Petri nets

- Petri nets (PN)
  high-level modelling formalism for concurrent and distributed systems (Petri, Carl A. (Ph. D. Thesis), University of Bonn, 1962)

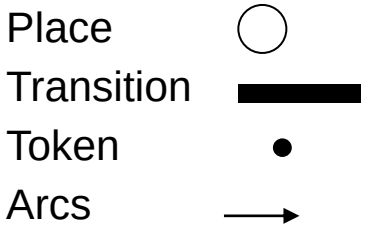
- Stochastic Petri nets (SPN)
  temporal and probabilistic information added to the model
  the approach aimed at equivalence between SPN and Markov Chains
  idea of associating an exponentially distributed random delay with the PN transitions (1990)

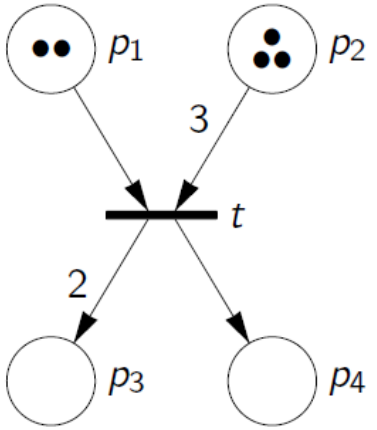
- Stochastic Activity networks (SAN)
  extension to Stochastic Petri nets (data structure, complex firing rules, global variables, ….) (2001)

Petri nets

A Petri net consists of places, transitions, and arcs. Arcs are from a place to a transition or vice versa. Places may contain a discrete number of marks called tokens.

Graphycally:

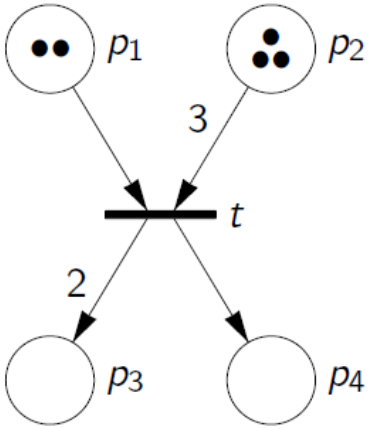




- the places with an arc that runs from the place to a transition are called the input places of the transition
- the places with an arc that runs from the transition to a place are called the output places of the transition

Petri nets

Marking of the network:
any distribution of tokens over the places. This represents a configuration of the network.



A transition of a Petri net may fire if it is enabled, i.e., there are sufficient tokens in all of its input places This is named the weight of the arc (by default 1).

When the transition fires, it consumes the required input tokens, and creates tokens in its output places.

Petri nets are well suited for modeling the concurrent behavior of distributed systems.

Petri nets

$$N_{P/T} = \langle P, T; F, W, M_0 \rangle$$

$$W : F \to \mathbb{N} - \{0\}$$

$$M_0 : P \to \mathbb{N}$$

P: places
T: transitions
F: arcs
W: weight
M_0 : initial marking

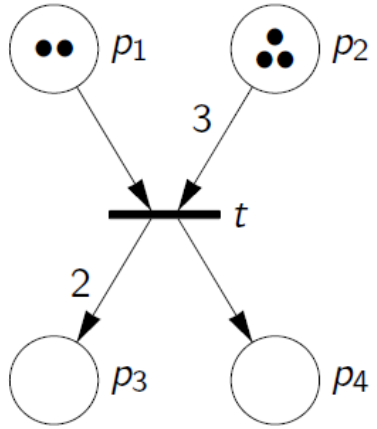
t, y transitions

$${}^\bullet t = \{p \in P \mid \langle p, t \rangle \in F\}$$ preset

$$y^\bullet = \{z \in P \mid \langle y, z \rangle \in F\}$$ postset

t enabled if: $\forall_{p \in {}^\bullet t}\ M(p) \geq W(\langle p, t \rangle)$

we write $M\ [t\rangle$



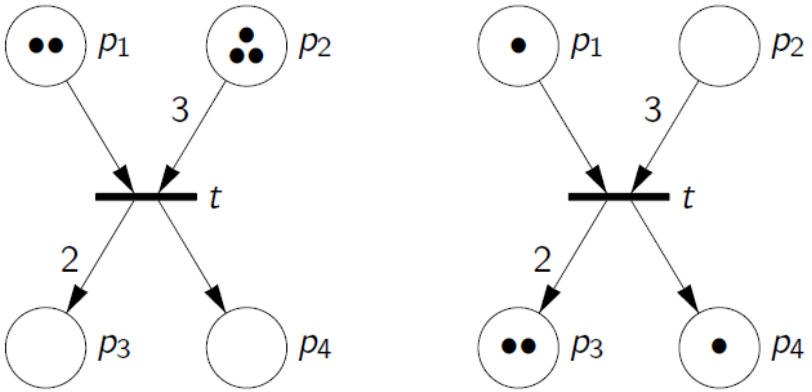
Transition firing

Firing rule

$$
\begin{array}{ll}
\forall_{p \in (\bullet t - t \bullet)} & M'(p) = M(p) - W(\langle p, t \rangle) \\
\forall_{p \in (t \bullet - \bullet t)} & M'(p) = M(p) + W(\langle t, p \rangle) \\
\forall_{p \in (\bullet t \cap t \bullet)} & M'(p) = M(p) - W(\langle p, t \rangle) + W(\langle t, p \rangle) \\
\forall_{p \in P - (\bullet t \cup t \bullet)} & M'(p) = M(p)
\end{array}
$$

We write $M\,[t\rangle\,M'$

M0 [t>M1



M0=(2,3,0,0)

M1=(1,0,2,1)

The firing of a transition is atomic (a single non-interruptible step)

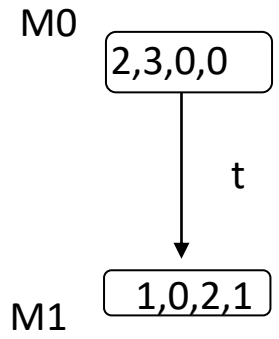
Transition sequence

Reachable marking $R_N(M)$:

$$M \in R_N(M)$$

$$M' \in R_N(M) \wedge \exists_{t \in T} M' \, [t\rangle \, M'' \Rightarrow M'' \in R_N(M)$$

Reachability graph



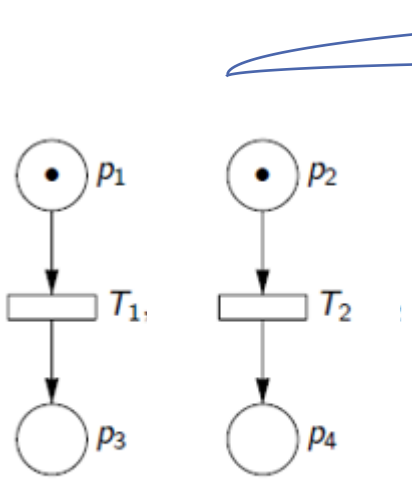
Transition sequence

Let $M \, [t_1\rangle \, M' \wedge M' \, [t_2\rangle \, M''$ then $M \, [t_1 t_2\rangle \, M''$

If $M \, [t_1 \ldots t_n\rangle \, M^{(n)}$ then $t_1 \ldots t_n$ is a transition sequence

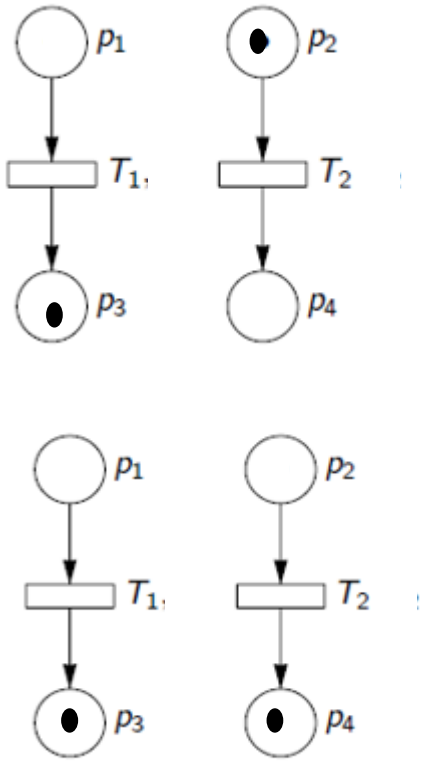
Evolution of the net

At any time, one of the enabled transitions is executed at a time
If multiple transitions are enabled, these transitions will fire in any order. The execution of Petri nets is nondeterministic

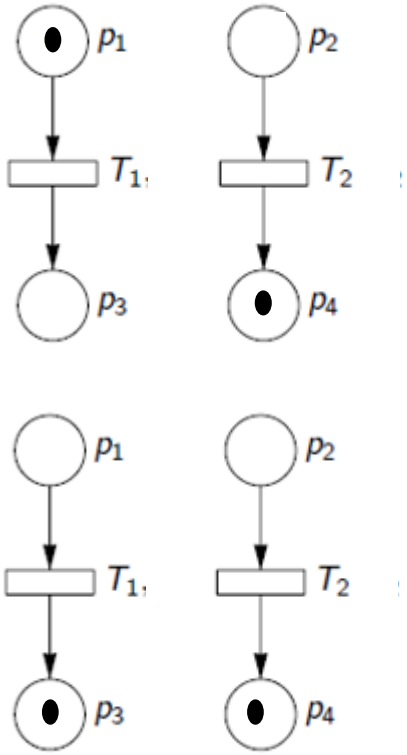


Two possible evolutions

Since T1 and T2 are not in conflict, then M** = M'



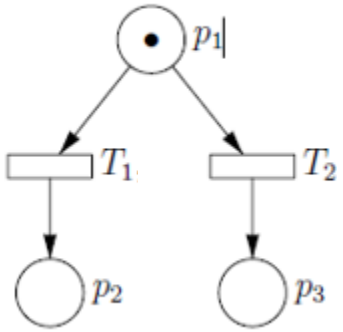
M0 [T1 T2> M**

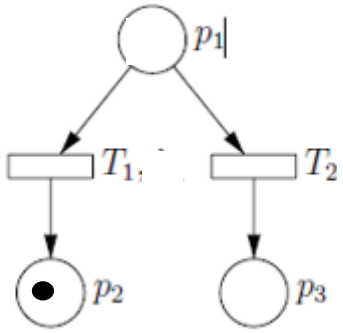


M0 [T2 T1> M''

Conflict resolution

T1, T2 are in conflict
For example, when T1 fires,
T2 is no more enabled

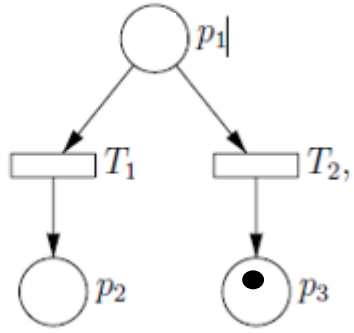




M0=(1,0,0,) M1=(0,1,0)

M0 [T1> M1

T2 not enabled in M1

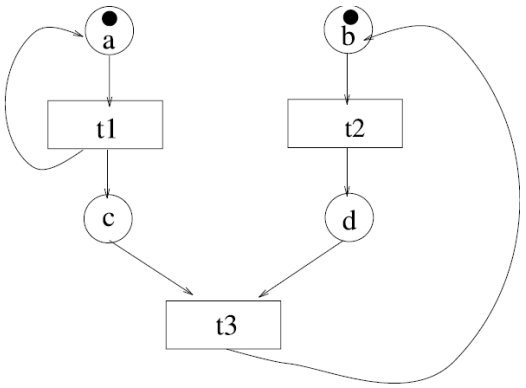


M0=(1,0,0,) M2=(0,0,1)

M0 [T2> M2

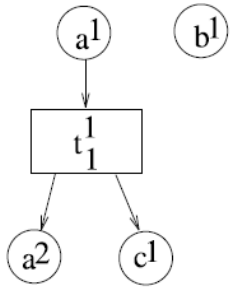
T1 not enabled in M2

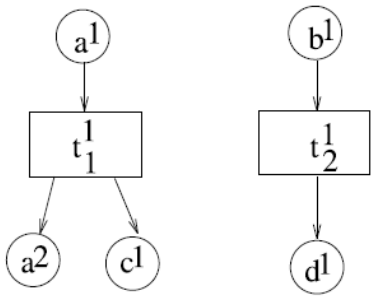
Non-sequential processes



Petri net

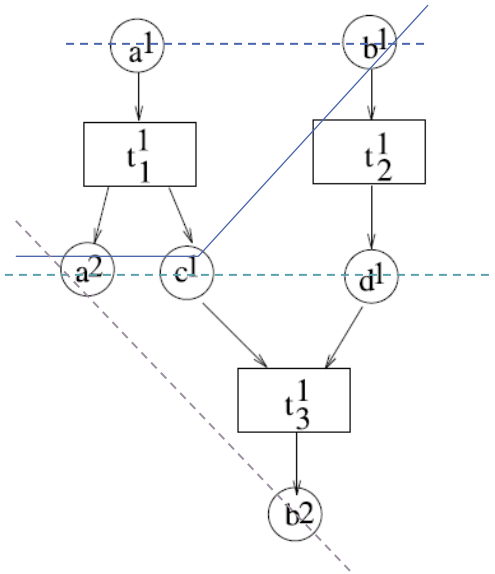
Building a process step by step

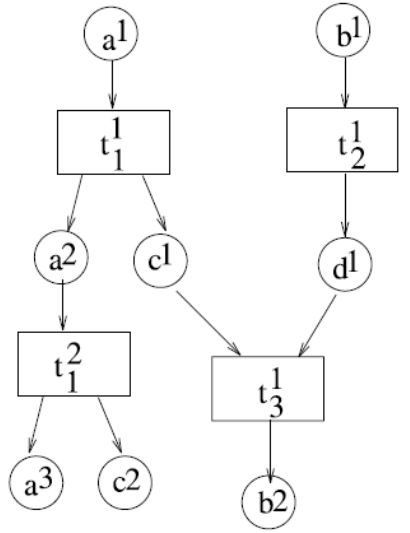




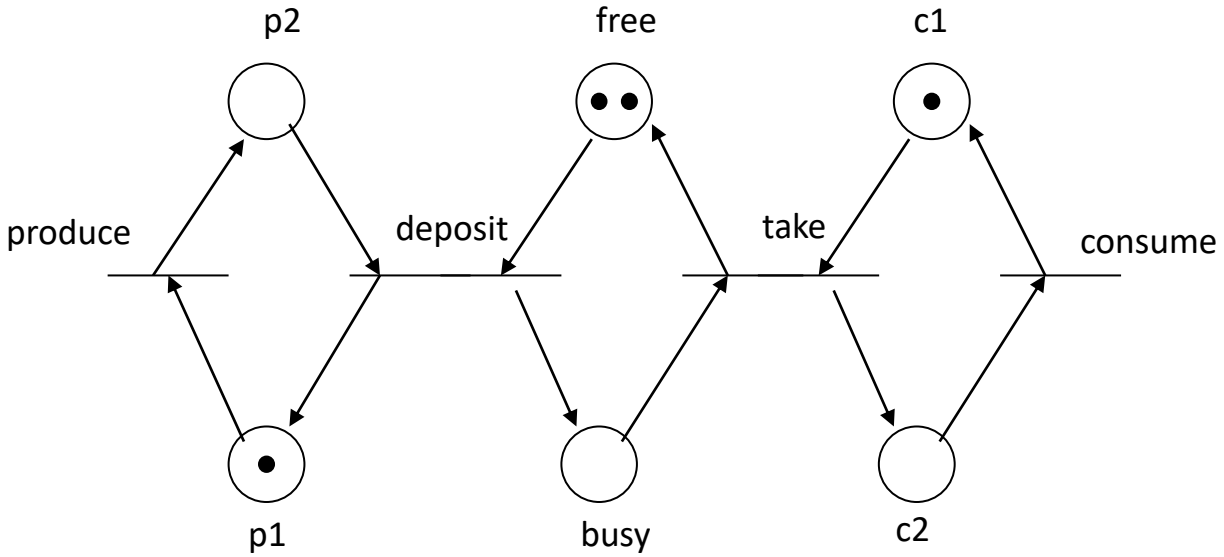
Simulation of concurrent and distributed systems

degree of paralleism: maximum number of transitions enabled in a cut of the process





Producer/Consumer example



Producer

Buffer
(2 slots)

Consumer

free + busy = 2

M=(p1, p2, free, busy, c1, c2)

(1, 0, 2, 0, 1, 0)

[produce>

(0, 1, 2, 0, 1, 0)

[deposit>

(1, 0, 1, 1, 1, 0)

[produce>

[take>

(0, 1, 1, 1, 1, 0)

(1, 0, 2, 0, 0, 1)

[deposit>

(0, 1, 0, 2, 1, 0)

…….

…….

Analysis

- Reachability graph
  the reachability graph may be infinite, i.e. if there is no bound on the number tokens on some place.
  The coverability graph computes an over-approximation (remains exact as long as the number of markings is finite)

- Conditions on reachable markings
  used to check whether some 'bad' state can occur (e.g., violation of mutual exclusion property)

- Transition invariants
  indicate a possible loop in the net (a sequence of transitions which leads back to the marking it starts in)

- Place invariants
  global properties of markings alsways satisfied: the number of tokens in all reachable markings satisfies some linear invariant (e.g., all reachable markings M satisfy: M(p1) + M(p4) = 5)

- Deadlock
  A marking is called a deadlock if no transition is enabled in the marking. A net is deadlock-free if no reachable marking is a deadlock
- ……

Dependabilty evaluation and Petri nets

System description with Petri nets

- Place:
  - a system component (one for every component)
  - a class of system components (CPU , Memory, ..)
  - components in a given state (CPU, FaultyCPU, ..)
  - …….
- Token:
  - number of components (number of CPUs)
  - Occurrence of an event (fault, ..)
  - …..
- Transitions:
  - Occurrence of an event (Repair, CPUFaulty, …)
  - Execution of a computation step
  - …

Timed transitions

- Timed transition: an activity that needs some time to be executed

  -assigning a delay (local time d) to each transition

  -assigning a global time to the PN

- When a transition is enabled a local timer is set to d

  - the timer is decresed

  - when the timer elapses, the transition fires and remove tokens from input places

- If the transitions is desabled before the firing, the timer stops

  Handling of the timer (two alternatives):

  Continue:

  the timer maintains the value and it will be used when the transition is enabled again

  Restart:

  the timer is reset

Timed transitions

Sequence of timed transitions:

$$(\tau_{k1}, t_{k1}) \ldots (\tau_{kn}, t_{kn})$$

where

- $\tau_{k1} <= \tau_{k2} <= \tau_{kn}$
- $[\tau_{ki}, \tau_{ki+1})$

  is the period of time between the firing of two transitions

  in this period of time the net does not change the marking

STOCHASTIC PETRI NET:

when the delay d of a timed transition is a random variable

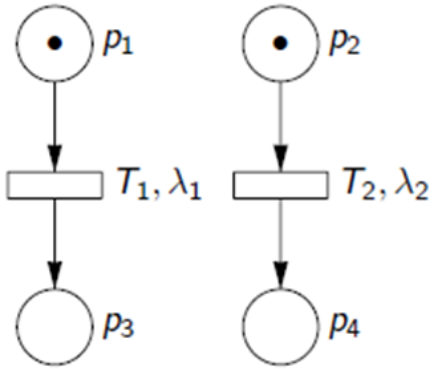
Stochastic Petri nets (SPN)

when the delay is a negative exponential distribution random variable,
the reachability graph is a Markov chain

Transition rate: λk is the transition rate (positive real number) of transition Tk

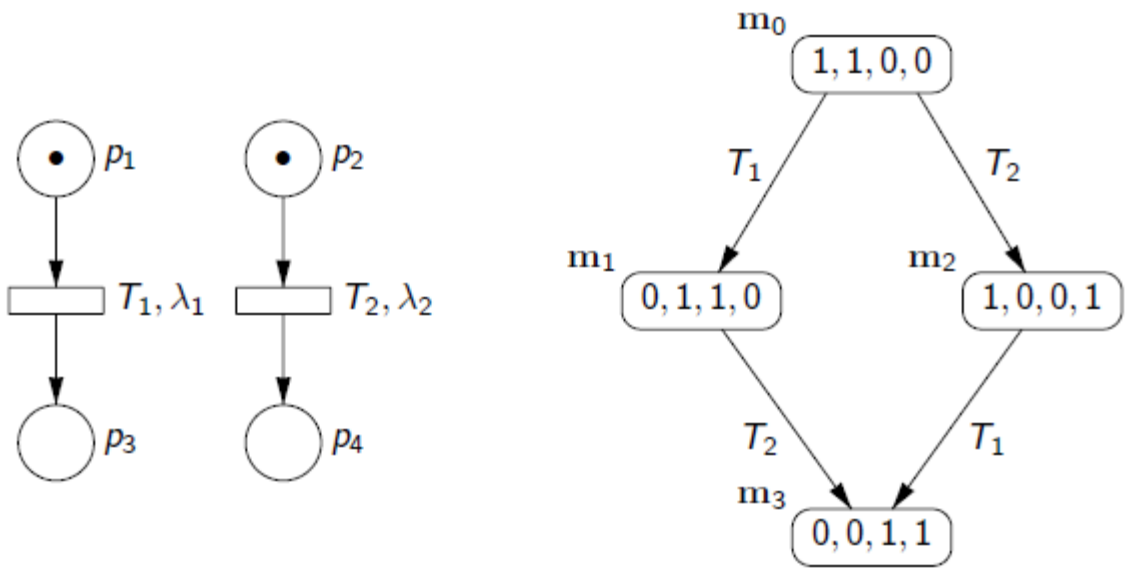
- delay of transition T1 random variable exponentially distributed with parameter $\lambda 1$
- delay of transition T2 random variable exponentially distributed with parameter $\lambda 2$

Assumption: delay of transitions independent



Stochastic Petri nets (SPN)

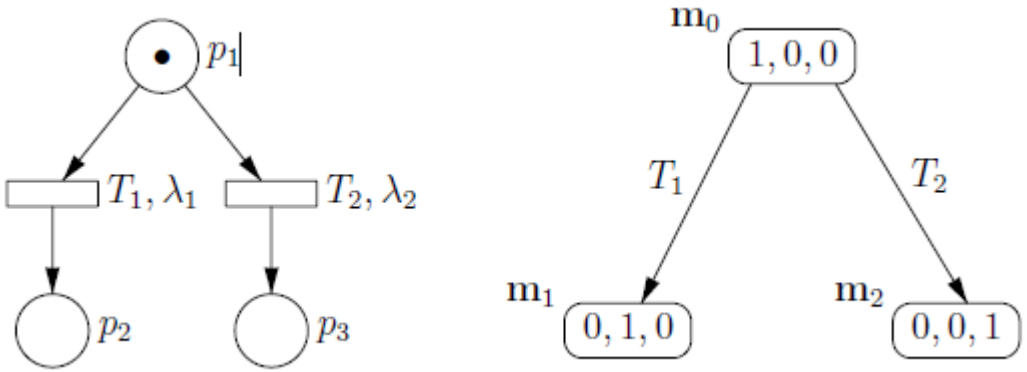
Reachability graph



a timed transition T enabled at time t, with d the random value for the transition delay, fires at time t+d, if it remains enabled in the interval [t, t+d)

Stochastic Petri nets (SPN)

Conflict resolution

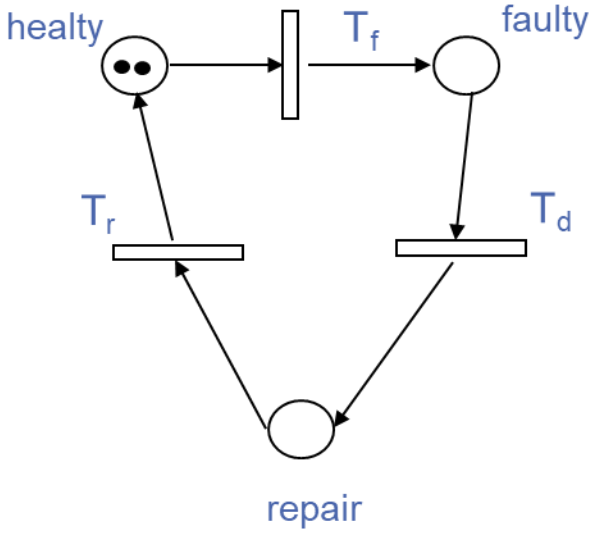


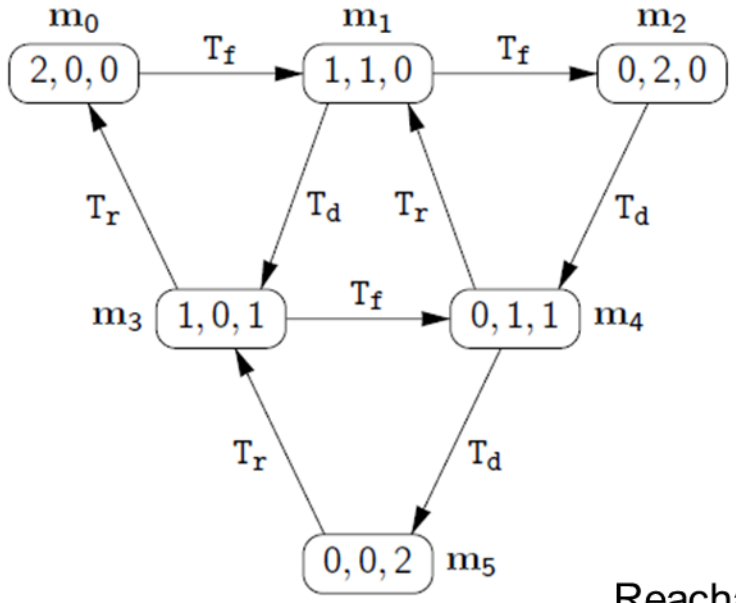
the resolution of the conflict depends on the delay of T1 and T2

A redundant system with repair

- Two identical CPUs
- Failure of the CPU: exponentially distributed with parameter λ
- Fault detection: exponentially distributed with parameter δ
- CPU repair: exponentially distributed with parameter μ

SPN





Reachability graph

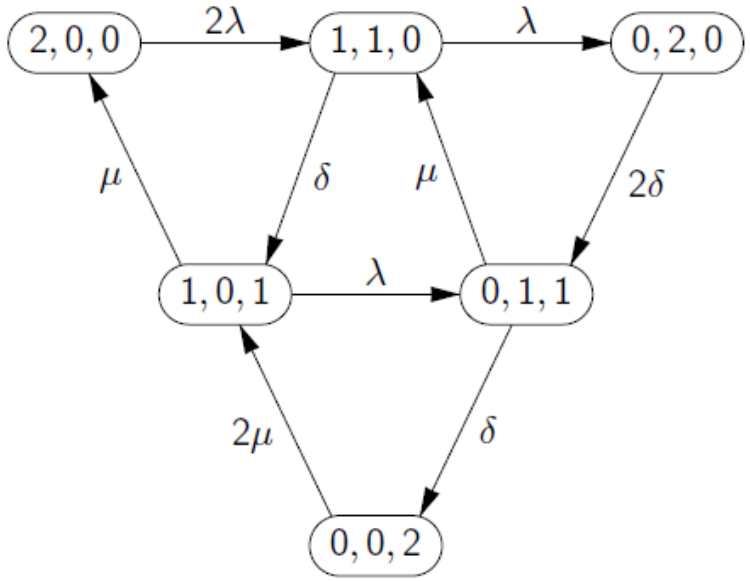
A redundant system with repair

Markov chain

failure rate = λ

detection rate = δ

repair rate = μ



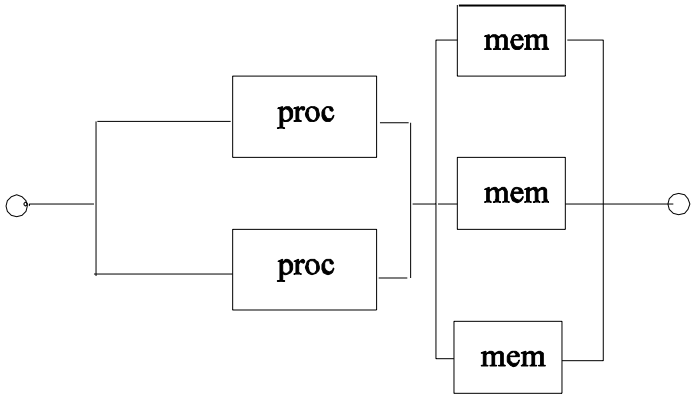
Properties

- Steady-state probability that both processors behave correctly
- Steady-state probability of one undetected faulty processor
- Steady state probability that both processors must be repaired
- .....

M. Ajmone Marsan. Stochastic Petri nets: An elementary introduction. In G. Rozenberg, editor, *Advances in Petri Nets 1989*, volume 424 of *LNCS*, pages 1–29. Springer Verlag, 1990.

Example

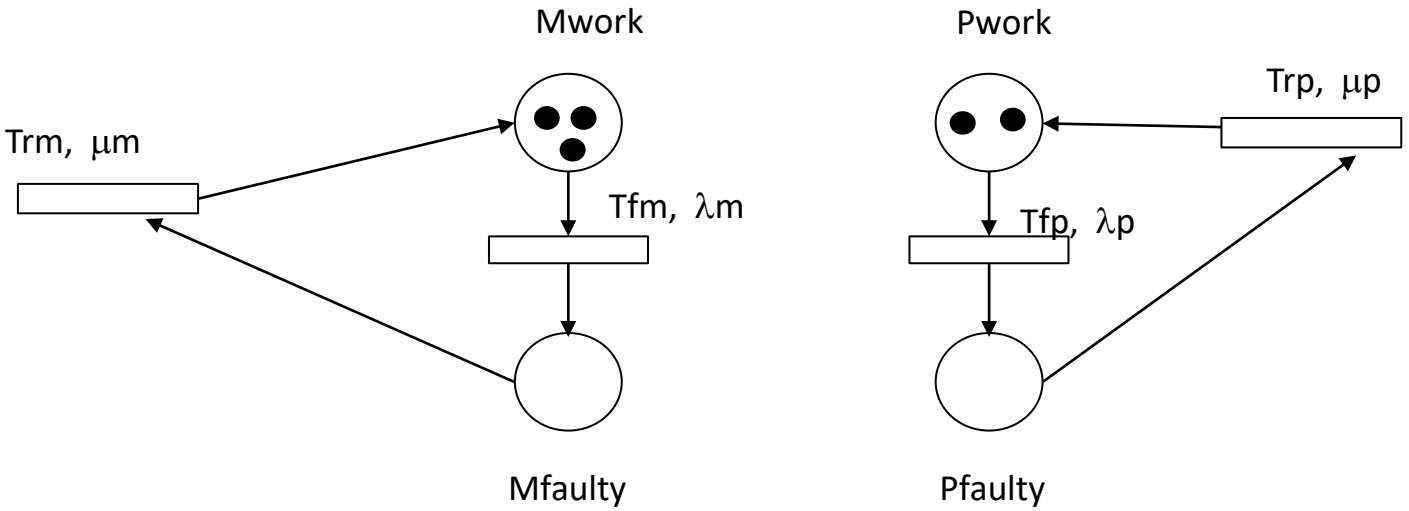
Multiprocessor system with 2 processors and 3 shared memories system.
System is operational if at least one processor and one memory are operational.



λ_m failure rate for memory
λ_p failure rate for processor

μm repair rate for memory
μ_p repair rate for processor

Example

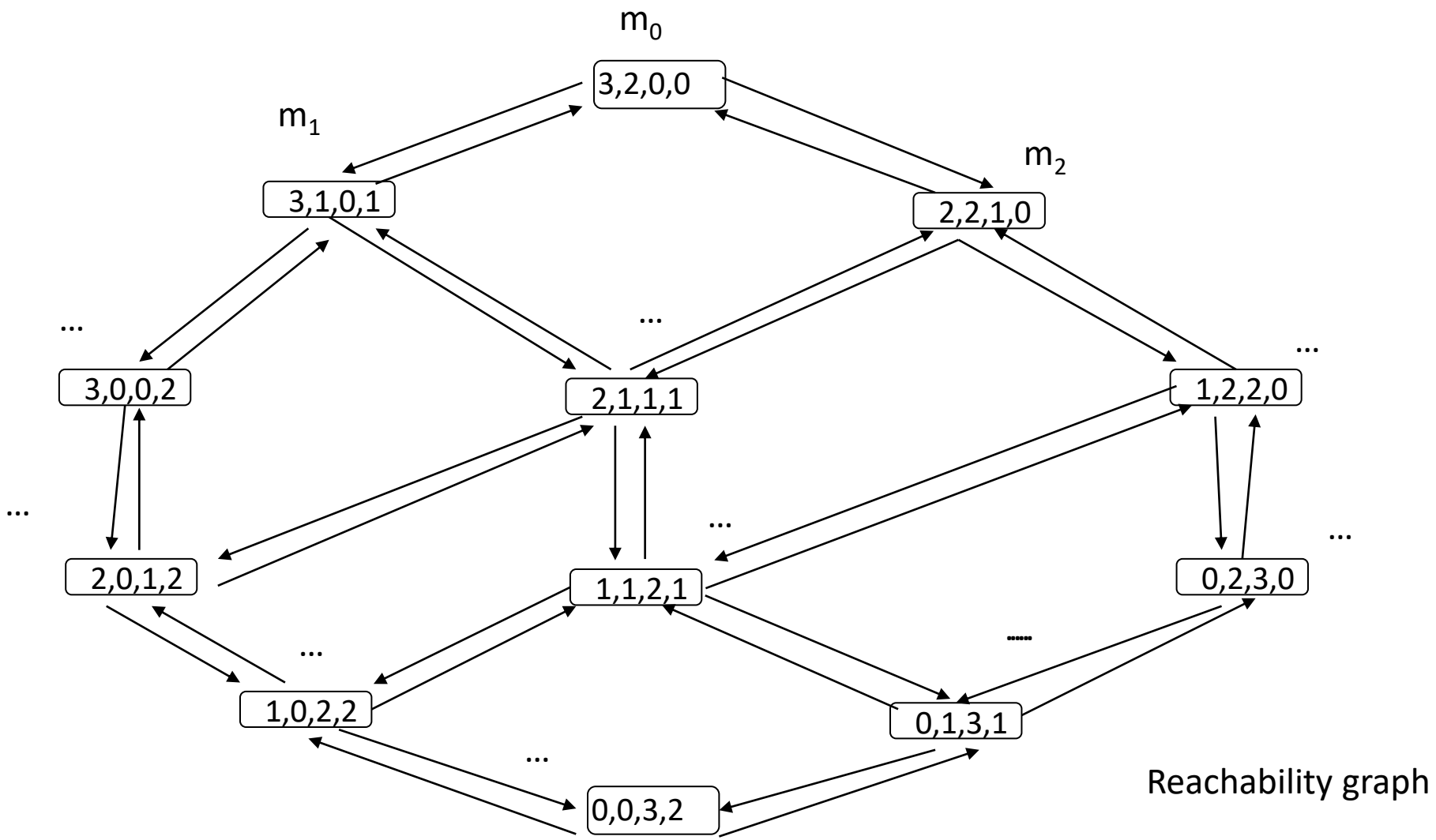


System state
M= (Mwork, Pwork, Mfaulty, Pfaulty)

Initial marking M_0= (3,2,0,0)

(3,2,0,0) [Tfp> (3,1,0,1) [Tfm> (2,1,1,1) ……..

Example



Reachability graph

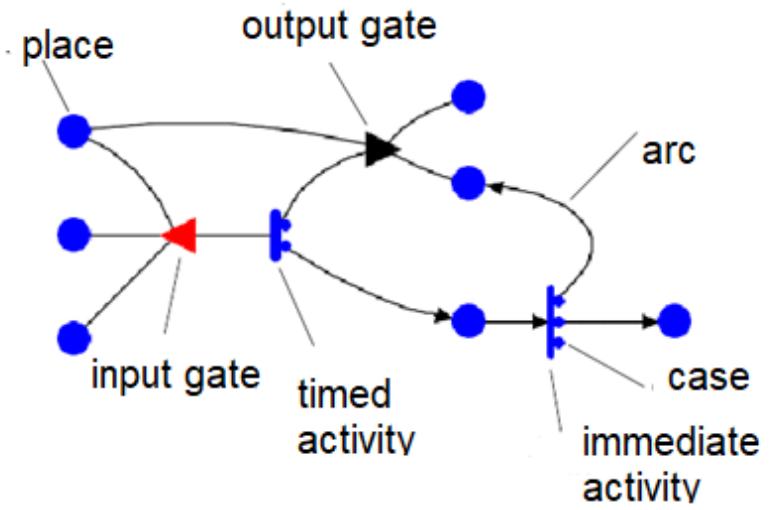
Stochastic Activity Networks (SAN)

A variant of stochastic Petri nets

A system is described with SAN through four disjoint sets of nodes:

- places
- input gates
- output gates
- activities

Cases can be used to model probabilistic behaviors.



Graphically, places are drawn as circles, input (output) gates as left-pointing (right-pointing) triangles, instantaneous activities as narrow vertical bars, and timed activities as thick vertical bars. Cases are drawn as small circles on the right side of activities.

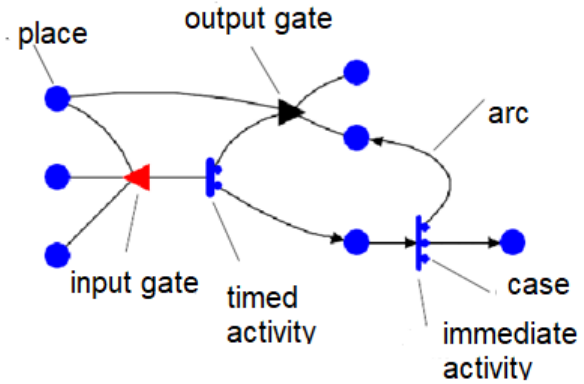
W. H. Sanders and J. F. Meyer. Stochastic activity networks: Formal definitions and concepts. In E. Brinksma, H. Hermanns, and J. P. Katoen, editors, *Lectures on Formal Methods and Performance Analysis*, volume 2090 of *LNCS*, pages 315–343. Springer Verlag, 2001.

Stochastic Activity Networks (SAN)

activity:

- instantaneous
system activities that are completed in a negligible amount of time

- **timed** activities of the modeled system whose durations impact the system's ability to perform
(the duration is expressed via a time distribution function)



input gate:

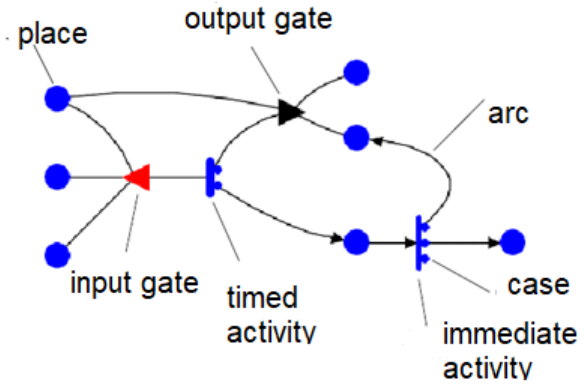
- has a finite set of inputs and one output
- enabling predicate (boolean function on the marking)
- input function defined for all values for which the enabling predicate is true
(marking of input places update)

output gate:

has a finite set of outputs and one input
output function (marking update)

Gates are introduced to permit greater flexibility in defining enabling and completion rules
Each input or output gate is connected to a single activity

Stochastic Activity Networks (SAN)



Cases associated with activities permit the realization of two types of *uncertainty:*

- uncertainty about which activities are enabled in a certain state is realized by cases associated with instantaneous activities
- uncertainty about the next state assumed upon completion of a timed activity is realized by cases associated with that activity

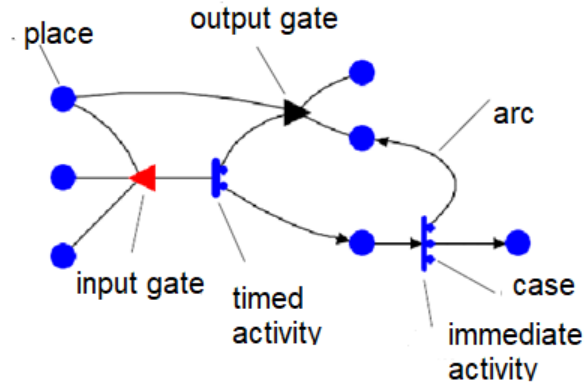
cases and case distribution:

instantaneous or timed activity may have mutually exclusive outcomes,
called cases, chosen probabilistically according to the case distribution of the activity.
Output gates allow the definition of different next marking rules for the cases of the activity

Stochastic Activity Networks (SAN)

Other extensions to PN:

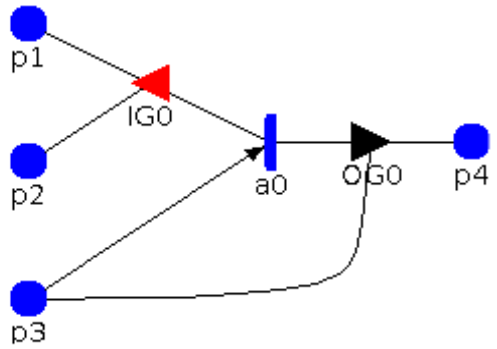
Modules
(describe a system as a composition/replication of modules)



Shared variables
(global objects that can be used to exchange information among modules)

Extended places
(places whose marking is a complex data structure instead of a non-negative integer)

Example

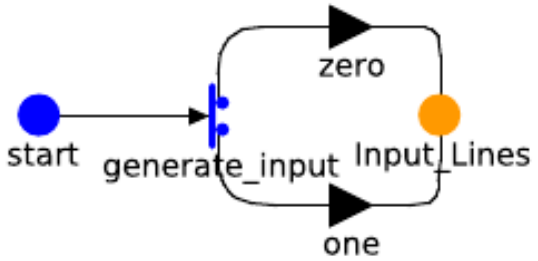


Gate	Definition
IG0	Enabling predicate if ((p1 → Mark() > 0 ∧ p2 → Mark() ==0) || (p1→ Mark()==0 ∧ p2 → Mark()> 0)) return 1; else return 0;
OG0	Output function p4 → Mark()++; if (p3 → Mark() == 0) p3 → Mark()=1;

Enabling predicates, and input and output gate functions are usually expressed using pseudo-C code.

Example

Example of Case: *activity generate_input*
Bit 0 or bit 1 is generated according to the probability assigned to the cases of the activity



The behavior of an activity network is a characterization of the possible completions of activities, selection of cases, and changes in markings.

Example

SAN evolution starting from a given marking M

(i) the instantaneous activities enabled in M complete in some unspecified order;

(ii) if no instantaneous activities are enabled in M, the enabled (timed) activities become active;

(iii) the completion times of each active (timed) activity are computed stochastically, according to the respective time distributions; the activity with the earliest completion time is selected for completion;

(iv) when an activity (timed or not) completes, the next marking M' is computed by evaluating the input and output functions;

(v) if an activity that was active in M is no longer enabled in M', it is removed from the set of active activities.

Mobius

Mobius tool provides a comprehensive framework for evaluation of system dependability and performance.

The main features of the tool include:

(i) a hierarchical modeling paradigm, allowing one to build complex models by specifying the behavior of individual components and combining them to create a complete system model;

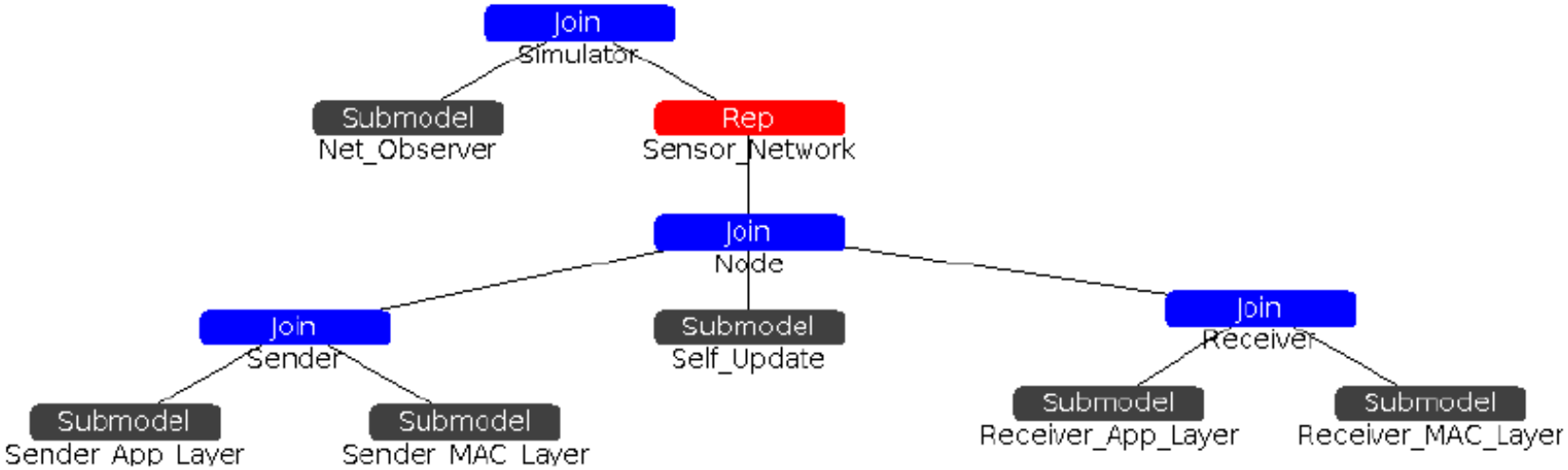
(ii) customised measures of system properties;

(iii) distributed discrete-event simulation;

(iv) numerical solution of Markov models.

Hierarchical model: an example

sensor network protocol + network observer



Rep operator:

constructs a number of replicas of an individual component

Join operator:

groups individual and/or replicated components.

Activities: sensor network protocol

Activities

Activity	Duration	Description
Sender MAC Layer		
`start`	instantaneous	transmission start
`test`	instantaneous	listening the channel
`backoff`	exponential	backoff period
`tx_chunk`	$c(\texttt{chunk\_length} \rightarrow Mark())$	transmit the preamble
`tx_data`	constant	transmit the data packet
`ready_to_send`	instantaneous	check interval end
Receiver MAC Layer		
`duty_on`	constant	wake up and channel check
`rx_chunck`	exponential	reception of chunck
`sleep`	$c'(\texttt{sleep\_length} \rightarrow Mark())$	sleep period between chunck and data packet
`rx_data`	constant	reception of a data packet
`duty_off`	$c"(\texttt{duty\_off\_length} \rightarrow Mark())$	duty off period
`restart`	instantaneous	check interval end/restart
Self_Update		
`time`	constant	periodic update of timers
`update`	instantaneous	update timers

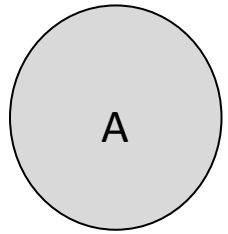
Simulation and Reward functions

Mobius allows system simulation and allows properties of interest to be specified with **reward functions**

ASSIGNING REWARDS TO STATES or TRANSITIONS defines a Markov reward model.

RATE rewards (in the case of assignment to states)
or
IMPLULSE rewards (in the case of assignment to transition)

Simulation and Reward functions



RATE rewards

Assume place A in the model is the only state that represents the failure of the system

Consider the performance variable Rel, which measures reliability

Generally the value of the performance variable is the **mean** of the values returned by the associated reward function

The reward funtion is:

```
if  (Model-name->A->Mark()==0) return 1;
                          else return 0;
```

Simulation and Reward functions

Reward function:
specifies how to measure a property on the basis of the SAN marking.

Measurements can be made at specific time instants, over periods of time, or when the system reaches a steady state.

A desired confidence level is associated to each reward function.

At the end of a simulation the Mobius tool is able to evaluate for each reward function whether the desired confidence level has been attained or not thus ensuring high accuracy of measurements.

Confidence interval: accuracy of the measure of the property

Solutions on measures

Numerical solver
model is the reachability graph.
Only for models with deterministic and exponentially distributed activities
Finite state, small states-space description

Simulation
model is a discrete event simulator.
For all models
- arbitrary large state-space description
- provide statistically accurate solutions within some user specifiable confidence interval