:
    Init ISAN Event;
variable:
    x TYPE DISCRETE Real;

behaviour:
    ONEVENT Init DO
        PRINTLOG("NaN"), PRINTLOG("NaN"),..., PRINTLOG("\n");
    END;

    ONEVENT Comm.In1.Print or Comm.In2.Print DO
        PRINTLOG(Base::Time),PRINTLOG(" "),PRINTLOG(Base::Time-x),
        PRINTLOG(NEW(Comm.In1.State)), PRINTLOG(" "),
        PRINTLOG(NEW(Comm.In2.State)), PRINTLOG("\n");
        NEW(x):=Base::Time;
    END;
END;
```

PRINTLOG(" ") results in a blank, PRINTLOG("\n") in the sign for end of line. The initial NaN:s are used to separate different simulations that are logged on the same file. Matlab interprets NaN as "not a number" and this is used in the analysis of the Monte Carlo simulations in chapter 4.

The other approach is to log the state exactly once every time step 5 in the CEF-algorithm in 2.3.4 has been executed. This will give the development of the marking which is normally associated with a PN [5],[12] and this is the printer type which has been used in the simulations in chapter 4.

```
Printer2 ISA Model WITH
terminal:
    Comm ISA PrinterInConn;
event:
    Init ISAN Event;
variable:
    x TYPE DISCRETE Real;
    Print TYPE DISCRETE Integer;

behaviour:
    ONEVENT Init DO
        PRINTLOG("NaN"), PRINTLOG("NaN"),..., PRINTLOG("\n");
    END;

    ONEVENT Comm.In1.Print or Comm.In2.Print DO
        NEW(Print):=1;
    END;

    ONEVENT Print DO
        PRINTLOG(Base::Time),PRINTLOG(" "),PRINTLOG(Base::Time-x),
        PRINTLOG(NEW(Comm.In1.State)), PRINTLOG(" "),
        PRINTLOG(NEW(Comm.In2.State)), PRINTLOG("\n");
        NEW(x):=Base::Time;
    END;

    ONEVENT Print>A0 DO NEW(Print):=0; END;
END;
```

It should be noted that a printer must be declared before the objects which state it is going to log regardless of printertype used.

3. Discrete Markov-Processes in Continuous Time

3.1 Stochastic Petri-Nets

Standard PN do not include any time concept. The introduction of time into standard PN can be done in at least two separate ways [1]. A first possibility is to associate with each transition a non-negative real value that indicates the delay time between the enabling and the firing of the transition. Such a transition is called a delayed transition. A second possibility of introducing the concept of time is to let a token which reaches a step become available only after a certain time. Thus tokens in a step can be divided into two groups: available and not available tokens, and only available tokens can enable a transition and be fired. Steps with the these properties are called delayed steps. The differences between these two ways to introduce time should be noted. In the first case the token is not a reserved resource and can be "stolen" by a transition with a shorter delaying time. In the second case the token is always a reserved resource.

The so called timed PN (TPN) above can be extended further by letting the delaying time be a non-negative stochasic variable and thus a stochastic PN is obtained. This thesis will deal with generalised stochastic PN (GSPN) which contain a mix of delayed and undelayed objects.

3.2 Implementation in Omola

The undelayed objects are the same as in chapter 2. The delayed objects consist of a delayed step which is connected to different random generators. The code from the final implementation is found in Appendix A. The following terminals are used:

```
RandomCallTerminal ISA RecordTerminal WITH
    CallForAdmission ISAN EventOutput;
    GoAhead ISAN EventInput;
END;

RandomAnswerTerminal ISA RecordTerminal WITH
    CallForAdmission ISAN EventInput;
    GoAhead ISAN EventOutput;
END;
```

Principle definition of a random generator:

```
ExpGen ISA Model WITH
event:
    Change ISAN Event;
parameter:
    Lambda ISA Parameter WITH default :=1.0 END;
terminal:
    Comm ISA RandomAnswerTerminal;

behaviour:
    ONEVENT Comm.CallForAdmission DO
        SCHEDULE(Change,ln(1-(1-rect(1.0)))/Lambda);
    END;

    ONEVENT Change CAUSE Comm.GoAhead;
END;
```

Definition of a delayed step:

```
DelayedStep ISA GrafObject WITH
variables:
    State, AvailState TYPE DISCRETE Integer;
terminals:
    Upper ISA SteUTerminal;
    Lower ISA StepLTerminal
    Comm ISA RandomCallTerminal;

behaviour:
    ONEVENT Upper.Trigg CAUSE Comm.CallForAdmission;

    ONEVENT Upper.Trigg DO
        NEW(State):=State+Upper.Weight;
    END;

    ONEVENT Comm.GoAhead DO
        NEW(AvailState):=AvailState+Upper.Weight;
    END;

    ONEVENT Lower.Trigg DO
        NEW(State):=State-Lower.Weight;
        NEW(AvailState):=AvailState-Lower.Weight;
    END;
    Lower.State:=AvailState;
END;
```

When a transition that has an outgoing arc connected to a DelayedStep fires, the number of tokens in the step is increased and a signal is sent to the random generator. This schedules an event Change at a time which is exponentially distributed. When Change occurs the tokens are made available to the following transitions. It should be noted that the scheduled event is started for a group of tokens that are made available at the same time when Change occur. The size of the group is equal to the weight on the input arc. To be able to initialise the SPN with groups of tokens of different sizes scheduled at different times an InitDelayedStep has been implemented and will give the right log of the initial marking. A SingleDelayedStep has also been implemented.

A delayed transition can be defined in a similar way, scheduling events each time (Upper.State) div(Upper.Weight) increases. Div denotes the integer part of Upper.State divided by Upper.Weight and the number of scheduled events should be equal to the increase in the div expression. The use of the div operator is motivated by the following example. Assume that a delayed transition has one ingoing arc with weight two and assume that a group of three tokens appears at the preceding step. A schedule for two of the tokens are made as the div expression above increases with one unit, but one token will be left over without a schedule. Now assume that somewhat later a single token arrives at the preceding step. The div expression is now increased and the new token together with the token which was left over from the first group of three lead to a schedule.

The construction has at least one major drawback as can be seen by the following example. Assume that a delayed Transition is enabled and that the random generator schedules a delayed event. Then assume that the div expression above increases again leading to another scheduled event. If the tokens which caused the first schedule are removed by the firing of another delayed transition, the scheduled event caused by these tokens will still be in the queue of the SEF-algorithm. This can lead to that the second group of tokens are fired in advance thus disturbing the probability distribution function of the transition. This thesis focuses on GSPN where delayed steps are the only stochastic objects.

3.3 Analysis of Markov-Processes Modelled by Petri-Nets

Models of a special stochastic process: the discrete Markov-process in continuous time, have been focused upon. The performance will be analysed by studying the asymptotical probability distribution and the time dependent probability distribution function, provided that these exist. The anaysis is made in two different ways using probability theory on one hand (3.3.2) and Monte Carlo simulations and statistics on the other (3.3.3).

3.3.1 Basic Properties from Probability Theory

A Markov-process {X(t)} satisfies the well-known Markov-condition:

$P[X(t_n)=x_n \mid X(t_1)=x_1, X(t_2)=x_2,....X(t_{n-1})=x_{n-1}]=P[X(t_n)=x_n \mid X(t_{n-1})=x_{n-1}]$

for all $n\geq 3$ and all $t_1<t_2<...<t_n$.

The property can be intuitively explained by saying that the future evolution of the process from the instant t_{n-1} is indepemdent of the past history of the process, i. e., the state $X(t_{n-1})$ contains all relevant information about the past process history. A discrete Markov-process in continuous time is defined as a stochastic process with the possible states E_1, E_2, ..., E_n which satisfies the Markov-condition above and is constituted so that:

$P[E_i \rightarrow E_j \text{ in } (t, t+h)]=a_{ij}h+o(h) \qquad i\neq j$

Five results from probability theory have consequences for which PN that can be regarded as Markov-processes. The proofs of statements below is found in [4].

1. The time interval between two consecutive events in a Markov-process is exponentially distributed.

   Implication: The random generators connected to the DelayedSteps should have exponential distribution.

2. The smallest value of n exponentially distributed independent stochastic variables with intensity λ_i is exponentially distributed with intensity $\lambda=\Sigma\lambda_i$.

   Implication: The transition intensity from one marking to another will be marking dependent. An example is seen in fig7a.

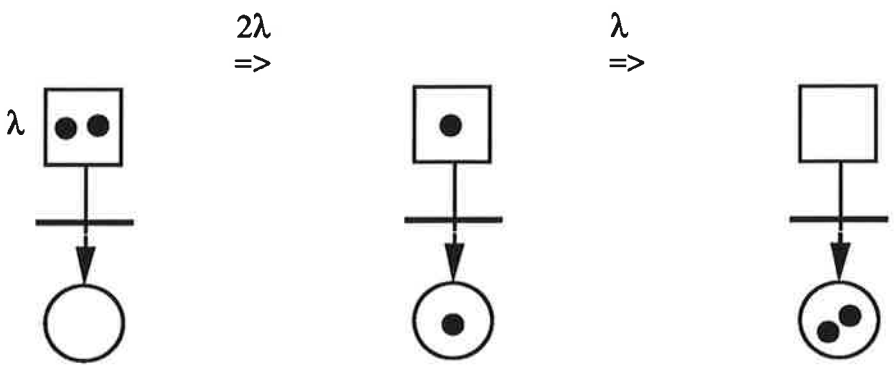


Fig7a A square has been used to denote a dealyed step.

Note: A more precise formulation is that the transition intensity is dependent on how many groups of tokens with the same size that are in a delayed step. In fig7b there are two groups of tokens with different sizes (one respectively two tokens). Since the groups

are of different size, the transition intensity is the same as the intensity associated with the delayed step.

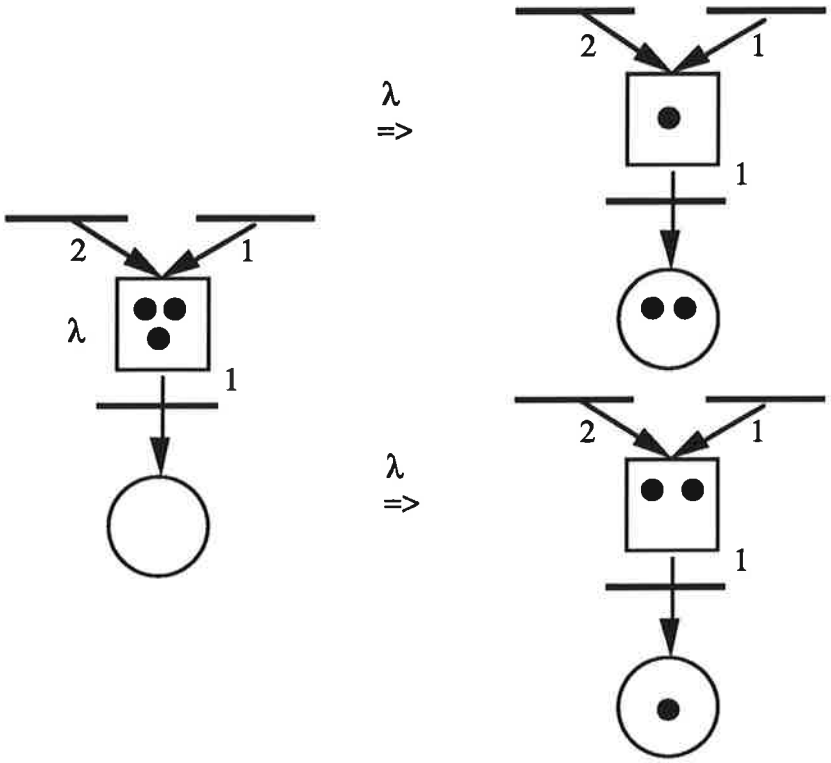


Fig7b The transition intensity is dependent on how many groups of same size that are in a delayed step.

3.The largest value of n exponentially distributed independent stochastic variables is a stochastic variable that is not exponentially distributed and a process which is generated by this stochastic variable is not a Markov-process.

Implication: Concurrency involving more than one delyed step does not model a Markov-process and the structure in fig 8 is not allowed.

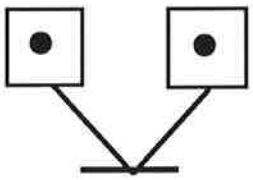


Fig8 This structure does not model a Markov-process.

4. The recurrent lifetime of a Markovian life-death process with failure rate λ is exponentially distributed with intensity λ.

Implication: Concurrency involving one delayed step and one or more undelayed steps represent a Markov-process. An example of such a process is given in fig9.

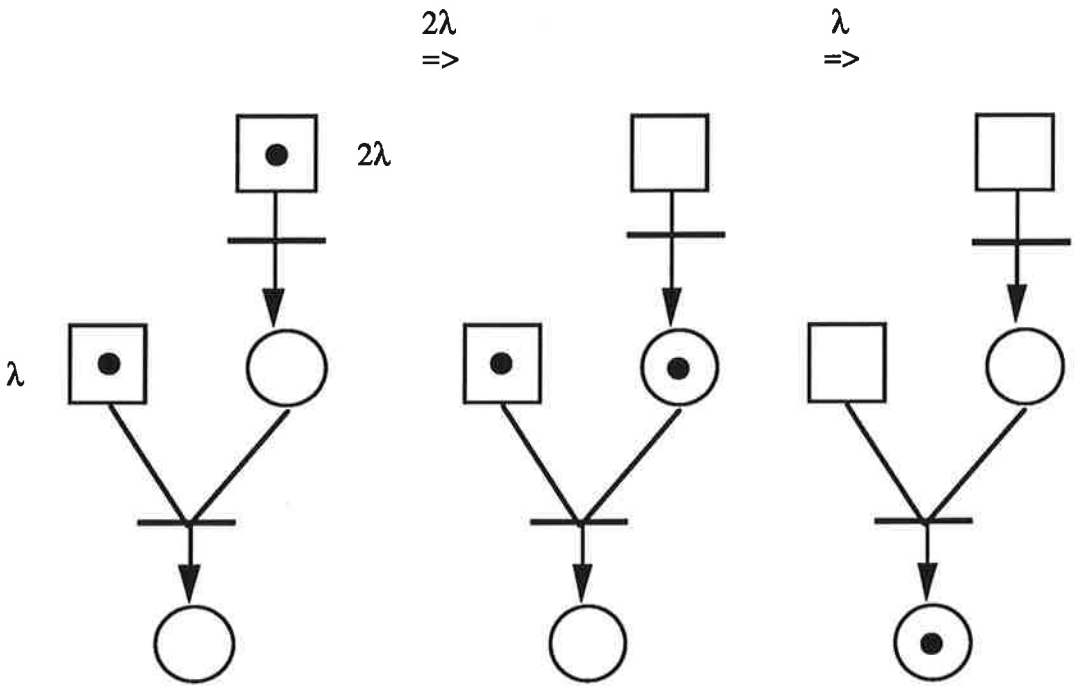


Fig9 Concurrency involving one delayed and one undelayed step.

Below an unstable state is defined as a marking which results from running the CEF-algorithm in 2.2.4 once (step 1-7) if the following execution of the algorithm do not break at step 3. An unstable marking has no duration in simulation time. A stable state is defined as a marking that is not an unstable state. A stable state has duration in simulation time.

3.3.2 Analysis Using Probability Theory

To be able to calculate the probability properties above the intensity matrix of a Markov-process must first be calculated. In order to do this the stable states and the transition probabilities between these states of the PN must first be decided.

Stochastic Petri nets

If the PN consists only of delayed steps the finding of stable states is equivalent to solving the reachability problem of a standard PN. This has been done in Matlab using a matrix representation of PN [12] and the code from the final implementation is presented in Appendix B. The following example deals with the net in fig10.

Example:

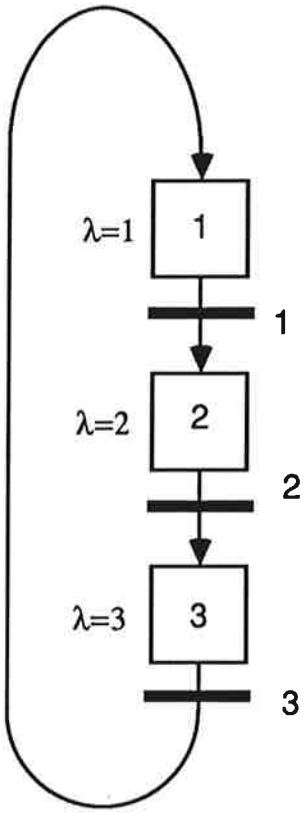


Fig10 A simple SPN.

Associate an integer with each step and let the marking M of the net be represented by a row vector where the integer at the i:th index corresponds to the number of tokens in the step with number i.

If an integer is associated with each transition a matrix consisting of the weights on the ingoing arcs (to the transition) can be written

$$D^- = \begin{array}{c} \begin{matrix} p1 & p2 & p3 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{array} \begin{matrix} t1 \\ t2 \\ t3 \end{matrix}$$

A corresponding matrix consisting of the weights on the outgoing arcs can be written.

$$D^+ = \begin{array}{c} \begin{matrix} p1 & p2 & p3 \end{matrix} \\ \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \end{array} \begin{matrix} t1 \\ t2 \\ t3 \end{matrix}$$

A transition t_j is enabled if and only if

$M_i >= (x_j * D^-)$ for all i=1...NoOfSteps

where

$x_j=(0,0,...1,...,0)$ (1 at index j)

When transition t_j fires the new marking is given by

$M=M_0+x_j*(D^+-D^-)$

Using the matrix representation a recursive algorithm has been implemented to solve the reachability problem. When a new state is found it is put in a matrix containing the different states that have been found and a matrix containing the state transition probabilities is updated.

```
function [IntensMatr,StateMatr]=generate(Marking,OldStateMatr,OldIntensMatr)

for k:=1 to NoOfTransitions
    if transition k can be fired with the present Marking then
        fire transition k giving NewMarking
        if NewMarking is in OldStateMatr then
            update OldIntensMatr giving IntensMatr
        else
            add NewMarking to OldStateMatr giving StateMatr and update OldIntensMatr
            giving IntensMatr
            [Intensmatr,StateMatr]=generate(NewMarking, StateMatr,IntensMatr)
    end
end
```

When checking if a transition can be fired possible conflicts must be resolved with the same result as in the Omola CEF-algorithm. The check and fire algorithm could be extended to:

```
if the transition k is enabled then
    if there are no transitions with higher priority than transition k that are enabled then
        while transition k is enabled
            fire transition k
            if there are transitions with lower priority that are enabled then
                fire these transitions in order of priority
            end
        end
    else
        break
    end
else
    break
end
```

Notice that the firing of transitions with lower priority does not lead to new stable states. These firings only occur when a group consisiting of more than one token has become available in a delayed step. In this case the stable state occur when all of these tokens have been fired.

The present Matlab implementation contains no checks if a PN is bounded, but this could easily be added using an algorithm in [12]. The implementation can only handle steps which contain groups of tokens of the same size. This implies that all ingoing arcs to a step must have the same weights and the initial state of a delayed step must be a multiple of this weight. The implementation consists of three functions of which one is recursive The user function has the heading:

```
function [IntensMatr, StateMatr, Bounded, Ok]=pdsmarkint(Dplus,
Dminus, Intensity, InitMarking, NoOfGroups, PriolList);
```

Pdsmarkint=pure delayed steps markov intensity

IntensMatr: The NoOfStates*NoOfStates matrix which contains the intensity matrix for the modelled Markov-process.

StateMatr: The NoOfStates*NoOfSteps matrix which contains the stable states of the Markov-process.

Bounded: The number of recursive calls are limited. If the limit is reached the algorithm is interrupted and Bounded is set to 0.

Ok: This variable is 1 if all ingoing arcs to transitions have the weight one else 0. This is a stronger version of condition 3 in 3.3 and can be removed.

Dplus, Dminus: The NoOfTransitions*NoOfSteps matrices which have been presented above.

Intensity: The 1*NoOfSteps rowvector which contains the intensities λ of the diffrent delayed steps.

InitMarking: The 1*NoOfSteps row vector which contains the initial marking of the net.

NoOfGroups: The 1*NoOfSteps row vector which contains the number of groups that the tokens in a step can be divided into.

In the Omola implementation of delayed steps every group of tokens has a scheduled event Change in the corresponding random generator. Here each step is only allowed to contain tokens sorted in groups with the same size. (The size can vary between different steps.)

PrioList: The 1*NoOfTransitions row vector which contains the numbers of the transitions sorted in the same way as they have been declared in the Omola implementation.

When this thesis is written the user must give all of the input information by himself. However, the information is available in the Omola representation of the net and an interface between Omola and Matlab could be written.

The user function calls on the recursive function pdsgenerate (pure delayed steps generate) which contains the recursive algorithm above. The latter calls on pdscheckfire (pure delayed steps check and fire) which contains the firing algorithm above.The result of running pdsmarkint is the intensity matrix

$$A = \begin{bmatrix} -1 & 1 & 0 \\ 0 & -2 & 2 \\ 3 & 0 & -3 \end{bmatrix}$$

and the state matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Once the intensity matrix is known it is easy to calculate the probability distributions that is wanted. It can be shown , [1], that a SPN is ergodic if the initial marking is reachable from any marking (state) in the state matrix. It follows that it is possible to calculate the asymptotical probability distribution π by solving:

$\pi A=0$

$\Sigma\pi_i=1$

where A is the transition matrix above and π is a 1*3 row vector containing the asymptotical probabilities.

Further the probability distribution function P(t) is given by

$dP/dt=PA$ $\quad\quad$ $P(t)=P_0\exp(At)$

$P(0)=P_0$ $\quad\quad <=>$ $\quad\quad \Sigma P_i=1$

$\Sigma P_i=1$

where A is the transition matrix above, P_0 is a 1*3 row vector with the initial probabilities of the states and P(t) is the 1*3 rowvector which contains the probability distribution functions of the three states.

Generalised Stochastic Petri-Nets

Thus far we have studied nets which only contain delayed steps. In a generalised stochastic PN a mixture of delayed and undelayed steps are allowed and as before the goal is to find the intensity matrix for the corresponding Markov-process. The matrix representation is used in the same way as above but the net is divided into a delayed and one undelayed part. The undelayed part of the net consists of the transitions which have no input arcs from a delayed step and the steps that are proceded and followed by such transitions. The rest of the net is the delayed part. In fig11 an example is shown where the undelayed part of the net is inside the polygon.

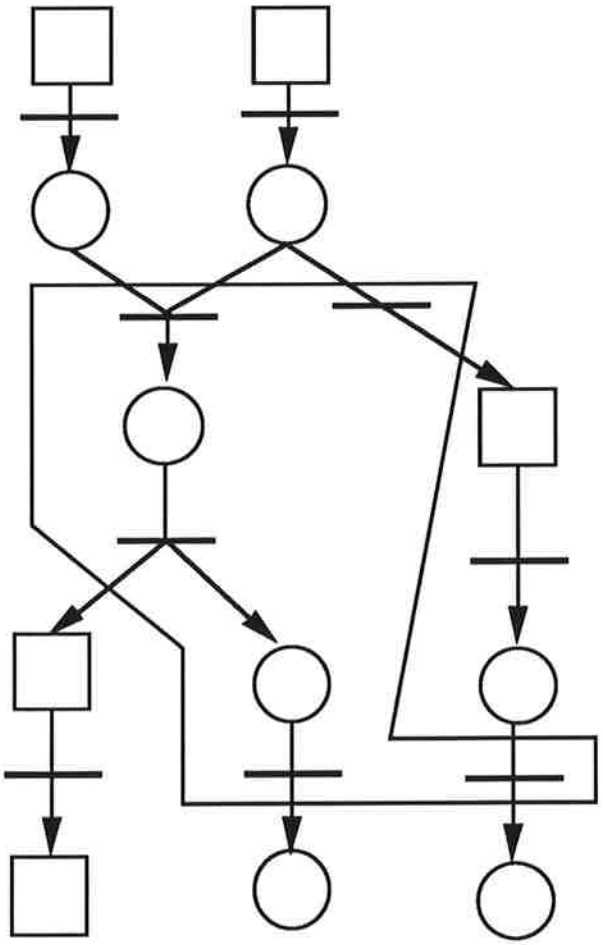


Fig11 An example of a GSPN. The transitions and steps inside the polygon is defined as the undelayed part of the net, the objects outside constitute the delayed part of the net.

The Matlab algorithms can now be extended to handle GSPN. Note that two nested recursive algorithms have been used.

```
function [IntensMatr, StabStateMatr, InstabStateMatr]=mixgenerate(Marking, OldIntensMatr,
OldStabStateMatr, OldInstabStateMatr)

for k=1 to NoOfTransitions in the delayed part of the net
    if transition k can be fired with the present Marking then
        fire transition k giving NewMarking
        [InstabStateMatr,NewMarking]=pudgenerate(NewMarking, OldInstabStateMatr)
        if NewMarking is in OldStabStateMatr then
            update OldIntensMatr giving IntensMatr
        else
            add NewMarking to OldStabStateMatr giving StabStateMatr and update
            OldIntensmatr giving IntensMatr
            [IntensMatr, StabStateMatr, InstabStateMatr]=mixgenerate(NewMarking,
            IntensMatr, StabStateMatr, InstabstateMatr)
        end
    end
end
```

The check and fire algorithm is the same as in SPN. The pure undelayed generate algorithm (pudgenerate) is given by:

```
function [InstabStateMatr,StableMark]=pudgenerate(Marking, OldInstabStateMatr)

if any transition in the undelayed part of the net is fireablewith the present Marking then
    fire it giving StableMarking
    while a transition has been fired
        if Marking is not found in InstabStateMatr then
            add Marking to InsatbStateMatr
            Marking:=StableMarking
            if any transition in the undelayed part of the net is fireablewith the present
            Marking then
                fire it giving StableMarking
            end
        end
    end
end
```

In the undelayed parts of the net possible conflicts must also be resolved giving the same result as in Omola. The undelayed check and fire alogorithm is given by:

```
make a listof the enabled undelayed transitions sorted with decreasing priority.
for k:=1 to the number of elements in this list
    if the transition on the k:th place in the list is still enabled then fire it
end
```

Notice that the last algorithm is exactly the same as the CEF-algorithm in 2.3.4. If the latter should be changed this is the only of the Matlab algorithms that must be modified.

The implementation can handle concurrency involving one delayed step and several undelayed steps. (Concurrency involving two or more delayed steps does not model a Markov-process as we have seen above and a check is made that a net does not contain such a structure). The restrictions on the weights on the arcs made above are also valid for the delayed part of a net in this paragraph.

Five functions of which two are recursive handles the algorithms. The user function has the heading:

```
function [IntensMatr, StabStateMatr, InstabStateMatr, Bounded, Ok]
=mixmarkint(Dp, Dm, Intensity, InitMarking, NoOfGroups,
PriolDList, PrioUList);
```

Mixmarkint=mix of delayed and undelayed steps markov intensity

IntensMatr: The NoOfStabStates*NoOfStabStates matrix which contains the intensity matrix for the modelled Markov-process.

StabStateMatr: The NoOfStabStates*NoOfSteps matrix which contains the stable states of the Markov-process.

InstabStateMatr: The NoOfInstabStabStates*NoOfSteps matrix which contains the unstable states of the net.

Bounded: The number of recursive calls are limited. If the limit is reached the algorithm is interrupted and bouned is set to 0.

Conc: If the net contains concurrency between two delayed steps the algorithms are interrupted and conc=1 else 0.

Dp, Dm: The NoOfTransitions*NoOfSteps (delayed and undelayed) matrices which have been presented above.

Intensity: The 1*NoOfSteps (delayed and undelayed) row vector which contains the intensities λ of the steps. The undelayed steps have zero intensity.

InitMarking: The 1*NoOfSteps (delayed and undelayed) row vector which contains the initial marking of the net.

NoOfGroups: The 1*NoOfSteps (delayed and undelayed) row vector which contains the number of groups that the tokens in a step can be divided into.

PrioDList: The 1*(NoOfTransitions proceded by delayed steps) row vector which contains the numbers of the transitions sorted in the same way as they have been declared in the Omola implementation.

PrioUList: The 1*(NoOfTransitions not proceded by delayed steps) row vector which contains the numbers of these transitions sorted in the same way as they have been declared in the Omola implementation.

The user function calls on the recursive function mixgenerate which contains the first algorithm above. The latter function calls on pdstcheckfire (pure delayed steps check and fire) and the recursive function pudgenerate (pure undelayed steps generate) which contains the second algorithm above. Pdstcheckfire contains the same algorithm as the corresponding algorithm in SPN. Pudgenerate then calls on pudcheckfire (pure undelayed steps check and fire) representing the third algorithm above.

At the same time as the algorithms in this and the preceding paragraph was developed, algorithms generating the intensity matrix for nets with delayed transitions was also implemented. The user function pdtmarkint (pure delayed transitions markov intensity) together with the recursive function pdtgenerate and the function checkfiresimple are found in the appendix B. These algorithms have been tested on a simple example in [11].

3.3.3 Monte Carlo Simulations

By statistical analysis of the logged data from different simulations the asymptotical probability distribution and the probability distribution function can be estimated. The output from the Omola printer objects has the format:

```
   NaN       NaN       NaN  NaN  NaN
0.000000  0.000000     1    0    0
0.000000  0.000000     0    1    0
0.523768  0.523768     0    0    1
1.000000  0.476232     1    0    0
...

9.624250  0.476232     1    0    0
   NaN       NaN       NaN  NaN  NaN
0.000000  0.000000     1    0    0
0.000000  0.000000     0    1    0
0.483683  0.493683     0    0    1
...
9.762548  0.213672     0    1    0
```

The table above shows the result from simulating a GSPN twice from simulation time 0 to 10. The log of each simulation is started with a row of NaN:s to help the Matlab algorithms below to separate the different simulation runs. The code from the final implementation is found in Appendix C. When a transition is fired the simulation time is logged in column1. To preserve the number of significant digits the difference between this time and the simulation time when the last firing occured is logged in column2. Column 3-5 consists of the new marking after the firing and each step has a column of its own. In this example the net is made up of three steps. The marking is logged according to 2.3.3 where the printer objects are defined. The first two rows of each simulation is an example where the firing of an undelayed part of the net has been logged.

Asymptotical probability distribution

The asymptotical distribution function is estimated assuming that it exists. One simulation is run a sufficiently long time (in some meaning). The analysing algorithm for the logged data is then:

```
convert the logged data into a Matlab matrix.
for the second row to the second last row in this matrix
      if the row represents an unstable state then
            add the state to the InstabStateMatr if it is not already included
      else
            add the state to the StateMatr if it is not already included
            SumTime(State):=SumTime(State)+time  to the next firing occurs
      end
end
do the same procedure for the last row in the matrix but let
      SumTime(State):=SumTime(State)+simulation stop time
end
for all states in StateMatr
      ProbabilityVector(state):=SumTime(State)/simulation time
end
```

The algorithm detects all stable states and the asymptotical probabilities are estimated by dividing the time which the PN has been in the state with the total simulation time [8]. The algorithm has been implemented in the Matlab function asymdistr which has the heading:

```
function [StateMatr,InstabstateMatr,ProbVect]=
asymdistr(SimTime,InitDelay)
```

StateMatr: The NoOfStableStates*NoOfSteps matrix which contains the marking of the stable states that have occured during the simulation.

InstabStateMatr: The NoOfUnstableStates*NoOfsteps matrix which contains the marking of the unstable states that have occured during the simulation.

ProbVect: The 1*NoOfStableStates rowvector which contains the estimated probabilities.

SimTime: The simulated time.

InitDelay: If there are InitDelayedSteps in the PN this should be one otherwise zero.

Probability distribution function

The time dependent probability function is estimated. A sufficient number of simulations (in some meaning) are run in the simulation time interval in which the function is to be estimated. The analysing algorithm for the logged data is then:

convert the logged data to a matlab matrix
define in how many discrete time steps of the simulation time interval the function is to be calculated.
for each of these TimeStep
    for each simulation
        if the State(TimeStep,Simulation) is unstable then
            add the State to InstabSateMatr if it is not already included
        else
            add the State to StateMatr if it is not already included
            SumNoOfTimes(State(Timesteps, Simulation), TimeSteps)=
            SumNoOfTimes(State(Timesteps, Simulation), TimeSteps)+1
        end
    end
end
for each TimeStep
    for each State
        ProbabilityMatrix(State, TimeStep)=
        SumNoOfTimes(State, TimeStep)/sum(SumNoOfTimes(TimeStep)
    end
end

where sum(SumNoOfTimes(TimeStep) denotes the sum of the column TimeStep

The algorithm only detects the different states at the specific times TimeStep. The probability of the process being in a specific state at a specific TimeStep is estimated by dividing the sum of the simulations in which the net has been in the state at the specific time by the total number of simulations [8]. The algorithm has been implemented in the Matlab function ensemblemv (ensemble mean value) which has the heading

```
function [StateMtr, InstabStateMatr, ProbMatr]=ensemblemv(SimTime,
TimeInt, IntiDelay)
```

StateMatr: The NoOfStableStates*NoOfSteps matrix which contains the marking of the stable states that have occured during the simulation.

InstabStateMatr: The NoOfUnstableStates*NoOfsteps matrix which contains the marking of the unstable states that have occured during the simulation.

ProbMatr: The NoOfStableStates*NoOfTimeSteps matrix which contains the estimated probabilities of the net being in a state at a specific time.

SimTime: The simulated time.

TimeInt: The time interval between two Timesteps.
(TimeSteps= 0..round(SimTime/TimeInt)

InitDelay: If there are InitDelayedSteps in the PN this should be one otherwise zero.

In the function above every new marking that is found is a new state. It is also of interest just to study a single step and to identify the possible states of the step and the corresponding probabilities. This can be done by modifying the algorithm above so that the the probabilities of states which have the same number of tokens in a specific step at the same time are added. By doing this the probability of a specific step having a certain number of tokens at a certain time is obtained. The implementation is called stepan (step analysis) and has the heading:

```
function [StateMatr,ProbMatr]=stepan(Steps, SimTime, TimeInt,
InitDelay)
```

StateMatr is a (max(NoOfStates(AnalysedStepi))*NoOfAnalysedSteps where i= 1.. NoOfAnalysedSteps) matrix. The analysed steps which have fewer states than the analysed step with the most states will have -1 on the elements in the matrix which have not been used. Steps is a 1*NoOfAnalysedSteps row vector which contains the indexes of the steps that are to be analysed in the way described above.

To facilitate the corresponding analysis using an intensity matrix generated by a markint function in 3.3.2 the function ensprob (ensemble probability) has been implemented with the heading:

```
function [Result,StepStateMatr]=ensprob(IntensMatr, InitState,
StateMatr,Steps, Time, Interval)
```

where Result corresponds to the matrix ProbMatr above and IntensMatr and StateMatr are the results generated by a markint function in 3.3.2. Steps correspond to the row vector above and Time to the simulation time. Interval corresponds to TimeInt. The algorithm calculates $P(t)=P_0\exp(At)$ at the specific time steps and summarizes the probabilities that can be connected to a marking of the step(s) that is analysed.

4. Modelling Example: Kanban Production System

The inspiration for this example has been sought in [7] which contains a mathematical model of a Kanban system and in [9] which uses PN for modelling pull-systems.

4.1 Petri-Net Modelling

The principle is shown in fig12.

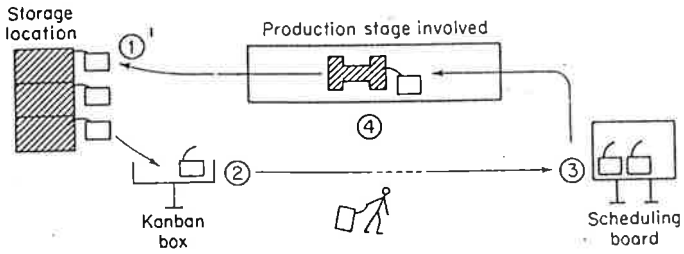


Fig12 Illustration of Kanban principle

The production is split into a number of stages. At each stage a certain number of operations are carried out. The parts processed at a stage is put in a material carrier. A kanban (a note of what is in the carrier and in which amounts) is attached to the carrier which is stored at a location designated by the kanban (1 in fig12 above). When a succeding process withdraws material, a worker lifts of the kanban and puts it in a kanban box (2). Kanbans are collected from the box at regular intervals and are hung on a scheduling board. The sequence of the various kanbans on the board shows workers the dispatching job in the process (3).

A worker produces various items in accordance with the sequence of the kanbans on the board. The kanban itself moves in the process with the first unit of the batch (4). The procedure 1 to 4 is repeated as the production continues. It should be noted that if the succeeding process never withdraws material from the preceding process, then the kanbans are neither collected from the box nor hung on the scheduling board. Consequently the item is never processed at this stage of the process.

It is quite simple to describe a kanban system in a few difference equations. However, it is hard to find analytical solutions to these equations except for some special cases and therefore it is convenient to make computer simulations.

Fig 13 shows the GSPN model of a Kanban production system that will be analysed.

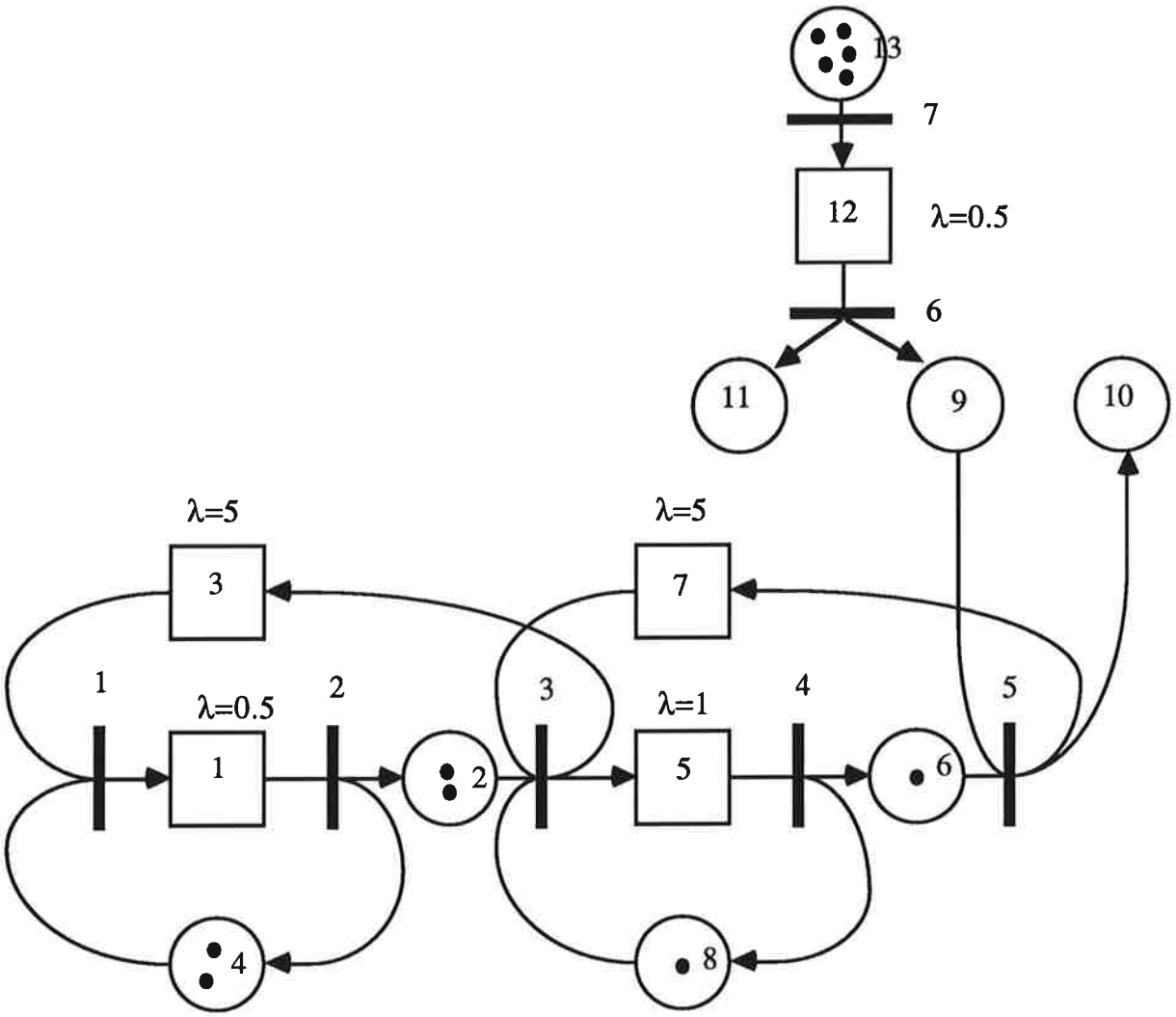


Fig13 The GSPN which has been analysed

Each stage in the process has a similar structure and is called a link in the implementation. A link consists of two DelayedSteps, two Steps and two Transitions. The system above is built of two links and is connected to an environment which models the demand and withdrawal of finished products. The function of a link is easily understood. The tokens in step6 (FilledCarriers in the implementation), represent the number of carriers which are filled with products that have been finished by the link. In this example we assume that a carrier only contains one product. When the succeeding process withdraws the carrier the kanban which has been attached to it is put in the step7 (KanbanBox). This box is emptied in time intervals that are assumed to be exponentially distributed and hung on the scheduling board. The number of tokens in step8 (AvailMach) represents the number of machines that are available for production in the link. When there is a kanban on the scheduling board, an available machine and a filled materialcarrier in the proceding link one token is withdrawn from each of the corresponding steps and a token is added in step5 (ProdInmach), which represents the very production. When the carrier is withdrawn from the proceding step the attached kanban is put in the corresponding kanban box that is step3. The machining time is assumed to be exponentially distributed and when the material carrier has been filled with products it is transfered to the stock represented by step6 (FilledCarriers).

In the first (leftmost) link the production is started as soon as there are kanbans on the scheduling board and available machines, ie the supply of raw material is assumed to be infinite. When the simulation is started five tokens are put in step13 (Initial) which represents the total potential orderstock. As the tokens in step12 (Demand) become available tokens are put in step11 (Order) registrating orders received and in step9 (Backlog) representing received orders that have not been

delivered yet. As products are delivered from the final link in the production system they are registered in step10 (Delivered). There are two available machines in link1 and one in link2 when the simulation is started. Further there is two filled materialcarriers in link1 and one in link2.

The Omola implementation consists of three layers of objects, where the first layer are the primitives defined in chapter 2 and 3. The second layer consists of the link objects described above (see Appendix D) and the third layer is a definition of the production system:

```
ProdNet ISA Model WITH
submodels:
Pr ISA Printer2;

L1 ISA FirstLink WITH
    AvailMach.IniState:=2;
    FilledCarrirs.InitState:=2;
    MachiningTime.Lambda:=0.5;
    TransportKanbanTime.Lambda:=5;
END

L1 ISA LastLink WITH
    AvailMach.IniState:=1;
    FilledCarrirs.InitState:=1;
    MachiningTime.Lambda:=1.0;
    TransportKanbanTime.Lambda:=5;
END

Controller ISAN Environment WITH
    Initial.InitState:=5;
    Random.Lambda:=0.5;
END;

Connections:
    L1.Out AT L2.In;
    L2.EnvirConn AT Controller.InNet;

    L1.PrinterComm AT Pr.Comm1;
    L2.PrinterComm AT Pr.Comm2;
END;
```

In Appendix D a more complex model of the Kanban production system has also been implemented, but this has not been studied any further (1prodprim, 1net).

4.2 Performance Analysis

The utilization of the machines in link1 and 2 in the net in fig13 will now be studied. This will be done by calculating the probability distribution function of the states in step 1 and 5. Step1 has 3 states: 0, 1 respectively 2 tokens, step5 has 2 states: 0 and 1 token. The GSPN has been simulated 2500 times in the simulation time interval 0-15. The five distribution functions are estimated in the interval by ProbMatr in Matlab:

```
[StateMatr,ProbMatr]=stepan(Steps, SimTime, TimeInt, InitDelay)
```

where

Steps=[1 5]
SimTime=15
TimeInt=0.1

InitDelay=0

The distribution functions are then calculated using the probability theory approch. The intensitymatrix is given in Matlab by:

```
[IntensMatr, StabStateMatr, InstabStateMatr, Bounded, Ok]
=mixmarkint(Dp, Dm, Intensity, InitMarking, NoOfGroups,
PriolDList, PrioULIst);
```

where

Dp is a 7*13 matrix with zeros in all elements except for ones in the elements with the indexes:

(1,1) (2,2) (2,4) (3,3) (3,5) (4,6) (4,8) (5,7) (5,10) (6,9) (6,11) (7,12)

Dm is a 7*13 matrix with zeros in all elements except for ones in the elements with the indexes:

(1,3) (1,4) (2,1) (3,2) (3,7) (3,8) (4,5) (5,6) (5,9) (6,12) (7,13)

Intensity is an 1*13 row vector with zeros on all elements except for the following indexes:

(1,1): 0.5 (1,3): 5 (1,5): 1 (1,7): 5 (1,12): 0.5

InitMarking is an 1*13 row vector with zeros on all elements except for the following indexes:
(1,2): 2 (1,4): 2 (1,6): 1 (1,8): 1 (1,13): 5

NoOfGroups is the same row vector as InitMarking

PrioDList=[1 2 3 4 6]

PrioUList=[7 5]

The function identifies 156 stable states and 6 unstable states. The result is then processed by:

```
[Result,StepStateMtr]=ensprob(IntensMatr, InitState, StateMatr,
Steps, Time, Interval)
```

where

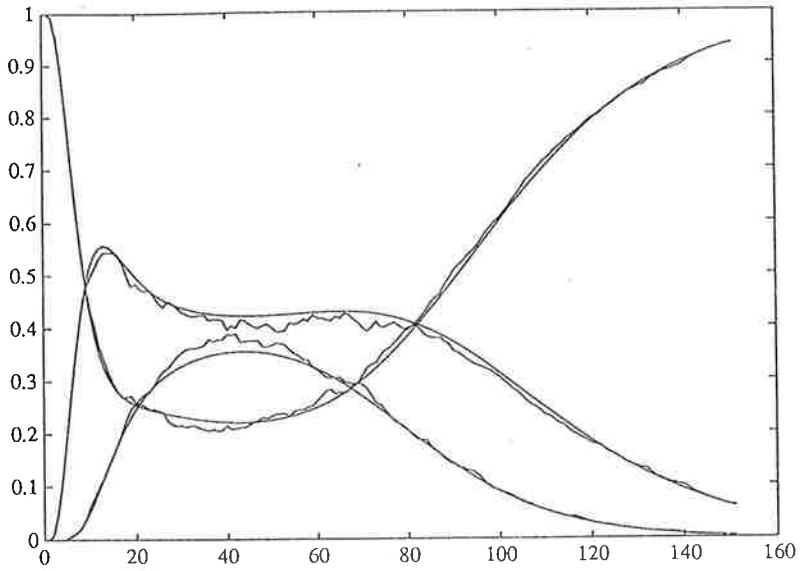
IntensMatr: The 156*156 matrix resulting from mixmarkint.

InitState: The same as InitMarking above.

StateMatr: The 156*13 matrix StabStateMatr resulting from mixmarkint.

Steps=[1 5]
Time=15
Interval=0.1

The result from the two different approaches is shown in fig 14 and 15.



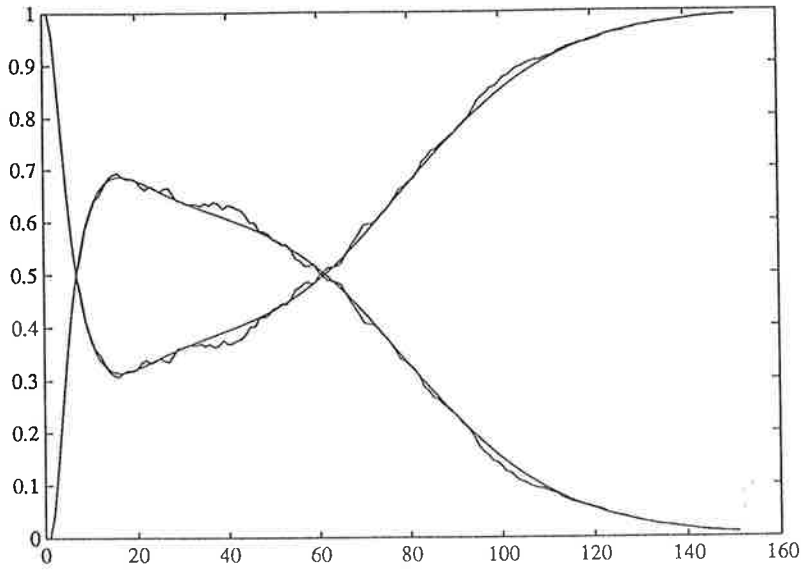


Fig14 and 15.
Figure 14 at the top of the page shows the distribution functions for step1, fig15 the distribution functions for step2. The y-axis represents the probability, the x-axis (simulation time)*10.

The smooth curves origins from running ensprob, the others from running stepan. In fig 14 the distribution function of step 1 is shown. The curve which is 1 at t=0 shows the distribution of the state with zero running machines, the curve with the first peak the state with one running machine and the third curve the state with two running machines. In the same way the curve which is 1 at t=0 in fig15 shows the distribution of the state of no running machines in step5, the other curve the distribition of one running machine. The two different approaches coincide quite well. (As the number of simulations goes to infinity the difference between the two different types of curves should go towards zero). No attempts to decide confidence intervals for the statistical estimation have been made.

5. Conclusions

Omola is usable for implementing PN. The main problem is that the instantiation time is quite long even for rather small PN. To some extent this is explained by the fact that the primitives have been implemented to handle general structures and each primitive contains several different events. Simpler but more specialised primitives could be implemented and this would decrease the instantiation time.

Even a simple PN can have a large set of stable states. The Matlab algorithms for state space generation tend to be quite slow and memory consuming. Once the intensity matrix has been computed it still remains to calculate the corresponding exponential matrix at a large number of times to decide the probability distribution function. This is time consuming for matrices with large dimensions.

5.1 Simulation

The scope of this work has been to investigate the possibilities of Omola and a variety of implementations have been tested. The instantiation time of the objects is quite long and to reduce this a vector handling concept is needed so that the number of terminals on each object can be adapted. The vector concept is also needed to facilitate implementation of coloured PN.

In 2.2 the impression is given that Grafcet can be modelled with the extended PN-model. This is not perfectly true in the current implementation. There is no real Grafcet step-primitive (with a boolean state) and this implies that only Grafcets that are denoted sound in [5] can be correctly modelled. However, it is of no principle difficulty to add a Grafcet step-primitive and corresponding Condition and PulseTransitions which work with a boolean state variable.

It can be discussed if the implementation shall be able to handle conflicts between Pulse- and ConditionTransitions. If this possibility is excluded the transition-primitives can be made simpler thus reducing the instatiation time.

The declaration rule in connection with fig5 in 2.4.2 is not very user friendly and perhaps should a special conflict-primitive be implemented to guarante that the transitions are declared in the right order.

One of the main features with PN is that it is a graphical tool. Thus a system for animation and graphical editing is needed. The log-file generated by the printer objects could probably be of some help to the animator.

5.2 Markovian State Space Generation

An interface is needed that is able to convert the Omola representation of the PN to the matrices used by the Matlab functions that have been implemented. The algorithms in these funcions have to be improved so that input arcs to delayed steps with different weights will be allowed. This increases the combinatorial complexity, since the order in which different groups in a delayed step is fired in has to be considered. Further the boundary check algorithm in [12] should be implemented.

5.3 Performance analysis

In this thesis we have modelled a process and made a very simple performance analysis without any real conclusions. The next step would be to make a meaningful analysis of the extracted data and to take control actions based on these data. The control decisions should then be fed back to the PN model giving an automatic control system. This control system could be implemented as a PN.

6. References

[1] Ajmone M., G. Balbo, G.Conte (Year Unknown): *Performance Models of Multiprocessor Systems*, The MIT Press

[2] Andersson M. (1990): Omola-An Object-Oriented Language for Model Representation, TFRT-3208, Licenciate Thesis, Department ofAutomatic Control, Lund Institute of Technology, Sweden

[3] Andersson M. (1991): Event fundamentals, Note, Department of Automatic Control, Lund Institute of Technology, Sweden

[4] Blom G. (1984): *Sannolikhetsteori med tillämpningar*, Studentlitteratur

[5] David R., H. Alla (1989):*Du Grafcet aux réseaux de Petri*, Edition Hermès, Paris

[6] IEC (1988): *Preparation of function charts for control systems*, CEI IEC 848 Publication 848:1988, International Electrotechnical Comission

[7] Kimura O., H. Terada (1981): Design and Analysis of Pull System, a method of multistage production control, INT. J. PROD. RES. 1981, VOL. 19 NO. 3

[8] Lindgren G., H. Rootzén (1985): Stationära Stokastiska Processerför E, KF Sigma Tryck 1985

[9] Mascolo, M. (1990): Modélisation et évaluation de performances de systèmes de production gérés en Kanban, Doctoral dissertation, Laboratorie d' Automatique de Grenoble, l'Institut Bational Polytechnique de Grenoble, Grenoble, France

[10] Moler C.,J.Little and S.Bangert (1987): *PRO-Matlab, User's Guide*, The Math Works Inc, Sherborn, MA, USA

[11] Murata T. (1989): Petri Nets:Properties, Analysis, Applications, Proc IEEE, VOL. 77, NO.4, 1989

[12] Peterson J.L. (1981): *Petri Net theory and the Modelling of Systems,* Prentice-Hall 1981

[13] Reisig W. (1982): *Petri Nets-An Introduction*, Springer-Verlag 1982

Appendix A. Petri-Net Primives

```
LIBRARY PetriBase;

GrafObject ISA Model;


IntParameter ISA Parameter WITH
value TYPE Integer;
default TYPE Integer :=0;
END;


DiscIntTerminal ISA SimpleTerminal WITH
value TYPE  DISCRETE Integer;
default TYPE Integer := 0;
END;


BooleanTerminal ISA SimpleTerminal WITH
value TYPE DISCRETE Real;
END;

%*****************************************************************************


StepLTerminal ISA RecordTerminal WITH
        State ISA DiscIntTerminal;
        Trigg ISAN EventInput;
        Weight ISA DiscIntTerminal;
END;

StepUTerminal ISA RecordTerminal WITH
        Trigg ISAN EventInput;
        Weight ISA DiscIntTerminal;
END;


TransitionLTerminal ISA RecordTerminal WITH
        Trigg ISAN EventOutput;
        Weight ISA DiscIntTerminal;
END;


TransitionUTerminal ISA RecordTerminal WITH
        State ISA DiscIntTerminal;
        Trigg ISAN EventOutput;
        Weight ISA DiscIntTerminal;
END;



RandomCallTerminal ISA RecordTerminal WITH
CallForAdmission ISAN EventOutput;
GoAhead ISAN EventInput;
END;


RandomAnswerTerminal ISA RecordTerminal WITH
CallForAdmission ISAN EventInput;
GoAhead ISAN EventOutput;
END;


RandomCallSetTerminal ISA RecordTerminal WITH
%Is used in connection with DelayedStep
InitChannel1,InitChannel2,InitChannel3,
Channel1, Channel2 ,Channel3 ISA RandomCallTerminal;
END;


RandomAnswerSetTerminal ISA RecordTerminal WITH
%Is used in connection with DelauyedStep
InitChannel1,InitChannel2,InitChannel3,
Channel1, Channel2, Channel3 ISA RandomAnswerTerminal;
END;



PrinterInTerminal ISA RecordTerminal WITH
        State ISA DiscIntTerminal;
        Print ISA EventInput;
END;


PrinterOutTerminal ISA RecordTerminal WITH
        State ISA DiscIntTerminal;
        Print ISA EventOutPut;
END;

PrinterInConn ISA RecordTerminal WITH
                In1,In2,In3,In4,In5,In6,In7 ISA PrinterInTerminal;
        END;

PrinterOutConn ISA RecordTerminal WITH
                Ch1,Ch2,Ch3,Ch4,Ch5,Ch6,Ch7 ISA PrinterOutTerminal;
END;
```

```
LIBRARY Petri;
USES PetriBase;



%*****************************************************************************
%
% Transitions
%
%*****************************************************************************


% Standard Petri-nets



ConditionTransition ISA GrafObject WITH
parameters:
        Produce1 ISA IntParameter WITH default :=1; END;
        Consume1 ISA IntParameter WITH default :=1; END;

        Produce2 ISA IntParameter WITH default :=0; END;
        Consume2 ISA IntParameter WITH default :=0; END;

        Produce3 ISA IntParameter WITH default :=0; END;
        Consume3 ISA IntParameter WITH default :=0; END;

%Notice! Unused connections should  have zero Consume otherwise the
%test ONEVENT Cond AND ... gives a wrong result

terminals:
        Upper1,Upper2,Upper3 ISA TransitionUTerminal;
        Lower1,Lower2,Lower3 ISA TransitionLTerminal;
        Condition ISA BooleanTerminal;
        Init ISAN Event;

variable:
        Fireable TYPE DISCRETE Integer;

behaviour:

        ONEVENT Init DO
                NEW(Lower1.Weight):=Produce1;
                NEW(Upper1.Weight):=Consume1;

                NEW(Lower2.Weight):=Produce2;
                NEW(Upper2.Weight):=Consume2;

                NEW(Lower3.Weight):=Produce3;
                NEW(Upper3.Weight):=Consume3;

        END;

        WHEN Condition>0.0 AND
             Upper1.State>=Upper1.Weight AND
             Upper2.State>=Upper2.Weight AND
             Upper3.State>=Upper3.Weight
        DO
                new(Fireable) := Upper1.State>=Upper1.Weight AND
                                 Upper2.State>=Upper2.Weight AND
                                 Upper3.State>=Upper3.Weight;
        END;

        ONEVENT Fireable > 0 AND
                Upper1.State>=Upper1.Weight AND
                Upper2.State>=Upper2.Weight AND
                Upper3.State>=Upper3.Weight

        CAUSE
                Upper1.Trigg, Lower1.Trigg,
                Upper2.Trigg, Lower2.Trigg,
                Upper3.Trigg, Lower3.Trigg;

        ONEVENT Fireable>0  DO NEW(Fireable):=0; END;

END;%ConditionTransition



Transition ISA ConditionTransition WITH
Condition:=1.0;
END; %Transition



%-----------------------------------------------------------------------------
%
% Extended Petri-nets



PulseTransition ISA GrafObject WITH
parameters:
        Produce1 ISA IntParameter WITH default :=1; END;
        Consume1 ISA IntParameter WITH default :=1; END;

        Produce2 ISA IntParameter WITH default :=0; END;
        Consume2 ISA IntParameter WITH default :=0; END;

        Produce3 ISA IntParameter WITH default :=0; END;
        Consume3 ISA IntParameter WITH default :=0; END;

terminals:
        Upper1,Upper2,Upper3 ISA TransitionUTerminal;
        Lower1,Lower2,Lower3 ISA TransitionLTerminal;
        Condition ISA BooleanTerminal;
        Init ISAN Event;

variable:
        Fireable TYPE DISCRETE Integer;

behaviour:

        ONEVENT Init DO
                NEW(Lower1.Weight):=Produce1;
                NEW(Upper1.Weight):=Consume1;

                NEW(Lower2.Weight):=Produce2;
                NEW(Upper2.Weight):=Consume2;
```

```
                NEW(Lower3.Weight):=Produce3;
                NEW(Upper3.Weight):=Consume3;
        END;
        ONEVENT Condition>0.0 DO
            NEW(Fireable):= Upper1.State>=Upper1.Weight AND
                            Upper2.State>=Upper2.Weight AND
                            Upper3.State>=Upper3.Weight;
        END;
        ONEVENT Fireable>0 AND Upper1.State>=Upper1.Weight AND
                               Upper2.State>=Upper2.Weight AND
                               Upper3.State>=Upper3.Weight
        CAUSE
                Upper1.Trigg, Lower1.Trigg,
                Upper2.Trigg, Lower2.Trigg,
                Upper3.Trigg, Lower3.Trigg;

        ONEVENT Fireable>0  DO NEW(Fireable):=0; END;
END;%PulseTransition



InhCondTransition ISA GrafObject WITH

parameters:
        Produce1 ISA IntParameter WITH default :=1; END;
        Consume1 ISA IntParameter WITH default :=0; END;

%Upper1 is the inhibitorentrance. If the Step that is connected to this
%entrance has a state that differs from Consume1 then the InhTransition
%is not enabled.

        Produce2 ISA IntParameter WITH default :=0; END;
        Consume2 ISA IntParameter WITH default :=1; END;

        Produce3 ISA IntParameter WITH default :=0; END;
        Consume3 ISA IntParameter WITH default :=0; END;


%Notice! The following values must be attached to Consumei corresponding to
%the connection Upperi:

%Not-inhibited entrances which are connected: Consumei must be >0
%Not-inhibited entrances which are not connected: Consumei must be 0

terminals:
        Upper1,Upper2,Upper3 ISA TransitionUTerminal; %Upper1ar inhibitoringang
        Lower1,Lower2,Lower3 ISA TransitionLTerminal;
        Condition ISA BooleanTerminal;
        Init ISAN Event;

variable:
        Fireable TYPE DISCRETE Integer;

behaviour:

        ONEVENT Init DO
                NEW(Lower1.Weight):=Produce1;
                NEW(Upper1.Weight):=0;

                NEW(Lower2.Weight):=Produce2;
                NEW(Upper2.Weight):=Consume2;
```

```
                NEW(Lower3.Weight):=Produce3;
                NEW(Upper3.Weight):=Consume3;

        END;
        WHEN Condition>0.0 AND
             Upper1.State == Upper1.Weight AND
             Upper2.State>=Upper2.Weight AND
             Upper3.State>=Upper3.Weight
        DO
        new(Fireable) := Upper1.State == Upper1.Weight AND
                                 Upper2.State>=Upper2.Weight AND
                                 Upper3.State>=Upper3.Weight;
        END;

        ONEVENT Fireable > 0 AND
                Upper1.State == Upper1.Weight AND
                Upper2.State>=Upper2.Weight AND
                Upper3.State>=Upper3.Weight

        CAUSE
                Lower1.Trigg,
                Upper2.Trigg, Lower2.Trigg,
                Upper3.Trigg, Lower3.Trigg;

        ONEVENT Fireable>0  DO NEW(Fireable):=0; END;

END; %InhCondTransition



InhTransition ISA InhCondTransition WITH
Condition:=1.0;
END; %InhTransition



InhPulseTransition ISA GrafObject WITH

%Notice! The comments in InhCondtransition

parameters:
        Produce1 ISA IntParameter WITH default :=1; END;
        Consume1 ISA IntParameter WITH default :=0; END;


        Produce2 ISA IntParameter WITH default :=0; END;
        Consume2 ISA IntParameter WITH default :=1; END;

        Produce3 ISA IntParameter WITH default :=0; END;
        Consume3 ISA IntParameter WITH default :=0; END;

terminals:
        Upper1,Upper2,Upper3 ISA TransitionUTerminal;
        Lower1,Lower2,Lower3 ISA TransitionLTerminal;
        Condition ISA BooleanTerminal;
        Init ISAN Event;

variable:
        Fireable TYPE DISCRETE Integer;

behaviour:
```

```
        ONEVENT Init DO
                NEW(Lower1.Weight):=Produce1;
                NEW(Upper1.Weight):=Consume1;

                NEW(Lower2.Weight):=Produce2;
                NEW(Upper2.Weight):=Consume2;

                NEW(Lower3.Weight):=Produce3;
                NEW(Upper3.Weight):=Consume3;
        END;
        ONEVENT Condition>0.0 DO
            NEW(Fireable):= Upper1.State == Upper1.Weight AND
                            Upper2.State>=Upper2.Weight AND
                            Upper3.State>=Upper3.Weight;
        END;
        ONEVENT Fireable>0 AND Upper1.State == Upper1.Weight AND
                               Upper2.State>=Upper2.Weight AND
                               Upper3.State>=Upper3.Weight
        CAUSE
                Lower1.Trigg,
                Upper2.Trigg, Lower2.Trigg,
                Upper3.Trigg, Lower3.Trigg;


        ONEVENT Fireable>0  DO NEW(Fireable):=0; END;

END;% InhPulseTransition



%*****************************************************************************
%
%Steps
%
%*****************************************************************************


% Standard Petri-nets


%InitStep2 see further down!

InitStep1 ISA GrafObject WITH
attribute:
        State  TYPE DISCRETE Integer;
variable:
        Init ISAN Event;
        Dummy TYPE DISCRETE Real;
parameter:
        InitState1 ISA IntParameter;
terminals:
        Upper1, Upper2, Upper3 ISA StepUTerminal;
        Lower1, Lower2, Lower3 ISA StepLTerminal;
        InsertNew ISAN EventInput;
        Printer ISA PrinterOutTerminal;
behaviour:
        ONEVENT Init DO
                NEW(State):=InitState1;
        END;
```

```
        ONEVENT InsertNew DO new(State):=State+1;END;

        ONEVENT Upper1.Trigg DO new(State):=State+Upper1.Weight;END;
        ONEVENT Lower1.Trigg DO new(State):=State-Lower1.Weight;END;



        ONEVENT Upper2.Trigg DO new(State):=State+Upper2.Weight;END;
        ONEVENT Lower2.Trigg DO new(State):=State-Lower2.Weight;END;

        ONEVENT Upper3.Trigg DO new(State):=State+Upper3.Weight;END;
        ONEVENT Lower3.Trigg DO new(State):=State-Lower3.Weight;END;

        Lower1.State:=State;
        Lower2.State:=State;
        Lower3.State:=State;


%Printer-extra
        ONEVENT Init OR InsertNew CAUSE Printer.Print;

%Printer
        ONEVENT Lower1.Trigg OR Lower2.Trigg OR Lower3.Trigg OR
                Upper1.Trigg OR Upper2.Trigg OR Upper3.Trigg  CAUSE
                Printer.Print;

        Printer.State:=State;
END;%InitStep1



Step ISA GrafObject WITH
attribute:
        State TYPE DISCRETE Integer;
parameter:
        InitState ISA IntParameter;
terminals:
        Upper1, Upper2, Upper3 ISA StepUTerminal;
        Lower1, Lower2, Lower3 ISA StepLTerminal;
        Printer ISA PrinterOutTerminal;
events:
        Init ISAN Event;
behaviour:
        ONEVENT Init DO
                NEW(State):=InitState;
        END;
        ONEVENT Upper1.Trigg DO new(State):=State+Upper1.Weight;END;
        ONEVENT Lower1.Trigg DO new(State):=State-Lower1.Weight;END;

        ONEVENT Upper2.Trigg DO new(State):=State+Upper2.Weight;END;
        ONEVENT Lower2.Trigg DO new(State):=State-Lower2.Weight;END;

        ONEVENT Upper3.Trigg DO new(State):=State+Upper3.Weight;END;
        ONEVENT Lower3.Trigg DO new(State):=State-Lower3.Weight;END;

        Lower1.State:=State;
        Lower2.State:=State;
        Lower3.State:=State;

%Printer
        ONEVENT Lower1.Trigg OR Lower2.Trigg OR Lower3.Trigg OR
                Upper1.Trigg OR Upper2.Trigg OR Upper3.Trigg  CAUSE
                Printer.Print;

        Printer.State:=State;

END;%Step



SingleStep ISA GrafObject WITH
%This Step must be used when a step is connected to an in- and out-arc of the
%same transition. Is typically used in loops with only one step.

attribute:
        State TYPE DISCRETE Integer;
events:
        Init ISAN Event;
parameter:
        InitState ISA IntParameter WITH Default:=1; END;
terminals:
        Upper1, Upper2, Upper3 ISA StepUTerminal;
        Lower1, Lower2, Lower3 ISA StepLTerminal;
        Printer ISA PrinterOutTerminal;
behaviour:
     ONEVENT Init DO
                NEW(State):=InitState;
     END;

     ONEVENT Lower1.Trigg DO new(State):=State+Upper1.Weight-Lower1.Weight;END;

     ONEVENT Lower2.Trigg DO new(State):=State+Upper2.Weight-Lower2.Weight;END;

     ONEVENT Lower3.Trigg DO new(State):=State+Upper3.Weight-Lower3.Weight;END;

     Lower1.State:=State;
     Lower2.State:=State;
     Lower3.State:=State;

%Printer
      ONEVENT Lower1.Trigg OR Lower2.Trigg OR Lower3.Trigg
      CAUSE Printer.Print;

      Printer.State:=State;


END;%SingleStep



%-------------------------------------------------------------------------------
%
%Timed and Stochastic Petri-nets


%InitDelayedStep see further down!


DelayedStep ISA GrafObject WITH

attribute:
        State,AvailState TYPE DISCRETE Integer;
```

```
terminals:
        Upper1, Upper2, Upper3 ISA StepUTerminal;
        Lower1, Lower2, Lower3 ISA StepLTerminal;
        Comm ISA RandomCallSetTerminal;
        Printer ISA PrinterOutTerminal;

behaviour:

        ONEVENT Upper1.Trigg CAUSE Comm.Channel1.CallForAdmission;
        ONEVENT Upper2.Trigg CAUSE Comm.Channel2.CallForAdmission;
        ONEVENT Upper3.Trigg CAUSE Comm.Channel3.CallForAdmission;

        ONEVENT Upper1.Trigg DO
                NEW(State):=State+Upper1.Weight;
        END;
        ONEVENT Upper2.Trigg DO
                NEW(State):=State+Upper2.Weight;
        END;
        ONEVENT Upper3.Trigg DO
                NEW(State):=State+Upper3.Weight;
        END;


        ONEVENT Comm.Channel1.GoAhead DO
                 new(AvailState):=AvailState+Upper1.Weight;
        END;
        ONEVENT Comm.Channel2.GoAhead DO
                 new(AvailState):=AvailState+Upper2.Weight;
        END;
        ONEVENT Comm.Channel3.GoAhead DO
                 new(AvailState):=AvailState+Upper3.Weight;
        END;


        ONEVENT Lower1.Trigg DO
                 NEW(State):=State-Lower1.Weight;
                 NEW(AvailState):=AvailState-Lower1.Weight;
        END;
        ONEVENT Lower2.Trigg DO
                 NEW(State):=State-Lower2.Weight;
                 NEW(AvailState):=AvailState-Lower2.Weight;
        END;
        ONEVENT Lower3.Trigg DO
                 NEW(State):=State-Lower3.Weight;
                 NEW(AvailState):=AvailState-Lower3.Weight;
        END;

        Lower1.State:=AvailState;
        Lower2.State:=AvailState;
        Lower3.State:=AvailState;

%Printer
        ONEVENT Lower1.Trigg OR Lower2.Trigg OR Lower3.Trigg OR
                Upper1.Trigg OR Upper2.Trigg OR Upper3.Trigg  CAUSE
                Printer.Print;

        Printer.State:=State;


END;    %DelayedStep
```

```
%-------------------------------------------------------------------------------


InitStep2 ISA GrafObject WITH
%This undelayed initial step should be used in nets also containing initial
%delayed steps.
attribute:
        State  TYPE DISCRETE Integer;
parameter:
        InitState1 ISA IntParameter;
terminals:
        Upper1, Upper2, Upper3 ISA StepUTerminal;
        Lower1, Lower2, Lower3 ISA StepLTerminal;
        InsertNew ISAN EventInput;
        Printer ISA PrinterOutTerminal;

submodels:
%The initialisation-net is needed when there are both InitSteps and
%InitDelayedSteps in a net in order to get the right print-out of the initial
%state.
        InitStep1 ISA Step WITH InitState:=InitState1; END;
        InitStep2 ISA Step;
        InitTrans1,InitTrans2 ISA Transition WITH
                        Consume1:=InitState1;
                        Produce1:=InitState1;
        END;

connections:

        InitStep1.Lower1 AT InitTrans1.Upper1;
        InitTrans1.Lower1 AT InitStep2.Upper1;
        InitStep2.Lower1 AT InitTrans2.Upper1;


behaviour:

        ONEVENT InitTrans2.Lower1.Trigg DO
                NEW(State):=State+InitTrans2.Lower1.Weight;
        END;

        ONEVENT InsertNew DO new(State):=State+1;END;

        ONEVENT Upper1.Trigg DO new(State):=State+Upper1.Weight;END;
        ONEVENT Lower1.Trigg DO new(State):=State-Lower1.Weight;END;

        ONEVENT Upper2.Trigg DO new(State):=State+Upper2.Weight;END;
        ONEVENT Lower2.Trigg DO new(State):=State-Lower2.Weight;END;

        ONEVENT Upper3.Trigg DO new(State):=State+Upper3.Weight;END;
        ONEVENT Lower3.Trigg DO new(State):=State-Lower3.Weight;END;

        Lower1.State:=State;
        Lower2.State:=State;
        Lower3.State:=State;

%Printer-extra

        ONEVENT InitTrans2.Lower1.Trigg CAUSE Printer.Print;

%Printer
        ONEVENT Lower1.Trigg OR Lower2.Trigg OR Lower3.Trigg OR
                Upper1.Trigg OR Upper2.Trigg OR Upper3.Trigg  CAUSE
```

```
                Printer.Print;

        Printer.State:=State;
END;%InitStep2


InitDelayedStep ISA GrafObject WITH

attribute:
        State,AvailState TYPE DISCRETE Integer;
parameter:
        InitState1 ISA IntParameter;
        InitState2 ISA IntParameter;
        InitState3 ISA IntParameter;
        GroupSize1 ISA IntParameter;
        GroupSize2 ISA IntParameter;
        GroupSize3 ISA IntParameter;
terminals:
        Upper1, Upper2, Upper3 ISA StepUTerminal;
        Lower1, Lower2, Lower3 ISA StepLTerminal;
        Comm ISA RandomCallSetTerminal;
        Printer ISA PrinterOutTerminal;


submodels:
        InitStep1 ISA Step WITH InitState:=InitState1; END;
        InitStep2 ISA Step WITH InitState:=InitState2; END;
        InitStep3 ISA Step WITH InitState:=InitState3; END;
        InitCollect ISA Step;
        InitTrans1 ISA Transition WITH Produce1:=GroupSize1;
                                       Consume1:=GroupSize1;
                                  END;
        InitTrans2 ISA Transition WITH Produce1:=GroupSize2;
                                       Consume1:=GroupSize2;
                                  END;
        InitTrans3 ISA Transition WITH Produce1:=GroupSize3;
                                       Consume1:=GroupSize3;
                                  END;
        InitCollectTrans ISA Transition WITH
                 Consume1:=5;
                %Consume1:=InitState1+InitState2+InitState3;
        END;
connections:
        InitStep1.Lower1 AT InitTrans1.Upper1;
        InitStep2.Lower1 AT InitTrans2.Upper1;
        InitStep3.Lower1 AT InitTrans3.Upper1;

        InitTrans1.Lower1 AT InitCollect.Upper1;
        InitTrans2.Lower1 AT InitCollect.Upper2;
        InitTrans3.Lower1 AT InitCollect.Upper3;

        InitCollect.Lower1 AT InitCollectTrans.Upper1;

%By choosing Producek=Consumek>=1 in InitTrans schedules for groups can be
%obtained in the random-generators. Put the tokens that is going to be part in
%a group of the size Producek in step InitStepk. The intialisation-net is
%used for getting the right print-out of the iniyial marking by just calling
%the printer once after the initialisation.
%This object can also be used to model  a delayed SingleStep. When this
%application is wanted the InitDelayedStep should be connected with a
%transition and a Step.
```

```
behaviour:

      ONEVENT InitTrans1.Lower1.Trigg CAUSE Comm.InitChannel1.CallForAdmission;
      ONEVENT InitTrans2.Lower1.Trigg CAUSE Comm.InitChannel2.CallForAdmission;
      ONEVENT InitTrans3.Lower1.Trigg CAUSE Comm.InitChannel3.CallForAdmission;

        ONEVENT Upper1.Trigg CAUSE Comm.Channel1.CallForAdmission;
        ONEVENT Upper2.Trigg CAUSE Comm.Channel2.CallForAdmission;
        ONEVENT Upper3.Trigg CAUSE Comm.Channel3.CallForAdmission;



        ONEVENT InitCollectTrans.Lower1.Trigg DO
           NEW(State):=State+InitState1+InitState2+InitState3;
        END;


        ONEVENT Upper1.Trigg DO new(State):=State+Upper1.Weight;END;

        ONEVENT Upper2.Trigg DO new(State):=State+Upper2.Weight;END;

        ONEVENT Upper3.Trigg DO new(State):=State+Upper3.Weight;END;



        ONEVENT Comm.InitChannel1.GoAhead DO
                 new(AvailState):=AvailState+InitTrans1.Lower1.Weight;
        END;
        ONEVENT Comm.InitChannel2.GoAhead DO
                 new(AvailState):=AvailState+InitTrans2.Lower1.Weight;
        END;
        ONEVENT Comm.InitChannel3.GoAhead DO
                 new(AvailState):=AvailState+InitTrans3.Lower1.Weight;
        END;


        ONEVENT Comm.Channel1.GoAhead DO
                 new(AvailState):=AvailState+Upper1.Weight;
        END;
        ONEVENT Comm.Channel2.GoAhead DO
                 new(AvailState):=AvailState+Upper2.Weight;
        END;
        ONEVENT Comm.Channel3.GoAhead DO
                 new(AvailState):=AvailState+Upper3.Weight;
        END;


        ONEVENT Lower1.Trigg DO
                 NEW(State):=State-Lower1.Weight;
                 NEW(AvailState):=AvailState-Lower1.Weight;
        END;
        ONEVENT Lower2.Trigg DO
                 NEW(State):=State-Lower2.Weight;
                 NEW(AvailState):=AvailState-Lower2.Weight;
        END;
               ONEVENT Lower3.Trigg DO
                 NEW(State):=State-Lower3.Weight;
                 NEW(AvailState):=AvailState-Lower3.Weight;
        END;

        Lower1.State:=AvailState;
        Lower2.State:=AvailState;
        Lower3.State:=AvailState;
```

```
%Printer-extra

        ONEVENT InitCollectTrans.Lower1.Trigg CAUSE Printer.Print;

%Printer
        ONEVENT Lower1.Trigg OR Lower2.Trigg OR Lower3.Trigg OR
        Upper1.Trigg OR Upper2.Trigg OR Upper3.Trigg  CAUSE Printer.Print;

        Printer.State:=State;


END; %InitDelayedStep



SingleDelayedStep ISA GrafObject WITH
%See the comments under SingleStep.

attribute:
        State,AvailState TYPE DISCRETE Integer;
parameter:
        InitState1 ISA IntParameter;
        InitState2 ISA IntParameter;
        InitState3 ISA IntParameter;
        GroupSize1 ISA IntParameter;
        GroupSize2 ISA IntParameter;
        GroupSize3 ISA IntParameter;
terminals:
        Upper1, Upper2, Upper3 ISA StepUTerminal;
        Lower1, Lower2, Lower3 ISA StepLTerminal;
        Comm ISA RandomCallSetTerminal;
        Printer ISA PrinterOutTerminal;


submodels:
        InitStep1 ISA Step WITH InitState:=InitState1; END;
        InitStep2 ISA Step WITH InitState:=InitState2; END;
        InitStep3 ISA Step WITH InitState:=InitState3; END;
        InitCollect ISA Step;
        InitTrans1 ISA Transition WITH Produce1:=GroupSize1;
                                       Consume1:=GroupSize1;
                                END;
        InitTrans2 ISA Transition WITH Produce1:=GroupSize2;
                                       Consume1:=GroupSize2;
                                END;
        InitTrans3 ISA Transition WITH Produce1:=GroupSize3;
                                       Consume1:=GroupSize3;
                                END;
        InitCollectTrans ISA Transition WITH
                  Consume1:=InitState1+InitState2+InitState3;
        END;

connections:
        InitStep1.Lower1 AT InitTrans1.Upper1;
        InitStep2.Lower1 AT InitTrans2.Upper1;
        InitStep3.Lower1 AT InitTrans3.Upper1;

        InitTrans1.Lower1 AT InitCollect.Upper1;
        InitTrans2.Lower1 AT InitCollect.Upper2;
        InitTrans3.Lower1 AT InitCollect.Upper3;

        InitCollect.Lower1 AT InitCollectTrans.Upper1;

%By choosing Producek=Consumek>=1 in InitTrans schedules for groups can be
%obtained in the random-generators. Put the tokens that is going to be part in
%a group of the size Producek in step InitStepk. The intialisation-net is
%used for getting the right print-out of the iniyial marking by just calling
%the printer once after the initialisation.
%This object can also be used to model  a delayed SingleStep. When this
%application is wanted the InitDelayedStep should be connected with a
%transition and a Step.

behaviour:

      ONEVENT InitTrans1.Lower1.Trigg CAUSE Comm.InitChannel1.CallForAdmission;
      ONEVENT InitTrans2.Lower1.Trigg CAUSE Comm.InitChannel2.CallForAdmission;
      ONEVENT InitTrans3.Lower1.Trigg CAUSE Comm.InitChannel3.CallForAdmission;

        ONEVENT Upper1.Trigg CAUSE Comm.Channel1.CallForAdmission;
        ONEVENT Upper2.Trigg CAUSE Comm.Channel2.CallForAdmission;
        ONEVENT Upper3.Trigg CAUSE Comm.Channel3.CallForAdmission;



        ONEVENT InitCollectTrans.Lower1.Trigg DO
           NEW(State):=State+InitState1+InitState2+InitState3;
        END;

        %ONEVENT Upper1.Trigg DO schedule(Dummy1,0.0); END;
        %ONEVENT Dummy1 DO new(State):=State+Upper1.Weight;END;

        %ONEVENT Upper2.Trigg DO schedule(Dummy2,0.0); END;
        %ONEVENT Dummy2 DO new(State):=State+Upper2.Weight;END;

        %ONEVENT Upper3.Trigg DO schedule(Dummy3,0.0); END;
        %ONEVENT Dummy3 DO new(State):=State+Upper3.Weight;END;


        ONEVENT Comm.InitChannel1.GoAhead DO
                new(AvailState):=AvailState+InitTrans1.Lower1.Weight;
        END;
        ONEVENT Comm.InitChannel2.GoAhead DO
                new(AvailState):=AvailState+InitTrans2.Lower1.Weight;
        END;
        ONEVENT Comm.InitChannel3.GoAhead DO
                new(AvailState):=AvailState+InitTrans3.Lower1.Weight;
        END;


        ONEVENT Comm.Channel1.GoAhead DO
                new(AvailState):=AvailState+Upper1.Weight;
        END;
        ONEVENT Comm.Channel2.GoAhead DO
                new(AvailState):=AvailState+Upper2.Weight;
        END;
        ONEVENT Comm.Channel3.GoAhead DO
                new(AvailState):=AvailState+Upper3.Weight;
        END;


        ONEVENT Lower1.Trigg DO
                NEW(State):=State+Upper1.Weight-Lower1.Weight;
                NEW(AvailState):=AvailState-Lower1.Weight;
        END;
```

```
        ONEVENT Lower2.Trigg DO
                NEW(State):=State+Upper2.Weight-Lower2.Weight;
                NEW(AvailState):=AvailState-Lower2.Weight;
        END;
                ONEVENT Lower3.Trigg DO
                NEW(State):=State+Upper3.Weight-Lower3.Weight;
                NEW(AvailState):=AvailState-Lower3.Weight;
        END;

        Lower1.State:=AvailState;
        Lower2.State:=AvailState;
        Lower3.State:=AvailState;

%Printer
        ONEVENT Lower1.Trigg OR Lower2.Trigg OR Lower3.Trigg
        CAUSE Printer.Print;

        Printer.State:=State;

END; %SingleDelayedStep



%*****************************************************************************
%
% Other primitives
%
%*****************************************************************************

%Standard Petri-nets
%
%Notice! Only the Sync-primitives is really needed. The Fork-primitives have
%been implemented to support a logical grafical presentatiom.


Fork_2 ISA GrafObject WITH
terminals:
Upper ISA StepUTerminal;
Lower_1,Lower_2 ISA TransitionLTerminal;
behaviour:
        ONEVENT Upper.Trigg CAUSE
                Lower_1.Trigg, Lower_2.Trigg;

        Lower_1.Weight:=Upper.Weight;
        Lower_2.Weight:=Upper.Weight;
END; %Fork_2


Fork_3 ISA GrafObject WITH
terminals:
Upper ISA StepUTerminal;
Lower_1,Lower_2,Lower_3 ISA TransitionLTerminal;
behaviour:
        ONEVENT Upper.Trigg CAUSE
                Lower_1.Trigg, Lower_2.Trigg, Lower_3.Trigg;

        Lower_1.Weight:=Upper.Weight;
        Lower_2.Weight:=Upper.Weight;
        Lower_3.Weight:=Upper.Weight;
END; %Fork_2


Sync_2 ISA GrafObject WITH
terminals:
        Upper_1, Upper_2 ISA TransitionUTerminal;
        Lower ISA StepLTerminal;
behaviour:
        Lower.State:=min(Upper_1.State, Upper_2.State);
        Upper_1.Weight:=Lower.Weight;
        Upper_2.Weight:=Lower.Weight;
        ONEVENT Lower.Trigg CAUSE Upper_1.Trigg, Upper_2.Trigg;
END;%Sync_2


Sync_3 ISA GrafObject WITH

terminals:
        Upper_1, Upper_2, Upper_3 ISA TransitionUTerminal;
        Lower ISA StepLTerminal;
behaviour:
        Lower.State:=min(min(Upper_1.State, Upper_2.State), Upper_3.State);
        Upper_1.Weight:=Lower.Weight;
        Upper_2.Weight:=Lower.Weight;
        Upper_3.Weight:=Lower.Weight;
        ONEVENT Lower.Trigg CAUSE Upper_1.Trigg, Upper_2.Trigg, Upper_3.Trigg;
END;%Sync_3



%*****************************************************************************
%
% Random Generators
%
%*****************************************************************************


%Stochastic Petri-nets


RectGen1 ISA Model WITH
terminal:
        Signal ISAN EventOutput;
parameter:
        LengthOfInterval ISA Parameter WITH default :=10.0;END;
variable:
        Change,Init ISAN Event;
behaviour:
        ONEVENT Init DO
                schedule(Change,rect(LengthOfInterval));
        END;
        ONEVENT Change CAUSE Signal;
        ONEVENT Change DO schedule(Change,rect(LengthOfInterval)); END;

END;%RectGen1



RectGen2 ISA Model WITH
%Has been used for simple tests of the Conditiontransition;
terminal:
        Switch ISA BooleanTerminal;
```

```
parameter:
        LengthOfInterval ISA Parameter WITH default :=10.0;END;
variable:
        Change,Init ISAN Event;
behaviour:
        ONEVENT Init DO
                New(Switch.value):=0.0;
                schedule(Change,rect(LengthOfInterval));
        END;
        ONEVENT Change DO
                NEW(Switch):=if Switch ==1.0 then 0.0 else 0.0; %not Switch
                schedule(Change,rect(LengthOfInterval));
        END;
END;%RectGen2



ExpGen1 ISA Model WITH
terminal:
        Signal ISAN EventOutput;
parameter:
        Lambda ISA Parameter WITH default :=1.0;END;
variable:
        Change,Init ISAN Event;
behaviour:
        ONEVENT Init DO
                schedule(Change,ln(1/(1-rect(1.0))));
        END;
        ONEVENT Change CAUSE Signal;
        ONEVENT Change DO schedule(Change,ln(1/(1-rect(1.0)))); END;
END;%ExpGen1



ExpGen2 ISA Model WITH

%Shakes hand through the Comm-Terminal where the time between the call
%and answer is exponentially distributed with mean 1/Lambda.

event:
        InitChange1,InitChange2,InitChange3 ISAN Event;
        Change1,Change2,Change3 ISAN Event;
parameter:
        Lambda ISA Parameter WITH default :=1.0;END;
terminal:
        Comm ISA RandomAnswerSetTerminal;


behaviour:
        ONEVENT Comm.InitChannel1.CallForAdmission DO
                schedule(InitChange1,ln(1/(1-rect(1.0)))/Lambda);
        END;
        ONEVENT InitChange1 CAUSE Comm.InitChannel1.GoAhead;

        ONEVENT Comm.InitChannel2.CallForAdmission DO
                schedule(InitChange2,ln(1/(1-rect(1.0)))/Lambda);
        END;
        ONEVENT InitChange2 CAUSE Comm.InitChannel2.GoAhead;

        ONEVENT Comm.InitChannel3.CallForAdmission DO
                schedule(InitChange3,ln(1/(1-rect(1.0)))/Lambda);
        END;
        ONEVENT InitChange3 CAUSE Comm.InitChannel3.GoAhead;



        ONEVENT Comm.Channel1.CallForAdmission DO
                schedule(Change1,ln(1/(1-rect(1.0)))/Lambda);
        END;
        ONEVENT Change1 CAUSE Comm.Channel1.GoAhead;

        ONEVENT Comm.Channel2.CallForAdmission DO
                schedule(Change2,ln(1/(1-rect(1.0)))/Lambda);
        END;
        ONEVENT Change2 CAUSE Comm.Channel2.GoAhead;

        ONEVENT Comm.Channel3.CallForAdmission DO
                schedule(Change3,ln(1/(1-rect(1.0)))/Lambda);
        END;
        ONEVENT Change3 CAUSE Comm.Channel3.GoAhead;
END;%ExpGen2



NormGen ISA Model WITH
%Shakes hand through the Comm-Terminal where the time between the call
%and answer is normally distributed with mean Mean and stdev StDev.


variable:
        InitChange1,InitChange2,InitChange3 ISAN Event;
        Change1,Change2,Change3 ISAN Event;
parameter:
        Mean ISA Parameter WITH default :=0.0;END;
        StDev ISA Parameter WITH default :=1.0;END;
terminal:
        Comm ISA RandomAnswerSetTerminal;

behaviour:
        ONEVENT Comm.InitChannel1.CallForAdmission DO
                schedule(InitChange1,Norm(Mean,StDev));
        END;
        ONEVENT InitChange1 CAUSE Comm.InitChannel1.GoAhead;

        ONEVENT Comm.InitChannel2.CallForAdmission DO
                schedule(InitChange2,Norm(Mean,StDev));
        END;
        ONEVENT InitChange2 CAUSE Comm.InitChannel2.GoAhead;

        ONEVENT Comm.InitChannel3.CallForAdmission DO
                schedule(InitChange3,Norm(Mean,StDev));
        END;
        ONEVENT InitChange3 CAUSE Comm.InitChannel3.GoAhead;



        ONEVENT Comm.Channel1.CallForAdmission DO
                schedule(Change1,Norm(Mean,StDev));
        END;
        ONEVENT Change1 CAUSE Comm.Channel1.GoAhead;
```

```
        ONEVENT Comm.Channel2.CallForAdmission DO
                schedule(Change2,Norm(Mean,StDev));
        END;
        ONEVENT Change2 CAUSE Comm.Channel2.GoAhead;

        ONEVENT Comm.Channel3.CallForAdmission DO
                schedule(Change3,Norm(Mean,StDev));
        END;
        ONEVENT Change3 CAUSE Comm.Channel3.GoAhead;
END;%NormGen




%*****************************************************************************
%
% Simple-IO
%
%*****************************************************************************


% Standard Petri-nets

Plot ISA Model WITH

%Is used when plotting a variable on an oscilloscope. The latter accepts only
%real values and that is the reason why this object is needed.

In ISA DiscIntTerminal;
Offset ISA Parameter WITH default := 0.0; END;
 x TYPE Real;

behaviour:
        x := Offset+In;
END;%Plot



Counter ISA Model WITH
%Is used to count the number of firings of a transition.

  In ISA StepUTerminal;

  x TYPE DISCRETE Real;

  OnEvent In.Trigg DO
    new(x) := x + 1.0;
  END;
END;%Counter




Printer1 ISA Model WITH
%This printer prints every alteration in the connected states. Compare with
%Printer2.
%A printer's In-terminal should be connected to a step's PrinterTerminal
%The printer must be connected to at least one InitStep (Delayed or Undelayed)
%and it must be declared before the objects whos states it is to plot. The
%number of initialsteps (delayed+undelayed) plus the number of

%SingleDelayedSteps (there is no SingleSteps adjustment to this printertype but%this$
                                ^ could easily be made as in InitStep) must be given by$
                                                                ^ NoOfInitials.The
%last print with -1 on time is the initial marking.

parameter:
        NoOfInitials ISA IntParameter;
        %The number of InitSteps + number of InitDelayedSteps in
        %the net.

variables:
        x,T TYPE DISCRETE Real;
        Count TYPE DISCRETE Integer;
terminals:
        Comm1,Comm2 ISA PrinterInConn;
event:
                Init ISAN event;
behaviour:


ONEVENT Init DO
        printlog("NaN"); printlog(" "); printlog("NaN"); printlog(" ");

        printlog("NaN"); printlog(" "); printlog("NaN"); printlog(" ");
        printlog("NaN"); printlog(" "); printlog("NaN"); printlog(" ");
        printlog("NaN"); printlog(" "); printlog("NaN"); printlog(" ");
        printlog("NaN"); printlog(" ");

        printlog("NaN"); printlog(" "); printlog("NaN"); printlog(" ");
        printlog("NaN"); printlog("\n");
END;



ONEVENT Comm1.In1.Print OR Comm1.In2.Print OR Comm1.In3.Print OR
        Comm1.In4.Print OR Comm1.In5.Print OR Comm1.In6.Print OR
        Comm1.In7.Print OR

        Comm2.In1.Print OR Comm2.In2.Print OR Comm2.In3.Print
DO
                NEW(T):=if Count<NoOfInitials then -1 else Base::Time;
                printlog(NEW(T)); printlog(" ");
                printlog(Base::Time-x); printlog(" ");
                printlog(NEW(Comm1.In1.State)); printlog(" ");
                printlog(NEW(Comm1.In2.State)); printlog(" ");
                printlog(NEW(Comm1.In3.State)); printlog(" ");
                printlog(NEW(Comm1.In4.State)); printlog(" ");
                printlog(NEW(Comm1.In5.State)); printlog(" ");
                printlog(NEW(Comm1.In6.State)); printlog(" ");
                printlog(NEW(Comm1.In7.State)); printlog(" ");

                printlog(NEW(Comm2.In1.State)); printlog(" ");
                printlog(NEW(Comm2.In2.State)); printlog(" ");
                printlog(NEW(Comm2.In3.State)); printlog("\n");


                NEW(x):=Base::Time;
                NEW(Count):=Count+1;
END;

END; %Printer1
```

```
Printer2 ISA Model WITH
%This alternative printer can be used in nets without InitDelayedStep
%and SingleDelayedStep. This printer prints the result every time the
%the OMOLA SEF-algorithm has been run.
%It has the advantage that the number of InitSteps need not to be given as
%NoOfInitials as in the printer above. Further it will give the correct initial
%marking without printing -1. If this printer is used it should be declarated
%before the objects whos state it is to print.
%A printer's In-terminal should be connected to a step's PrinterTerminal
%The net must contain at least one InitStep  which one must be connected to
%the printer.


variables:
        x,T TYPE DISCRETE Real;
        Count,Print TYPE DISCRETE Integer;
terminals:
        Comm1,Comm2,Comm3 ISA PrinterInConn;
event:
        Init ISAN event;

behaviour:


ONEVENT Init DO
        printlog("NaN"); printlog(" "); printlog("NaN"); printlog(" ");

        printlog("NaN"); printlog(" "); printlog("NaN"); printlog(" ");
        printlog("NaN"); printlog(" "); printlog("NaN"); printlog(" ");
        printlog("NaN"); printlog(" "); printlog("NaN"); printlog(" ");
        printlog("NaN"); printlog(" ");

        printlog("NaN"); printlog(" "); printlog("NaN"); printlog(" ");
        printlog("NaN"); printlog(" "); printlog("NaN"); printlog(" ");
        printlog("NaN"); printlog(" "); printlog("NaN"); printlog(" ");
        printlog("NaN"); printlog(" ");

        printlog("NaN"); printlog(" "); printlog("NaN"); printlog(" ");
        printlog("NaN"); printlog(" "); printlog("NaN"); printlog(" ");
        printlog("NaN"); printlog("\n");



END;



ONEVENT Comm2.In1.Print OR Comm2.In2.Print OR Comm2.In3.Print OR
        Comm2.In4.Print OR Comm2.In5.Print OR Comm2.In6.Print OR
        Comm2.In7.Print OR

        Comm3.In1.Print OR Comm3.In2.Print OR Comm3.In3.Print OR
        Comm3.In4.Print OR Comm3.In5.Print OR Comm3.In6.Print OR
        Comm3.In7.Print OR

        Comm1.In1.Print OR Comm1.In2.Print OR Comm1.In3.Print OR
        Comm1.In4.Print OR Comm1.In5.Print

DO      NEW(Print):=1; END;


ONEVENT Print>0 DO
                printlog(Base::Time); printlog(" ");
                printlog(Base::Time-x); printlog(" ");
                printlog(NEW(Comm2.In1.State)); printlog(" ");
                printlog(NEW(Comm2.In2.State)); printlog(" ");
                printlog(NEW(Comm2.In3.State)); printlog(" ");
                printlog(NEW(Comm2.In4.State)); printlog(" ");
                printlog(NEW(Comm2.In5.State)); printlog(" ");
                printlog(NEW(Comm2.In6.State)); printlog(" ");
                printlog(NEW(Comm2.In7.State)); printlog(" ");

                printlog(NEW(Comm3.In1.State)); printlog(" ");
                printlog(NEW(Comm3.In2.State)); printlog(" ");
                printlog(NEW(Comm3.In3.State)); printlog(" ");
                printlog(NEW(Comm3.In4.State)); printlog(" ");
                printlog(NEW(Comm3.In5.State)); printlog(" ");
                printlog(NEW(Comm3.In6.State)); printlog(" ");
                printlog(NEW(Comm3.In7.State)); printlog(" ");

                printlog(NEW(Comm1.In1.State)); printlog(" ");
                printlog(NEW(Comm1.In2.State)); printlog(" ");
                printlog(NEW(Comm1.In3.State)); printlog(" ");
                printlog(NEW(Comm1.In4.State)); printlog(" ");
                printlog(NEW(Comm1.In5.State)); printlog("\n");

                NEW(x):=Base::Time;
                NEW(Count):=Count+1;
END;

ONEVENT Print>0 DO NEW(Print):=0; END;


END; %Printer2



%The following printers are of the same type as Printer2 but prints different
%numbers of states.


Printer3 ISA Model WITH

variables:
        x,T TYPE DISCRETE Real;
        Count,Print TYPE DISCRETE Integer;
terminals:
        Comm1,Comm2,Comm3 ISA PrinterInConn;
event:
        Init ISAN event;

behaviour:


ONEVENT Init DO
        printlog("NaN"); printlog(" "); printlog("NaN"); printlog(" ");

        printlog("NaN"); printlog(" "); printlog("NaN"); printlog(" ");
        printlog("NaN"); printlog(" "); printlog("NaN"); printlog(" ");
```

```
        printlog("NaN"); printlog(" "); printlog("NaN"); printlog(" ");
        printlog("NaN"); printlog(" "); printlog("NaN"); printlog(" ");

        printlog("NaN"); printlog(" "); printlog("NaN"); printlog(" ");
        printlog("NaN"); printlog(" "); printlog("NaN"); printlog(" ");
        printlog("NaN"); printlog("\n");
END;

ONEVENT Comm2.In1.Print OR Comm2.In2.Print OR Comm2.In3.Print OR
        Comm2.In4.Print OR

        Comm3.In1.Print OR Comm3.In2.Print OR Comm3.In3.Print OR
        Comm3.In4.Print OR

        Comm1.In1.Print OR Comm1.In2.Print OR Comm1.In3.Print OR
        Comm1.In4.Print OR Comm1.In5.Print

DO      NEW(Print):=1; END;


ONEVENT Print>0 DO
                printlog(Base::Time); printlog(" ");
                printlog(Base::Time-x); printlog(" ");
                printlog(NEW(Comm2.In1.State)); printlog(" ");
                printlog(NEW(Comm2.In2.State)); printlog(" ");
                printlog(NEW(Comm2.In3.State)); printlog(" ");
                printlog(NEW(Comm2.In4.State)); printlog(" ");

                printlog(NEW(Comm3.In1.State)); printlog(" ");
                printlog(NEW(Comm3.In2.State)); printlog(" ");
                printlog(NEW(Comm3.In3.State)); printlog(" ");
                printlog(NEW(Comm3.In4.State)); printlog(" ");

                printlog(NEW(Comm1.In1.State)); printlog(" ");
                printlog(NEW(Comm1.In2.State)); printlog(" ");
                printlog(NEW(Comm1.In3.State)); printlog(" ");
                printlog(NEW(Comm1.In4.State)); printlog(" ");
                printlog(NEW(Comm1.In5.State)); printlog("\n");

                NEW(x):=Base::Time;
                NEW(Count):=Count+1;
END;

ONEVENT Print>0 DO NEW(Print):=0; END;


END; %Printer3



Printer4 ISA Model WITH

variables:
        x,T TYPE DISCRETE Real;
        Count,Print TYPE DISCRETE Integer;
Terminals:
        Comm1,Comm2,Comm3 ISA PrinterInConn;


event:
        Init ISAN event;
```

```
behaviour:


ONEVENT Init DO
        printlog("NaN"); printlog(" "); printlog("NaN"); printlog(" ");

        printlog("NaN"); printlog(" "); printlog("NaN"); printlog(" ");
        printlog("NaN"); printlog(" "); printlog("NaN"); printlog(" ");


        printlog("NaN"); printlog(" "); printlog("NaN"); printlog(" ");
        printlog("NaN"); printlog(" "); printlog("NaN"); printlog(" ");
        printlog("NaN"); printlog("\n");
END;



ONEVENT Comm2.In1.Print OR Comm2.In2.Print OR Comm2.In3.Print OR
        Comm2.In4.Print OR


        Comm1.In1.Print OR Comm1.In2.Print OR Comm1.In3.Print OR
        Comm1.In4.Print OR Comm1.In5.Print

DO      NEW(Print):=1; END;

ONEVENT Print>0 DO
                printlog(Base::Time); printlog(" ");
                printlog(Base::Time-x); printlog(" ");
                printlog(NEW(Comm2.In1.State)); printlog(" ");
                printlog(NEW(Comm2.In2.State)); printlog(" ");
                printlog(NEW(Comm2.In3.State)); printlog(" ");
                printlog(NEW(Comm2.In4.State)); printlog(" ");


                printlog(NEW(Comm1.In1.State)); printlog(" ");
                printlog(NEW(Comm1.In2.State)); printlog(" ");
                printlog(NEW(Comm1.In3.State)); printlog(" ");
                printlog(NEW(Comm1.In4.State)); printlog(" ");
                printlog(NEW(Comm1.In5.State)); printlog("\n");

                NEW(x):=Base::Time;
                NEW(Count):=Count+1;
END;

ONEVENT Print>0 DO NEW(Print):=0; END;


END; %Printer4
```

Appendix B. Funcions Generating an Intensitymatrix

```
function [IntensMatr,StateMatr,Bounded,Ok]= ...
pdsmarkint(Dplus,Dminus,Intensity,InitMarking,NoOfGroups,PrioList)

%function [IntensMatr,StateMatr,Bounded,Ok]= ...
%pdsmarkint(Dplus,Dminus,Intensity,InitMarking,NoOfGroups,PrioList)
%
%Generates a transition- and a statematrix for a discrete Markovian
%process in continouos time described by a Petri-net with delayed steps and
%undelayed transitions. Dplus and Dminus are (NoOfTransitions*NoOfSteps)
%matrices and Intensity is a (1*NoOfSteps) rowmatrix which contains the
%available-intensities of the Steps. InitMarking is a (1*NoOfSteps) rowvector
% containing the initial marking of the net. PrioList is a 1*NoOfTransitions
%rowmatrix which contains the indexes of the transitions ordered in the
%same order as they are declared in the OMOLA-code. This is used to resolve
%possible conflicts in the same way that they are resolved by the OMOLA SEF-
%algorithm. NoOfGroups is a 1*NoOfSteps rowvector that contains the number of
%groups that the tokens in a step can be divided in. Typically a group of two
%tokens is created when a transition with  weight=2 on its output arc fires.
%
%If the Petri-net with only delayed-steps is to be modelled as a Markovian
%process there must be no concurrency in the net. Further restrictions must be
%made on the weight of the arcs connected to a delayed step. One sufficient
%condition is that the largest weight on the outgoing (from a delayed Step)
%arcs must be <= the smallest weight on the ingoing arcs. Another sufficient
%condition is that no outgoing (from the step) arc with weight>1 is allowed
%(ie Consumei=1 or 0 are the only possibilities at the following transitions).
%Ok will be 1 if the net does not contain concurrency and has no transitions
%with an input arc with weight>1. This implementation can handle the first
%sufficient condition provided that the input arcs (to the delayed step) all
%have the same weight and if this is wanted the Ok check below should be
%removed.
%
%This implementation can only handle steps which contain groups of token with
%the same size. This implies that all the ingoing arcs to a step must have the
%same weight. The initial state of a delayed step must be a multiple of a
%group with the same size as the weight on the input-arcs to the delayed step,
%provided that there is an input-arc. The multiple should be indicated in
%NoOfGroups.
%
%If the net is unbounded the algorithm will be interrupted by Limit which
%gives the maximum numbers of recursive calls.



Limit=100;
Count=1;
Ok=~any(1.-sum(Dminus'));
if Ok
        D=Dplus-Dminus;
        [m,n]=size(D);
        [ResultMatr,StateMatr,NoOfCalls]= ...
        pdsgenerate(D,Dplus,Dminus,Intensity,InitMarking,NoOfGroups,[0],...
        1,InitMarking,Limit,Count,m,PrioList);
        if NoOfCalls<=Limit
                Bounded=1;
                [m,n]=size(ResultMatr);
                eyedum=eye(m);
                sumdum=sum(ResultMatr')';
                for k=1:m
                        DiagVal(k,:)=sumdum(k,1).*eyedum(k,:);
                end
                IntensMatr=ResultMatr-DiagVal;
        else
                Bounded=0;
        end
end
```

```
function [IntensMatr,StateMatr,NoOfCalls]= ...
pdsgenerate(D,Dplus,Dminus,Intensity,Marking,NoOfGroups,OldIntensMatr,  ...
        OldStateIndex,OldStateMatr,Limit,Count,NoOfTrans,PrioList)

%function [IntensMatr,StateMatr,NoOfCalls]= ...
%pdsgenerate(D,Dplus,Dminus,Intensity,Marking,NoOfGroups,OldIntensMatr, ..
%OldStateIndex,OldStateMatr,Limit,Count,NoOfTrans,PrioList)
%
%Main algorithm for generating the markovian state-space in Petri-nets where
%all steps but no transitions  are delayed. Intensity, marking
%are rowvectors. Limit and Count are integers and the other parameters are
%matrices.Limit gives the upper limit for the number of recursive calls and
%limits the search for different states. No try to detect Concurrency is
%made. See comments in pdsmarkint.

NoOfCalls=Count;
if Count<=Limit
  OldSM=OldStateMatr;
  OldIM=OldIntensMatr;
  StateMatr=OldStateMatr;
  IntensMatr=OldIntensMatr;
  for k=1:NoOfTrans;
        [Fire,NewMarking,NewNoOfGroups,Steps]=checkfirepds..
        (D,Dplus,Dminus,Marking,NoOfGroups,k,PrioList);
        if Fire
                [Found,Index]=rowinmatr(NewMarking,OldSM);
                if Found
                  IntensMatr=OldIM;
                  IntensMatr(OldStateIndex,Index)= ...
                  OldIM(OldStateIndex,Index)+ ...
                  Intensity(1,Steps)- ...
                  Intensity(1,Steps)*(1-NoOfGroups(1,Steps));
                  %The previous row handles the marking dependent case.
                  StateMatr=OldSM;
                  OldIM=IntensMatr;
                else
                  [m,n]=size(OldIM);
                  x=zeros(m,1);
                  x(OldStateIndex,1)= ...
                  Intensity(1,Steps)- ...
                  Intensity(1,Steps)*(1-NoOfGroups(1,Steps));
                  %The previous row handles the marking dependent case.
                  y=zeros(1,n+1);
                  IntensMatr=[OldIM x;y];
                  StateMatr=[OldSM;NewMarking];
                  [IntensMatr,StateMatr,NoOfCalls]= ...
                  pdsgenerate(D,Dplus,Dminus,Intensity,NewMarking,..
                  NewNoOfGroups, IntensMatr,n+1,StateMatr,Limit,Count+1,...
                  NoOfTrans,PrioList);
                  OldSM=StateMatr;
                  OldIM=IntensMatr;
                end
        end %If Fire
  end %for
end
```

```
function [Fire,NewMarking,NewNoOfGroups,Step] = checkfirepds..
         (D,Dplus,Dminus,Marking,NoOfGroups,Trans,NoOfTrans,PrioList)

%function [Fire,NewMarking,NewNoOfGroups,Step,MDepend] = checkfirepds..
%         (D,Dplus,Dminus,Marking,NoOfGroups,Trans,NoOfTrans,PrioList)
%
%Checks if the transition with index trans(>0) is fireable and fires it.
%The new marking after the firing is put in NewMarking. The index of the Step
%from which token are removed are put in Step. Marking,NewMarking and Step
%are rowvectors. If the transition is in conflict with a transition with
%higher priority the firing is cancelled.


[NoOfTrans NoOfSteps=size(D);
x=zeros(1,NoOfTrans);
x(1,Trans)=1;
Consume=x*Dminus;
Step=(Consume>0);
if all(Marking>=Consume);
        k=1;
        Prohibited=0;
        MyPrio=find(Trans==PrioList);
        while k<=(MyPrio-1) & ~Prohibited
          y=zeros(1,NoOfTrans);
          y(1,PrioList(k))=1;
          Consumey=y*Dminus;
          if Consumey(1,Step)>0
                        Prohibited=1; %Possible Conflict with a transition
          end                         %with a higher priority
          k=k+1;
        end
        if ~Prohibited
                Fire=1;
                NewNoOfGroups=NoOfGroups;
                NewMarking=Marking;
                k=MyPrio;
                TokenLeftInGroup=NewMarking(1,Step)/NoOfGroups(1,Step);
                NewNoOfGroups(1,Step)=NewNoOfGroups(1,Step)-1;
                while TokenLeftInGroup>0
                        y=zeros(1,NoOfTrans);
                        y(1,PrioList(k))=1;
                        Consumey=y*Dminus;
                        if Consumey(1,Step)>0 %Resolving potential conflict
                          TokenLeftInGroup=TokenLeftInGroup-1;
                          NewMarking=y*D+NewMarking;
                          Stepy=(y*Dplus>0);
                          NewNoOfGroups(1,Stepy)=1.+NewNoOfGroups(1,Stepy);
                        end
                        k=(k+1-(NoOfTrans-MyPrio+1)*fix(k/NoOfTrans));
                end
        else
                Fire=0;
        end
else
        Fire=0;
end;
```

```
function [found,row]=rowinmatr(vector,matrix)

%Investigates if the rowvector v is a row in matrix m. If that is the
%case found is 1  and row gives the first rownumber in the matrix which
%is identical with vector. If Vector not is found, found is 0.

ready=0;
k=1;
[m,n]=size(matrix);
if n==0
        found=0;
else
        while ~ready & k<=m
                ready=all(vector==matrix(k,:));
                k=k+1;
        end
        found=ready;
        if ready
                row=k-1;
        else
                row=NaN;
        end
end
```

```
function [IntensMatr,StabStateMatr,InstabStateMatr,Bounded,Conc]= ...
mixmarkint(Dp,Dm,Intensity,InitMarking,NoOfGroups,PrioDList,PrioUList)

%function [IntensMatr,StabStateMatr,InstabStateMatr,Bounded,Ok]= ...
%mixmarkint(Dp,Dm,Intensity,InitMarking,NoOfGroups,PrioDList,PrioUList)
%
%Generates a transition- and a statematrix for a discrete Markovian
%process in continouos time described by a Petri-net with delayed and
%undelayed steps combined with undelayed transitions. Dplus and Dminus are
%(NoOfTransitions*NoOfSteps) matrices and Intensity is a (1*NoOfSteps)
%rowmatrix which contains the available-intensities of the Steps. InitMarking
%is a (1*NoOfSteps) rowvector containing the initial marking of the net.
%PrioDList is a 1*NoOfDelayedTransition rowmatrix which contains the indexes of
%the delayed transitions ordered in the same way as they are declared in the
%OMOLA-code. A Delayed Transition is defined as a transition which has (at
%least) one ingoing arc which origins from a delayed step. PrioULIst is a
%1*NoOfUndelayedTransitions rowvector which contains the indexes of the
%undelayed transitions ordered in the same way as they are declared in the
%OMOLA-code. An undelayed transition is defined as a transition which has no
%ingoing arcs which origins from a delayed step. Conflicts are resolved so
%that the result is the same as when the Petri-net simulation is used.
%NoOfGroups is a 1*NoOfSteps rowvector that contains the number of groups that
%the tokens in a step can be divided in. Typically a group of twotokens is
%created when a transition with  weight=2 on its output arc fires.
%
%If the Petri-net with a mix of delayed and undelayed steps is to be modelled
%as a Markovian process there must be no concurrency involving more than one
%delayed step. Concurrency involving several undelayed steps are allowed as
%well as concurrency involving one delayed and several undelayed steps. Further
%restrictions must be made on the weight of the arcs connected to a dealyed
%step. One sufficient condition is that the largest weight on the outgoing
%(from a delayed Step) arcs must be <= the smallest weight on the ingoing arcs
%Another sufficient condition is that no outgoing (from the step) arc with
%weight>1 is allowed (ie Consumei=1 or 0 are the only possibilities at the
%following transitions). Ok will be 1 if the net does not contain concurrency
%between delayed steps. No checks of weights on arcs are made. This
%implementation can handle the first sufficient condition provided that the
%input arcs (to the delayed step) all have the same weight.
%
%This implementation can only handle steps which contain groups of token with
%the same size. This implies that all the ingoing arcs to a step must have the
%same weight. The initial state of a delayed step must be a multiple of a
%group with the same size as the weight on the input-arcs to the delayed step,
%provided that  there is an input-arc. The multiple should be indicated in
%NoOfGroups.
%
%If the net is unbounded the algorithm will be interrupted by Limit which
%gives the maximum numbers of recursive calls.



Limit=1000;
Count=1;
Conc=0;
%if size(PrioDList)>0 %Concurrencycheck. Works only for pure delayed nets
%       Conc=any(1.-sum(Dm(PrioDList,:)'));
%end
if ~Conc
        D=Dp-Dm;
        [m n]=size(D);
        [md nd]=size(PrioDList);
        [mu nu]=size(PrioUList);
        [InstabStateMatr,Marking,NewNoOfGroups,NoOfCalls]=pudgenerate...
        (D,Dp,Dm,InitMarking,NoOfGroups,[],Limit,Count,m,nd,PrioUList);

        [ResultMatr,StabStateMatr,InstabStateMatr,NoOfCalls,Conc]=mixgenerate..
        (D,Dp,Dm,Intensity,Marking,NewNoOfGroups,[0],1,Marking,...
        InstabStateMatr,Limit,Count,m,nd,PrioDList,PrioUList);
        if NoOfCalls<=Limit & ~Conc
                Bounded=1;
                [m,n]=size(ResultMatr);
                eyedum=eye(m);
                sumdum=sum(ResultMatr')';
                for k=1:m
                        DiagVal(k,:)=sumdum(k,1).*eyedum(k,:);
                end
                IntensMatr=ResultMatr-DiagVal;
        else
                Bounded=0;
        end
end
```

```
function [IntensMatr,StabStateMatr,InstabStateMatr,NoOfCalls,Conc]= ...
mixgenerate(D,Dp,Dm,Intensity,Marking,NoOfGroups,OldIntensMatr,...
OldStateIndex,OldStabStateMatr,OldInstabStateMatr,Limit,Count,..
NoOfTrans,NoOfDTrans,PrioDList,PrioUList)

%function [IntensMatr,StabStateMatr,InstabStateMatr,NoOfCalls,Conc]= ...
%mixgenerate(D,Dp,Dm,Intensity,Marking,NoOfGroups,OldIntensMatr,...
%OldStateIndex,OldStabStateMatr,OldInstabStateMatr,Limit,Count,..
%NoOfTrans,NoOfDTrans,PrioDList,PrioUList)
%
%
%Main algorithm for generating the markovian state-space in Petri-nets where
%some steps but no transitions  are delayed. Intensity, marking
%are rowvectors. Limit and Count are integers and the other parameters are
%matrices.Limit gives the upper limit for the number of recursive calls and
%limits the search for different states. Concurrency between delayed steps
%is detected. See also mixmarkint.

NoOfCalls=Count;
if (Count<=Limit)
  Conc=0;
  OldSM=OldStabStateMatr;
  OldIM=OldIntensMatr;
  StabStateMatr=OldStabStateMatr;
  IntensMatr=OldIntensMatr;
  InstabStateMatr=OldInstabStateMatr;
  for k=1:NoOfDTrans;
    if ~Conc
      [Fire,NewMarking,NewNoOfGroups,Steps]=checkfirepdst..
      (D,Dp,Dm,Marking,NoOfGroups,PrioDList(k),NoOfTrans,NoOfDTrans,PrioDList);
       if Fire
            [InstabStateMatr,NewMarking,NewNoOfGroups,NoOfCalls]=...
            pudgenerate(D,Dp,Dm,NewMarking,NewNoOfGroups,OldInstabStateMatr,..
            Limit,Count+1,NoOfTrans,NoOfDTrans,PrioUList);
            [Found,Index]=rowinmatr(NewMarking,OldSM);
            if Found
               Conc=(sum(Intensity(1,Steps)>0)>1); %Concurrency check
               if ~Conc
                  IntensMatr=OldIM;
                  [mi,ni]=size(Intensity(1,Steps));
                  IntensMatr(OldStateIndex,Index)= ...
                  OldIM(OldStateIndex,Index)+Intensity(1,Steps)*ones(mi,ni)'+..
                  Intensity(1,Steps)*(NoOfGroups(1,Steps)-ones(mi,ni))';
                  %The previous row handles the marking dependent case.
                  StabStateMatr=OldSM;
                  OldIM=IntensMatr;
               end
            else
               Conc=(sum(Intensity(1,Steps)>0)>1);  %Concurrency check
               if ~Conc
                  [m,n]=size(OldIM);
                  x=zeros(m,1);
                  [mi,ni]=size(Intensity(1,Steps));
                  x(OldStateIndex,1)=...
                  Intensity(1,Steps)*ones(mi,ni)'+...
                  Intensity(1,Steps)*(NoOfGroups(1,Steps)-ones(mi,ni))';
                  %The previous row handles the marking dependent case.
                  y=zeros(1,n+1);
                  IntensMatr=[OldIM x;y];
                  StabStateMatr=[OldSM;NewMarking];
                  [IntensMatr,StabStateMatr,InstabStateMatr,NoOfCalls,Conc]= ..
                  mixgenerate(D,Dp,Dm,Intensity,NewMarking,NewNoOfGroups,...
                  IntensMatr,n+1,StabStateMatr,InstabStateMatr,..
                  Limit,NoOfCalls+1,NoOfTrans,NoOfDTrans,PrioDList,..
                  PrioUList);
                  OldSM=StabStateMatr;
                  OldIM=IntensMatr;
               end
            end
       end %If Fire
    end %if ~Conc
  end %for
end
```

```
function [InstabStateMatr,StableMark,NoOfGroups,NoOfCalls]=pudgenerate...
(D,Dp,Dm,Marking,OldNoOfGroups,OldInstabStateMatr,Limit,Count,NoOfTrans,..
NoOfDTrans,PrioUList)

%function [InstabStateMatr,StableMark,NoOfGroups,NoOfCalls]=pudgenerate...
%(D,Dp,Dm,Marking,OldNoOfGroups,OldInstabStateMatr,Limit,Count,NoOfTrans,..
%NoOfDTrans,PrioUList)
%
%Main algorithm for generating the markovian state-space in Petri-nets whith no
%delayed objects. Conflicts are resolved in the same way as in the OMOLA CEF-
%algorithm. See also comments in  mixmarkint.


NoOfCalls=Count;
if Count<=Limit
        StableMark=Marking;
        Oldsm=Marking;
        NoOfGroups=OldNoOfGroups;
        InstabStateMatr=OldInstabStateMatr;
        [Fire,StableMark,NoOfGroups]=checkfirepud..
        (D,Dp,Dm,Marking,NoOfGroups,NoOfTrans,NoOfDTrans,PrioUList);
        while Fire
                [Found,Index]=rowinmatr(Oldsm,InstabStateMatr);
                if ~Found
                        InstabStateMatr=[InstabStateMatr;Oldsm];
                end
                Oldsm=StableMark;
                [Fire,StableMark,NoOfGroups]=checkfirepud..
               (D,Dp,Dm,StableMark,NoOfGroups,NoOfTrans,NoOfDTrans,PrioUList);
        end
end
```

```
function [Fire,NewMarking,NewNoOfGroups,Step] = checkfirepdst..
(D,Dplus,Dminus,Marking,NoOfGroups,Trans,NoOfTrans,NoOfDTrans,PrioDList)

%function [Fire,NewMarking,NewNoOfGroups,Step] = checkfirepdst..
%(D,Dplus,Dminus,Marking,NoOfGroups,Trans,NoOfTrans,NoOfDTrans,PrioDList)
%
%Checks if the transition with index trans(>0) is fireable and fires it.
%The new marking after the firing is put in NewMarking. The index of the Step
%from which token are removed are put in Step. Marking,NewMarking and Step
%are rowvectors.


x=zeros(1,NoOfTrans);
x(1,Trans)=1;
Consume=x*Dminus;
Step=(Consume>0);
if all(Marking>=Consume);
        k=1;
        Prohibited=0;
        MyPrio=find(Trans==PrioDList);
        while k<=(MyPrio-1) & ~Prohibited
          y=zeros(1,NoOfTrans);
          y(1,PrioDList(k))=1;
          Consumey=y*Dminus;
          if Consumey(1,Step)>0
                        Prohibited=1;  %Possible Conflict with a transition
          end                          %with a higher priority
          k=k+1;
        end
        if ~Prohibited
                Fire=1;
                NewNoOfGroups=NoOfGroups;
                NewMarking=Marking;
                k=MyPrio;
                TokenLeftInGroup=round(NewMarking(1,Step)/NoOfGroups(1,Step));
                %Notice! The result of division must be an integer
                %The round has been added to ensure that the reference
                %index in the matrix is an integer.(An error was obtained
                %before that.)
                NewNoOfGroups(1,Step)=NewNoOfGroups(1,Step)-1;
                while TokenLeftInGroup>0
                        y=zeros(1,NoOfTrans);
                        y(1,PrioDList(k))=1;
                        Consumey=y*Dminus;
                        if Consumey(1,Step)>0 %Resolving potential conflict
                          TokenLeftInGroup=TokenLeftInGroup-1;
                          NewMarking=y*D+NewMarking;
                          Stepy=(y*Dplus>0);
                          NewNoOfGroups(1,Stepy)=1.+NewNoOfGroups(1,Stepy);
                        end
                        k=(k+1-(NoOfTrans-MyPrio+1)*fix(k/NoOfTrans));
                end
        else
                Fire=0;
        end
else
        Fire=0;
end;
```

```
function [Fire,NewMarking,NewNoOfGroups,Steps]=checkfirepud...
(D,Dp,Dm,Marking,NoOfGroups,NoOfTrans,NoOfDTrans,PrioUList)

%function [Fire,NewMarking,NewNoOfGroups,Steps]=checkfirepud...
%(D,Dp,Dm,Marking,NoOfGroups,NoOfTrans,NoOfDTrans,PrioUList)
%
%This algorithm works exactly the same as the OMOLA CEF-algorithm.

Enable=[];
NoOfUTrans=NoOfTrans-NoOfDTrans;
Fire=0;
NewMarking=Marking;
NewNoOfGroups=NoOfGroups;
for k=1:NoOfUTrans
        x=zeros(1,NoOfTrans);
        x(1,PrioUList(k))=1;
        if all(Marking>=x*Dm)
                Enable=[Enable PrioUList(k)];
        end
end
[me,ne]=size(Enable);
if ne>0
        for k=1:ne
                x=zeros(1,NoOfTrans);
                x(1,Enable(k))=1;
                if all(NewMarking>=x*Dm)
                        NewMarking=NewMarking+x*D;
                        Steps=(x*Dm>0);
                        NewNoOfGroups(1,Steps)=-(1.-NewNoOfGroups(1,Steps));
                        Steps=(x*Dp>0);
                        NewNoOfGroups(1,Steps)=1.+NewNoOfGroups(1,Steps);
                end
        end
        Fire=1;
end
```

```
function [IntensMatr,StateMatr,Bounded]= ...
pdtmarkint(Dplus,Dminus,Intensity,InitMarking,MDepend)

%function [IntensMatr,StateMatr,Bounded]= ...
%pdtmarkint(Dplus,Dminus,Intensity,InitMarking,MDepend)
%
%Generates a transition- and a statematrix for a discrete Markovian process in
%continouos time described by a Petri-net with delayed transitions and
%undelayed steps. Dplus and Dminus are (NoOfTransitions*NoOfSteps) matrices
%and Intensity is a (1*NoOfTransitions) rowmatrix which contains the firing-
%intensities of the transitions. InitMarking is a (1*NoOfSteps) rowvector
%containing the initial marking of the net. MDepend is a
%(NoOfTransitions*NoOfSteps) matrix and is usedfor marking dependent
%intensities. In a row i the different columns j containsthe proportionality
%constant for the step j so that lamda(transition ti)=
%[sumj=1:NoOfSteps(MDepend(i,j)*NoOfTokensInStep(j))]*Intensity(i))
%No check if the net is bounded is made. If the net is unbounded the algorithm
%will be interrupted by Limit which gives the maximum numbers of recursive
%calls.

Limit=100;
Count=1;
D=Dplus-Dminus;
[m,n]=size(D);
[ResultMatr,StateMatr,NoOfCalls]= ...
pdtgenerate(D,Dminus,Intensity,InitMarking,MDepend,[0],1,InitMarking, ...
            Limit,Count,m);
if NoOfCalls<=Limit
        Bounded=1;
        [m,n]=size(ResultMatr);
        eyedum=eye(m);
        sumdum=sum(ResultMatr')';
        for k=1:m
                DiagVal(k,:)=sumdum(k,1).*eyedum(k,:);
        end
        IntensMatr=ResultMatr-DiagVal;
else
        Bounded=0;
end
```

```
function [IntensMatr,StateMatr,NoOfCalls]= ...
pdtgenerate(D,Dminus,Intensity,Marking,MDepend,OldIntensMatr,  ...
        OldStateIndex,OldStateMatr,Limit,Count,NoOfTrans)

%function [IntensMatr,StateMatr,NoOfCalls]= ...
%pdtgenerate(D,Dminus,Intensity,Marking,MDepend,OldIntensMatr,  ...
%        OldStateIndex,OldStateMatr,Limit,Count,NoOfTrans)
%
%Main algorithm for generating the markovian state-space and transition matrix
%in Petri-nets where all transitions but no steps are delayed. Intensity and
%Marking are rowvectors. Limit and Count are integers and the other parameters
%are matrices.Limit gives the upper limit for the number of recursive calls and
%limits the search for different states. See comments in pdtmarkint.


NoOfCalls=Count;
if Count<=Limit
  Change=0;
  OldSM=OldStateMatr;
  OldIM=OldIntensMatr;
  for k=1:NoOfTrans
        [Fire,NewMarking,Steps]=checkfiresimple(D,Dminus,Marking,k);
        if Fire
                Change=1;
                [Found,Index]=rowinmatr(NewMarking,OldSM);
                if Found
                  IntensMatr=OldIM;
                  IntensMatr(OldStateIndex,Index)= ...
                  OldIM(OldStateIndex,Index)+ ...
                  Intensity(1,k)- ...
                  Intensity(1,k)*((1.-Marking(1,Steps))*MDepend(k,Steps)');
                  %The previous row handles the marking dependent case.
                  StateMatr=OldSM;
                  OldIM=IntensMatr;
                else
                  [m,n]=size(OldIM);
                  x=zeros(m,1);
                  x(OldStateIndex,1)= ...
                    Intensity(1,k)- ...
                    Intensity(1,k)*((1.-Marking(1,Steps))*MDepend(k,Steps)');
                  %The previous row handles the marking dependent case.
                  y=zeros(1,n+1);
                  IntensMatr=[OldIM x;y];
                  StateMatr=[OldSM;NewMarking];
                  [IntensMatr,StateMatr,NoOfCalls]= ...
                  pdtgenerate(D,Dminus,Intensity,NewMarking,MDepend, ...
                  IntensMatr,n+1,StateMatr,Limit,Count+1,NoOfTrans);
                  OldSM=StateMatr;
                  OldIM=IntensMatr;
                end
        end
  end %for
  if ~Change
        StateMatr=OldStateMatr;
        IntensMatr=OldIntensMatr;
  end
end
```

```
function [Fire,NewMarking,Steps] = checkfiresimple (D,Dminus,Marking,Trans)

%function [Fire,NewMarking,Steps] = checkfire (D,Dminus,Marking,Trans)
%
%Checks if the transition with index trans(>0) is fireable and fires it.
%The new marking after the firing is put in NewMarking. The index of the steps
%from which token are removed are put in Steps. Marking,NewMarking and Steps
%are rowvectors.

[NoOfTrans,NoOfSteps]=size(Dminus);
x=zeros(1,NoOfTrans);
x(1,Trans)=1;
if all(Marking>=x*Dminus)
        Fire=1;
        NewMarking=x*D+Marking;
        Steps=x*Dminus>0;
else
        Fire=0;
end;
```

Appendix C. Statistical functions

```
function [StateMatr,InstabStateMatr,ProbVect]=...
                                        asymdistr(SimTime,InitDelay);

%function [StateMatr,InstabStateMatr,ProbVect]=...
%                                       asymdistr(InitDelay);
%
%Calculates the assymptotical probability distribution from the data in
%the OMOLA-file eventlog.t. This data has been generated using print in
%library Petri and simulating the system one time.
%No checks are made that a asymptotical distribution really excists!
%InitDelayed should be 1 if any of the steps that has been connected to
%the Petri-net printer is a InitDelayedStep otherwise 0.

load eventlog.t
StateMatr=[];
InstabStateMatr=[];
ProbVect=[];
[m,n]=size(eventlog);

%Removes redundant columns
k=3;
Found=0;
while (k<=n) & ~Found
        Found=~any(eventlog(2:m,k));
        k=k+1;
end
if Found
        n=k-2;
else
        n=k-1;
end

SampleMatr=eventlog(1:m,1:n);

%Removes the instable and uninteresting initial states in one or more
%InitDelayedStep. Notice that there can be no interesting instable states
%caused by firing transitions at time 0 when the initial step is delayed.

if InitDelay>0
        DeleteRow=1;
        while DeleteRow
                DeleteRow=(SampleMatr(3,1)==-1);
                if DeleteRow
                        SampleMatr(2,:)=[];
                        m=m-1;
                end
        end
end


StateMatr=[StateMatr; SampleMatr(2,3:n)];
ProbVect=[ProbVect; SampleMatr(3,2)];
for i=3:(m-1)
        if SampleMatr(i+1,2)~=0.0
          [found,row]=rowinmatr(SampleMatr(i,3:n),StateMatr);
          if ~found
             StateMatr=[StateMatr; SampleMatr(i,3:n)];
             ProbVect=[ProbVect; SampleMatr(i+1,2)];
          else
           ProbVect(row,1)=ProbVect(row,1)+SampleMatr(i+1,2);
          end
        else
          [found,row]=rowinmatr(SampleMatr(i,3:n),InstabStateMatr);
          if ~found
             InstabStateMatr=[InstabStateMatr; SampleMatr(i,3:n)];
          end
        end
end
%LastRow. Represents always a stable state.
          [found,row]=rowinmatr(SampleMatr(m,3:n),StateMatr);
          if ~found
             StateMatr=[StateMatr; SampleMatr(m,3:n)];
             ProbVect=[ProbVect; (SimTime-SampleMatr(m,1))];
          else
           ProbVect(row,1)=ProbVect(row,1)+(SimTime-SampleMatr(m,1));
          end

ProbVect=sum(ProbVect).\ProbVect;
```

```
function [StateMatr,InstabStateMatr,ProbMatr]=...
                                        ensemblemv(SimTime,TimeInt,InitDelay)

%function [StateMatr,InstabStateMatr,ProbMatr]=...
%                                       ensemblemv(SimTime,TimeInt,InitDelay)
%
%Computes the state-probability of a number of ensembles in SampleMatr.
%The probabilities are computed at the times  0+k*TimeInt where
%k=0..round(SimTime/TimeInt). Only the states that are detected at the
%specific times above are included in StateMatr. All instabile states are put
%in InstabStateMatr. InitDelayed should be 1 if any of the steps that has been
%connected to the Petri-net printer is a InitDelayedStep otherwise 0.

load eventlog.t
[m,n]=size(eventlog)



%Removes redundant columns

FoundLast=0;
FirstRows = find(isnan(eventlog(1:m,1)));
FirstRows=[FirstRows; m];
[mfr,nfr]=size(FirstRows);
Column=3;
while (Column<=n) & ~FoundLast
        i=1;
        NextCol=0;
        while ~NextCol & (i<mfr)
             NextCol=any(eventlog((FirstRows(i)+1):(FirstRows(i+1)-1),Column));
                i=i+1;
        end
        if ~NextCol
                FoundLast=1;
        end;
        Column=Column+1;
end
if FoundLast
        n=Column-2;
else
        n=Column-1
end
SampleMatr=eventlog(1:m,1:n);
%Copy=eventlog(1:m,1:n);


%Removes instable states caused by an InitDelayedStep.

if InitDelay>0
  for k=1:(mfr-1)
    DeleteRow=1;
    while DeleteRow
        DeleteRow=(SampleMatr(FirstRows(k)+2,1)==-1);
        if DeleteRow
                SampleMatr(FirstRows(k)+1,:)=[];
                m=m-1
        end
     end
   end
end
```

```
%Removes unstable states.
%Removes lines with the same time (same value in the first column) only the
%last line in a group with the same times is left in SampleMatr. The removed
%rows are put in InstabStateMatr

InstabStateMatr=[];
k=1;
while k<m
        if SampleMatr(k,1)==SampleMatr(k+1,1)
                [found,row]=rowinmatr(SampleMatr(k,3:n),InstabStateMatr);
                if ~found
                        SampleMatr(k,3:n);
                        InstabStateMatr=[InstabStateMatr; SampleMatr(k,3:n)];
                end
                SampleMatr(k,:)=[];
                m=m-1;
        else
                k=k+1;
        end
end



%Main algorithm

StateMatr(1,:)=SampleMatr(2,3:n);
NoOfStates=1;
ProbMatr(1,1)=1;
FirstRows=[find(isnan(SampleMatr(1:m,1)));(m+1)];
[NoOfSim,dummy]=size(FirstRows);
NoOfSim=NoOfSim-1;
j=2.*ones(1,NoOfSim);
for k=1:round(SimTime/TimeInt)
        ProbMatr=[ProbMatr zeros(NoOfStates,1)];
        for i=1:NoOfSim
          ready=0;
          while ((FirstRows(i)+j(1,i))<FirstRows(i+1)) & ~ready
                ready=~(SampleMatr(FirstRows(i)+j(1,i),1)<k*TimeInt);
                j(1,i)=j(1,i)+1;
          end
          if ready
                j(1,i)=j(1,i)-1;
          end
        [found,row]=rowinmatr(SampleMatr(FirstRows(i)+j(1,i)-1,3:n),StateMatr);
          if ~found
                StateMatr=[StateMatr; SampleMatr(FirstRows(i)+j(1,i)-1,3:n)];
                ProbMatr=[ProbMatr; zeros(1,k) 1]; %nollor fran index 0..t-1
                NoOfStates=NoOfStates+1;
          else
                ProbMatr(row,k+1)=ProbMatr(row,k+1)+1;
          end
        end %for
        k

end
[mp,np]=size(ProbMatr);
s=sum(ProbMatr)
for k=1:np
        ProbMatr(:,k)=s(1,k).\ProbMatr(:,k);
end;
```

```
function [StateMatr,ProbMatr]=statean(Steps,SimTime,TimeInt,InitDelay)

%function [StateMatr,ProbMatr]=statean(SimTime,TimeInt,InitDelay)
%
%Identifies the different states of the steps with index in Steps.
%Steps is a row vector with the indexes of the steps that are to be analysed.
%The probability of the analysed steps being in a state is calculated at the
%times t=k*Timeint where k=0,1.. and k*TimeInt<=SimTime. Only the states that
%are detected at the specific times above are included in StateMatr.
%StateMatr is a (max(NoOfStates(AnalysedStepi)))*NoOfAnalysedSteps matrix.
%The analysed steps which have fewer states than the analysed step with
%most states will have -1 on the places not used.
%A more thoroughly tested but slower version is found in stepan2.m



load eventlog.t;
[m,n]=size(eventlog)
SampleMatr=eventlog;
InternSteps=2.+Steps;


%Main algorithm
[mint,nint]=size(InternSteps);
FirstRows=[find(isnan(SampleMatr(1:m,1)));(m+1)];

%Removes instable states caused by an InitDelayedStep.
%The last row with -1 is the true initial state.
if InitDelay>0
  for k=1:(mfr-1)
    DeleteRow=1;
    while DeleteRow
        DeleteRow=(SampleMatr(FirstRows(k)+2,1)==-1);
        if DeleteRow
                SampleMatr(FirstRows(k)+1,:)=[];
                m=m-1;
        end
     end
   end
end


[NoOfSim,dummy]=size(FirstRows);
NoOfSim=NoOfSim-1;
StateMatr=[];
msm=0;
ProbMatr=[];
for r=1:nint
    ISM=[];
    k=2;
    while SampleMatr(k,1)==0
        k=k+1;
    end
    k=k-1;
    ISM(1,:)=SampleMatr(k,InternSteps(r));
    NoOfStates=1;
    IPM=[];
    IPM(1,1)=1;
    j=2.*ones(1,NoOfSim);
    Skip=zeros(1,NoOfSim);
    for k=1:round(SimTime/TimeInt)
      k
      IPM=[IPM zeros(NoOfStates,1)];
      for i=1:NoOfSim
        ready=0;
        while ((FirstRows(i)+j(1,i))<FirstRows(i+1)) & ~ready
                Quit=0;
                while  ((FirstRows(i)+j(1,i))<FirstRows(i+1)-1) & ~Quit
                        Quit=(SampleMatr((FirstRows(i)+j(1,i)),1)~= ...
                                SampleMatr((FirstRows(i)+j(1,i)+1),1));
                        j(1,i)=j(1,i)+1;  %Skips unstable states
                        Skip(1,i)=Skip(1,i)+1;
                end
                if Quit
                        j(1,i)=j(1,i)-1;
                        Skip(1,i)=Skip(1,i)-1;
                end
                ready=~(SampleMatr(FirstRows(i)+j(1,i),1)<k*TimeInt);
                if ~ready
                        Skip(1,i)=0;
                end
                j(1,i)=j(1,i)+1;
        end
        if ready
                j(1,i)=j(1,i)-1;
        end
        [found,row]=rowinmatr...
        (SampleMatr((FirstRows(i)+j(1,i)-Skip(1,i)-1),InternSteps(r)),ISM);
        if ~found
           ISM=[ISM;...
                SampleMatr((FirstRows(i)+j(1,i)-Skip(1,i)-1),InternSteps(r))];
           IPM=[IPM; zeros(1,k) 1]; %zeros for index 0..t-1
           NoOfStates=NoOfStates+1;
         else
                IPM(row,k+1)=IPM(row,k+1)+1;
         end
      end %for i
    end %for k
    [mp,np]=size(IPM);
    s=sum(IPM);
    for k=1:np
       IPM(:,k)=s(1,k).\IPM(:,k);
    end;

    ProbMatr=[ProbMatr;IPM];
    if msm>NoOfStates
       StateMatr=[StateMatr [ISM; -ones((msm-NoOfStates),1)]];
    else
       StateMatr=[[StateMatr;-ones((NoOfStates-msm),(r-1))] ISM];
       msm=NoOfStates;
    end
end %for r
```

```
function [StateMatr,ProbMatr]=stepan2(Steps,SimTime,TimeInt,InitDelay)

%function [StateMatr,ProbMatr]=statean2(SimTime,TimeInt,InitDelay)
%
%Identifies the different states of the steps with index in Steps.
%Steps is a row vector with the indexes of the steps that are to be analysed.
%The probability of the analysed steps being in a state is calculated at the
%times t=k*Timeint where k=0,1.. and k*TimeInt<=SimTime. Only the states that
%are detected at the specific times above are included in StateMatr.
%StateMatr is a (max(NoOfStates(AnalysedStepi)))*NoOfAnalysedSteps matrix.
%The analysed steps which have fewer states than the analysed step with
%most states will have -1 on the places not used.

load eventlog.t;
[m,n]=size(eventlog)
SampleMatr=eventlog;
InternSteps=2.+Steps;

%Removes instable states caused by an InitDelayedStep.

if InitDelay>0
  for k=1:(mfr-1)
    DeleteRow=1;
    while DeleteRow
        DeleteRow=(SampleMatr(FirstRows(k)+2,1)==-1);
        if DeleteRow
                SampleMatr(FirstRows(k)+1,:)=[];
                m=m-1
        end
     end
   end
end




%Removes unstable states.
%Removes lines with the same time (same value in the first column) only the
%last line in a group with the same times is left in SampleMatr.

k=1;
while k<m
        if SampleMatr(k,1)==SampleMatr(k+1,1)
                SampleMatr(k,:)=[];
                m=m-1;
        else
                k=k+1;
        end
end



%Main algorithm
[mint,nint]=size(InternSteps);
FirstRows=[find(isnan(SampleMatr(1:m,1)));(m+1)];
[NoOfSim,dummy]=size(FirstRows);
NoOfSim=NoOfSim-1;
StateMatr=[];
msm=0;
ProbMatr=[];
for r=1:nint
    ISM=[];
    ISM(1,:)=SampleMatr(2,InternSteps(r));
    NoOfStates=1;
    IPM=[];
    IPM(1,1)=1;
    j=2.*ones(1,NoOfSim);
    for k=1:round(SimTime/TimeInt)
      IPM=[IPM zeros(NoOfStates,1)];
      for i=1:NoOfSim
        ready=0;
        while ((FirstRows(i)+j(1,i))<FirstRows(i+1)) & ~ready
                ready=~(SampleMatr(FirstRows(i)+j(1,i),1)<k*TimeInt);
                j(1,i)=j(1,i)+1;
        end
        if ready
                j(1,i)=j(1,i)-1;
        end
        [found,row]=...
        rowinmatr(SampleMatr(FirstRows(i)+j(1,i)-1,InternSteps(r)),ISM);
        if ~found
           ISM=[ISM; SampleMatr(FirstRows(i)+j(1,i)-1,InternSteps(r))];
           IPM=[IPM; zeros(1,k) 1]; %nollor fran index 0..t-1
           NoOfStates=NoOfStates+1;
        else
                IPM(row,k+1)=IPM(row,k+1)+1;
        end
      end %for i
      k
    end %for k
    [mp,np]=size(IPM);
    s=sum(IPM);
    for k=1:np
       IPM(:,k)=s(1,k).\IPM(:,k);
    end;

    ProbMatr=[ProbMatr;IPM];
    if msm>NoOfStates
       StateMatr=[StateMatr [ISM; -ones((msm-NoOfStates),1)]];
    else
       StateMatr=[[StateMatr;-ones((NoOfStates-msm),(r-1))] ISM];
       msm=NoOfStates;
    end
end %for r
```

```
function [Result,StepStateMatr]=ensprob...
(IntensMatr,InitState,StateMatr,Steps,Time,Interval)

%function [Result,State]=ensprob...
%(IntensMatr,InitState,StateMatr,Steps,Time,Interval)
%
%Computes the solution to dpi/dt=pi*IntensMatr; pi(0)=InitState from t=0
%to t=Time. The solution is calculated in intervals Interval.
%Steps is a rowvector with the indexes of the steps which state is analysed.
%The state of a Step is part of several states in StateMatr. The probabilities
%of the different states in StateMatr which contain the same state of an
%analysed step is added.


%Add building algorithm

StepStateMatr=[];
mss=0;
mi=0;
Add=[];
[ms ns]=size(StateMatr);
[mst,nst]=size(Steps);
for i=1:nst
  NoOfStSt=0;
  for k=1:ms
      if NoOfStSt==0
        found=0;
      else
      [found,row]=rowinmatr(StateMatr(k,Steps(i)),ISSM((mi-NoOfStSt+1):mi,1));
      end
      if ~found
                ISSM=[ISSM;StateMatr(k,Steps(i))];
                mi=mi+1;
                NoOfStSt=NoOfStSt+1;
                Dummy=zeros(1,ms);
                Dummy(1,k)=1;
                Add=[Add;Dummy];
      else
                Add((row+mi-NoOfStSt),k)=1;
      end
  end
  if mss>NoOfStSt
        StepStateMatr=..
        [StepStateMatr [ISSM((mi-NoOfStSt+1):mi,1); -ones((mss-NoOfStSt),1)]];
  else
      StepStateMatr=...
      [[StepStateMatr;-ones((NoOfStSt-mss),(i-1))] ISSM((mi-NoOfStSt+1):mi,1)];
      mss=NoOfStSt;
  end
end



%Calculating algorithm

M=IntensMatr';
I=InitState';
Result=[];
for k=1:(round(Time/Interval)+1)
        Result=[Result Add*(expm((k-1)*Interval*M)*I)];
end

save
```

Appendix D. Production System Primitives

```
LIBRARY ProdPrim2;
USES PetriBase,Petri;


FirstLink ISA Model WITH
Submodels:
      AvailMach,FilledCarriers ISAN InitStep1;
      ProdInMach, KanBanBox ISA DelayedStep;
      T1 ISA Transition WITH Consume2:=1; END;
      T2 ISA Transition;
      MachiningTime,TransportKanbanTime ISAN ExpGen2;
      P1AvailMach,P1FilledCarriers,P1ProdInMach,P1KanbanBox ISA Plot;

Terminals:
        Out ISA RecordTerminal WITH
                OutFilledCarriers ISA StepLTerminal;
                InKanbanBox ISA StepUTerminal;
        END;

        PrinterComm ISA PrinterOutConn;

connections:

        T1.Upper1 AT AvailMach.Lower1;
        T1.Upper2 AT KanbanBox.Lower1;
        T1.Lower1 AT ProdInMach.Upper1;


        T2.Upper1 AT ProdInMach.Lower2;
        T2.Lower1 AT FilledCarriers.Upper1;
        T2.Lower1 AT AvailMach.Upper1;


        Out.OutFilledCarriers AT FilledCarriers.Lower1;
        Out.InKanbanBox AT KanbanBox.Upper1;


        MachiningTime.Comm AT ProdInMach.Comm;
        TransportKanbanTime.Comm AT KanbanBox.Comm;


        PrinterComm.Ch1 AT ProdInMach.Printer;
        PrinterComm.Ch2 AT FilledCarriers.Printer;
        PrinterComm.Ch3 AT KanbanBox.Printer;
        PrinterComm.Ch4 AT AvailMach.Printer;



        P1ProdInMach.In:=ProdInMach.State;
        P1AvailMach.In:=AvailMach.State;
        P1FilledCarriers.In:=FilledCarriers.State;
        P1KanbanBox.In:=KanbanBox.State;

END; %FirstLink


MiddleLink ISA Model WITH
Submodels:
      AvailMach,FilledCarriers ISAN InitStep1;
      ProdInMach, KanbanBox ISA DelayedStep;
      T1 ISA Transition WITH Consume2:=1; Consume3:=1; END;
      T2 ISA Transition;
      MachiningTime,TransportKanbanTime ISAN ExpGen2;
      P1AvailMach,P1FilledCarriers,P1ProdInMach,P1KanbanBox ISA Plot;

Terminals:
        In ISA RecordTerminal WITH
                InT1 ISA TransitionUTerminal;
                OutT1 ISA TransitionLTerminal;
        END;

        Out ISA RecordTerminal WITH
                OutFilledCarriers ISA StepLTerminal;
                InKanbanBox ISA StepUTerminal;
        END;

        PrinterComm ISA PrinterOutConn;

Connections:
        T1.Upper1 AT AvailMach.Lower1;
        T1.Upper2 AT KanbanBox.Lower1;
        T1.Upper3 AT In.InT1;
        T1.Lower1 AT ProdInMach.Upper1;
        T1.Lower1 AT In.OutT1;


        T2.Upper1 AT ProdInMach.Lower1;
        T2.Lower1 AT FilledCarriers.Upper1;
        T2.Lower1 AT AvailMach.Upper1;



        Out.OutFilledCarriers AT FilledCarriers.Lower1;
        Out.InKanbanBox AT KanbanBox.Upper1;


        MachiningTime.Comm AT ProdInMach.Comm;
        TransportKanbanTime.Comm AT KanbanBox.Comm;


        PrinterComm.Ch1 AT ProdInMach.Printer;
        PrinterComm.Ch2 AT FilledCarriers.Printer;
        PrinterComm.Ch3 AT KanbanBox.Printer;
        PrinterComm.Ch4 AT AvailMach.Printer;


        P1ProdInMach.In:=ProdInMach.State;
        P1AvailMach.In:=AvailMach.State;
        P1FilledCarriers.In:=FilledCarriers.State;
        P1KanbanBox.In:=KanbanBox.State;

END; %MiddleLink



LastLink ISA MiddleLink WITH
Submodels:
        T3 ISA Transition WITH Consume2:=1;END;

Terminals:
        EnvirConn ISA RecordTerminal WITH
                Deliver ISA TransitionLTerminal;
                Produce ISA TransitionUTerminal;
        END;

connections:
```

```
        T3.Upper1 AT FilledCarriers.Lower1;
        T3.Upper2 AT EnvirConn.Produce;

        T3.Lower1 AT EnvirConn.Deliver;
        T3.Lower1 AT KanbanBox.Upper1;

END; %LastLink



FirstAndLastLink ISA FirstLink WITH
Submodels:
        T3 ISA Transition WITH Consume2:=1;END;

Terminals:
        EnvirConn ISA RecordTerminal WITH
                Deliver ISA TransitionLTerminal;
                Produce ISA TransitionUTerminal;
        END;

connections:
                                                                        $
                                              ^ T3.Upper1 AT FilledCarriers.Lower1;
        T3.Upper2 AT EnvirConn.Produce;

        T3.Lower1 AT EnvirConn.Deliver;
        T3.Lower1 AT KanbanBox.Upper1;
END; %FirstAndLastLink



Environment1 ISA Model WITH
SubModels:
        Order,BackLog  ISAN InitStep1;
        Delivered ISA Step;
        PlOrder,PlBackLog,PlDelivered ISA Plot;
        Demand ISA ExpGen1;

Terminal:
        InNet ISA RecordTerminal WITH
                DelConn ISA StepUTerminal;
                LogConn ISA StepLTerminal;
        END;
        PrinterComm ISA PrinterOutConn;


Connections:
        BackLog.Lower1 AT InNet.LogConn;
        Delivered.Upper1 AT InNet.DelConn;

        Demand.Signal AT Order.InsertNew;
        Demand.Signal AT BackLog.InsertNew;

        PrinterComm.Ch1 AT Order.Printer;
        PrinterComm.Ch2 AT BackLog.Printer;
        PrinterComm.Ch3 AT Delivered.Printer;

        PlOrder.In:=Order.State;
        PlBackLog.In:=BackLog.State;
        PlDelivered.In:=Delivered.State;

END; %Environment1



Environment2 ISA Model WITH
SubModels:
        Initial  ISAN InitStep1;
        Demand ISA DelayedStep;
        Order,BackLog,Delivered ISA Step;
        T1,T2 ISA Transition;
        PlInitial,PlDemand,PlOrder,PlBackLog,PlDelivered ISA Plot;
        Random ISA ExpGen2;


Terminal:
        InNet ISA RecordTerminal WITH
                DelConn ISA StepUTerminal;
                LogConn ISA StepLTerminal;
        END;
        PrinterComm ISA PrinterOutConn;


Connections:
        T1.Upper1 AT Initial.Lower1;
        T1.Lower1 AT Demand.Upper1;

        T2.Upper1 AT Demand.Lower1;
        T2.Lower1 AT BackLog.Upper1;
        T2.Lower1 AT Order.Upper1;

        BackLog.Lower1 AT InNet.LogConn;
        Delivered.Upper1 AT InNet.DelConn;

        Random.Comm AT Demand.Comm;

        PrinterComm.Ch1 AT Initial.Printer;
        PrinterComm.Ch2 AT Demand.Printer;
        PrinterComm.Ch3 AT Order.Printer;
        PrinterComm.Ch4 AT BackLog.Printer;
        PrinterComm.Ch5 AT Delivered.Printer;

        PlOrder.In:=Order.State;
        PlBackLog.In:=BackLog.State;
        PlDelivered.In:=Delivered.State;
        PlInitial.In:=Initial.State;
        PlDemand.In:=Demand.State;

END; %Environment2
```

```
LIBRARY ProdNet;
USES Petri,ProdPrim2;

Net2 ISA Model WITH

% This is the net that has been analysed theoreticallyr

submodels:

Pr ISA Printer3;

L1 ISA FirstLink WITH
        AvailMach.InitState1:=2;
        FilledCarriers.InitState1:=2;
        MachiningTime.Lambda:=0.5;
        TransportKanbanTime.Lambda:=5;
END;



L2 ISA LastLink WITH
        AvailMach.InitState1:=1;
        FilledCarriers.InitState1:=1;
        MachiningTime.Lambda:=1;
        TransportKanbanTime.Lambda:=5;
END;

Controller ISAN Environment2 WITH
         Initial.InitState1:=5;
         Random.Lambda:=0.5;
END;



Connections:
        L1.Out AT L2.In;
        L2.EnvirConn AT Controller.InNet;
        %Controller.PrinterComm AT Pr.Comm1;
        L1.PrinterComm AT Pr.Comm2;
        L2.PrinterComm AT Pr.Comm3;

END; %Net2
```

```
LIBRARY ProdPrim1;
USES PetriBase,Petri;


FirstLink ISA Model WITH
Submodels:
      AvailMach,FilledCarriers ISAN InitStep1;
      ProdInMach, KanBanBox ISA DelayedStep;
      NoOfOrdProd,MaterialCarrier,CollectBox ISA Step;
      T1 ISA Transition WITH Consume2:=1; END;
      T2,T3,T4,T5 ISA Transition;
      MachiningTime,TransportKanbanTime ISAN ExpGen2;
      P1AvailMach,P1FilledCarriers,P1ProdInMach,P1NoOfOrdProd,P1KanbanBox,
      P1MaterialCarrier,P1CollectBox ISA Plot;

Terminals:
        Out ISA RecordTerminal WITH
                OutFilledCarriers ISA StepLTerminal;
                InCollectBox ISA StepUTerminal;
        END;

        PrinterComm ISA RecordTerminal WITH
                Ch1,Ch2,Ch3,Ch4,Ch5,Ch6,Ch7 ISA PrinterOutTerminal;
        END;

connections:

        T1.Upper1 AT AvailMach.Lower1;
        T1.Upper2 AT NoOfOrdProd.Lower1;
        T1.Lower1 AT ProdInMach.Upper1;


        T2.Upper1 AT ProdInMach.Lower2;
        T2.Lower1 AT MaterialCarrier.Upper1;
        T2.Lower1 AT AvailMach.Upper1;

        T3.Upper1 AT MaterialCarrier.Lower1;
        T3.Lower1 AT FilledCarriers.Upper1;


        T4.Upper1 AT CollectBox.Lower1;
        T4.Lower1 AT KanbanBox.Upper1;


        T5.Upper1 AT KanbanBox.Lower1;
        T5.Lower1 AT NoOfOrdProd.Upper1;



        Out.OutFilledCarriers AT FilledCarriers.Lower1;
        Out.InCollectBox AT CollectBox.Upper1;


        MachiningTime.Comm AT ProdInMach.Comm;
        TransportKanbanTime.Comm AT KanbanBox.Comm;


        PrinterComm.Ch1 AT ProdInMach.Printer;
        PrinterComm.Ch2 AT MaterialCarrier.Printer;
        PrinterComm.Ch3 AT FilledCarriers.Printer;
        PrinterComm.Ch4 AT CollectBox.Printer;
        PrinterComm.Ch5 AT KanbanBox.Printer;
        PrinterComm.Ch6 AT NoOfOrdProd.Printer;
        PrinterComm.Ch7 AT AvailMach.Printer;


        P1ProdInMach.In:=ProdInMach.State;
        P1AvailMach.In:=AvailMach.State;
        P1FilledCarriers.In:=FilledCarriers.State;
        P1KanbanBox.In:=KanbanBox.State;
        P1NoOfOrdProd.In:=NoOfOrdProd.State;
        P1MaterialCarrier.In:=MaterialCarrier.State;
        P1CollectBox.In:=CollectBox.State;

END; %FirstLink


MiddleLink ISA Model WITH
Submodels:
      AvailMach,FilledCarriers ISAN InitStep1;
      ProdInMach, KanbanBox ISA DelayedStep;
      NoOfOrdProd,MaterialCarrier,CollectBox ISA Step;
      T1 ISA Transition WITH Consume2:=1; Consume3:=1; END;
      T2,T3,T4,T5 ISA Transition;
      MachiningTime,TransportKanbanTime ISAN ExpGen2;
      P1AvailMach,P1FilledCarriers,P1ProdInMach,P1NoOfOrdProd,P1KanbanBox,
      P1MaterialCarrier,P1CollectBox ISA Plot;

Terminals:
        In ISA RecordTerminal WITH
                InT1 ISA TransitionUTerminal;
                OutT1 ISA TransitionLTerminal;
        END;

        Out ISA RecordTerminal WITH
                OutFilledCarriers ISA StepLTerminal;
                InCollectBox ISA StepUTerminal;
        END;

        PrinterComm ISA RecordTerminal WITH
                Ch1,Ch2,Ch3,Ch4,Ch5,Ch6,Ch7 ISA PrinterOutTerminal;
        END;

Connections:
        T1.Upper1 AT AvailMach.Lower1;
        T1.Upper2 AT NoOfOrdProd.Lower1;
        T1.Upper3 AT In.InT1;
        T1.Lower1 AT ProdInMach.Upper1;
        T1.Lower1 AT In.OutT1;


        T2.Upper1 AT ProdInMach.Lower1;
        T2.Lower1 AT MaterialCarrier.Upper1;
        T2.Lower1 AT AvailMach.Upper1;


        T3.Upper1 AT MaterialCarrier.Lower1;
        T3.Lower1 AT FilledCarriers.Upper1;


        T4.Upper1 AT CollectBox.Lower1;
        T4.Lower1 AT KanbanBox.Upper1;


        T5.Upper1 AT KanbanBox.Lower1;
        T5.Lower1 AT NoOfOrdProd.Upper1;
```

```
        Out.OutFilledCarriers AT FilledCarriers.Lower1;
        Out.InCollectBox AT CollectBox.Upper1;


        MachiningTime.Comm AT ProdInMach.Comm;
        TransportKanbanTime.Comm AT KanbanBox.Comm;


        PrinterComm.Ch1 AT ProdInMach.Printer;
        PrinterComm.Ch2 AT MaterialCarrier.Printer;
        PrinterComm.Ch3 AT FilledCarriers.Printer;
        PrinterComm.Ch4 AT CollectBox.Printer;
        PrinterComm.Ch5 AT KanbanBox.Printer;
        PrinterComm.Ch6 AT NoOfOrdProd.Printer;
        PrinterComm.Ch7 AT AvailMach.Printer;



        Pl1ProdInMach.In:=ProdInMach.State;
        Pl1AvailMach.In:=AvailMach.State;
        Pl1FilledCarriers.In:=FilledCarriers.State;
        Pl1KanbanBox.In:=KanbanBox.State;
        Pl1NoOfOrdProd.In:=NoOfOrdProd.State;
        Pl1MaterialCarrier.In:=MaterialCarrier.State;
        Pl1CollectBox.In:=CollectBox.State;

END; %MiddleLink




MiddleLink1 ISA FirstLink WITH
%Does not pass the instantiation
Submodel:
        T1 ISA Transition WITH Consume2:=1; Consume3:=1; END;

Terminals:
        In ISA RecordTerminal WITH
                InT1 ISA TransitionUTerminal;
                OutT1 ISA TransitionLTerminal;
        END;

connections:

        T1.Upper1 AT AvailMach.Lower1;
        T1.Upper2 AT NoOfOrdProd.Lower1;
        T1.Upper3 AT In.InT1;

        T1.Lower1 AT ProdInMach.Upper1;
        T1.Lower1 AT In.OutT1;

END; %MiddleLink1



LastLink ISA MiddleLink WITH
Submodels:
        T6 ISA Transition WITH Consume2:=1;END;

Terminals:
        EnvirConn ISA RecordTerminal WITH
                Deliver ISA TransitionLTerminal;
                Produce ISA TransitionUTerminal;
        END;

connections:

        T6.Upper1 AT FilledCarriers.Lower1;
        T6.Upper2 AT EnvirConn.Produce;

        T6.Lower1 AT EnvirConn.Deliver;
        T6.Lower1 AT CollectBox.Upper1;

END; %LastLink




Environment1 ISA Model WITH
SubModels:
        Order,BackLog  ISAN InitStep1;
        Delivered ISA Step;
        Pl0rder,Pl1BackLog,Pl1Delivered ISA Plot;
        Demand ISA ExpGen1;

Terminal:
        InNet ISA RecordTerminal WITH
                DelConn ISA StepUTerminal;
                LogConn ISA StepLTerminal;
        END;
        PrinterComm ISA RecordTerminal WITH
                Ch1,Ch2,Ch3 ISA PrinterOutTerminal;
        END;


Connections:
        BackLog.Lower1 AT InNet.LogConn;
        Delivered.Upper1 AT InNet.DelConn;

        Demand.Signal AT Order.InsertNew;
        Demand.Signal AT BackLog.InsertNew;

        PrinterComm.Ch1 AT Order.Printer;
        PrinterComm.Ch2 AT BackLog.Printer;
        PrinterComm.Ch3 AT Delivered.Printer;

        Pl0rder.In:=Order.State;
        Pl1BackLog.In:=BackLog.State;
        Pl1Delivered.In:=Delivered.State;

END; %Environment1



Environment2 ISA Model WITH
SubModels:
        Initial  ISAN InitStep1;
        Demand ISA DelayedStep;
        Order,BackLog,Delivered ISA Step;
        T1,T2 ISA Transition;
```

```
        PlInitial,PlDemand,PlOrder,PlBackLog,PlDelivered ISA Plot;
        Random ISA ExpGen2;


Terminal:
        InNet ISA RecordTerminal WITH
                DelConn ISA StepUTerminal;
                LogConn ISA StepLTerminal;
        END;
        PrinterComm ISA RecordTerminal WITH
                Ch1,Ch2,Ch3,Ch4,Ch5 ISA PrinterOutTerminal;
        END;


Connections:
        T1.Upper1 AT Initial.Lower1;
        T1.Lower1 AT Demand.Upper1;

        T2.Upper1 AT Demand.Lower1;
        T2.Lower1 AT BackLog.Upper1;
        T2.Lower1 AT Order.Upper1;

        BackLog.Lower1 AT InNet.LogConn;
        Delivered.Upper1 AT InNet.DelConn;

        Random.Comm AT Demand.Comm;

        PrinterComm.Ch1 AT Initial.Printer;
        PrinterComm.Ch2 AT Demand.Printer;
        PrinterComm.Ch3 AT Order.Printer;
        PrinterComm.Ch4 AT BackLog.Printer;
        PrinterComm.Ch5 AT Delivered.Printer;

        PlOrder.In:=Order.State;
        PlBackLog.In:=BackLog.State;
        PlDelivered.In:=Delivered.State;
        PlInitial.In:=Initial.State;
        PlDemand.In:=Demand.State;

END; %Environment2
```

```
LIBRARY ProdNet;
USES Petri,ProdPrim1;

Net1 ISA Model WITH
submodels:

Pr ISA Printer2;

L1 ISA FirstLink WITH
        AvailMach.InitState1:=2;
        FilledCarriers.InitState1:=2;
        MachiningTime.Lambda:=0.5;
        TransportKanbanTime.Lambda:=5;
        T2.Consume1:=1; %NoOfItems/Kanban
        T2.Produce1:=1; %NoOfItems/Kanban
        T3.Produce1:=1; %NoOfItems/Kanban
        T5.Consume1:=1; %NoOfItems/Kanban
END;


L2 ISA LastLink WITH
        %2 InitSteps
        AvailMach.InitState1:=2;
        FilledCarriers.InitState1:=2;
        MachiningTime.Lambda:=0.5;
        TransportKanbanTime.Lambda:=5;
        T1.Consume3:=1; %NoOfInputProducts/Item
        T2.Consume1:=1; %NoOfItems/Kanban
        T2.Produce1:=1; %NoOfItems/Kanban
        T3.Produce1:=1; %NoOfItems/Kanban
        T5.Consume1:=1; %NoOfItems/Kanban
END;

Controller ISAN Environment2 WITH
         Initial.InitState1:=10;
         Random.Lambda:=0.1;
END;


Connections:
        L1.Out AT L2.In;
        L2.EnvirConn AT Controller.InNet;

        %Controller.PrinterComm AT Pr.Comm1;
        L2.PrinterComm AT Pr.Comm2;
        L1.PrinterComm AT Pr.Comm3;

END; %Net1
```