# Handout III - ICTiB/Process modelling

Pieter Kwantes

January 17, 2010

## Contents

# 1 Exercises

## 1.1 Getting acquainted with CPN Tools

To get acquainted with CPN-tools we will start by creating the Petri net model of the Driving school, described in the book "Workflow Management. Models, Methods and Systems" chapter 2, exercise 2.5. a), in CPN/tools. We already did that exercise using Woped in the second exercise class (see Handout II/Exercise 1.1.). We will now translate that same model into a Coloured Petri net using CPN-tools. You may recall that the model we created in Woped was a PT-system, which has only "black" tokens. However, we could consider a PT-system to be a Coloured Petri net with just one "colour", i.e. black.

The next assignment, exercise 1.1.1. will help you to build your first CPN-model. You can use the "getting started" web page of CPN Tools for reference. The address of the web page is:

`http://wiki.daimi.au.dk/cpntools-help/getting_started_with_cpn_.wiki`

On the "getting started" web page of CPN Tools you will find a link to the **main help** which contains a lot of detailed information on working with CPN tools. You can use this for reference when you make the next assignment, exercise 1.1.1., translating the "Driving school" model you made in Woped into an equivalent model in CPN-tools.

**1.1.1. Creating a model in CPN tools** To create the "Driving school" model in CPN tools you should execute the following steps:

**Step 1: Start CPN Tools** After you start up CPN Tools you should see the *user interface* as shown in figure 1.

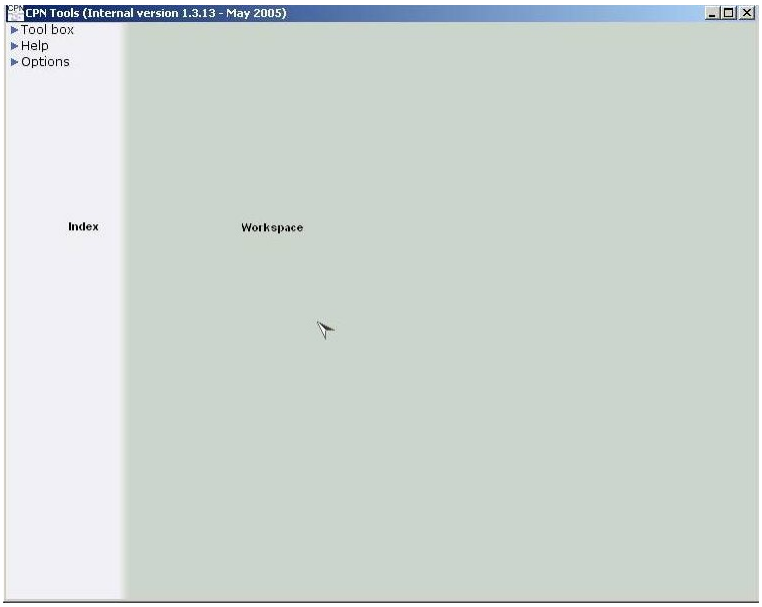


Figure 1: the user interface of CPN tools

If you click on the *Tool box* menu item, you should see the Toolbox sub menu as shown in figure 2.


Figure 2: the Tool box sub menu

**Step 2: Create a new Net** Use the *Net* option under the *Toolbox* menu to create a new net. Just drag and drop the menu item to the right into the workspace. A small window (*binder*) with a number of menu sub options will appear in the workspace as shown in figure 3. Choose the *create new net* option (i.e. the option in the upper left corner). A new (net) binder

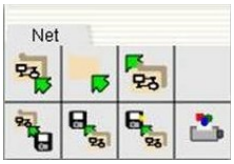


Figure 3: the "Net" option in the "Toolbox" menu in CPN tools

will then appear, as shown in figure 4, which allows you to define a new net .

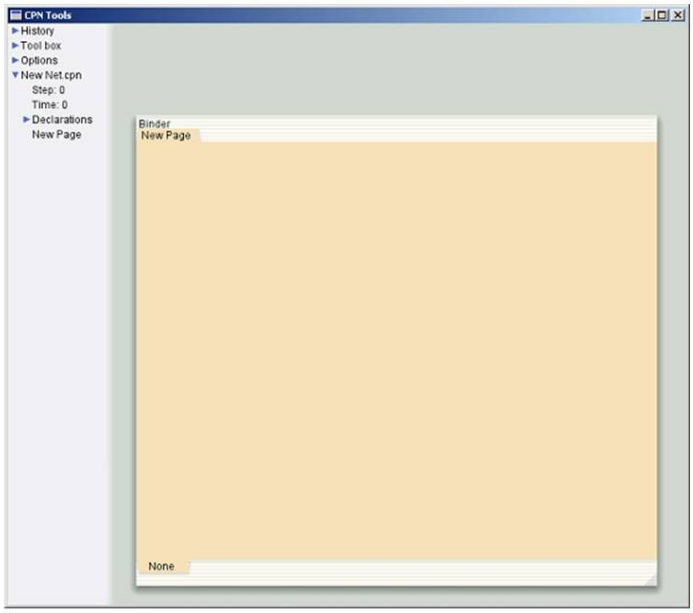


Figure 4: An empty page in CPN tools for defining a new net

**Step 3: Define coloursets** For this exercise we will only use one of the built-in coloursets of CPN-tools: UNIT. You must now extend the definition of the colourset UNIT which you can find under the menu *Declarations*, shown in figure 4, with the clasuse **with e**. The declaration will then look as follows: **colset UNIT = unit with e;**. This now means that you have defined a colourset with one constant value **e**. This constant value **e** we will use to represent our black tokens.

**Step 4: Create the places in the net** Open the *Create* submenu under the *Toolbox* menu (again drag and drop), shown in figure 5. To create places in the net you must select the oval shape in the *Create* submenu and drop it in the new net binder you created earlier (shown before in figure 4).

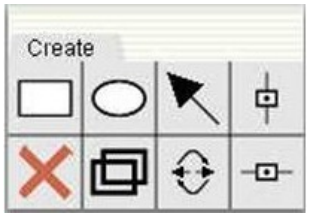


Figure 5: The Create submenu

Do that as often as you need places (as many as you had in the Woped version of the "Driving school" model), and deselect the oval shape when you are done. Next you must define the correct colourset for each place. You can do that by selecting a place and then push the *Tab* key once. The text "PLACE TYPE" will appear. Then push the *arrow down* key as often as you need to select the correct type : UNIT. By doing this CPN-tools now knows that this place only contains tokens of type UNIT, i.e. "black" tokens.

**Step 5: Add transitions** Add the transitions you need (as many as in the Woped solution) using the same *Create* submenu you used to create the places in step 4, but now selecting the rectangular shapes.

**Step 6: Complete the net** Connect the places and the transitions using the arrow option in the *Create* binder. In CPN-tools you must specify an *expression* for each arrow. This expression specifies the number and type of tokens that can be *bound to* ("transported across" so to speak) that arrow. For this exercise each arrow simply has the expression **e**. This means that each arrow will " transport" one black token.

**Step 7: Put the initial marking in the net** The net must have an initial marking before it can do anything.Now for each place you must define its initial marking. You do that by selecting a place and then push the *Tab* key twice. The text "INIT MARK" will appear. Here you specify the

number of tokens in the place as initial marking. One token is specified as 1‘e. Two tokens is specified as 2‘e, etcetera.

**Step 8: Save the model**

**1.1.2. Executing the model** One of the advantages of modeling processes with Petri nets, like you just did, is that Petri net models in principle are *executable*. This allows you a visual inspection of the *behaviour* of the model to determine whether that is according to your expectations. You can use the *Simulation* submenu, shown in figure 6 under Toolbox menu for executing the model. You can start with the single step option, the one exactly in the middle.


Figure 6: The simulation menu

**1.1.3. Extension of the driving school model with colour** Create the Petri net model (a Coloured Petri net) described in the book "Workflow Management. Models, Methods and Systems" chapter 2, exercise 2.5. b) .

To extend the model, you must execute the following steps:

**Step 1: Define appropriate colourset** In this assignment the driving school students must take 10 lessons before taking the exam and will drop out if they fail 3 times. To account for this extra information in your model you must keep track of the number of lessons each student has taken and how many exams he has failed. Therefore you must define a new colourset, eg. STUDENT, to represent this information. This colourset represents the set of driving school students, each which has three relevant attributes: *(student id, number of lessons taken, number of exams failed)*. You can define such a colourset in CPN tools using the menu option *Declarations* which you encountered earlier, as follows:

**colset STUDENT = product INT * INT * INT;**

**Step 2: Define variables** You need to define variables to "transport" the tokes defined in your colorset. You could for instance declare two variable *s and t* of type STUDENT, below the declaration of the colourset STUDENT, as follows

**var s,t: STUDENT;**

**Step 3: Assign colourset** Next, you assign each relevant place in your model this new colourset STUDENT (the same way that you assigned the colourset UNIT in the exercise 1.1.1. step 4).

**Step 4: Assign expressions** You have to assign an expression to each arc in the model (*arc inscriptions*), the same way that you did in exercise 1.1.1. (step 6) when you assigned the constant **e** to each arc, to bind the tokens to ("transport across") the arcs. In this case each token of type STUDENT can be bound to the variable **s** you defined. To adjust the model from exercise 1.1.1. you just have to replace each occurrence of the constant **e** that you have assigned to an arc, by the variable **s**.

**Step 5: Counting of lessons and exams** There are a number of ways you can keep track of the number of lessons and exams taken by each student. One way, that is explained here, is to create *transition inscriptions*. You can do that by selecting a transition and then push the *Tab* key until you see the inscription template as the one shown in figure 7.

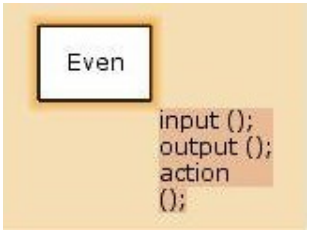


Figure 7: Transition inscription

Then you should fill in the template, eg. to count the lessons, as follows:

*input(s)*;
*output(t)*;
*action((#1 s,(#2 s)+1,#3 s)*;

You can understand the meaning of this by first recalling that the variable *s* in the input clause consists of three numbers *(student id, number of lessons taken, number of exams failed)*, because that is the way you defined the colourset that you assigned to this variable. When the transition, with this inscription attached to it, fires, the first number in the variable *s* : *#1 s* is copied to the first number in variable *t* in the output clause. The same happens to the third number in the variable *s*. The second number, the number of lessons, is increased by one. So if the input token was *(1,0,0)*, representing student with id *1* with zero lessons and zero exams, the output token will be *(1,1,0)* representing the same student with one lesson. You should of course take care that you assign the

inscription to the correct transition, eg. the transition that models the end of a lesson. In a similar fashion you can count the exams (remember to reset the lessons count after each exam!).

**Step 6: Routing of tokens** when a student has taken 9 lessons, he cannot take an exam yet, but first has to take another lesson. If he has taken 10 lessons, then he must not take another lesson, but take the exam. So the route taken by the token, representing the student, through the net depends on the values in the token. To model that you need to add another type of transition inscription, which are called **guards**. You can do that by selecting a transition and then push the *Tab* key until you see the inscription template consisting of two square brackets on the left upper corner of the transition. An example of a guard you could define on the transition that represents the taking of the exam is : *[(#2 s) = 10]*. This means that if the second number in the input variable *s*, i.e. the number of lessons, is equal to 10, only then can the transition fire.

**1.1.4. Extension of the driving school model with time** Create the Petri net model (a Coloured Petri net) described in the book "Workflow Management. Models, Methods and Systems" chapter 2, exercise 2.5. c). You can do that by extending the model you created in exercise 1.1.3. You will have to extend the model in two ways:

- Change the colourset STUDENT into a timed colourset by adding the word "timed" at the end of the declaration of the colourset as shown below :

  **colset STUDENT = product INT * INT * INT timed;**

  This will ensure that CPN tools will automatically maintain timestamps on each token fo type STUDENT.

- Add time clauses to expressions. For example, if you want to model the fact that a lesson at the driving school has a duration of one hour, you can attach a time clause to the outgoing arc of the transition "start lesson", that looks lie this ***s@+60*** (where *s* is the variable of colourset STUDENT). This will ensure that each token produced by the transition "start lesson" will get a time stamp that is equal to the *firing time* of the transition + the value in the time clause, i.e. 60 minutes. This produced token will then be available for the next transition, eg. "end lesson", as indicated by the timestamp, i.e. 60 minutes after the lesson started.

Execute the model again and see what happens. Try to explain the value of the time stamps you see.

## 1.2 Performance analysis of an assembly line

Consider (part of) a business process of producing a car on an assembly line: assume that three of the steps in the assembly line are to install the engine, install the hood, and install the wheels (in that order). A car on the assembly line can have only one of the three steps done at once. After the first car moves on to wheel installation, the second car moves to the hood installation, and a third car begins to have its engine installed. There are three different installation stations, an engine station, a hood station and a wheels station, each operated by one worker. Each station can handle one car at a time.

We want to analyze the performance of the assembly line with a timed Petri net model. It is given that engine installation takes 18 minutes, hood installation takes 4 minutes, and wheel installation takes 8 minutes. You can assume that all cars are handled the same in the assembly line. Assume further that in the initial state of this part of the assembly line there are 4 unfinished cars waiting at the first station. Figure 8 shows a timed PT system (created with CPN Tools) of the engine installation station of the assembly line.

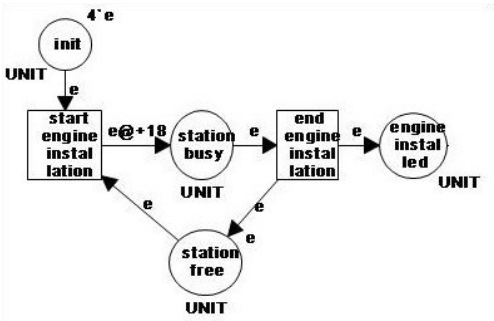


Figure 8: A timed PT-system of the engine installation station

**1.2.1.** how much time does it take to produce one single car?

**1.2.2.** make a sketch on paper of a timed PT system of the part of the assembly line with the three stations described above.

**1.2.3.** assume that the first task (engine installation) on the first car starts on a working day at time 09.00 (model time $t = 0$). The initial marking (at $t = 0$) of the Petri net in figure 8 is given next[1]:

$\{(init, 4`e@0), (station\_busy, empty), (station\_free, 1`e@0), (engine\_installed, empty)\}$

Describe the state, i.e. the marking (don't forget the time stamps on the tokens!) that the Petri net model can have reached respectively at 9.00, at 9.30, at 9.50 and at 10.10 (i.e. model time $t = 0$, $t = 30$, $t = 50$ and $t = 70$). We assume that the transitions are *eager*, which means they fire as soon as they can (i.e. when they are *enabled*)

**1.2.4.** How can you calculate average completion time of the four cars from the information in the Petri net when all cars are finished? Explain and do the calculation. Calculate also the average waiting time.

**1.2.5.** How many cars can be produced in one working day (assuming 8 working hours in one working day) ? Explain.

**1.2.6.** Now suppose you can add one extra station (either an engine station, a hood station or a wheels station) and a worker to operate it, which station would you add, if you wanted to maximize the amount of cars produced per day? How would you change the Petri net model to count for this extra station? How

[1] Using the CPN-tools notation see e.g. sheets 8 and 9, 6th lecture.

many cars a day can you produce after adding the extra station? Explain your answer.

**1.2.7.** Create the Petri net you have drawn as a solution to exercise 1.2.2. in CPN tools.

**1.2.8.** Verify your solution to exercise 1.2.3. by executing the Petri net you created in exercise 1.2.7. (You might need to add a "clock" to your model to answer the questions. You can do that by simply linking two transitions, where one transition increases the time by one when it fires.)

**1.2.9.** In this exercise you will prepare the Petri net model of the assembly line for automated performance analysis with CPN-tools. First of all you need to extend the model with a so called "data collector" that will gather the measurements of the model simulations. Because you want to calculate the average completion time for the cars running of the assembly line, you will have to gather the completion time for each car. You can do this by gathering the time stamp of each token when it reaches the end of the assembly line. You can do that in CPN tools by using the "data collector" option in the "monitoring" menu (see figure 9) and apply that to the transition that marks the end of the assembly line. This will result in a template for a monitor declaration. We will have to

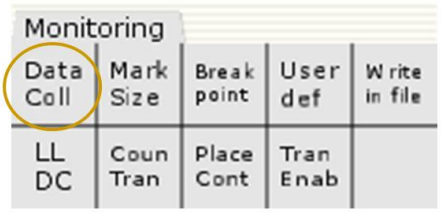


Figure 9: the "data collector" option in the "monitoring" menu in CPN tools

modify the definition of the observer function so that it will look like this:

```
fun obs (bindelem) =
let
  fun obsBindElem (DSp1'end_exam (1, {})) = intTime()
      | obsBindElem _ = ~1
in
  obsBindElem bindelem
end
```

The observer function as defined above will lookup the model time, using the function intTime(), when the transition to which you attached the data collector fires and and save the token and the model time for analysis.

We also have to define the function intTime() as a separate declaration follows:

```
fun intTime() = IntInf.toInt(time())
```

**1.2.10. Performance analysis with CPN-tools** In this exercise you will use the features of CPN tools for performance analysis to verify the correctness of performance statistics you derived by hand in exercise 1.2.4. Extension of the model with a data collector as we did in the previous exercise enables us to measure the completion time of the process. We are now ready to sun a simulation of the model. You can do that by using the simulation submenu, and activate the fast forward option. After you have run the model you can find a performance report in a subdirectory (called "output") of the directory where the model is saved. View this performance report and interpret the information in it. It will give you the average completion time.

## 1.3 Comparing process alternatives

This exercise is about performance analysis of alternative process models in order to choose an optimal process design. We will use the alternatives presented in the book of van der Aalst, "Workflow Management: Models, Methods and Systems" in paragraph 4.4. summarized in figure 4.30. The exercise is to create a model of each of the alternatives, run a simulation of each of the models and compare the results. (Hint:To model the arrival of 20 cases per hour as a *negative exponential* process in CPN tools you can create a transition with a time clause on one of its outgoing arc $v@+(poisson(3.0))$, where $v$ is some variable defined to bind the tokens representing the cases. This means that there will be a delay that is distributed around 3.0 in a negative exponential way, resulting in on average 20 cases per hour.)

## 1.4 Advanced features for performance analysis with CPN Tools

**1.3.1. Measuring performance of the driving school model** In this exercise we will run the model of the driving school to assess its performance.

We will be analyzing the "completion time" of the process, which we will define here as the time between the start of a series of lessons for one student and the end of the associated exam (which need not be successful).

**1.3.1.a. Extend the model with a data-collector** We will have to change the model slightly and extend it with a data-collector to gather the data that is produced by execution of the model. We will need that data to calculate the completion time as defined above.

First of all we have to extend the colourset we have used to represent students ( in exercise 2.5. b) in the book "Workflow Management. Models, Methods and Systems" chapter 2), with an extra colour "start time" (INT), to represent the start of a series of lessons.

Secondly we have to attach a so called "data collector" to the transition that we have used to model the end of the exam. We can do that in CPN tools by using the "data collector" option in the "monitoring" menu (see figure 9) and apply that to the "end exam" transition.

This will result in a predefined monitor declaration. We will have to change the definition of the observer function so that it will look like this:

```
fun obs (bindelem) =
let
  fun obsBindElem (DSp1'end_exam (1, {s,t})) = (intTime() - #4 s)
      | obsBindElem _ = ~1
in
  obsBindElem bindelem
end
```

The variable $s$ is the variable attached to the input arc of the transition "end exam"and the variable "t" is attached to the output arc of transition "end exam". The observer function as defined above will calculate the difference between the model time when the transition "end exam" fires and the start time we saved in the token that is being process and that is bound to the variable $s$ (i.e. $\#4\ s$ gives the start time).

We also have to define the function intTime() as a separate declaration follows:

```
fun intTime() = IntInf.toInt(time())
```

Thirdly we will have to set the start time for each student when he begins a new series of lessons to the current model time given by the function $inTime()$.

**1.3.1.b. Gather data by running the model** Extension of the model with a data collector as we did in the previous exercise 2.2.a. enables us to measure the completion time of the process. But before we are going to run the model we will simplify it a bit, to make analysis a bit easier. We eliminate the option to

"drop out" and we start by assuming that there are 10 students, that there is one examiner and one instructor. Next we run the model by using the "simulation menu" (50 steps option, double speed) until all students passed the exam (for the third time). After we have run the model we can find a performance report in a subdirectory (called "output") of the directory where the model is saved. View this performance report and interpret the information in it. It will give you the average completion time.

**1.3.1.c. Analyze the influence of increase in capacity** Rerun the model 1.3.1.b. with respectively 2, 3, 5 and 10 instructors, write down the average completion time for each of these cases and explain the results.

**1.3.1.d. Analyze the influence of resource flexibility** The model in 1.3.1.b. assumed that an instructor, only can give instruction, and an examiner, is only allowed to take exams. Now change the model so that the instructor can also take exams and the examiner can also give lessons. Then rerun the model and write down the results and compare them with the results you found in 1.3.1.b. and 1.3.1.c. Explain the results.

**1.3.2. : Increase reliability of measurements by using multiple subruns** Do the performance analysis described in the Queue system example. You can find the example net in the subdirectory "CPN Tools/samples/Queuesystem"

**1.3.2.a. Manual simulation** First execute the Queue system by using the fast forward option on the simulation menu. Then investigate the Performance report ($PerfReport$) in the sub directory named "output" in the directory where the Queue system model is saved.

**1.3.2.b. Multiple subruns** Execute the statement shown in the figure below. You will notice that after you executed this statement, a subdirectory in the


Figure 10: Statement to execute a simulation with multiple subruns

output directory is created named "reps_n", whereby $n$ is increased by one each

time you run the statement. The subdirectory "reps_n" contains the output produced by CPN tools during the $n$th execution of the statement above. Within the subdirectory "reps_n" you will find a performance report $PerfReportIID$ and another 5 subdirectories "sim_m", where $m$ is 1 to 5, containing the output of 5 simulations of the Queue-system. Each subdirectory "sim_m" contains a performance report $PerfReport$ with the performance statistics based on data gathered during one simulation. The $PerfReportIID$ is based on the data from all 5 simulations.

Now write down the average queue delays calculated for each of the 5 simulations you just executed and the associated standard deviations. Also write down the average queue delay and standard deviation based on all 5 simulations. Explain the difference between the first 5 sets of numbers and the last one.

**1.3.3. : Multiple subruns per model variant** Do the performance analysis described in the Queue system configuration example.

**1.3.3.a. Manual variation** In this exercise you will analyze the influence of changing the number of servers (NOS) and the distribution of processing time of the servers in the Queue system configuration example. Run the model 4 times by executing the statement $CPN$'$replications.nreplications$ 5 4 times, each time with a different configuration as described below:

1. NOS = 4, Average processing time = 360, distribution of processing time = DISCRETE
2. NOS = 5, Average processing time = 360, distribution of processing time = DISCRETE
3. NOS = 4, Average processing time = 180, distribution of processing time = DISCRETE
4. NOS = 5, Average processing time = 180, distribution of processing time = DISCRETE

Write down the average queue delay you find by running the simulations for each different configuration.

**1.3.3.b. Automated variation** Run the model 4 times by executing the statement $simulateConfigs(5)$ one time and write down the results like you did in the previous exercise.

## 1.5 The Lista order process

In this exercise you will start with the creation of a Petri net model in CPN tools of the Lista order process based on the sketches you made in the previous exercise classes.

**3.4.1. Create appropriate colourset** The Lista order process is, of course, all about processing of orders. The orders that are handled by the order process can be considered as the **cases** in the workflow. The first thing then to do is to define an ORDER colourset. We can start simple and define this colourset to contain only "black" tokens, the similar to way we did it in exercise 3.1.1. step 3. To do that we include the following declarations:

**colset UNIT = unit with e;**
**colset ORDER = UNIT;**
. You should also add a declaration of a variable of type ORDER, eg.
**var o1: ORDER;**
We can extend the colourset ORDER at a later stage.

**3.4.2. Create top level Petri net model** Create transitions, like you did for exercise 3.1.1. step 4, but only for entities/processes on the highest level of your dataflow diagram (eg. entities like "LISTA" and "Customer"). Next create the places to connect these transitions, like you did for exercise 3.1.1. step 3, to represent the communication between the entities (based on the dataflow diagram and the first sketch of the Petri net model you made in the previous exercise classes). Finally, you connect the places and transitions with arcs,like you did in exercise 3.1.1. step 5. However, in this case, you should add the expression **o1** to each arc, in stead of **e** like you did in exercise 3.1.1. step 5.

**3.4.2. Create lower level Petri net models** Now you need to create the lower levels of you Petri net model. For instance you want to zoom in on the "LISTA" entity you modeled and model the "insides" of it. For instance you can model the departments, of which LISTA is composed. To do this you can use the *Hierarchy submenu* in the *Toolbox menu*, which is show in figure 11. To

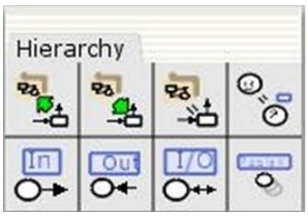


Figure 11: Hierarchy submenu

create a subnet for a transition, "zoom in" on a transition, you should select the *move to subpage* option in the upper left corner of the submenu shown in figure 11. After you selected the option, you apply it to the transition you want to zoom in to. CPN tools then creates a new page which contains the transition you selected and all the places attached to it. You can now delete the transition

on this new page, and create a subnet that represents a lower level detail of the transition with which this page is associated.

**3.4.3. Complete the model** For each transition on the toplevel you can create a subnet as described in exercise 3.4.2., if that is appropriate. You can repeat the procedure for the transitions on the subnets you created. You will then end up with a tree-like structure of subnets, where the top-level Petri net model is the root of that tree. It is certainly possible that, while working on a subnet, somewhere in the lower levels of the tree, you discover that you have forgotten a place, a transition or some connection on a higher level. In that case you need to switch to the "bottum-up" level development feature of CPN-tools, which is explained in the help menu of CPN tools and will also be discussed in one of the upcoming exercise classes and lectures.

**3.4.4. Validate the model** Use the *simulation menu* of CPN Tools to execute the model in a "step by step" fashion. By executing the model in "slow motion" you can detect undesired behaviour and correct any errors you made in the model.

**3.4.5. Add time information to the model** To use the Petri net model for performance analysis you must add time information. First of all you must change the definition of the colourset ORDER to make it a *timed* colourset. Secondly, to measure the completion time of an order, you must know its *start time*. Therefore you could change the definition of the colourset so it will then look like this:

**colset ORDER = INT timed;**

Next you must add time clauses to the Petri net model to represent the duration of the activities in the model. If, for example, there is an activity "Send sales documents to customer", that takes 1 day to complete, you can amend the expression on the outgoing arc from **o1** into **o1@+(24*60)** (so the duration is measured as 24 times 60 minutes).

**3.4.6. Add routing information to the model** An order can follow different routes within the Lista order process. For instance, in activity "Check of offer documents" (task 84 on the process chart), in 10 % of the cases, the next step is "Deliver by mail" (task 86 in the process chart) and in the remaining 90 % of the cases the next task is "Consultation of customer" (task 96 on the process chart). The are different ways you can model this. One simple solution is suggested below.

**Add colourset ROUTE** defined as **colset ROUTE = int with 1..100;**

**Add a routing attribute to the ORDER colourset** change the colourset ORDER into **colset ORDER = product INT * ROUTE timed;**

**Add guards to the transitions** to represent to the routing decisions within the order process.

**3.4.7. Add data collector** to the appropriate transition, eg. the transition that marks the completion of the order, like you did in exercise 3.2.9.. This will result in a data collector template. You should change the observer function in the template as follows :

```
fun obs (bindelem) =
let
  fun obsBindElem (DSp1'end_exam (1, {o1,o2})) = (intTime() - #1 o1)
      | obsBindElem _ = ~1
in
  obsBindElem bindelem
end
```

(NB: the symbols *o1* and *o2* are two variables on the input and output arc of the transition with which the data collator is associated. If you defined other names for the variables, they will change accordingly in the definition of the function.)

**3.4.6. Create a "generator" of orders** to represent the order flow that will arrive at the LISTA order process. This generator will create orders. Make sure that it fills the "start time" and the 'routing information" correctly. The start time is simply the current time you can derive from the function *intTime()* you already used earlier. For the routing information you can use a random number function like *discrete(1,100)*, which will generate a random number between 1 and 100.

**3.4.7. Calculate average completion time of an order** Let CPN tools calculate the average completion time of an order in the LISTA order process by running a simulation of the model you created, like you did in exercise 1.2.10 on page 1.2.

# 2 Timed Coloured Petri Nets.

## 2.1 Introduction

In the models considered until now there is no explicit notion of time. However timing is often a critical aspect of systems. This is for instance the case for real-time computer systems, communication protocols, logistics ([1]) and also, for the order process at LISTA, which is the subject of the running case.
Several methods for introducing time in the Petri net model have been proposed. The approach taken by Jensen in [4] will be described in this section.

Time is introduced by Jensen by attaching a **timestamp** to each token. Furthermore a global clock is available telling the **model time**. A token becomes available if the model time is equal to or greater than the timestamp assocciated with the token. A transition is **enabled** when all its input tokens have become available. The global clock will move to a next point in time after all transitions enabled at the current model time have fired.
The values and timestamps of the tokens produced may depend upon the consumed tokens. This relation between tokens consumed and tokens produced is described by **arc-expressions**. The timestamp of a token is calculated by adding a **delay**, defined in the arc-expressions, to the model time. The delay can be defined by an arbitrarily complex function, which for instance might include statistical distributions. By introducing this randomness a category of Stochastic Petri Nets is created. This is often appropriate because in reality it is not always possible to determine the exact timing of an event. The type of delay used does however influence the size of the occurrence graph and thus the complexity of the analysis of the Petri net. Stochastic delays increase the size of the occurrence graph and the complexity of analysis and are not considered in this document. The subject of stochastic Petri nets is however not covered here. The formal definition of a Timed Coloured Petri net given in the next paragraph is taken from [4].

## 2.2 Formal definition of Timed Coloured Petri Nets

First of all the state of the CP-net, as described in the handout for the second exercise class, must be extended with time information. This is accomplished by introducing the concept of a **timed multiset**. The definition of timed multisets is given below :

1. A timed multiset $tm$ over a non-empty set $V$ is defined as a function $tm : V \times R \rightarrow N$ such that

   $$tm(v) = \sum_{r \in R} tm(v, r)$$

   is finite for all $v \in V$. Analogous to the term $m(v)$ for ordinary multisets, defined in the second handout, $tm(v)$ is a non-negative integer representing the number of appearences of the element $v$ in the timed multiset $tm$.

2. The list $tm[v] = [r_1, r_2, \ldots, r_{tm_v}]$ consists of all the time values $r \in R$ for which $tm(v, r) \neq 0$ with $r_i \leq r_{i+1}$, $\forall i \in \{1, \ldots, tm_v - 1\}$ .

3. The timed multiset $tm$ over $V$ can now be represented as

   $$\sum_{v \in V} tm(v)`@tm[v].$$

   where $tm(v)$ is called the coefficient of $v$.

4. The set of all timed multisets is denoted by $V_{TMS}$ and the non-negative integers $\{tm(v) | v \in V\}$ are called the coefficients of the multiset $tm$ .

With the notation just introduced we can describe the following example of a timed multiset :

$3'1@[10, 11, 13] + 1'2[9]$

for which we have $tm[1] = [10, 11, 13]$ and $tm[2] = [9]$. This could for instance describe a marking of a place with 4 tokens, three with a value of 1 and one with a value of 2, each with its own time stamp.

The operations allowed on timed multisets are defined the same way as the corresponding operations on ordinary multisets except for the operations comparison and subtraction.
An ordinary multiset $V_1$ over a set $V$ is smaller than another multiset $V_2$ over $V$ if each element $v \in V$ that belongs to $V_1$ also belongs to $V_2$ and the multiplicity of the element in $V_1$ is never more than the multiplicity of the element in $V_2$. If $V_1$ and $V_2$ were timed multisets then $V_1 \leq V_2$ would require each element in $V_1$ to appear in $V_2$ with *exactly the same* time value. This requirement is too strong since we want tokens with a time stamp *smaller than* a specified time

value to become available. Comparing time stamps is defined as follows. For two ascending lists of time stamps $a = [a_1, a_2, \ldots, a_m]$ and $b = [b_1, b_2, \ldots, b_n]$ over $R$, $a \leq b$ iff $m \leq n$ and $a_i \leq b_i$ for all $i \in \{1, \ldots, m\}$. If $a \leq b$ then $b - a$ is defined as the list with length $n - m$, which is obtained from $b$ in the following way. From $b$ we remove the largest time value which is smaller than $a_1$. From the remaining list we remove the largest time value smaller than $a_2$. And so on, until finally, from the remaining list, we remove the largest time value which is smaller than $a_m$. This definition of subtraction is necessary to prevent a possible violation of a basic Petri net rule, the so called "diamond rule". This rule states that concurrently enabled steps can occur in any order with the same global effect.
The operations comparison and subtraction can now be defined for timed multisets as follows :

1. Let $tm_1, tm_2 \in S_{TMS}$, then:
   $tm_1 \leq tm_2 \Leftrightarrow \forall s \in S : tm_1[s] \leq tm_2[s]$.
2. If $tm_1 \leq tm_2$ then:
   $tm_2 - tm_1 = \sum_{s \in S}(tm_2(s) - tm_1(s))`s@(tm_2[s] - tm_1[s])$.

The consequence of these new definitions for comparison and subtraction is that tokens are not necessarily consumed in FIFO order and might cause a token to become "stuck" at a place, because the place may always have other tokens with the same colour and a usable timestamp which is higher(see pp. 151 and 152 in [4]).

The evaluation of initialization and arc expressions in a $TCPN$ might result in timed multisets. The timestamps attached to the tokens in these multisets represent a *time delay* after which the tokens will become available. So the time stamps of these tokens are calculated by adding the current model time to the time-delay. This is reflected in the following notation. For a timed multiset $tm \in S_{TMS}$ and a time-value $r \in R$ the multi-set $tm_r \in S_{TMS}$ is defined as follows: $tm_r = \sum_{s \in S} tm(s)@tm[s]_r$ where $tm[s]_r$ is the list obtained from $tm[s]$ by adding $r$ to each time-value.

**Structure of a TCP-net** A timed coloured Petri net can now be defined as a tuple $TCPN = (CPN, R, r_0)$ such that:

1. $CPN = (\Sigma, P, T, A, N, C, G, E, I)$, satisfies the requirements of a non-hierarchical CP-net when in the arc expression $E(a)$ and the initialization function $I(p)$, as described in the handout for the second exercise class, the $Type$ is allowed to be a timed or untimed multiset over $C(p(a))$ and $C(p)$ respectively.
2. $R$ is a set of time values or time stamps.It is also a subset of the set of real numbers $\mathcal{R}$ closed under addition and containing zero.

3. $r_0 \in R$ is the start time.

The set of bindings $B(t)$, token elements $TE$, binding elements $BE$ and steps $Y$ are defined in the same way as for untimed CP-nets in the handout for the second exercise class. The concept of a **marking** for TCPNs needs to be adapted to account for the additional timing information:

1. A marking is a timed multiset over $TE$.
2. The initial marking $M_0$ of $TCPN$ is the marking obtained by evaluating the initialization expressions :
   $\forall p \in P : M_0(p) = I(p)_{r_0}$.
3. A state is a pair $(M, r)$ where $M$ is a marking and $r$ a time value. The initial state is the pair $(M_0, r_0)$.

**Dynamics of a TCP-nets** The concepts of binding, token elements and steps for TCP-nets are defined in the same way as for CP-nets (see second handout). The concept of **enabling** in a CP-net has been introduced in the handout for the second exercise class to describe which tokens must be available before a transition is ready to fire and which tokens will be consumed if the transition fires. The definition of this concept for a TCP-net must be adapted to account for the fact that tokens must be consumed in the order of their timestamps in the following way :
A step $Y$ is enabled in a state $(M_1, r_1)$ at time $r_2$ iff the following properties are satisfied:

1. $$\forall p \in P \sum_{(t,b)\in Y} E(p,t)\langle b \rangle_{r_2} \leq M_1 p$$
2. $r_1 \leq r_2$
3. $r_2$ is the smallest element of $R$ for which there exists a step satisfying the previous two properties.

When a step $Y$ is enabled is a state $(M_1, r_1)$ at time $r_2$ the step may occur, changing the state $(M_1, r_1)$ to another state $(M_2, r_2)$, where $M_2$ is defined by:

$$\forall p \in P : M_2(p) = (M_1(p) - \sum_{(t,b)\in Y} E(p,t)\langle b \rangle_{r_2}) + \sum_{(t,b)\in Y} E(t,p)\langle b \rangle_{r_2}.$$

The first sum is called the removed tokens and the second is called the added tokens. Moreover : $(M_2, r_2)$ is said to be directly reachable from $(M_1, r_1)$ by the occurrence of step $Y$ at time $r_2$, which is also denoted by :
$(M_1, r_1)[Y, r_2\rangle, [M_2, r_2\rangle$.

Now we can define the concepts of occurrence sequence and reachability for a $TCPN = (CPN, R, r_0)$ :

1. a finite occurrence sequence is a sequence of states, steps and time values: $S_1[Y_1, r_2\rangle S_2[Y_2, r_3\rangle S_3 \ldots S_n[Y_n, r_{n+1}\rangle S_{n+1}$ such that $n \geq 0$ and $S_i[Y_i, r_{i+1}\rangle S_{i+1}$ for all $i \in \{1 \ldots n\}$. The state $S_1$ is called the start state, and the state $S_{n+1}$ is called the end state, where $n$ is the number of steps in or length of the occurrence sequence .

2. an infinite occurrence sequence is a sequence of marking, steps and time values:
   $S_1[Y_1, r_2\rangle S_2[Y_2, r_3\rangle S_3 \ldots$. The state $S_1$ is called the start state of the infinite occurrence sequence.

3. The set of all finite occurrence sequences $OSF$ of TCPN can be defined as $OSF(TCPN) =$
   $\{S_1[Y_1, r_2\rangle S_2[Y_2, r_3\rangle M_3 \ldots S_n[Y_n, r_{n+1}\rangle S_{n+1} : n \geq 0 \quad \wedge \quad S_1 = (M_0, r_0) \quad \wedge$
   $\forall p \in P \quad M_0 = I(p)_{r_0}\}$

4. The set of all infinite occurrence sequences $OSI$ of TCPN is defined as $OSI(TCPN) =$
   $\{S_1[Y_1, r_2\rangle S_2[Y_2, r_3\rangle M_3 \ldots : S_1 = (M_0, r_0) \quad \wedge \quad \forall p \in P \quad M_0 = I(p)_{r_0}\}$

5. The set of all occurrence sequences $OS$ of TCPN is defined as $OS(TCPN) = OSF(TCPN) \cup OSI(TCPN)$

6. A state $S''$ is reachable from a state $S'$ iff there exists a finite occurrence sequence having $S'$ as a start state and $S''$ as and end state. As a shorthand, we say that $S$ is reachable iff it is reachable from $S_0$. The set of states which are reachable from state $S$ is denoted by $[S\rangle$.

The example of a CP-net shown in the second handout is adapted to give an example of a TCP-net in figure 12 below. The formal specification is given by $TCPN = (CPN, R, r_0)$. The definition $CPN = (\Sigma, P, T, A, N, C, G, E, I)$ is equal to the specification of the $CPN$ given in the second handout except that one of the arc-expressions now includes a time-clause and one of the colourssets is defined as a timed multiset.

The new arc-expression function is:
$E = \{(a1, p), (a2, p), (a3, d), (a4, c), (a5, c), (a6, p@+1), (a7, p+1@+5), (a8, d), (a9, c), (a10, c@+1)\}$ and the definitions of the colourssets are extended with the clause "timed", denoting the fact that they are now *timed* multisets.

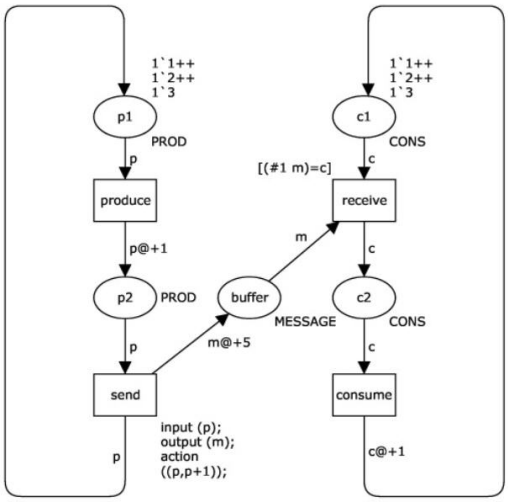


Figure 12: A TCP-net of a producer consumer process

The time clause @+1 in the expression associated with the arc between *produce* and $p2$ means that if the transition *produce* occurs it will produce tokens with a timestamp equal to the model time + 1. This token will become available for the transition *send* only after the model time has increased by one. An interpretation of this is that the activity *produce* has a duration of one time unit. Similarly, the activity *consume* takes also one time unit and the activity *send* takes five time units, which might represent for instance some kind of transmission delay. An important difference with CP-nets is that the enabling rule in a TCP-net is no longer local. To determine whether a transition has an enabled binding element at time $r_2$, we need to consider all transitions that have colour enabled binding elements. Otherwise we cannot know whether $r_2$ is minimal. In the example in figure 12 this means for instance that after the production and sending of 5 tokens by each producer,there are 15 tokens in the place $buffer$ and the transition *receive* is colour enabled. But the transition *produce* is colour and time-enabled by a token with a lower time-stamp (which, after producing and sending five tokens will have a value of 5) than the tokens in $buffer$ (the lowest time stamp will have a value of 1+5=6). Therefore the transition *receive* is not yet time-enabled. This will only happen after the production and sending of the sixth token in the place $buffer$.

**Occurrence graph of TCP-nets** The construction of an occurrence graph for a TCP-net is defined in the same way as for untimed nets, except that the nodes now represent states instead of markings. This means that each node contains a time value and a timed marking. Adding time however can have consequences for the size of the occurrence graph. The nature of this consequences is dependent on the type of delay chosen.
If a **fixed delay** is used, the occurrence graph of a non-cyclic TCP-net will in general be smaller, or at most as big as the occurrence graph of the untimed version of the same Petri net. The untimed version of a TCP-net $M = (CPN, R, r_0)$, can be obtained by taking the CP-net $CPN$ and deleting the timing information from the arc- and initialisation-expressions.(p. 157, [4]) The occurrence graph of a non-cyclic TCP-net is smaller than its untimed version because the timing information imposes a constraint on the dynamics of the CP-net. For a cyclic TCP-net, the occurrence graph usually becomes infinite. (p. 164, [4]).

# References


[1] W. van der Aalst; *Timed coloured Petri nets and their application to logistics*, Phd thesis Tue, 1992

[2] C. Girault, R. Valk; *Petri Nets for Systems Engineering*, Springer,2003

[3] K. Jensen; *Coloured Petri Nets; Basic Concepts, Analysis Methods and Practical Use; Vol. 1*, Springer-Verlag, 1992

[4] K. Jensen; *Coloured Petri Nets; Basic Concepts, Analysis Methods and Practical Use; Vol. 2*, Springer-Verlag, 1995

[5] A.V. Ratzer, L. Wells, H.M. Lassen; *CPN Tools for Editing, Simulating, and Analysing Coloured Petri Nets*, Department of Computer Science, University of Aarhus

[6] W. Reisig, G. Rozenberg, (Eds.); *Lectures on Petri nets: Basic Models*, *Lecture Notes in Computer Science*, 1491, Springer-Verlag, 1998

[7] G. Rozenberg, J. Engelfriet; *Elementary net systems*, in [6], pp. 12-121