The Petri net modelling and analysis Tool Π-Tool

User Guide

Version 1.2.0, September 2013

Institute for Quality, Safety and Transportation

Hermann-Blenk-Straße 22, 38108 Braunschweig, Germany
fon: +49 (0)531 317 326 4
fax: +49 (0)531 208 567 11
info@iqst.de
www.iqst.de

Contents

List of Figures

1 Introduction

A good introduction can be found in this video tutorial (in German language).
Nowadays several description means -for instance fault trees, event trees, markov chains-, methods and software tools are used to perform RAMS (Reliability, Availability, Maintainability, Safety) analysis.

With the help of Petri nets, all four properties can be analysed using just one model. However, modelling real systems by means of simple Petri nets often turns to be cumbersome. Such an approach can be outstandingly put into practice using Π-Tool.

The Π-Tool supports Generalized Stochastical Petri Nets (GSPN) as well as Coloured Petri Nets (CPN). Besides hierachical models are supported, i.e. sub-nets can be created, which are able to communicate wich each other be means of fusion places and fusion transitions. Furthermore, Reliability Block Diagrams can be use to complement Petri Nets to enhance the modelling of complex systems with interconection -e.g. parallel or series- of several units.

The following analysis mothods are supported by Π-Tool:

- Model validation with the help of the Token Game, i.e. animated transition firing.
- Reachability graph calculation, for modells with up to 50 million global states.
- Assessment of stationary transition firing rates and place occupancy rates. This includes
  - Analytical solution of Markov chains for Petri nets with negative exponential pdf and immidiate transitions as well as
  - Monte Carlo simulation for all Petri nets and Reliability Block Diagrams.
- Transient analysis to assess the cummulative distribution function for the time to firing of the selected transition.

The usual modelling approach of Π-Tool uses the *Bottom-Up* principle. Petri nets are state based models, which comprises both logical and dynamic -i.e. time based, also stochastic- model features. The supported analysis methods simulate the system behaviour for a long time -i.e. long enough, that the specified confidence interval associated to the simulation results is achieved-. The main simulation output is the transition firing rates and place occupancy rates. These provide essencial information on the system reliability properties.

In section 2 we present a brief guide to use Π-Tool to create GSPN, Coloured Petri net and Reliability Block Diagram models. Then section 3 handles the multiple analysis features supported by Π-Tool. Finally section 4 provides an example on how a real system -a level crossing- can be modelled using Π-Tool.

2 Modelling

2.1 Generalized stocastic Petri nets (GSPN)

Generalized stocastic Petri nets are a kind of low-level (i.e. with anonimous tokens) Petri nets that support the modelling of time dependent events, both deterministic and stochastic. IEC 60300-3-1 explicitly list Petri nets as applicable for general dependability as well as risk and safety tasks.

2.1.1 Introduction

In Petri nets, active and passive elements are differentiated (see figure 1). The passive elements are called **Places**, they model conditions, e. g. distinguishable elementary states with certain duration. Places are represented graphically by circles. **Transitions** represent the active elements which change the elementary states. They are drawn as bars or rectangular boxes. Transitions are *activated* when the necessary conditions are fulfilled, i.e. when the incomming places (i.e. places connected with the transition by means of an arc going from the place to the transition) carry the required number of tokens. This required number of tokens depends on the type of the connecting arc. If it is a normal or a conditional (dotted) arc, the required number is equal to the arc multiplicity, which is by default one. If it is an inhibitor arc, the required number is zero, i.e. the place must be empty for the transition to get activated.

After the transition delay time elapses (see figure 2 and section 2.1.2 for the different kinds of transitions supported), the transition fires, and two things happen:

- For each incomming place connected to the transition by a normal arc, a number of tokens equal to the arc multiplicity is removed from that place. Note that for incomming places connected to the transition by conditional or inhibitor arcs, no places are removed.

- From each outgoing place (i.e. places connected with the transition by means of an arc going from the transition to the place) connected to the transition by a normal arc, tokens are removed. The number of tokens removed is equal to the arc multiplicity. Places have limited capacity (maximal number of tokens it may contain, user defined). This constitutes an additional constraint for a transition to be able to fire: the unused capacity (i.e. the difference between the current number tokens it contains and the maximal number of tokens it may contain) of the outgoing places must be at least as large as the multiplicity of the connecting arc.

Through switching a transition, i.e. the instantaneous event, places connected to the firing transiiton may change their marking, to that transitions connected to those places (including the firing transition itself) may get activated or deactivated. Figure 2 shows the different kinds transitions supported by GSPNs within Π-Tool. Further information on Petri nets can be found in (Petri, 1962; Schnieder, 1999; German, 2000; Slovk, 2006).

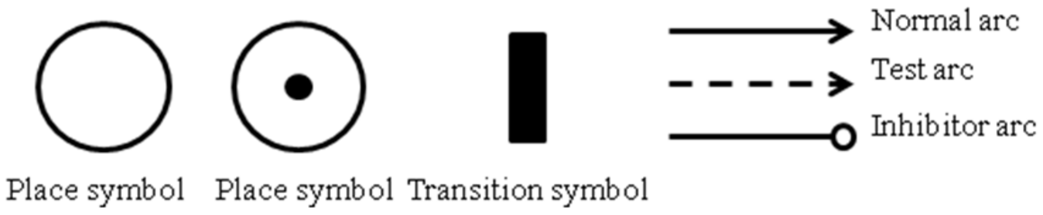


Figure 1: The basic elements of Petri nets and their graphical representations

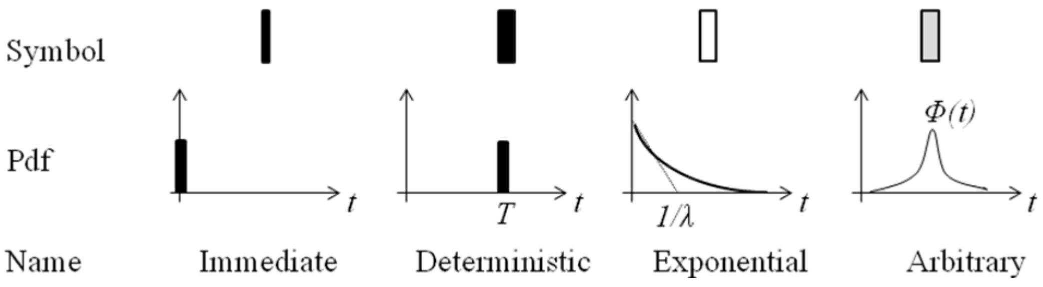


Figure 2: Petri net transition types: graphical symbols and corresponding modelled time distributions

In addition, Π-Tool also supports packages. This make possible to subdivide a Petri net into subnets. This is convenient when the model is so large that placing it in only one sheet would make it unmanageable, or when an object oriented modelling approach is desired. When subnets are used, the elements of different subnets can be connected with each other by menas of *fusion places* and *fusion transitions*. These are an additional representation of a place or transition. Fusion places are exemplarily used in the level crossing model presented in section 4.

2.1.2 Model building with Π-Tool

You can start a new GSPN model by selecting the Icon 🗎 or via the File menu. In the toolbar, there are selectable icons for each of the Petri net element types: places, arcs, transitions ad packages.

So, to create a place, just select the place icon ○ and click on the canvas at the position where the place should be created. To create an arc and a transition, just click on the place using the mouse middle button and release it at the position where the transition is desired. An arc and a place will be created. If yoor mouse does not have a middle button, you can also use the combination Ctrl key plus Left mouse button. A third way would be just to choose the arc icon ↗ and use the left mouse button.

In order to create sub packages, just choose the package icon 🗀 and click in the desired position. You can open the new sebnet by double clicking on the package or using the Petri net tree navigator on the upper left part of the window.

On the lower left part of the window you can find the object properties manager. With this widget you con parameterize all Petri net elements. the are as folows:

- Places
  - Name. Name of the local state this place represents, for example "System Up".
  - Capacity. The maximal number of token the place can contain.
  - Initial marking. Number of tokens in the place at the beginning of the simulation.
- Transitions
  - Name. Name of the event this transition represents, for example "failure".
  - Distribution. Type of time delay associated with the transition. It determines the time that must elapse between transition activation, i.e. the instant atr which all activation conditions for the transition are fulfilled, and firing. It can be deterministic

(immatiate of with a fixed delay value) or stocastic (negative exponential, normal, lognormal uniform, or weibull probability distribution function).

- Distribution parameters. These depend on the chosen distribution as follows:
    - Deterministic In case of deterministic transitions, the parameters are
        - Delay The time between activation and firing.
        - Priority This parameter is only relevant in case two immediate transitions are activated at the same time. In such case, the priority undefines which of them should fire first.
        - Wheight This paremeter defines the relative firing probability, in case more than one immediate transitions with equal priority are activated at the same time. The probability of each of these transitions to fire is its Wheight divided by the wheight of all other equal priority activated immediate transitions. This is useful for instance in case we would like to model to different types of failures, and it is known that, for examplye, failute type A occurs in 40% and failure type B in 60% of the failures. This could be modeled using the net presented in figure 3, where transition *failute type A* has wheight 0.4 and transition *failute type A* has wheight 0.6.
    - Exponential.
        - Rate. Firing rate λ, inverse of the mean.
    - Normal.
        - Mean value (μ).
        - Variance (σ^2).
    - Lognormal
        - Mean value (μ).
        - Variance (σ^2).
    - Uniform
        - From. Minimal possible firing delay.
        - To. Maximal possible firing delay.
    - Weibull
        - T. Characteristic lifetime, inverse of scale factor λ.
        - b. Shape factor.

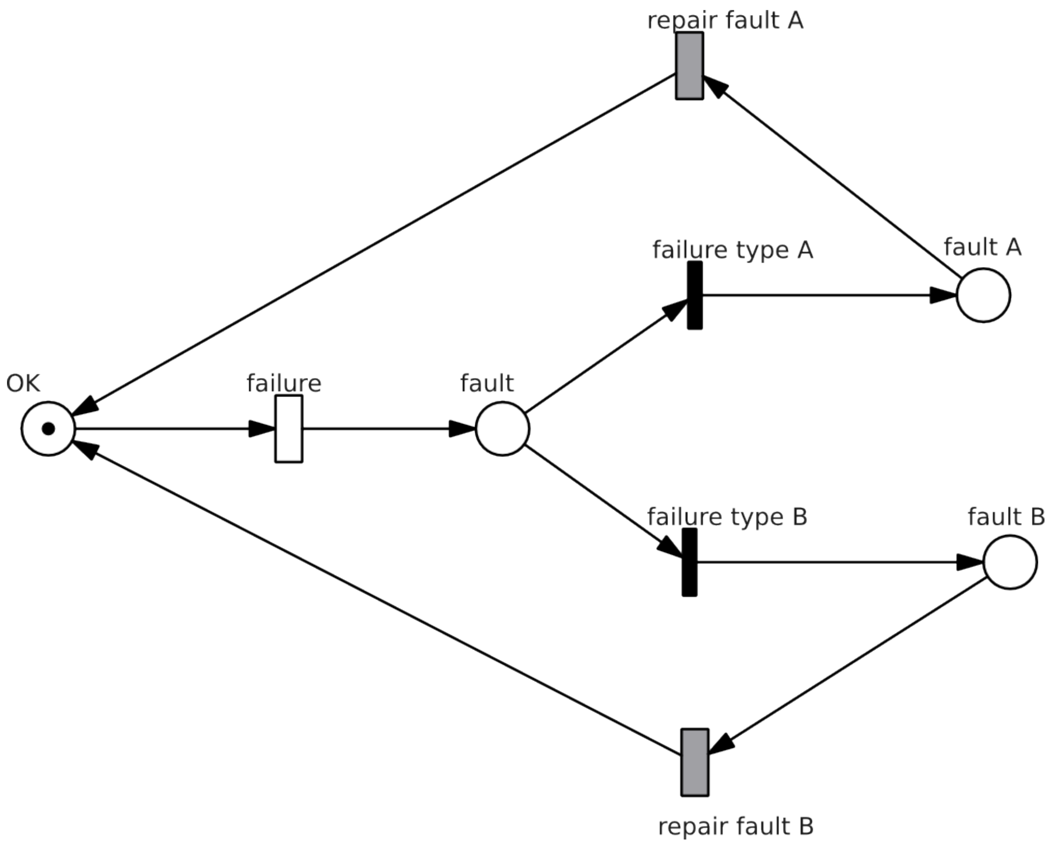


Figure 3: Model of a unit with two different failure types. The share of each failure type in the total failure rate is modeled by means of the *wheight* parameter of transitions *failure type A* and *failure type B*

Places and transitions can also be set to be *fusion places* or *fusion transitions* . These are representations of other places/transitions, and are usefull to connect elements in different subnets as well as to simplify the Petri net graph by avoiding long arcs, which eventually make the net unclear. To set a place/transition as *fusion*, just right-click on it and select *make fusion.* A tree with all net places/transitions will pop up for you to select the element to be fusioned. When you connect the fusioned element, it will have exacly the same (functional) effect as if you had connected the fusioned one.

To illustrate the use of fusion places, consideg the Peti net model of figure 4. It represents a motor which is repaired by a maintenance team, which is available at woking time only. The net has two subnets: the subnet of figure 5 represents the maintenance team availability, and the one in figure 6 represents the reliability of the motor itself. As you can see in figure 6, at the motor subnet the availability of the team is read out by means of a fusion place plus a conditional arc, so that the transition *repair* will only be active when the motos is faulty and the team is available.

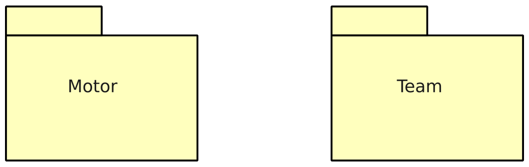


Figure 4: Model subnets: *Motor* for the motor reliability, *Team* for the maintenance team availability

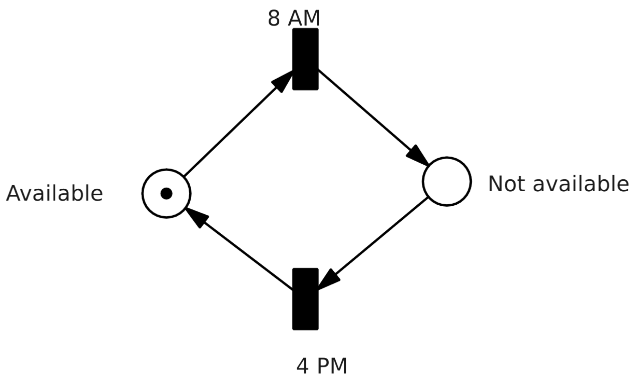


Figure 5: Maintenance team availability model. Since the team has fixed working time, the transitions are deterministic

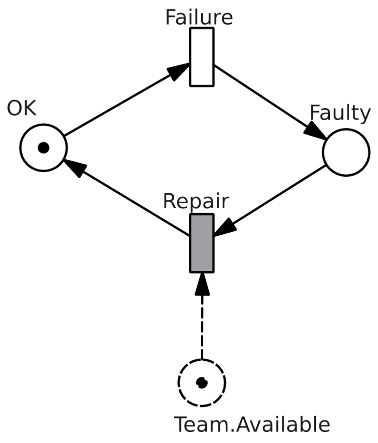


Figure 6: Motor reliability model. Note the use of a *fusion place* to make the repair depend on the maintenance team availability

After you have set your model, you can verify it by means of the *token game* (see paragraph 3.1) and analyse it using any of the analysis methods presented in section 3.

2.2 Coloured Petri nets

2.2.1 Introduction

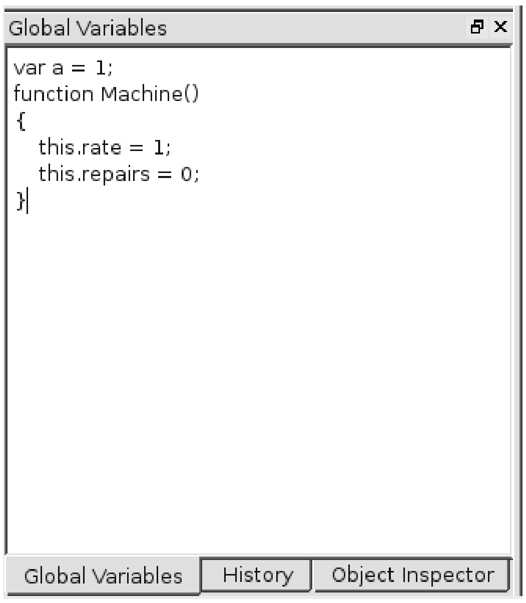
The Π-Tool also supports the generation and analysis of high-level, i.e. coloured Petri nets. The difference between low-level and high-level Petri nets is rooted in the fact that low-level net tokens do not contain any information, i.e. they are anonimous and indistinguishable, while in high-level nets each token is an object of a defined class -also called colour sets-. These classes can contain any user-defined variables. However the type of the variables is restricted. The following data types are supported:

- The three simple JavaScript data types, namely Number, bool, and String
- JavaScript Date
- Arrays
- Objects, that is, coulour sets

2.2.2 Model building with Π-Tool

The creation of a coloured Petri net involves the following steps:

- Declaration of global varialbes and colour sets. Colour sets are structured data types similar to classes or structures in object oriented programming laguages like C++ or Java. As shown in figure 7, gobal variables are declared with the keyword *var* and colour sets as object classes on JavaScript.
- For each place that should contain coloured tokens, the corresponding colour set must be defined, as well as the initial marking. When places are created, the have by default no colour set. In order to declare the desired colour set, right-click on the place and select *Set color*. Then write at the first line the colour set name, and on the folowing lines enter the initial values of the variables of each initial token. In each line one marking is initialized. The initial value of each single variable must be written in parenthesis, in the same order in which they appear



```
var a = 1;
function Machine()
{
   this.rate = 1;
   this.repairs = 0;
}
```

Figure 7: Declaration of global varialbes with initial value 1 and of the colour set Machine with variables *rate* and *repair*, with their respective initial values 1 and 0

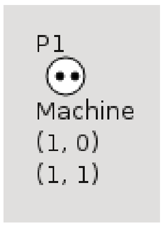


Figure 8: Declaration of the colour set *Machine* and of the initial marking with 2 tokens and the initial values $(1, 0)$ and $(1, 1)$

in the declaration of the colour set. Figure 8 presents an example of initial marking declaration.

- Each transition which removes or generates coloured tokens, has to be provided with three JavaScript scripts which determine exactly its behaviour: *isActive*, *calcNextTime* und *setPlaces*. The respective code editors can be opened by right-clicking on the transition and via the *Code* menu.

  Within the scripts, the oldest token of each of the incomming places is accessible via the place name. Incomming places are the places that are connected to the transition by an incomming arc, seen from the transition. If the connecting arc is normal, then the oldest token will be removed when the transition fires, that is, each coloured place works like a FILO (first in last out) stack.

As an example, consider the the simple coloured Petri net in figure 9. In any script of T1 the marking of the oldest token in P1 can be accessed simply as P1, as shown in script 1. In script 1, a variable called x is declared and assigned the value of the variable repair of the oldest token in place P1.

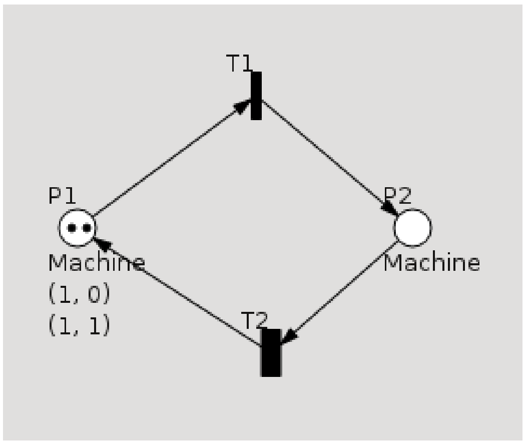


Figure 9: A simple coloured Petri net

Listing 1: Exemplary access to the marking of an incomming place

```
var x = P1.repairs;
```

Next the functionality of each of the three scripts is described.

- isActive determines additional conditions for the transition activation. As in low-level nets, the basic condition for the activation of a place is that the incoming places have the required number of incomming tokens. In coloured Petri nets, additional conditions can be stated in the *isActive* script. Thus, the Script *isActive* must return a boolean variable. For example, in script 2, for T1 in figure 9 to be activated it is required that the variable *Repairs* of the oldest token in P1 has a value equal or greater than 3.
- calcNextTime determines the transition firing time delay, i.e. the time that must elapse between activation and firing of the transi-

Listing 2: A simple *isActive* Script

```
return P1.repairs > 3;
```

tion. Thus, this script must return a number. In script 3, transition T1 of figure 9 will fire *P1.rate* time units after its activation.

Listing 3: A simple *calcNextTime* Script. The transition firing time delay is set to the value of the variable *rate* of the incomming place P1

```
return P1.rate;
```

– setPlaces determines the marking of the places that generated when the trasition fires. In the net of figure 9, when T2 fires, a token will be removed from P2 and a new one will be placed in P1. If script 4 is the *setPlaces* script of T2, then the variables of the token placed in P1 will have the value of the token removed from P2.

Listing 4: A simple *setPlace* Script. The token placed in P1 will be a copy of the token removed from P2

```
P3 = P2;
```

2.3 Reliability Block Diagrams

2.3.1 Introduction

Many systems make use of redundancy in order to enhance their reliability features. The different units of a redundant system can be represented by means of reliability blocks.

An outstanding feature of Pi-Tool is the possibility to combine Petri nets and reliability block diagrams in one model.

This makes possible to model individual units with Petri nets, and the series and/or parallel connections between the units with a reliability block diagram (RBD).

The combination of both representation means is achieved by means of a link of each reliability block to a Petri net place, i.e. the block will be Up when the pointed place is filled with at least one token, and Down if the place is empty. This way, the reliability of each block can be modelled with a Petri net. This extremely flexible moelling approach allows for modelling virtually any dependencies between reliability blocks.

2.3.2 Model building with Π-Tool

When a new model is created via the File menu, both a Petri net and a RBD are created. At first the RBD has only two blocks: Start and End. As usual in RBDs, the RBD will be considered to be Up whenever a path that goes from Start to End whose blocks are all Up.

After creating the Petri nets representing the system units, you can open the RBD and populate it with blocks by selecting the yellow icon with the label RBD from the toolbar. After you place each block, a menu window will request you to select the Petri net place this block will point to.

You can also add boolean logic to the RBD by means of the Logic blocks. To do this, select the yellow icon with the label Logic from the toolbar. Logic blocks have one parameter called Active Connections, which determines how many input blocks must be active for the logic block to be active. This way both AND and OR boolean logic can be implemented.

When you run the token animation via the Animation button in the toolbar, the reliability blocks will also change their colors according to the state of the place they point to. They will turn green if the place is filled with one or more tokens, or turn red if the pointed place is empty. This can help you verify the correctness of your RBD.

You can also run with RBDs the same analysis as with Petri nets, i.e. a steady state analysis and a transient analysis. The steady state analysis will provide you with

- failure rate,
- mean time between failures (MTBF),
- mean failure time and
- availability

for the whole RBD as well as for each of the reliability blocks.

The transient analysis generates a plot representing the cumulative distribution function of the RBD reliability.

3 Analysis

There are many Petri net analysis methods. Π-Tool supports three major groups.

3.1 Token game

The token game provides a basis for the verification of Petri net models. It consists of the visualisation of the transition firing sequence and the associated place markings. The token game can be initiated by clicking on the *Animation* button on the toolbar. Additionally, by selecting the Icon the *Script debugger* can be activated, which provides a debugging environment for debugging JavaScript scrips of coloured transitions.

3.2 State space based analysis

Π-Tool offers the state space based analysis features listed below. As such they concern structural model properties only, i.e. dynamic characteristics are not considered.

- For a RAMS analysis it is often interesting to assert if certain system states are reachable or not. This is made by Π-Tool by means of the reachability graph. Just select Reachability Graph in the Analysis menu.

- Similarly, you can quickly check if a given transition can ever fire, i.e. the transition is reachable from the initial Petri net marking. For this, press the *Animation* button in the toolbar, and right-click on the transition and select Shortest Path. If the transition is reachable, Π-Tool will also find the shortest possible firing sequence which would lead to the selected transition firing, starting from the initial marking. The shortest sequence will be shown in the *History* docking window in the lower left part of the Π-Tool main window. You can reproduce the sequence by clicking on the *Step Up* and *Step Down* buttons in the *History* docking window.

- Π-Tool can also calculate the net's place invariants. In the *Petrinet tree* docking window on the upper left part of the Π-Tool main window, just right click on the *Petrinets* element and select *P-Invariants*

3.3 Steady state Analysis

Particularly interesting results are the steady state, i.e. stationary, transition firing rates and place occupancy rates (the fraction of the total simulation time at which the place has had at least one token). To open the steady state analysis tab, just select *Steady state analisis* from the *Analysis* menu. On the left you will find two formulars. These will present the firing rates and the place occupancy rates. On the right hand, there is on the top a series of parameters to configurate the Monte Carlo simulation -see their description below-, a histogram of the simulated firing times of the observed transition -see definition of observed transition below-, and the buttons to start/end a simulation and save the results into files.

Π-Tool supports two steady state analysis methods:

1. Markovian Analysis. The Petri net is transformed into a Makov chain and the firing rates and place occupancy rates can be thus calculated by solving the Markov chain analytically. This is possible only for nets containing immediate transitions, i.e. deterministic transitions with delay equal to 0- and/or transitions with negative exponential pdf. However, in case the net has non markovian transitions -transitions with deterministic delay greater than 0 and/or transitions with pdfs other than the exponential distribution, for exmaple normal distribution- Π-Tool will replace internally these transitions by a markovian approximate and will retrieve approximate results. For exmaple, transitions with delay T are replaced by transitions with negative exponential distribution and $\lambda = 1/T$. In case any such replacement is necessary, a warning message will be displayed to let you know that the results are not exact.

2. Monte Carlo Simulation. For the general case, i.e. nets with transitions with arbitrary pdf and deterministic transitions with delay greater that 0, a Monte Carlo simulation will retrieve the same results. However, these results will be associated with a confidence interval, which can be set by the user before starting the simulation. The narrower the confidence intervall, the longer the simulation duration.

The Monte Carlo simulation takes some input parameters which can be modified in the top left sector of the simulation tab, namely:

- Observed transition. The Monte Carlo simulation results are associated with a confidence interval. In fact, the confidence intervall is difference for each result, i.e. for each firing rate and occupancy rate. Therefore, the user must choose for which trasition the selected confidence interval must hold. This is the *Observed transition.* The user should choose the transition which represents the event we are interested in. For example, if we are interested in assessing a failure rate, then the transition representing that failure should be chosen.

- Max samples. This is just an upper bound for the simulated transition firings. The simulation will stop if this number is reached, even is the desired confidence interval has not been reached yet.
- Simulation duration. This is just an upper bound for the simulation duration. The simulation will stop if this time is reached, even is the desired confidence interval has not been reached yet. By default it is disable. In order to enable it, just ener a value different from 00:00.
- Confidence interval. The desired confidence interval for the firing rate of the observed transition.
- Number of threads. In case you use a multicore processor, you can increase the simulation speed by setting the number of simulation threads.

The simulation output consists of many data, namely:

- The two formulars with the calculated transition firing rates and place occupancy rates.
- The histogram of simulated times between firings of the observed transition.
- The achived values for the simulation parameters: simulation samples, simulation duration, achived confidence interval (can be smalle than desired if the simulation was very short, or greater than desired if the simulation was stopped for havien reached the maximum simulation duration or the maximum number of samples), number of firings of observed transition and observed transition firing rate (this is also shown in the transition firing rates formular).

All this data can be saved into files in csv-like (readable by any apreadsheet application) format by selecting the button *Save Results.*

3.4 Transient analysis

Sometimes it is interesting to know, for a time intervall $0-T$, the probability of an event to have happened for the first time. Fro example it could be interesting to find out the probability that a failure has occured after a system has been working for 1, 2, 3 ... 10 hours. This is usually known as transient analysis and is also supported by Π-Tool. Just select *Transient Analysis* from the *Analysis* menu. A new simulation tab will open. It is just the same as the transient analysis tab, but in this case we focus explusively on one trasition, so the formulars are not present. The input parameters Observed transition, Max samples, Simulation duration, Confidence interval and number of threads are exactly the same as in the steady state analyis (see above). However there are two new, very important parameters:

- Transient step. It defines the resolution of the curve be calculated, i.e. how far one point will be from the other on the time axis. For example, if the time step is 0.1 then the first point will be for time 0.1, the next one at 0.2, the next one at 0.3 and so forth.

- Transient points. It defines the total number of points the curve will have. For example, if the *Transient step* is 0.1 and the transient points are 100, the curve will start at time 0.1 and and at time 10.0.

After setting the input parameters as desired, you can start the simulation by clickong on the button *Transient Analysis* on the bottom of the tab. The output of the transient analysis is a curve that represents, for each time point T, the probability that the observed transition has fired at least one time within the time interval $0-T$. The curve point can be saved into a file by choosing *Save Results* on the bottom transient simulation tab.

4 Modelling example with Π-Tool

In this example, the simplified model of a railway level crossing is presented. It is a hierarchical model, with the following sub-nets:

1. Car traffic

2. Railwaay traffic

3. Level-crossing protection system dependability

4. Accident occurrence

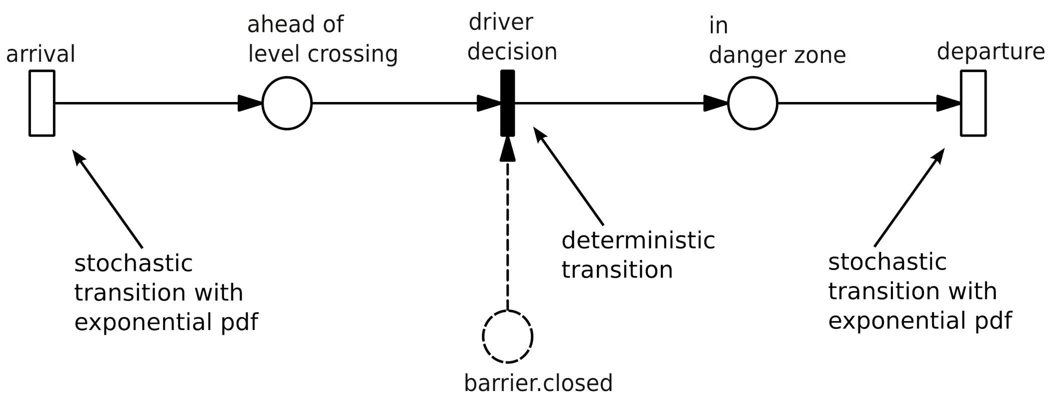


Figure 10: Simple Petri net model of the car traffic at a railway level crossing

The car traffic is modelled as shown in figure 10

The transition *arrival* generates cars and follows a negative exponential pdf with a firing rate $\lambda = 4$ cars/hour. Then the car driver decides if he enters the level crossing danger zone. This decision is represented by the transition *driver decides*. The time it takes for the car to cross the danger zone depends on the transition *departure*, which is normally distributed.

For the railway traffic a similar model is proposed, see figure 11.

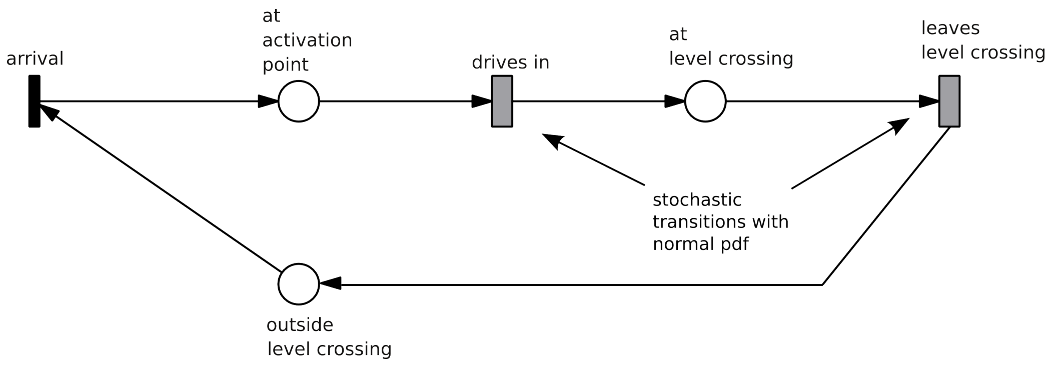


Figure 11: Petri net model for the railway traffic

The Transition *arrival* generates trains and follows a negative exponential pdf with a firing rate $\lambda = 2$. As soon as the place *at activation point* is filled, the level crossing protection is activated. The time it takes for the train to reach the danger zone depends on the transition *enter danger zone*, which is normally distributed. Likewise, the time it takes for the train to cross the danger zone depends on the transition *departure*, which is normally distributed with parameters $\mathcal{N}(80; 10)$ km/h.

The level-crossing protection system is modeled as shown in figure 12.

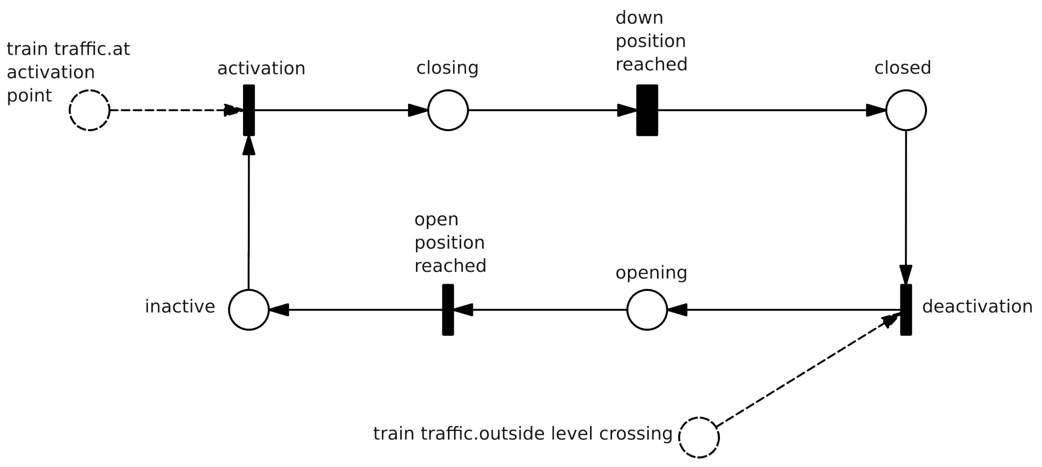


Figure 12: Petri net model of a full barrier

The dependability of the level-crossing protection system is modeled as shown in figure 13.

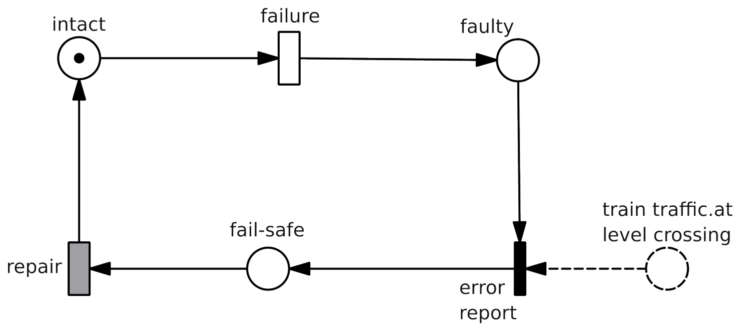


Figure 13: Dependability of the level-crossing protection system

As soon as a train enters the level crossing and notices that it is not protected, the train driver reports this failure to the traffic controller, which sets the level crossing protection in fail-safe mode, for example by blocking the track to any upcoming train.

The accident occurence model is presented in figure 14.

If at any time a car and a train are simultaneously at the danger zone, an accident occurs.

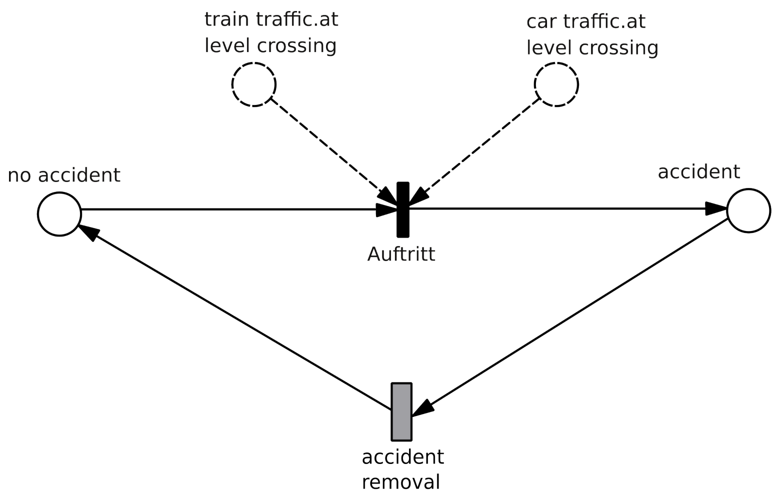

train traffic.at level crossing
car traffic.at level crossing
no accident
accident
Auftritt
accident removal

Figure 14: Petri net model of the accident occurence