UNIVERSITY OF TARTU

FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

Institute of Computer Science

Karl Blum

Open and Extensible Business Process Simulator

Master Thesis (30 EAP)

Supervisors: Luciano García-Bañuelos, PhD

Marlon Dumas, PhD

Author: ………………………………... "…... " June 2010

Supervisor: ………………………………... "…..." June 2010

Supervisor: ………………………………... "…..." June 2010

Professor: ………………………………... "…..." June 2010

TARTU 2010

Contents

1. Introduction

Business process simulation is a widely used technique for analyzing business process models with respect to performance metrics such as cycle time, cost and resource utilization. There are many commercial process simulation tools available that also incorporate a simulation engine, e.g. ARIS, Oracle Business Process Architect (Oracle BPA), FirstSTEP, TIBCO Business Studio, IBM Websphere Business Modeler and many others. Most of the currently available tools have two important architectural limitations that will be discussed within this thesis. These limitations are:

- They allow to simulate processes that are designed only in the same tool;
- The simulation engine is built-in and it is not extendable.

The aim of this thesis is to move towards overcoming these problems in existing tools. The core of this thesis is twofold. First we provide some of the commonly used Business Process Notation (BPMN) mappings to Coloured Petri Net (CPN) modules while considering the need to use these converted models for simulation purposes. This means that the mappings have to be able to handle simulation data and can generate simulation output into log files. Secondly we provide a new process model converter architecture that is open and extendable and it is responsible for generating a ready to simulate CPN models.

The following of this thesis is structured as follows. In section 2 we give an overview of business process management and process simulation. We discuss what they are and why are they important. In section 3 we give an overview of existing state of the art tools and conclude their shortcomings. In section 4 we discuss process mappings from BPMN elements to CPN modules with simulation support and also show how we can benefit from this conversion. In section 5 we provide an overview of our new simulation framework that has been developed with extendibility and openness in mind. In section 6 we look at how our converter architecture can play a part in an end-to-end process simulation lifecycle. After that in section 7 we describe the potential future work to extend our architecture and use it in an end-to-end process simulation environment.

2. Business Process Management

Business process is a collection of activities that take inputs and creates an output that is of value to the customer. Business processes consist of coordinated set of activities carried out by both automated systems and people to achieve a desired outcome. Michael Porter in his book [8] suggested that the activities within the organization and thus in the processes can be split into primary activities and support activities. Primary activities are essential for the company and support activities assist the primary activities. Primary activities are usually customer centric and customers interact with these activities directly (e.g. marketing and sales). Supporting activities provide more of the back-office and administrative activities (e.g. procurement).

People use the term "BPM" in many different ways. Some use BPM to refer to "Business Process Management" and others use BPM to refer to "Business Performance Management." Some use BPM to refer to a general approach to the management of business process change, while others use it more narrowly, to refer to the use of software techniques to control the runtime execution of business processes [19]. In this thesis we use BPM as "Business Process Management" to refer to a top-down methodology that is designed to organize, manage and measure the organization, based on the organization's processes. In this case business process management provides governance of a business process environment to improve performance through the optimization of business processes. Business processes are the key instruments to organizing activities in the process and to improve the understanding of their interrelationships [3]. Business process management activities can be grouped into four different categories as seen on Figure 1.
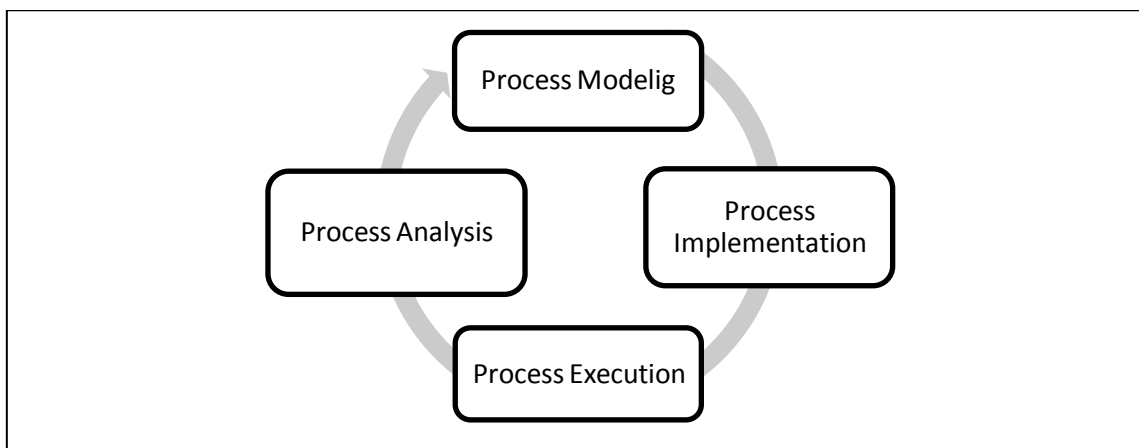

Figure 1 - BPMN lifecycle.

The lifecycle of BPM begins with process modelling where the order of tasks is specified in the currently used processes. This means that currently used processes will be modelled in some representable format. This is often called "as-is" modelling. In process implementation phase the processes will be converted into executable form. This can be done in the software simulation environment for example. In the next phase the implemented business processes will be executed and the outcomes will be recorded for analyzing in the next phase. In process analysis phase, business experts review the business issues to be resolved and in that context evaluate the end-to-end processes of the company, along with its strategy, governance documents and other process information. By defining key performance indicators and creating business cases, the business process expert identifies the needs of the business and defines the requirements to create and implement the new or enhanced processes and applications [2].

Business process modelling and simulation are an important part of BPM discipline. Process modelling allows abstracting real world processes and representing them in a graphical form. Graphical models can be used in discussions between different stakeholders as it provides understanding of the processes in a common way. Simulation is important part of BPM discipline because it gives the ability to test the outcome of a certain process. With simulation it is possible to evaluate the performance of the existing process model (as-is model) and to compare the current model with its modified versions to verify the increase in some key performance indicator.

3. Simulation of Business Process

Process model is a simplified representation of the actual process. It draws the boundaries in the process and is intended to promote understanding of how the real system works. Process simulation is a tool for validating and getting answers from the provided process model. Simulation can also be seen as a tool for managing change and it is an important part of BPM life-cycle. Practitioners of BPM know the critical importance of carefully leading organizations and people from old to new ways of doing business, and simulation is one way to accelerate change. This capability derives largely from the ability of simulation to bring clarity to the reasons for change. Simulation provides more than an answer: it shows you how the answer was derived; it enables you to trace from cause to effect; and it allows you to generate explanations for decisions [13]. It can be used in the design phase to support dynamic experiments with the process and evaluate the decisions that have been made to see how the process behaves under various circumstances, where the problems may lie and how should the process be improved. Simulation gives the process manager the possibility to experiment with different variables like staff size, working hours, etc. and it makes easier to correct problems or discover weaknesses before the new model is implemented and put into execution.

The needs for the simulator tool can vary depending on the modelled process, but the core requirements remain the same. A range of features desired from a simulation tool are: modelling flexibility, ease of use, animation, general simulation functions (e.g. warm-up period, multiple runs), statistical functions, interface with other software, product help and support, price and expandability [9]. The study presented in this article [9] also provides a list of 70 evaluation criteria or even the whole evaluation framework for simulation software selection methodology. Amongst other things the framework includes criteria regarding coding aspects (e.g. programming flexibility, access to source code, support, of programming concepts) and software compatibility (e.g. integration with statistical packages, integration with DBMS). Flexibility and access to the source code are considered to be of high importance if models are to be used for modelling complex or real time systems.

It is not realistic to expect any of the currently available tools to satisfy all these criteria. Increasing functionality tends to increase also the complexity of the tool. It then becomes

reasonable to choose the right tool with right functionality for given models. In the following we will discuss the extensibility aspect in current state of the art tools.

3.1. State of the art

Extensibility is a systemic measure of the ability to extend a system and the level of effort required to implement the extension. Extensions can be through the addition of new functionality or through modification of existing functionality. The central theme is to provide support for change while minimizing impact to existing system functions [10]. With an extendable tool it would mean that it is not so important whether it supports all of the needed functionality – workflow elements, simulation capabilities, analysis capabilities because the tool can be extended to support what is needed.

Currently most available simulation tools aim to provide end to end capabilities without any specific business domain in mind, e.g. TIBCO Business Studio and ARIS Business Simulation. Despite the list of currently available simulation tools, there has been made conclusions that most of them have architectural limitations [20] that decrease their support for extensibility. It could mean that the availability of different control flows, resource patterns or simulation measurements is fixed by the tool provider. We can see an evaluation of different commercial tools regarding the support of different workflow patterns here [23]. The limitations are not only in the inability to use some workflow pattern due to the lack of its support in the used simulation tool. Problems may also arise if the simulation to be done needs to measure something specific. For example most of the current tools use task duration and execution cost as the primary simulation data, but if we would like to add some specific tax calculation to the analysis, then implementing it might be complicated or even impossible. It is arguable that most of such analysis can be done later by a separate analysis tool from the data on the simulation logs, but this solution has another problem. That is because many of the current tools do not provide standardized simulation log output and therefore the analysis is limited to the functionality provided in the tool. In the following we will discuss some of the state of the art research prototypes and also some of the commercial tools.

3.1.1. Process simulation in research prototypes

In [5] and [6] some of the state of the art business process management tools were evaluated in respect of their simulation capabilities. In [5] the tools were divided into three different areas based on their capabilities. First category contains business process modelling tools that are specially developed to describe and analyze business processes. Protos was evaluated in this category. The second category was for business process management tools which core functionality is in the automation of the workflow processes. In this category FileNet and FLOWer where evaluated, although FLOWer lacks of simulation facilities. In the third category general purpose simulation tools were evaluated. General purpose tools are not tailored towards a particular domain, such as logistics, military or any other. In this category CPN Tools and Arena were evaluated. The overall conclusion in this study [5] is that the tools in business process management and process modelling category fell short on their simulation capabilities. Protos provides simulation capabilities based on the ExSpect simulation engine, but the interface between the two parts of the system omit important details with respect to data and resources. Within a research discussed in paper [16] a transformation from Protos models to CPNs was developed to provide extensive measurement and verification for processes. Protos2CPN is able to convert Protos XML output to CPN Tools file with XSL transformation. The simulation of Protos models in CPN Tools makes the running process visible by depicting the moving cases as tokens within the process model. It therefore allows for a detailed in-spection of the running process. In addition, the monitoring features of CPN Tools enable the generation of complex statistics. The models created by Protos2CPN transformation already include some basic measurements which can be extended by experienced users [16]. The conclusion was that the resulting CPN model has complex modelling and simu-lation capabilities, but although the simulation capabilities of CPN Tools is its good side, it requires profound knowledge to model in Petri Nets and the resulting models are hard to understand, especially by a non-technical background business analyst. Despite this short-coming CPN Tools is based on the formal modelling technique and opens many possibilities for complex process simulations. The thesis [6] also discussed process simulation in CPN and provided a complete simulation data metamodel. One of the differrences with this paper is that here we will provide more advanced CPN mappings and concentrate on building an extendable converter architecture and implementing a version of the converter prototype.

3.1.2. Process simulation in commercial tools

IBM WebSphere Business Modeler (WebSphere) is one of the comprehensive commercial business process analysis tool that offers modelling, simulation and analysis capabilities. In WebSphere it is possible to model, assemble and deploy business processes, then monitor and take actions based on key performance indicators (KPIs), alerts and triggers to continually optimize these processes. WebSphere also supports the capabilities of simulation, analysis and redesign. WebSphere offers robust functions for business process analysis as well as modelling capabilities for business processes, enterprises, essential data, artefacts, organizations, resources, timetables and locations [6]. It is possible to perform dynamic and static analyses. Static analyses are performed on process simulation profiles and also on processes without actually running the simulation. With this it is possible to recognize decisions and loops in the process flow and get a comprehensive idea of the possible paths through processes including costs and revenue generated by each possible path. Dynamic analysis is performed on the data from a simulation run and it can be done at three levels of granularity:

1. Aggregated analysis – Most broadly scoped of the dynamic analyses. They use all the data from the entire simulation run for their information. This analysis is used to gain understanding of the behaviour of the whole process.
2. Process case analysis – This analysis uses the data from specific process case and it is used to gain understanding of a certain process flow.
3. Process instance analysis – It is the most granular of the dynamic analysis and it uses data from a single process case instance for their information.

WebSphere process modelling notation is not fully BPMN-compliant, but it is influenced by BPMN. Some of the icons used in WebSphere process modelling environment are based on those from the BPMN specification. An example of a process model in WebSphere can be seen on Figure 2.
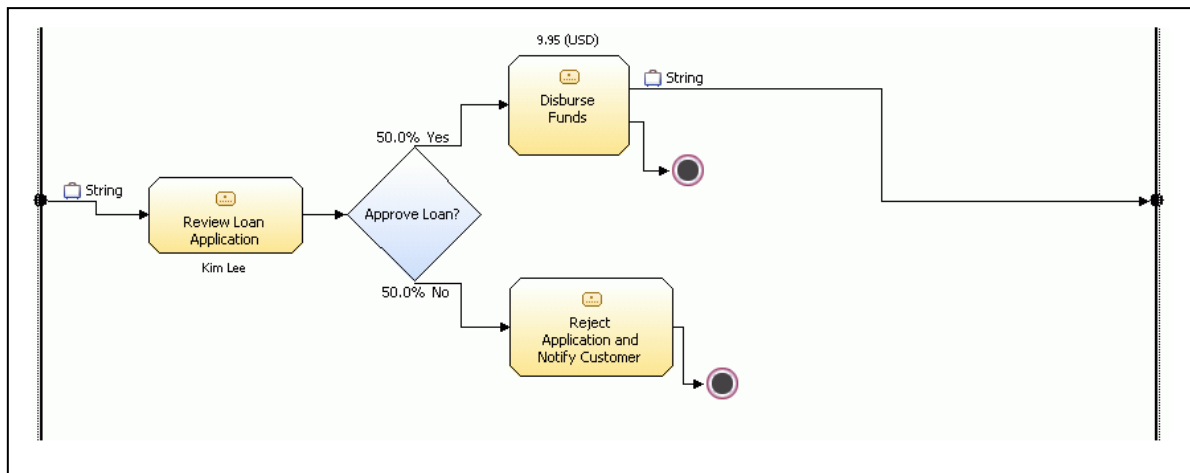
Figure 2 - Example process in IBM WebSphere Business Modeler.

Process simulation attributes that define conditions and behaviour for the whole process during a simulation run can be set from the settings dialog as seen on Figure 3. Setting the simulation data for activities is rather limited as can be seen on Figure 4 and Figure 5.

Figure 3 - Adding simulation data in IBM WebSphere

Figure 4 - Adding simulation data to a task.

Figure 5 - Adding cost and revenue data to a task.

Although the modelling notation used in WebSphere is influenced from BPMN, it does support only small subset of modelling elements like: start events, end events; forks; loops; merges; joins; timers and some others. With this subset of elements it is complicated or even impossible to model complex business processes to achieve more detailed simulation

results for further analysis. For example, it is not possible to model intermediate events as they are described in BPMN specification and thus such process cannot also be simulated.

Another state of the art commercial tool for business process modelling and simulation is TIBCO Business Studio. It has a better support for different BPMN elements as it claims to be 100% BPMN version 1.2 compliant. On the other hand, it only means that it is able to build process models with using all of the BPMN elements, but it does not mean that all of the BPMN elements can be used in models for simulation purposes. For example it is not possible to add an intermediate timer or message boundary event to a task for simulation purposes. Also it is not possible to use event based gateways or any other intermediate gateway in a simulation model. Example of a business process model in TIBCO can be seen on Figure 6.
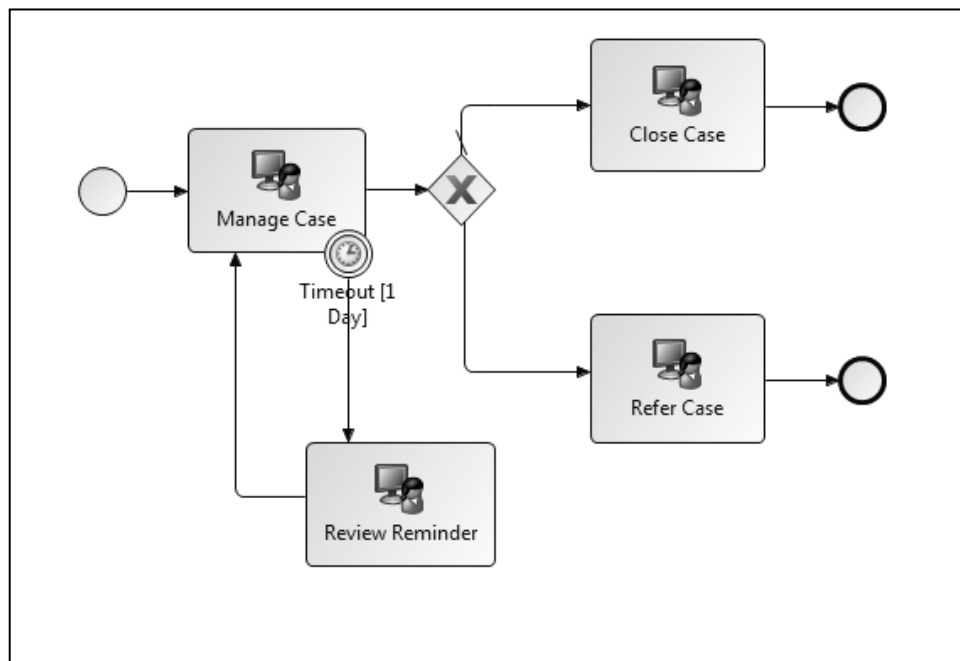
Figure 6 - Process model in TIBCO modelling environment.

ITP Commerce is another tool for process modelling in BPMN notation and it also provides full support for BPMN 1.2 and also BPMN 2.0 elements. Although it provides simulation support for also intermediate events and event-based gateway it is not possible to use boundary events for tasks. The simulation support for intermediate message events is also rather limited because for example it is not possible to use intermediate message with message receiving probability. The coverage for intermediate events and event-based

gateways for process simulation is a problem for all of these discussed tools. The support for this has been summarized in Table 1.

	BPMN 1.2 coverage for modelling	Intermediate events in simulation	Event-based gateways in simulation
TIBCO	Full support.	No simulation support.	No simulation support.
IBM WebSphere	Almost none	No modelling support.	No modelling support.
ITP Commerce	100%	Limited simulation support.	Limited simulation support.

Table 1 - BPMN events coverage coverage in state of the art tools.

3.2.   Conclusions

The tools described here are either research prototypes or expensive commercial tools that are the flagships of currently available business process simulation environments. Most of them provide support for the whole business process management lifecycle and have a huge number of features including support for process simulation. They also have good graphical interfaces with a drag and drop functionality for easy process modelling. Despite this they all tend to have limited support for simulation capabilities and they mostly use their own closed and built-in simulators that cannot be extended by the end user to satisfy their needs. CPN Tools is considered to have good support for complex process simulation, but it can be too difficult to use for business persons. In the following we will discuss how it would be possible to convert BPMN models to CPN models by presenting a set of mappings for different BPMN constructs.

4. Simulation of BPMN with CPN

Business Process Modelling Notation (BPMN) gives the possibility to model all kinds of business processes in a way that is easily readable to people with no technical background. It is a de-facto standard for modelling business processes. Although BPMN is the most widely used notation, it lacks the support for process simulation. Current version of BPMN provides only a few properties in elements that can be considered as simulation data (timer execution time for example). In the forthcoming BPMN version 2.0 there will be some additional support for simulation data but it will not be as complete as the modelling notation itself. BPMN provides only the visual representation of a business process model and it does not have any formal semantics. Without formal semantics it is impossible to automatically simulate an existing process model. To overcome this problem there have been made some proposals of formal semantics [11], but there is no standard semantics currently available.

Coloured Petri Net (CPN) is another graphical oriented modelling language that is used for designing and simulating processes. CPN syntax and semantics have a formal definition which is the basis for simulation of a process model. CPN can be used in places where formal semantics is essential in the modelling project. It is possible to model very complex business processes in CPN, but the modelling notation has only a small set of low level elements. The graphical representation of the model will not be as readable as BPMN model and might not be as understandable for business persons with no technical background.

In the following we will discuss the mappings from BPMN elements to Petri net modules. We start with the mappings to plain Petri nets and then discuss the mappings to CPN modules incrementally starting from the simple elements. Short introduction to BPMN and CPN can be seen in appendix A and 0 accordingly.

4.1.   BPMN to plain Petri nets

An example of a BPMN model for order processing can be seen on Figure 7.Figure 7 - Example of process model in BPMN. It is a simplified model with five tasks, exclusive gateway, and gateway, join, end event and start event. The leftmost circle represents the process start and the circles on the right represent process endpoints. The first gateway after the task "Check availability" is an exclusive gateway and exactly one of the two

output paths is always taken. The second gateway after order confirmation is a split gateway. It means that all the paths will be taken concurrently.
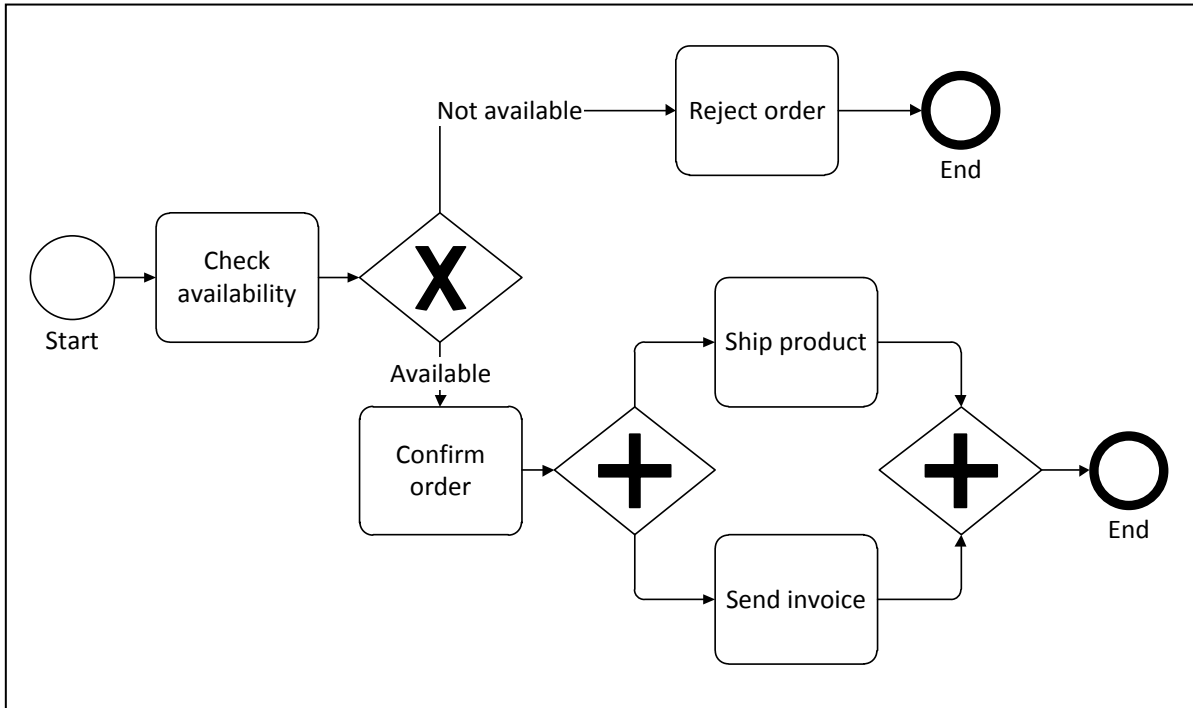
Figure 7 - Example of process model in BPMN.

The conversion from BPMN to plain Petri net model is done in a way that each control flow element has usually at least on central transition that is able to capture their routing behaviour. Examples of conversion mappings can be seen on Table 2.

Mapping from BPMN start event to Petri net is straightforward and it has been also discussed in earlier papers, for example here [20], [11]. Most of the conversions can be done by adding a silent transition to a Petri net for each BPMN element. For example the start event in BPMN will have a transition in Petri net that moves tokens out of the start place. In Petri net it is not possible to connect two places with an arc and this means that the tokens from the starting place can move only via the existing transition. Converting an end element is similar to the start event as there is an end place and the transition that is able to put a token to this place. There is no outgoing arc from the place because it represents the end of a process.
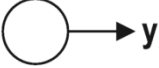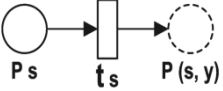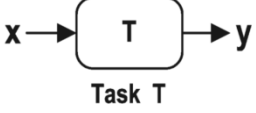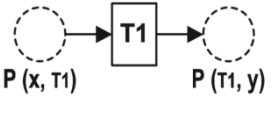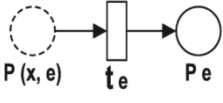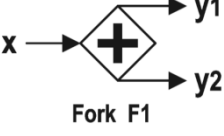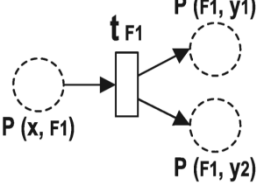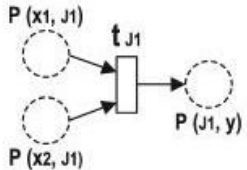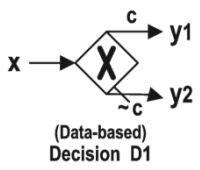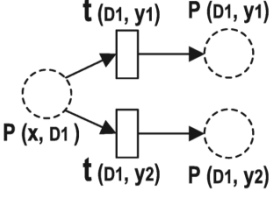
BPMN Object	Petri-net module	BPMN Object	Petri-net module
Start s	P_s t_s $P(s,y)$	x → T → y Task T	$P(x, T1)$ T1 $P(T1, y)$
x → End e	$P(x, e)$ t_e P_e	x → Fork F1 → y1, y2	t_{F1} $P(F1, y1)$ $P(x, F1)$ $P(F1, y2)$
x1, x2 → Join J1 → y	$P(x1, J1)$ t_{J1} $P(J1, y)$ $P(x2, J1)$	x → (Data-based) Decision D1 → c y1, ~c y2	$t_{(D1, y1)}$ $P(D1, y1)$ $P(x, D1)$ $t_{(D1, y2)}$ $P(D1, y2)$

Note: x,x1 and x2 represents an incoming object. y,y1 and y2 represent an outgoing object.

Table 2 - Petri net mappings.

BPMN Task can be simply represented as a transition in Petri net module as can be seen in Table 2. The transition in Petri net is needed to capture the behaviour of the model. AND gateways in BPMN mean that an incoming case will take all of the outgoing branches concurrently. Modelling AND gateway in Petri net is also done with a central transition and the splitting is done with two or more outgoing arcs. Each outgoing arc corresponds to one outgoing branch in BPMN model. In Petri net it means that the transition will consume one incoming token and generate two or more outgoing tokens (depending on the amount of outgoing branches. Data based exclusive gateways can be modelled in Petri net using several transitions. If we have a separate transition for all of the outgoing arcs then it means that each branch can be taken and if one of them fires then a token is consumed and the other transitions are not enabled anymore. Joining two process branches has to be done with one central Petri net transition that is able to fire only if there is a token from each incoming arc. This means that the transition will consume a token from all of the incoming places and produce only one output token.

Using these conversion patterns discussed before, we are able to convert a simple process modelled in BPMN as seen on Figure 7. The corresponding Petri net model can be seen on Figure 8. As Petri net uses only a small set of elements (place, transition, arc), it may sometimes not be as simple to read as a BPMN model. In this case if we use only simple tasks and gateways the CPN model is not complex. If we would add boundary events and resource management then the model would become more complex.
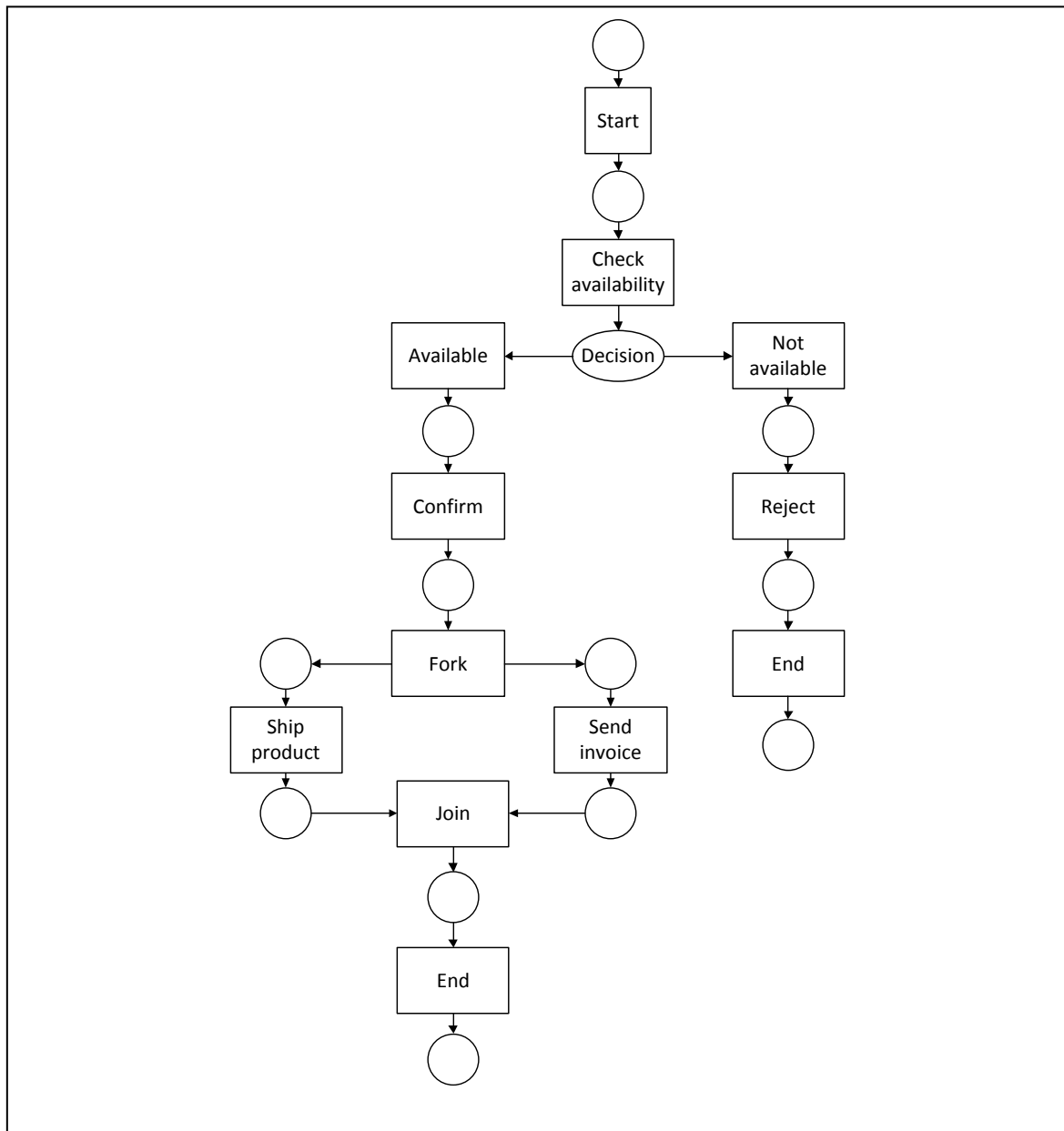
Figure 8 - Process model in Petri net.

The mappings previously discussed provide an easy way of converting BPMN processes to plain Petri Nets. This conversion is straightforward but this conversion does not help much

as it is not possible to do proper model simulation. In the converted model, as seen on Figure 8, it is only possible to manually "play" with the control flow by moving the token from the start place towards the end. To do this, we have to place a token to the first place on the left in our model. This token represents the process instance and the start transition is able to execute. This way we can manually simulate different process situations but it is not usable for advanced process analysis.

One aim of this thesis is to provide the mapping solutions from BPMN modelling elements to Petri net constructs in a way that the converted model is ready for simulation. In some cases the basic mappings can be still used but most of the mappings will become more complex if we add support for simulation data and resource consumption for example. In the following paragraph we discuss the solutions developed within this thesis for adding simulation and resource handling support. Because Petri net is very constraining for advanced process modelling, we decided to use higher level version of Petri net called Coloured Petri net (CPN).

4.2.   BPMN to CPN with simulation support

In the following we discuss the process conversion mappings to get a CPN model with simulation and resource management support. The mappings that were developed as part of this thesis have been done in a way that when doing a model conversion with any traversal method, then in each next conversion we do not have to remove CPN elements from the previous mappings. This means, that for example after mapping a gateway we start mapping a task, then we do not remove places or transitions from the gateway. This makes our mappings robust and stable because connecting the next element to the previous mapping does not change its structure although it can modify the arc inscriptions and transition texts or create a new structure by adding elements.

4.2.1. Case generation

In process simulation we can consider the start element as a process initiator. This is the starting point for the whole process. With this in mind we incorporated the process case generator into the start event itself and to model it in CPN we used a sub-page to encapsulate the whole generator into one virtual transition. It basically means that the start event produces process tokens or cases. Our implementation of the process generator is derived from the work [6] and the process generator itself consists of a central CPN

transition that after firing will generate a new case to an output port. The frequency of cases created and other variables in the process generator can be defined with the arc inscriptions as seen on Figure 9.
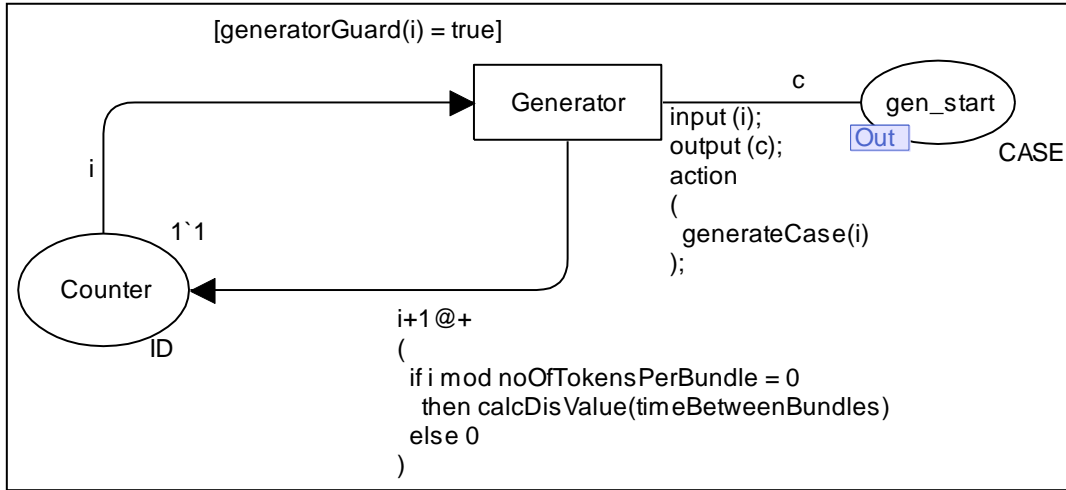
Figure 9 - Case generator in CPN.

The generator transition itself is connected to the counter place where exists only one token. This token is a simple integer type case identifier and after each time the generator transition uses it, the identification number (ID) is increased by one. If the generator transition puts the token back to the counter place, it increases also the timing to simulate a delay between two process instances. If the simulation needs a bundle of cases to be generated in the same time unit, the time is not increased when the full bundle has not yet been created. The generator transition has an assigned function *generateCase(i)* that is responsible for initializing the logging functionality. In this place the log file in a file system for this process case is generated and this file can be later updated to add more simulation events.

4.2.2. Control flow

In BPMN there are several ways to do branching in a process. One of the most used elements for process branching in BPMN is data based exclusive gateway [12] and an example of this can be seen on Figure 10.

To use this branching in a simulation model we have to add branching probabilities to know how many times each of the branches will be taken. To convert this kind of bran-

ching to a CPN model we developed a version of mapping that also supports branching probabilities. A simple example of this mapping can be seen on Figure 11.
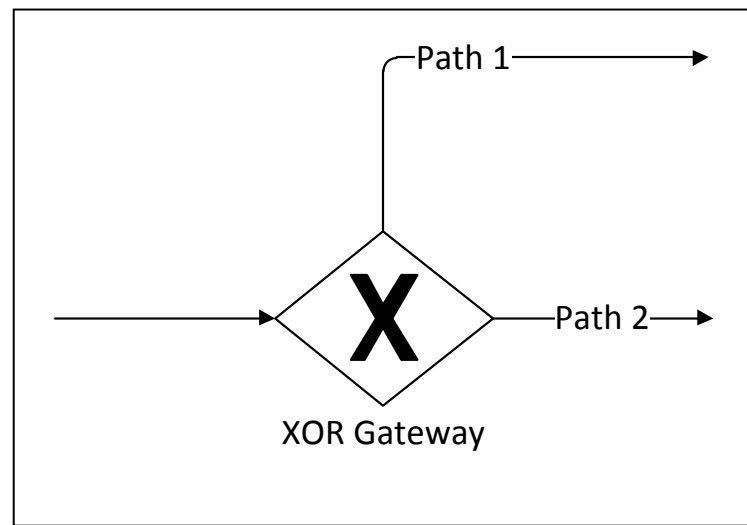
Figure 10 - Exclusive gateway in BPMN.
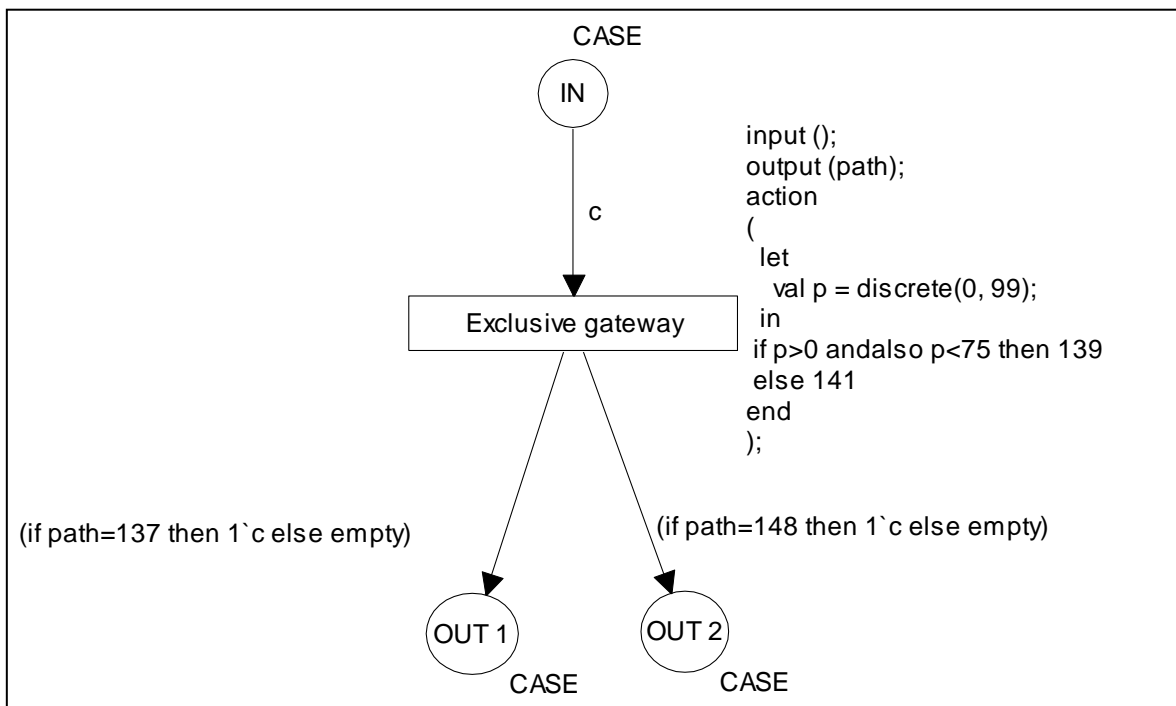
Figure 11 - Exclusive gateway in CPN.

The CPN model has one central transition that is responsible of determining the path to be taken. The simulation information will be added there to the transition action function. This function will pick a random value between 0 and 99 and with this number the

outgoing path is determined. Each outgoing arc from the transition has a conditional expression that will allow the token to pass only if it the token was intended to take this path.

4.2.3. Execution time

Extending our previously defined Petri-net conversion pattern for a task to support simple simulation with the ability to model processing time is really straightforward. In CPN we can define a delay in the outgoing arc inscription which means that after the transition fires the token will have its time changed. On Figure 12 we can see a task that has an execution time of 100 time units.
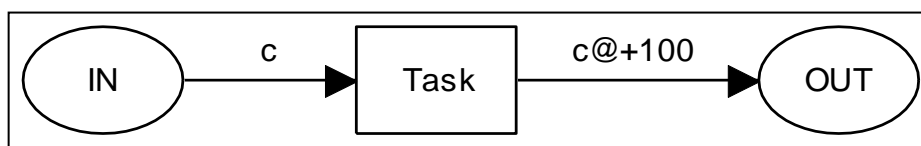
Figure 12 - Simple task in CPN.

In our mappings we use a method assigned to a task to calculate the total time consumed. If we add for example the resource management then the outgoing arc inscription also has to consider the resource waiting time. Instead of using fixed time values it is possible to used time functions in the arc inscriptions.

4.2.4. Resource management

To support resources in simulation we had to extend our mapping model a bit further. Our proposed solution for a task mapping can be seen on Figure 13. To add support for tasks that also need resources to execute, there has to be made a few enhancements that we will discuss in the following.
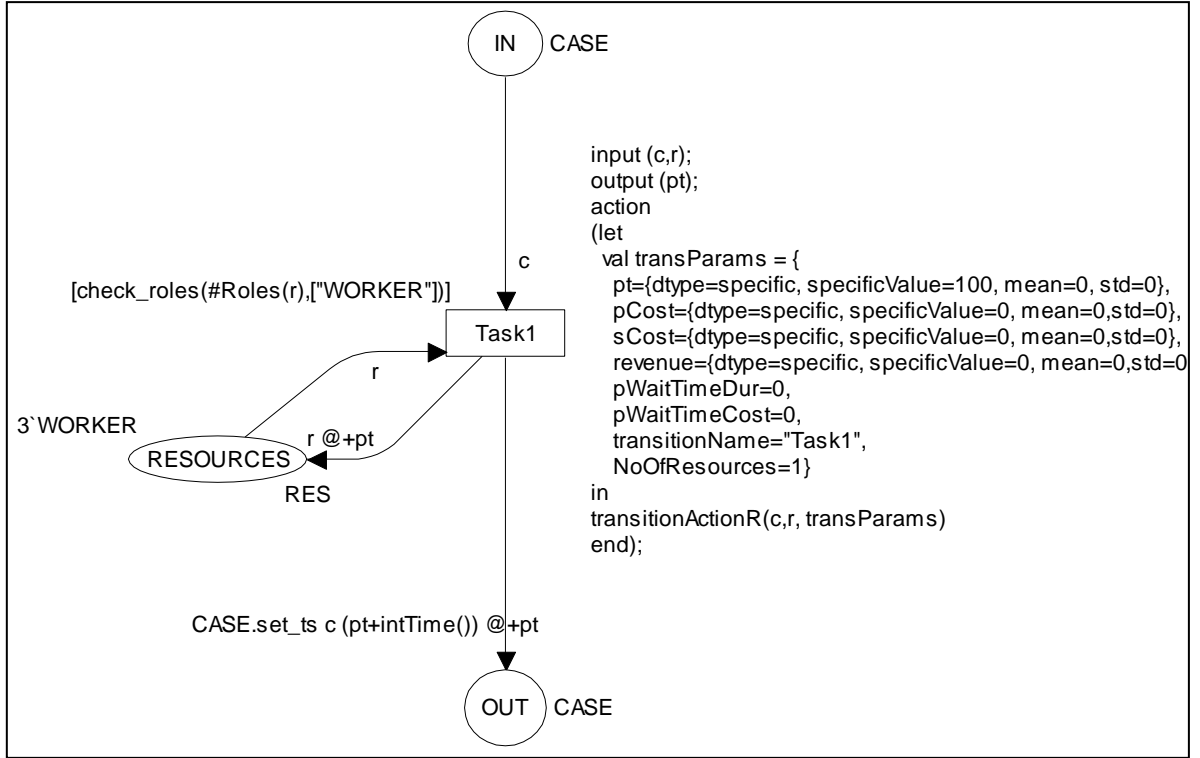
Figure 13 - Task with resource management support in CPN.

In our mapping we use only one central resource pool and its initial marking shows all of the available resources. In CPN it is possible to assign a colour to token and this can be used to differentiate between different types of resources. In each task transition there has to be a guard that checks the availability of a needed resource and can fire only if a certain resource is available. When a task uses a resource to execute then it will put the resource back to the resource pool after it has finished and also sets a new time to the used resource token. This way the resource is not available to other elements while the task is executing.

In our example model on Figure 13 the Task1 also uses resources to execute. The function defined to the right from the task is the transition function. This function calls the simulation logger with the defined parameters to generate log trail. This function we used is taken from [6].

4.3. Advanced constructs

Besides the previously discussed simple process modelling constructs there are also a lot of elements that can be considered more complex. In the following we discuss the mappings of some of these elements.

4.3.1. Intermediate events

Events can happen between any other activities and these are called intermediate events. Examples of timer intermediate event between two tasks and its corresponding CPN mapping can be seen on Figure 15. On Figure 14 we have a timer intermediate event between two tasks and its corresponding CPN mapping.
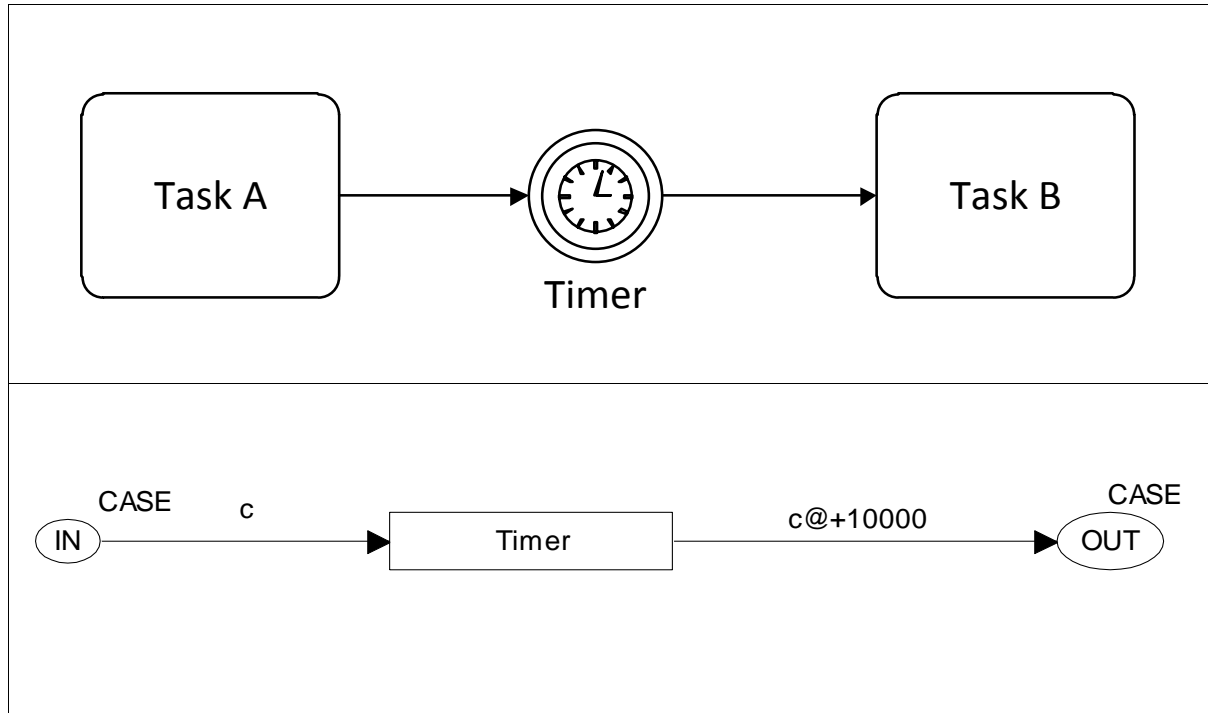
Figure 14 - CPN mapping (bottom) for BPMN intermediate timer event (top).

Intermediate events in the process flow mean that the execution of the process has to pause and wait for the certain event to happen before the execution can continue. Timer event means that the process has to wait for a specific amount of time and message event means that the process has to wait until the message has arrived before continuing. For the message event there can be various implementations of how to define the receiving of a message. Our implementation in the CPN mapping is done in a way that the messages will be received periodically after every defined amount of time. Considering this assumption we can also say that the messages are received in uniform distribution between 0 and period time. An example of intermediate message event and its mapping to CPN can be seen on Figure 15.
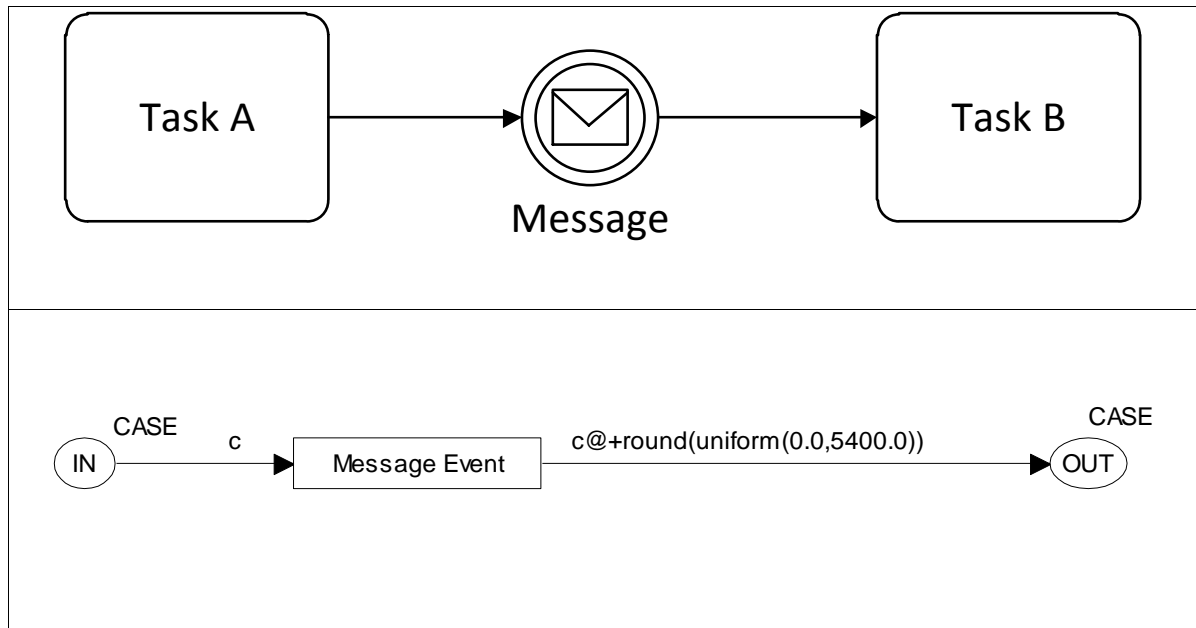
Figure 15 - CPN mapping (bottom) for BPMN intermediate message event (top).

Mapping for timer event is straightforward because an intermediate timer is intended to generate a fixed delay in process execution and therefore in CPN mapping we have to add a fixed time delay to the timer transition outgoing arc inscription as can be seen on Figure 14.

4.3.2. Task boundary events

We consider a task as an atomic transaction that cannot be cancelled while the execution has already started. This means that a certain task can be represented as a single transition in CPN. While simulating a model where resources are involved we might want to cancel an activity that has been waiting too long for a free resource and has not been started yet. This kind of behaviour can be modelled in BPMN with boundary events and these events can fire any time while waiting for the task to start executing. Another type of boundary event for a task is receiving a message. Example of a BPMN task with a timer and message boundary event can be seen on Figure 16.
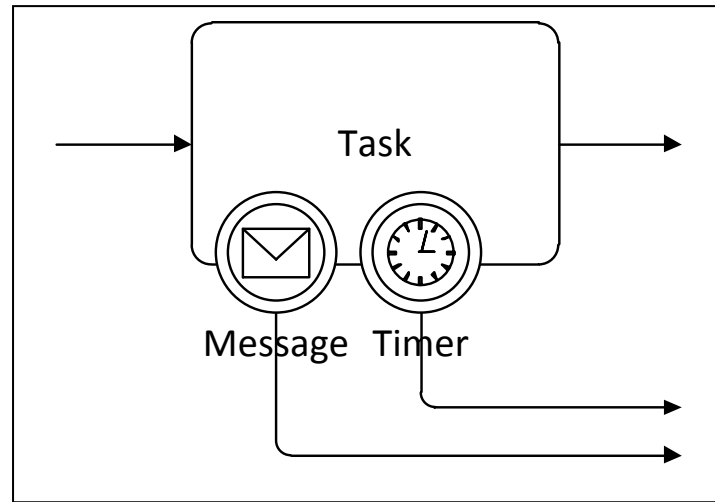
Figure 16 - BPMN task with message and timer boundary events.

To model this kind of behaviour in CPN, we used pre-empting time stamps in arc inscriptions. These timestamps are used in incoming arcs and it means, that the token can be consumed the expressed amount of time ahead of model time. On Figure 17 we can see the CPN mapping for a BPMN task with boundary timer and message events. If the task receives an input token then it is first evaluated if a message will arrive. This is done with a probability function before timer and task itself. It is so because the arriving of the message is derived from the probability from simulation data and it does not depend on the availability of resources. If the message will arrive, then the message transition will generate a delay that is between 0 and message arrival interval and the message output will be taken. The values for arrival time are taken in uniform distribution.

If a message does not arrive then the timer is started by putting a token with the added timer delay to the place just before timer event transition. It means that the timer can fire if the time has reached $@+1000$ in our example. The task is still able to fire because in its incoming arc inscription we use pre-emptive timing with the same amount of added time which means that the token is available before current simulation time. In our example if the token will arrive at time x. Then after the transition "Timer Events" the token will be available at time $x+1000$. But for the transition *Task* the token is available 1000 time units before and this means the token is available for task at time: $x+1000-1000 = x$. This gives to the task the ability to execute before the timeout. If the task is not executed within a given time then the timer output path will be taken. The time delay for timeout has then already been added.
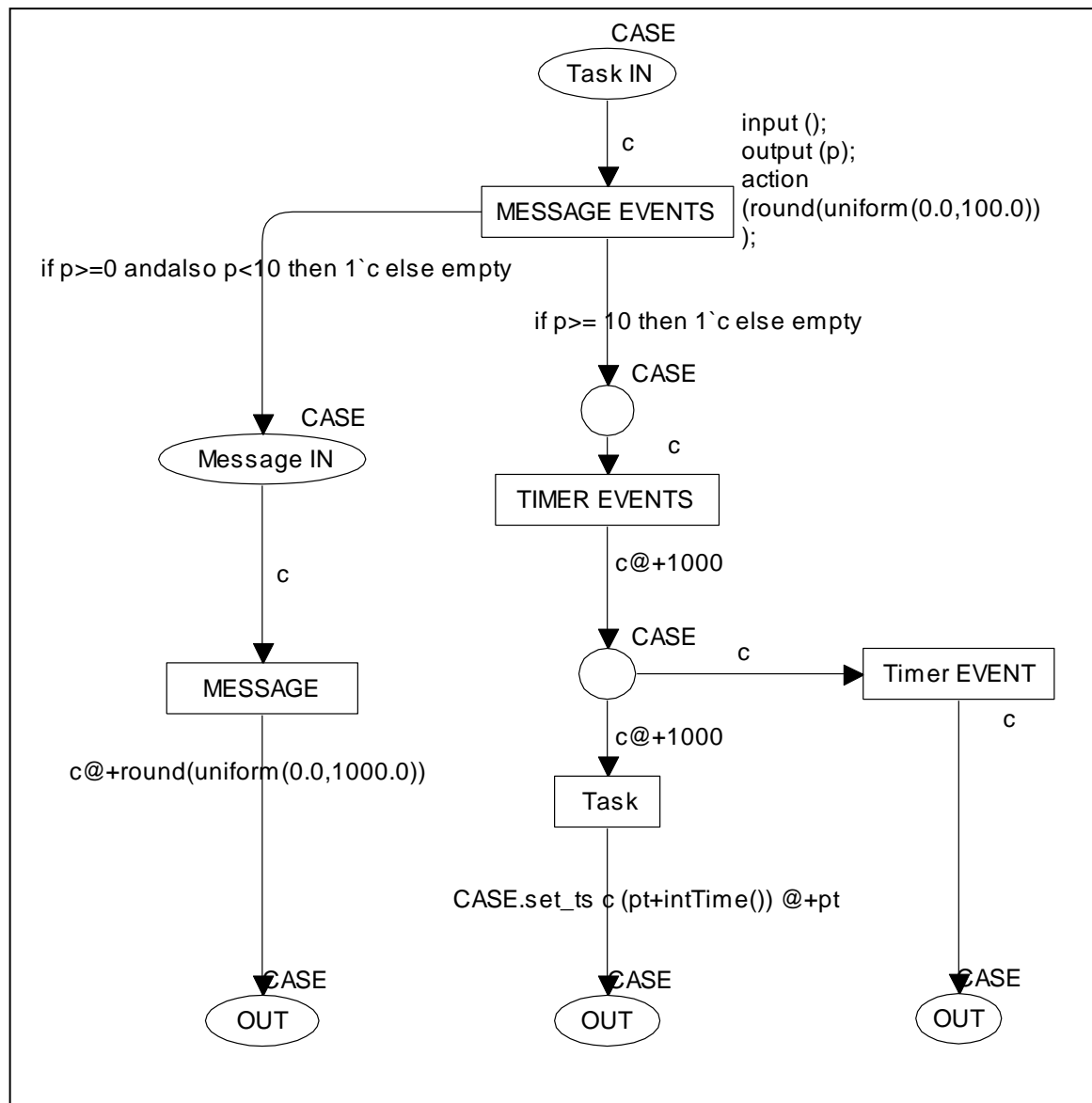
Figure 17 – CPN Mapping for a task with boundary timer and message events.

4.3.3. Event-based gateways

An event-based gateway is a BPMN flow control gateway where the path to be taken is decided on the event that occurs first. Event-based gateway can have different types of events connected to it, but only one timer. It is not prohibited to use more than one timer but if fixed time timers are used then we know that one of them fires always first and thus the others are never used. From this conclusion we can say that the event-based gateway output path can be derived from the occurrence probabilities of each event. Each event fires with some fixed probability and the timer fires when no other events occur. If the timer path is taken, we add the fixed timer delay. Message events can occur at a time between arriving to the gateway and timer triggering time. If we have more than one

message event, they all have the same delay function. An example of event-based gateway with a timer and two message events can be seen on Figure 18.
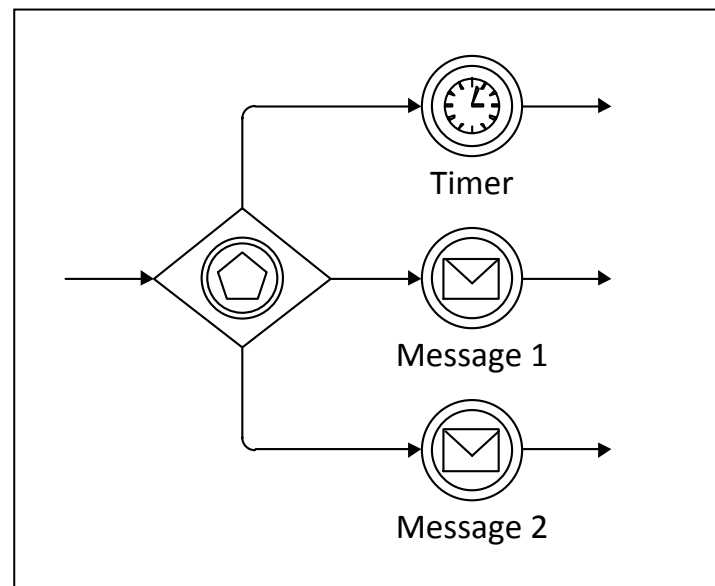
Figure 18 - BPMN event-based gateway with at timer and two message events.

The mapping for event-based gateway is rather straightforward when supporting timers and message events. For an example if we would have a process model with an event based gateway as seen on Figure 17, then we know that the branching probability can be predefined and is independent from the timer timeout limit. This allows us to calculate the branch taken right after the case has arrived in the gateway. It means that when both messages have a receiving probability of 25% of the cases, then the timeout occurs 50% of the time (100% - 25% - 25%). If the timeout is set to 1000 time units then if the timeout path is taken the delay will be exactly 1000 time units. But if any of the messages are received then it means the time delay is evenly distributed between 0 and timeout time. An example of this mapping can be seen on Figure 19.
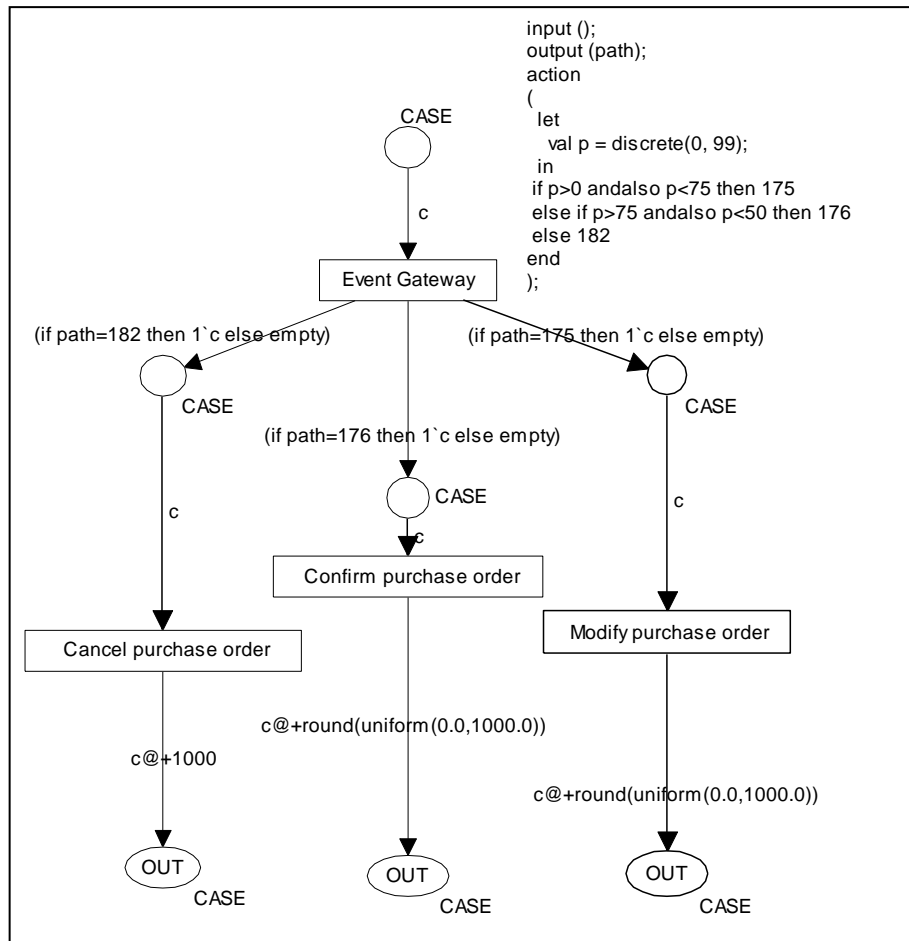
Figure 19 - CPN mapping for BPMN event based exclusive gateway.

4.3.4. Sub-processes

Another commonly used construct in a BPMN model is a sub-process. A sub-process is a collection of activities that can be viewed as a whole and it provides a natural way to draw a condensed top-down view. Process flow in the sub-activity cannot cross the boundaries of the sub-process. Example of a simple sub-process in BPMN can be seen on Figure 20.

A sub-process is in connection with the parent process through events. Most basic flow in a sub-process can be modelled with a start and end event and the mapping to a CPN model is then straightforward. The sub-process start has to be connected directly with the activity before the sub-process in the main process. The sub-process can also use intermediate events to throw events outside the sub-process boundaries. For example a BPMN sub-process can have an error, message or timer events that will be caught from the boundary of the sub-process. This means that the sub-process cannot be seen as an atomic activity

and the mapping for sub-processes with these events is more complex. We will discuss these mappings in more depth in the following chapters.
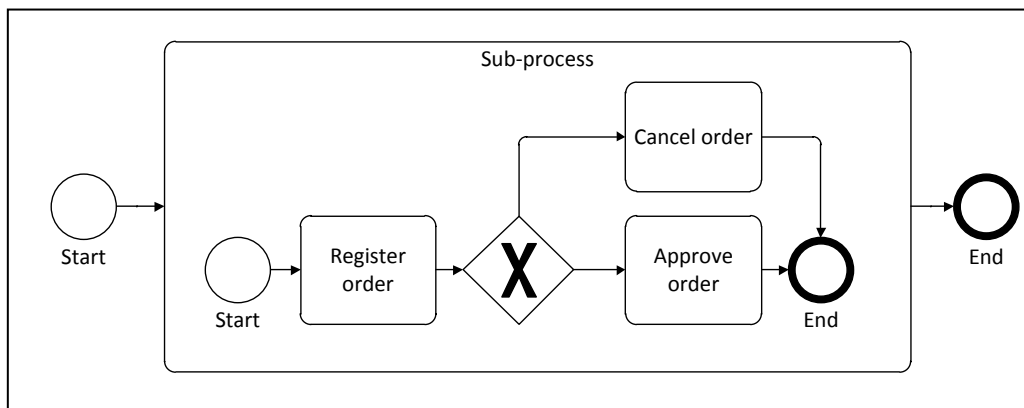
Figure 20 - Simple BPMN sub-process.

4.3.5. Sub-process timer

We can also attach a timer to a sub-process to control the amount of time the sub-process can run. If the certain amount of time has passed the process execution will be interrupted. Interrupting the sub-process flow with an intermediate timer event means that the whole execution of the sub-process needs to be cancelled just after already executing tasks will finish. This means that if timeout occurs, we cannot start new tasks and in a CPN model the tokens in a sub-process have to be taken out to stop executing the normal flow. To model this kind of behaviour we have to add some additional constructs to the model.
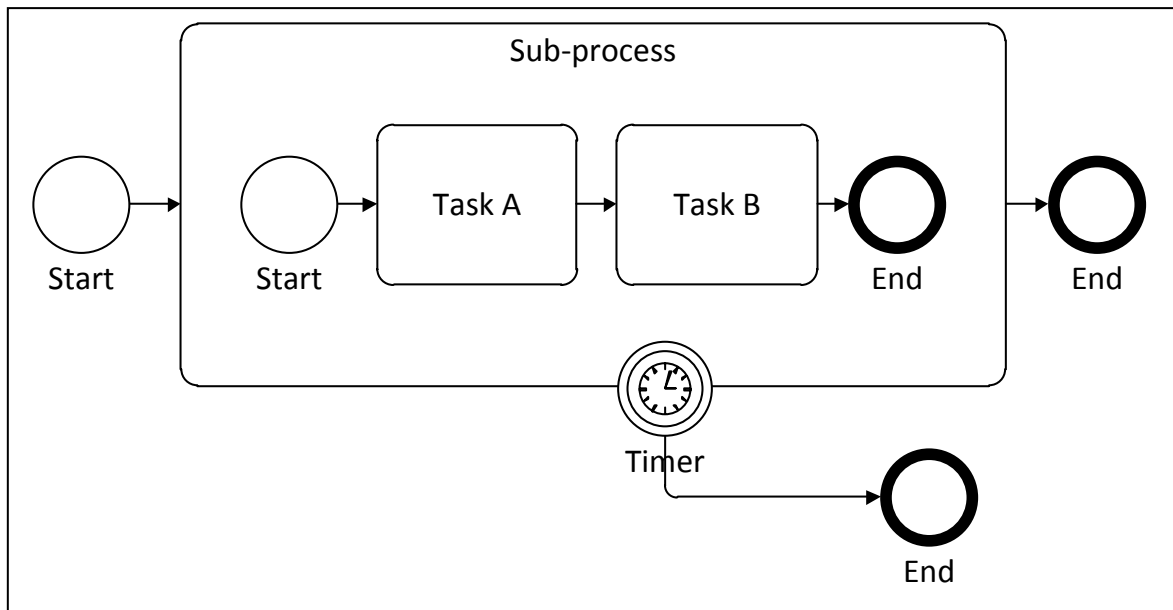
Figure 21 - Simple sub-process with a timer boundary event.

Each task in a sub-process with a timer boundary event has to have a skipper function as seen on the Figure 22. The corresponding BPMN model can be seen on Figure 21. At the input of the sub-process the token goes to a place where it indicates, that the process is ready to execute and the timeout has not happened yet. In our example this place is named *OK*. This status place is connected to each task in the sub-process and each task can start executing if a corresponding token is in the *OK* place. The token is put back after the execution of the task has been ended. If the task has finished it allows the timer event to execute if timeout has occurred. For this the exception transition fires and takes the token from *OK* (OK to continue) place and moves it to *NOK* (Not OK to continue) place. With a token in *NOK* place no other tasks can execute and all the task skip transitions are active. This means that artificial transitions transfer the tokens through the model without executing the real tasks and this way it is possible to move tokens out of the model and start an alternative flow (e.g. timeout). This also means that the time does not advance anymore in this sub-process because no task can execute and therefore no delay can happen.
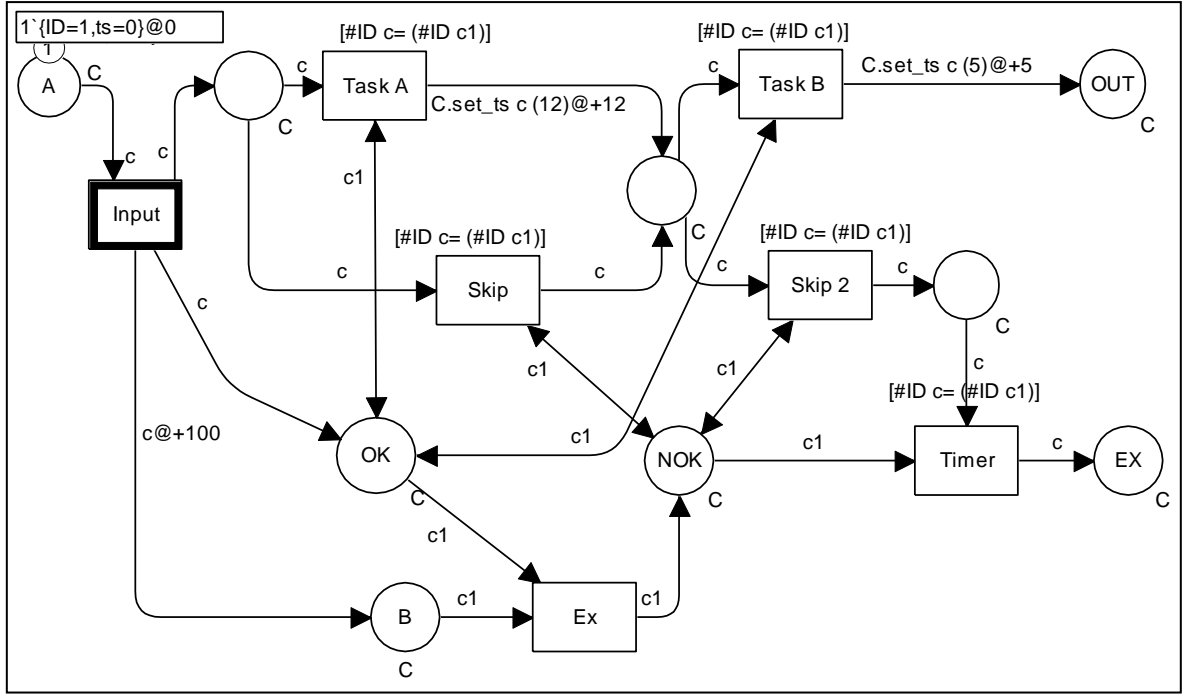
Figure 22 - CPN sub-process timer mapping.

It is important to note that the connection between task and *OK* place, and between task and *NOK* place, has to use different arc variables. Also the skip transitions and task transitions have to have additional guard functions. This function allows to fire either of these transitions if there is a token with a needed identifier in either *OK* or *NOK* place. It is important to use different variables in the incoming arcs to tasks and skip functions, because this allows us to use guard functions to check if only the identifiers mach. If the variables would be the same (e.g. only *c*), then without the guard function the transition will be active only if all incoming places have the same token available. It will be a problem when for the case token has been changed (for example after the execution of the first task).

The last skip transition is directly connected to the timeout exception output and the process flow goes to the event handler from there. If the timeout does not happen, then the process flow goes to the normal output. With this kind of mapping solution we can be sure that the model is able to handle concurrently running processes although it does not behave properly if the sub-process itself is part of a loop. If we have to support more than one alternate path (e.g. when using sub-process boundary message event), then this solution has to be extended. We will discuss this extension in the next chapter when we introduce message event mappings.

4.3.6. Sub-process messages

During the execution of a sub-process we can also model the arrival of different messages. For this sub-process boundary message events are used. These messages can be interrupting and non-interrupting. Interrupting message will cancel the normal sub-process execution exactly like the timer event we discussed earlier. The only difference between the timer and message event is that messages can arrive during the sub-process execution with some kind of time function. To add support for multiple sub-process boundary events we have to extend our previously introduced CPN mapping. An example model with two message events and a timer can be seen Figure 23.
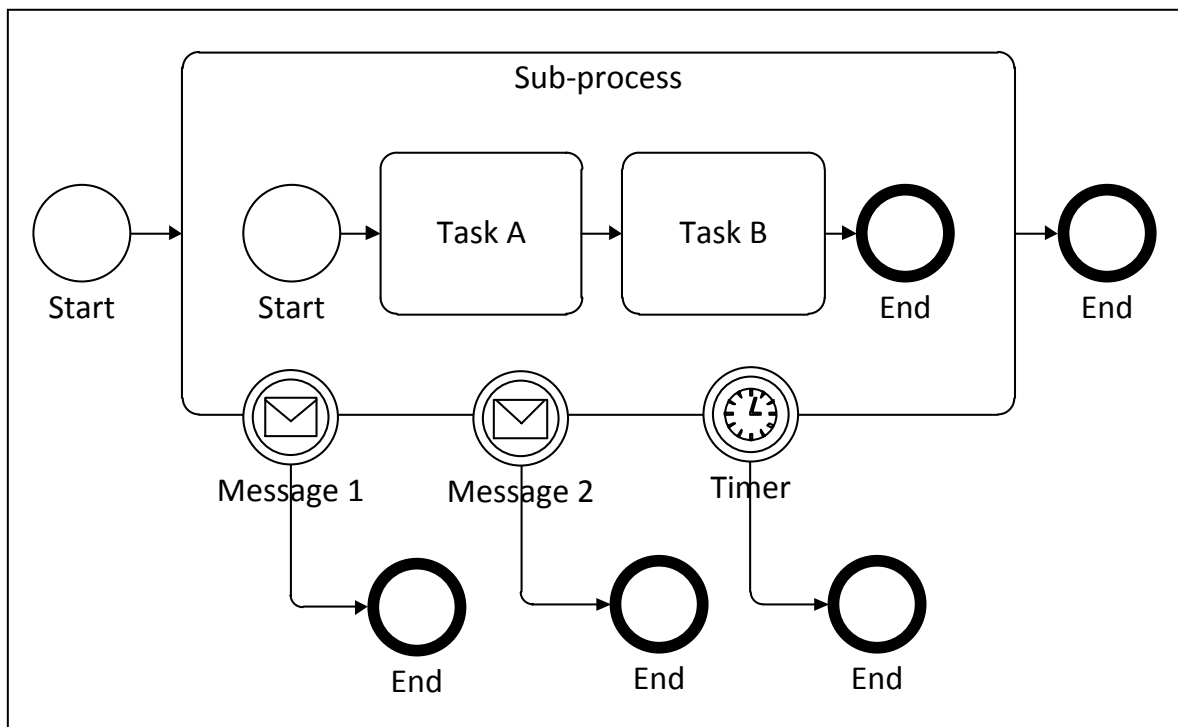
Figure 23 - BPMN model with multiple boundary events.

We previously introduced task skip functions to transfer process instance tokens out of the sub-process to the timer exception output. We do not have to add more skip functions if we are using more than one boundary event but we have to add additional data to the token that will go to the control flow (*OK* place, *NOK* place etc). The result of this extended mapping can be seen on Figure 24. First if the process instance token enters the sub-process the exception path will be selected immediately, because we are able to calculate the first occurring event in case any of them is able to fire. The ability to fire depends on how much time it takes for the tasks to execute and on other time consuming activities.

This selection is done in the transition action as can be seen in our example. The action function will decorate the token with the output exception path identifier that will be used when the exception occurs. For example if the arrival time for the second message will be 10 time units, then after the first task the exception will occur, the second task will be skipped and *Message 2* transition will fire.
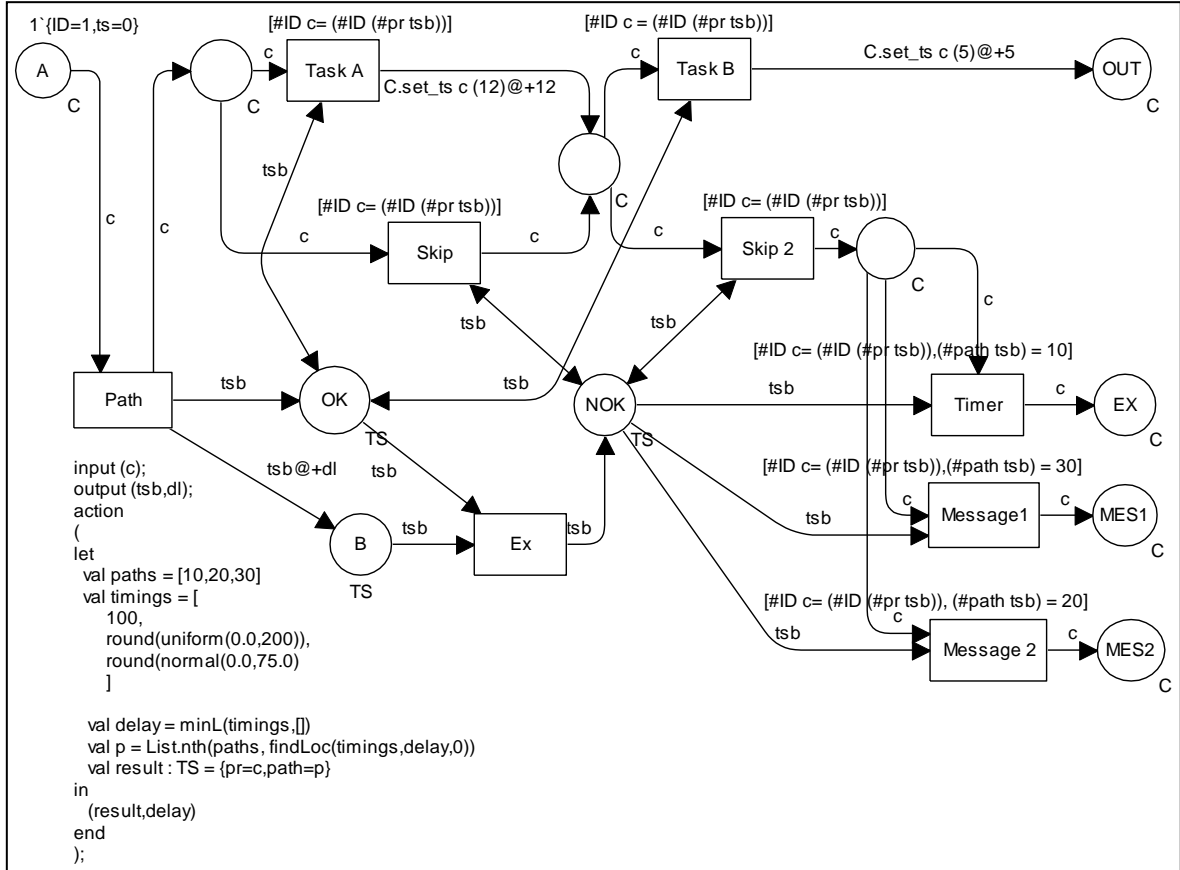
Figure 24 - CPN mapping for multiple sub-process boundary events.

4.3.7. Sub-process error events

Error events are used in a sub-process to cancel the execution in a sub-process instance. In a BPMN process model an error is represented as a special type of end node as can be seen on Figure 25, where we have an error event after the *Cancel order* task. The Error will be caught by the intermediate Error event which is on the boundary of the sub-process.

Converting this kind of error throwing and catching to a CPN module is rather complex, because we have to make sure that after an error has occurred, none of the tokens in a sub-process will move to the normal end. For example, if we have an error event after an AND split, then the path without error event has to be cancelled and the token has to be taken out

34

from the sub-process without executing any tasks and the token is also not allowed to continue its flow after the normal sub-process end event. To achieve this we have extended our previous mapping model on Figure 24. The result of this extension can be seen Figure 6, where we provide only the added exception task and status places.
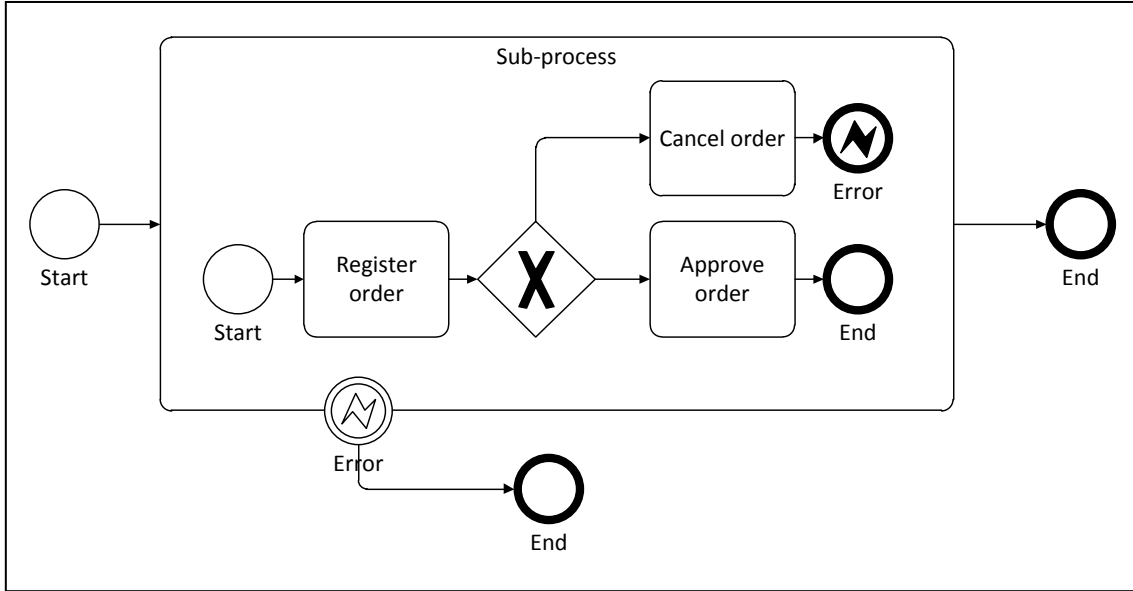
Figure 25 - Sub-process with an error event.

First when we trigger an error event after receiving a token in *Error IN* place, as seen on Figure 26, we have to activate the skip functions by moving the process instance token from *OK* place to *NOK* place. This is done by the error event transition. Note that the error transition can move the process instance directly to the error event output. Secondly we have to make sure that if there are more tokens in the sub-process then they do not start more tasks and do not go to the normal sub-process end event also via the skip transitions. To achieve this we place the status token from *OK* place to *NOK* place and set its path identifier to 0. This means that in the end of the sub-process flow this corresponding case token will not take normal output and is destroyed. On Figure 26, there is a transition *Clean* that will take the token from *NOK* place and from the last skip function end place and destroys them both, because the process flow has already continued from the exception handler.

Note that this solution behaves correctly, but it does leave a token in the place *B* if the error occurs, because the error event does not wait until the timeout has occurred. This problem can be solved by adding another cleaner function.

35
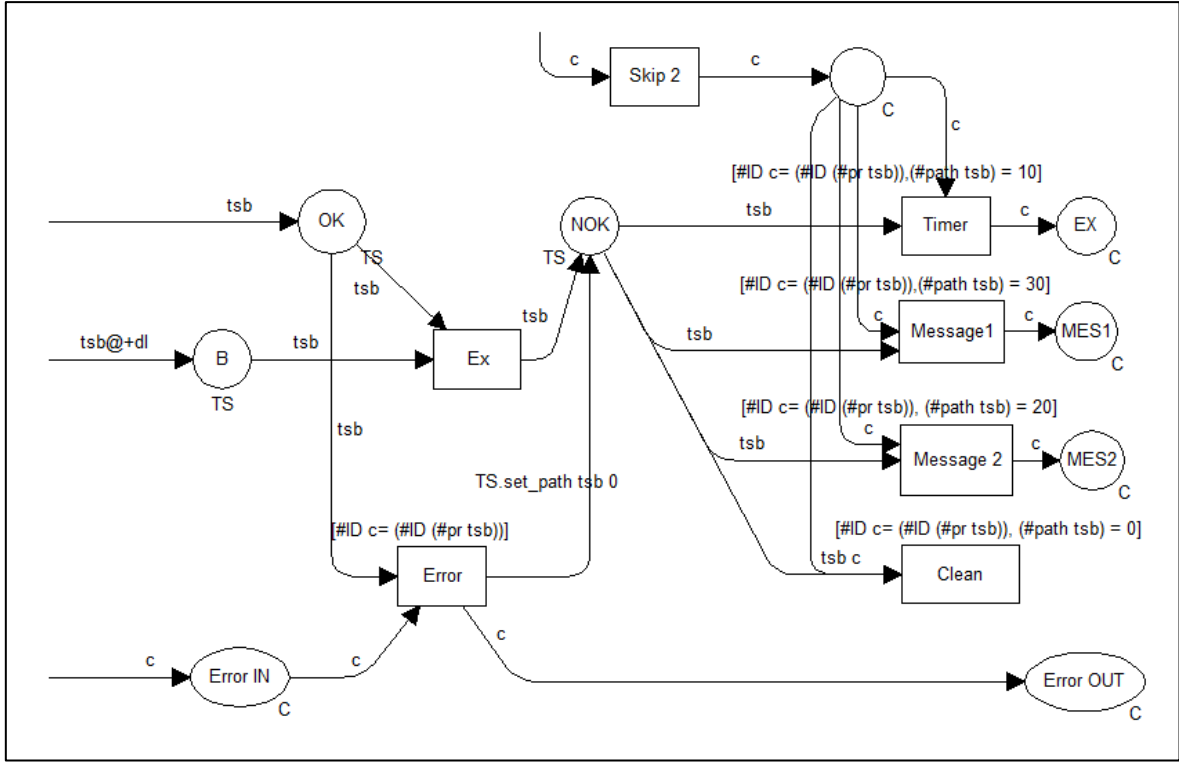
Figure 26 – CPN Mapping of sub-process error event.

4.4. *Conclusions*

BPMN is a widely used process modelling notation, but it lacks a standardized support for process simulation. To overcome this problem it is useful to convert BPMN process models into CPN models to support advanced simulation techniques. These previously presented mappings show us that it is possible to convert different BPMN elements into Petri net constructs. When using plain Petri nets then the end result is not usable for simulation purposes and the mappings are only important to gain understanding of the BPMN to CPN conversion method. To overcome the limitations and add support for advanced simulation models, some higher level Petri net has to be used. Coloured Petri net is highly flexible and allows us to convert more sophisticated constructs with also simulation support. Some of these mappings provided in this thesis have already been discussed in various papers. Our contribution is mostly in the advanced mapping constructs like task boundary events and sub-process boundary events.

5. Architecture

As a part of this thesis we designed and also implemented open and extensible process converter architecture. The architecture is designed in a way that it would be dependent on only a small subset of tools (e.g. CPN, CPN Tools simulator) and allow it to be used with many other existing software packages available. The core and fixed part of this architecture deals with a CPN model that will be the basis for this architecture. It means that the input process modelling notation; input file type; output format and even the CPN simulator itself are exchangeable and only the usage of CPN as the intermediate process description language is mandatory. This allows us to overcome the problematical limitations of current tools described also in the paper [20].
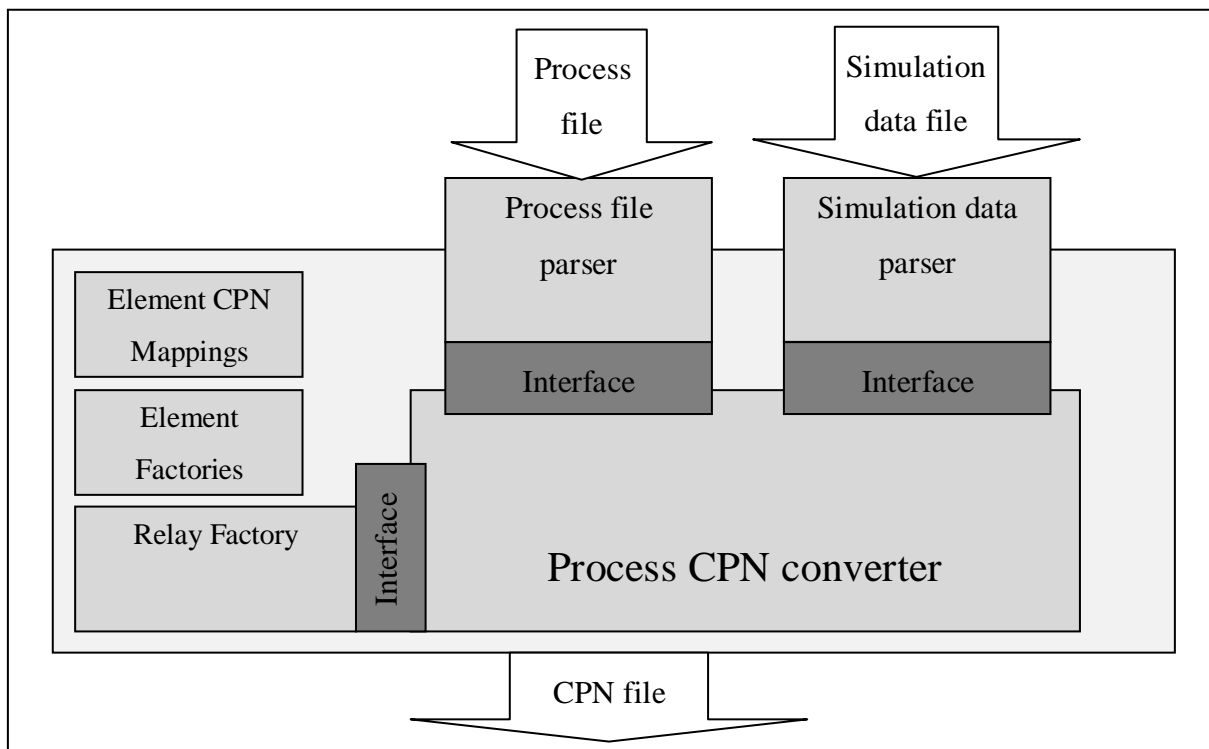
Figure 27 - Process converter architecture.

An overview of the architecture can be seen on Figure 27. The first part is the process converter that takes a process model and simulation data files as an input. The purpose of converter is to generate a simulation ready CPN model that can be handed over to the simulator. Simulation data can also be in the same file, but we have designed the architecture in a way that the simulation data can be also added independently from the process model itself. The file parsers have to implement an interface to assure a common interface

between the converter and the file parser. The same is also true for the element in-memory CPN mapping generation. This solution is discussed in more depth in the following.

In the converter architectural design we used design patterns to assure openness and extendibility. The design patterns we used include abstract/concrete factory pattern and strategy pattern. With abstract/concrete factory pattern we ensured that each model mapping can be changed to another implementation. For example it is possible to define the used mapping implementations from text file. This allows us to switch quickly and easily between different mappings.

To assure extendibility we had to make our architecture independent from process input file format, simulation data input file format and also from the modelling language. An overview of these extendibility aspects can be seen in Table 3.

Extendibility aspect	Motivation	Solution
Process input format.	To support different process serialization formats.	Data file is handled through a separate and independent parser that follows the predefined communication rules.
Simulation data input format.	To support different simulation data input formats. Can be from the process model file or from a separate file.	
Modelling language.	To support various available modelling notation and different subsets of their modelling elements.	Modelling notation is independent from the file parsers and the generation of modelling elements is handled through factories to allow changing and extending.
Different subsets of modelling elements.	Different processes need different subsets of modelling elements.	It is possible to add, remove or modify modelling elements that have to be converted.

Table 3 - Different extendibility aspects.

38

5.1. Independence from process input format

The architecture does not rely on any currently available process exchange file format. Currently there are various process serialization formats available, e.g. Business Process Modelling Language (BPML), Business Process Execution Language and XML Process Definition Language (XPDL). They are all developed with a different target usage in mind and each of them is good for a certain purpose (e.g. using in an execution environment to automate the process). Despite the differences in the target usage there is often a need to simulate the process regardless of its serialization format. This is why we need an extensible architecture for process simulation to handle any of these available process file formats.

In our architecture the process file is read into memory through a concrete file parser. This parser is independent from the other parts of the system and therefore it is easily extendable and changeable. The other parts can communicate with the parser through the communication rules defined in the interface file. It basically means, that any process file parser has to be able to answer simple questions like: what are the elements in the file; what is the type of a certain element and the parser itself does not have to know what will be converted and how. As a proof of concept to show how this architecture works, we have implemented an XPDL 2.0 parser in our prototype solution [17] (The prototype is also available on the accompanied CD-ROM. Look appendix CD ContentsE for more information). It is possible to start using the upcoming BPMN 2.0 standard serialization model by just creating a new concrete parser implementation.
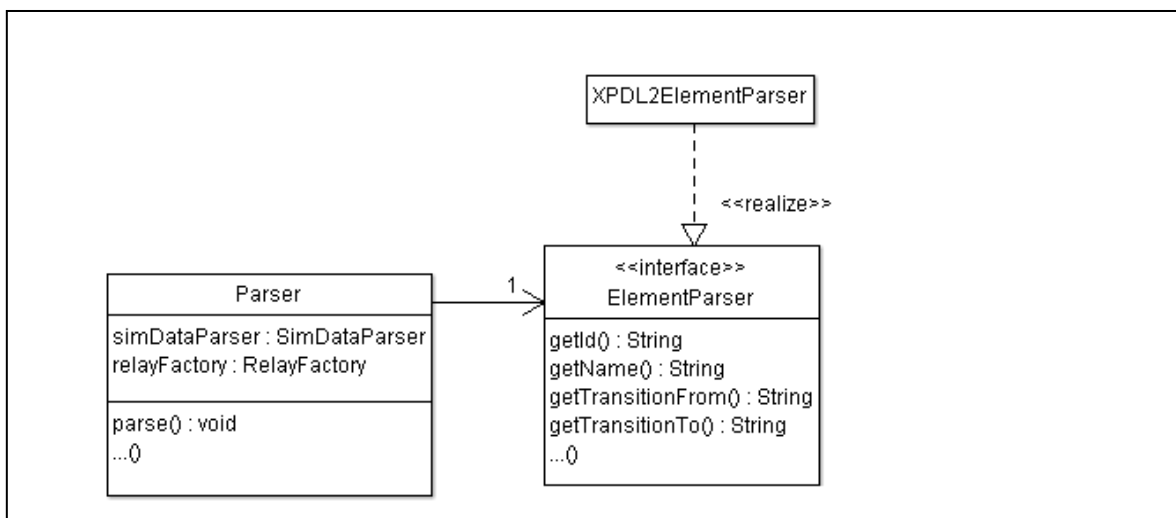
Figure 28 - Parser class diagram.

39

Part of this Java code is represented on Figure 28 as a class diagram. Class *Parser* is the central starting point for the whole conversion process and it has a reference to the concrete process file parser. In our example the *XPDL2ElementParser* is the concrete parser that implements *ElementParser* interface and is thus exchangeable. To use another concrete process file parser (e.g. BPMN 2.0), another parser class has to be created that implements *ElementParser* interface. The concrete element parser reference is set in the *Parser* class before the *parse()* method can be run.

5.2.   *Independence from the simulation data file format*

Not all currently available process model serialization formats and modelling environments support adding simulation data. Usually when there is no simulation engine built into the modelling environment it is also not possible to add simulation data. Therefore our architecture supports simulation data from a separate input file. Although some of the process exchange file formats support adding extended attributes, it is good if the original process file does not have to be changed. Therefore we have a separate simulation file parser interface defined in our architectural solution. It means that the parser for simulation data is independent from the process model parser, but it is also possible to use the same file for both parsers. If we consider BPMN 2.0 where there is support for some simulation data, this same file can be handled to both process parser and to simulation data parser. This part of the implementation in Java code is represented on Figure 29.
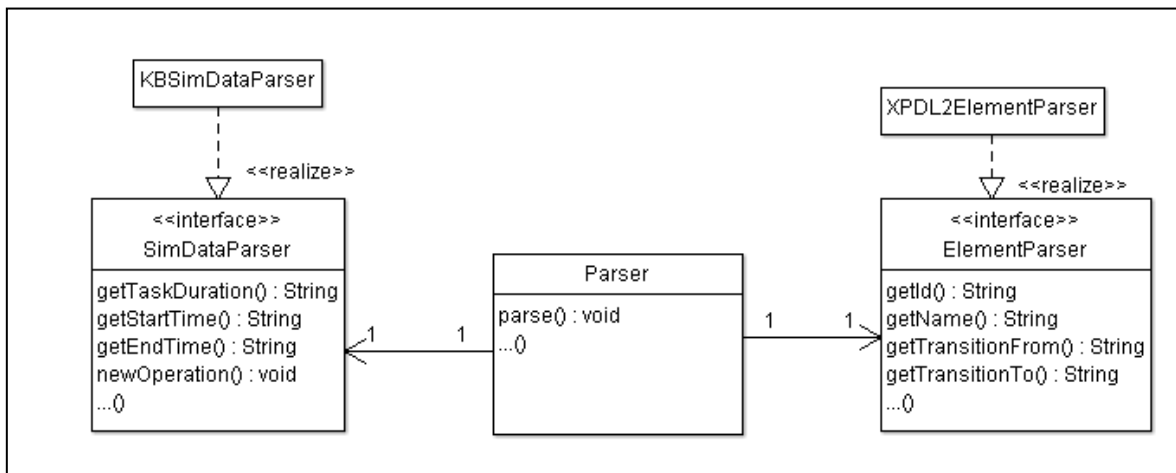
Figure 29 - Parser class diagram.

5.3.   Independence from modelling notation

As our goal was to design a process converter that is open and extensible, it must not impede the usage of any advanced modelling construct. It is important to support different modelling notations currently available and to be prepared for future changes in current modelling standards. For example in the next version of BPMN a new type of intermediate event will be introduced – non-interrupting event. Adding conversion support for this kind of event would be easy in our process converter.

The creation of CPN mappings for different modelling notation elements is handled through abstract/concrete factory pattern. The main parser instance has a reference to a central relay factory that handles the collection of different factories, one for each certain modelling notation element. For instance as seen on Figure 30 we have the abstract relay factory *RelayFactory* and this is extended by a concrete *BPMNRelayFactory* which is responsible for the registration of different BPMN element factories. *BPMNRleayFactory* is responsible for selecting the right factory to generate a needed CPN mapping for certain BPMN elements. To do this the concrete factory has to register the needed element factories (e.g. *BPMNTaskFactory, BPMNGatewayFactory*). When the factories are registered, then it is easy for the parser to just ask the relay factory, which it has reference to, to create CPN mapping for a certain element. This makes the main parser independent from the element factories and thus the mappings and factories are easily exchangeable.

Each modelling notation element has a CPN mapping in a separate Java class and instances of these classes will be generated through the element factories. In our prototype solution BPMN element *Task* CPN mapping is created from *BPMNTaskFactory.*

To add support for another type of modelling notation there has to be made another relay factory that will handle factories for each new modelling element. Also for each modelling element there has to be the CPN mapping class. For example if we are adding support for Event-driven Process Chain (EPC) models we would create *EPCRelayFactory*. From this new relay factory we would reference to the element factories (e.g. *EPCEventFactory, EPCFunctionFactory*). Each of these element factories would be responsible for generating one certain EPC modelling elements. In this way we can set the main parser to reference *EPCRelayFactory* to handle all the element generation. To add support for a new element we only have to make a new mapping implementation.
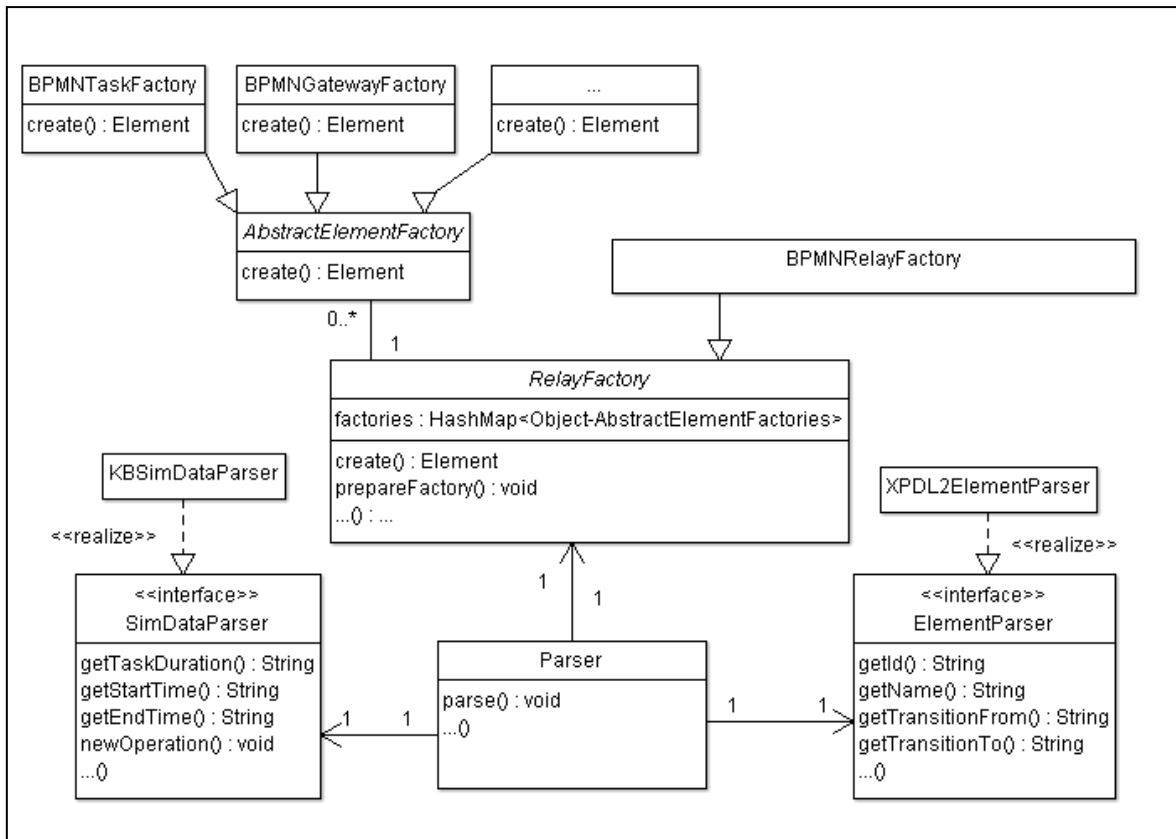
Figure 30 - Parser factories class diagram.

5.4.   Our converter in an end-to-end simulation system

We consider end-to-end simulation system to consist of three main components:

1. Modelling environment;
2. Process simulator;
3. Analytical capability.

There are different commercial and also open source tools available that implement at least one of these components. Some of the well known tools are for example:

1. Oryx-editor – Web-based process modelling environment that has the support for various modelling languages (e.g. BPMN, EPC);
2. CPN Tools simulator – Supports simulation of the Coloured Petri Net models and is able to generate simulation log output as a standardized MXML format;
3. ProM – Generic open-source process mining and analysis tool.

This list is just an example of one specific tool for each part of the process modelling environment. In this thesis we propose an extended version of this previously discussed simulation system. We have extended the initial end-to-end system in a way to support using these already available and previously mentioned freely available tools.
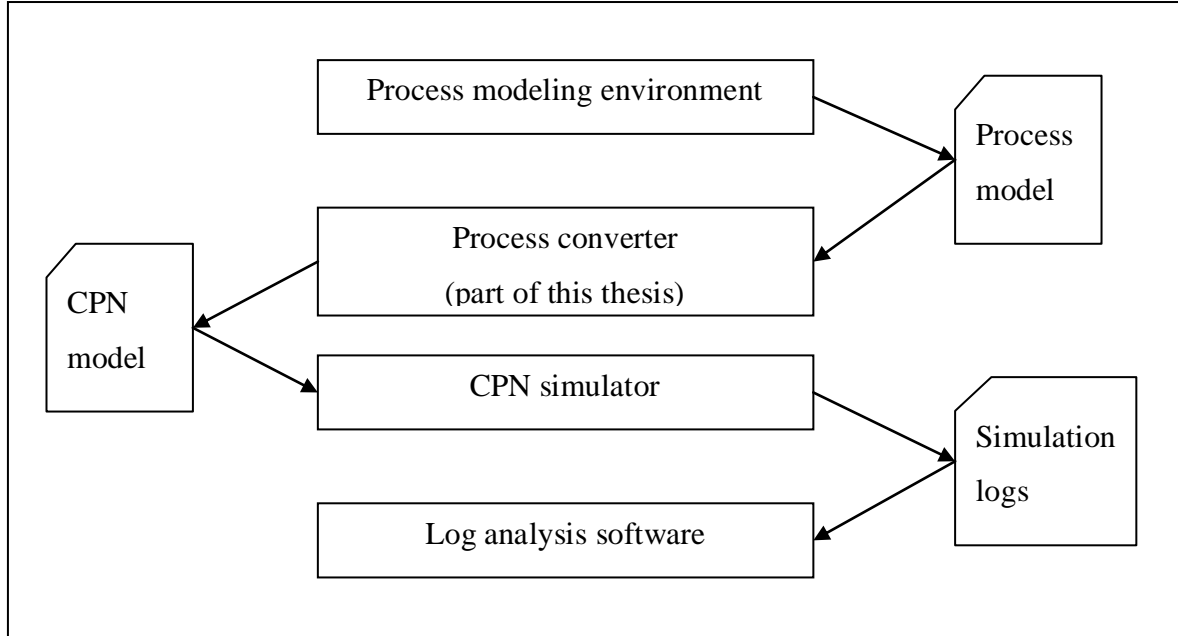
Figure 31 - Extended simulation system.

This system has been extended in a way that all of the business processes will be converted into CPN modelling language to support advanced simulation functionality that most current end-to-end tools lack of. CPN gives us enough flexibility to support almost any imaginable workflow pattern and simulation need. An overview of this extended system can be seen on Figure 31. We also implemented a prototype version of this converter.

5.5.   *Conclusions*

Most of current simulation tools provide a built-in simulation engine that has limited support for simulation functionality and is not extendable. Here we provided a process converter architecture that is the basis for building process model CPN converter for different modelling notations. Process models in CPN can be extended with the conversion mappings and thus can support advanced simulation constructs. In this thesis we also implemented a version of a converter prototype built on this architecture. It is able to convert BPMN models to simulation ready CPN models, based on the mappings provided in previous chapter.

6. Case of study

In the following we demonstrate the usage of our process converter as a part in the whole end-to-end simulation system. To support an end-to-end simulation system, we used different tools. These tools that we used throughout the case study are the following:

1. Modelling the process model – We used an open-source project *Sketchpad BPMN* as our modelling environment. This tool is currently under development and is available here [18].

2. Simulation data – To add simulation data for the process model we used a simple XML file created with text editor.

3. Process converter – We converted our BPMN model to a CPN simulation ready model with our prototype converter.

4. Simulation – To simulate the converted CPN model we used process simulator from the CPN Tools project. This simulator can be used separately from the CPN Tools editor, but as we did the simulation manually, we used the editor to execute the simulation.

5. Simulation log conversion - The simulator tool generated one file for each process instance. To merge these separate files we used ProM Import Framework. This tool has special support for merging log files that were generated from CPN Tools simulator.

6. Results analysis – To analyze the simulation results we used ProM 5.2. This is an open source project and it is available here [15].

6.1. Preparing the process model and simulation data

To demonstrate the whole simulation lifecycle we use a BPMN process model that is complete and ready for simulation and it can be seen on Figure 32.
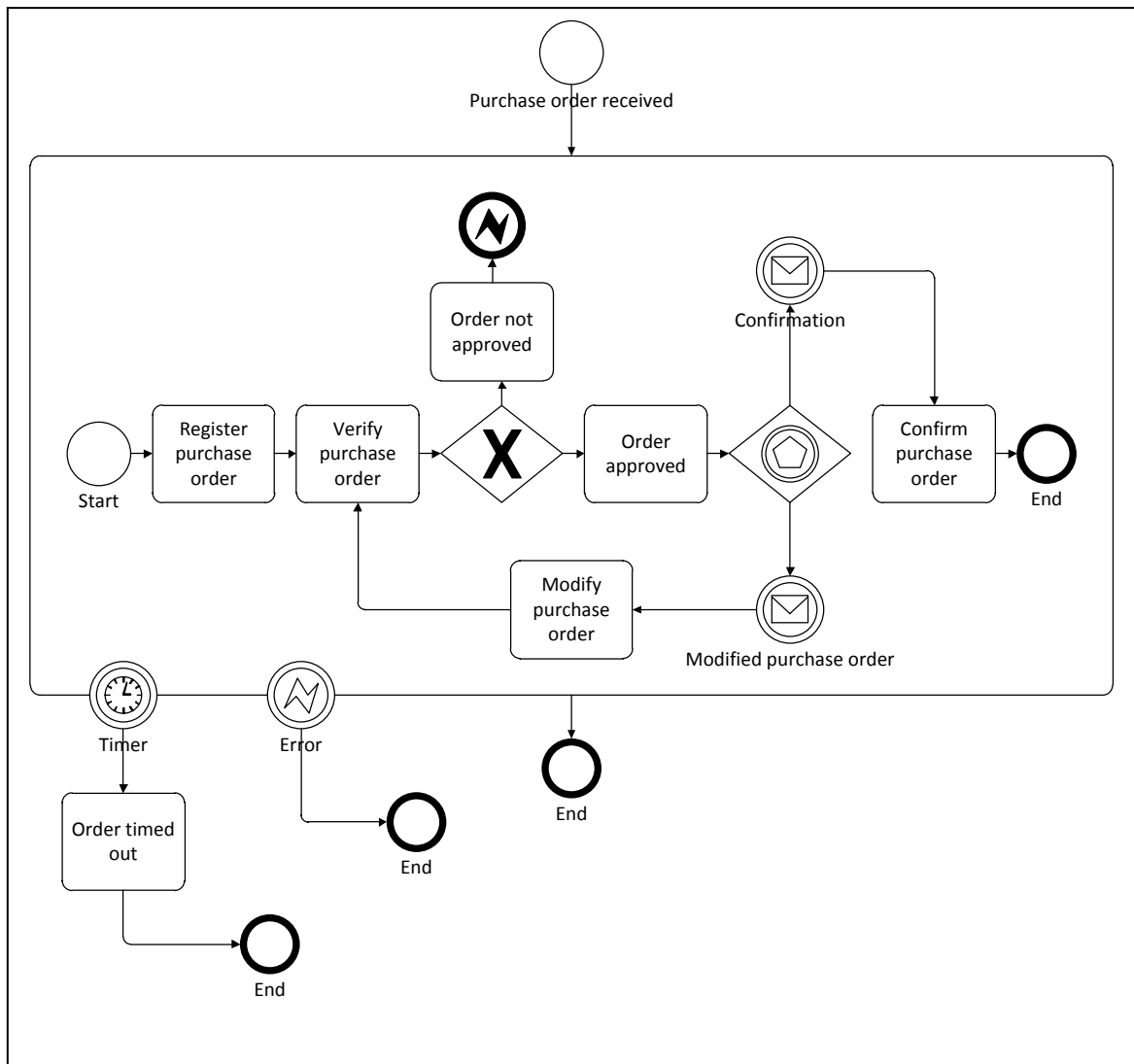
Figure 32 - Example BPMN model.

It is a simplified purchase order management process that after modelling in BPMN includes different BPMN elements which conversion is currently supported by our prototype CPN converter. It includes tasks, different gateways, a sub-process, sub-process timer and different end events.

The process starts when a new purchase order is received. The purchase is first automatically registered in the system (task in the model: Register purchase order). As this registration is done by the system automatically, it does not need the interaction of any worker and thus we consider the execution time as 0 minutes. After the registration there is a manual approval process that will be handled by a clerk. This task takes 20 minutes on average and after that the purchase order is either approved and the process continues or the order is not approved and the process ends with an error output in our BPMN model.

45

As can be seen on Figure 32, the error occurs in the sub-process and it is handled with the sub-process boundary event. This error handling ends the process as it is directly connected to the process end event. We also know that 30% of the approval requests are not approved and 70% of the time the case is approved accordingly. This is in our process modelled as an exclusive gateway. After the order has been approved it can be confirmed or modified by the order submitter. This confirmation or modification proposal is sent to the clerk as a written document. We also know that 20% of the time after the order has been approved the customer writes a modification proposal. 80% of the time the purchase order does not have to be changed anymore and the customer approves the order. The confirmation and modifications to the purchase order are received in uniform distribution between 0 and 1.5 hours. This part is in our BPMN model represented as an event-based gateway with two intermediate message events connected to it. If the confirmation is received from the buyer then it takes 10 minutes for the clerk to finally confirm the purchase order. If the buyer sends modifications, then it takes the clerk 20 minutes to process these modifications and this new modified version of this purchase order will be again verified. This verification can be done by any of the available clerks.

This process starting from registering the purchase order and ending with a non-approval or confirmation can take a maximum of 4 hours. If this time has passed after the order is registered, this order is not valid anymore and the whole approval process has to be cancelled. This means that after the time has passed no new activities in the process will be started and the process ends with a timeout.

We also assume that the initial orders are received after every 10 minutes and we have always 4 clerks working. This simulation data can be also seen in Table 4 and Table 5.

If the process has been modelled in BPMN Sketchpad tool, it allows saving the process into XPDL 2.0 file format. Our prototype converter has an implementation of XPDL 2.0 parser and therefore this file can be converted after adding the simulation data from a separate file.

To add simulation data we use separate XML file which has references to the previous BPMN model elements in the XPDL file. An example of this XML file can be also seen on Figure 33. We used our own simulation data metamodel and it only supports the values that were needed to run basic simulations with our prototype converter. The completion of

simulation schema was not part of this thesis because our converter architecture is independent from the simulation data input format and can be extended in the future. This example metamodel XSD schema can be seen in Appendix D.

Task	Performer	Duration
Register purchase order	System	0 (automatic process)
Verify purchase order	Clerk	20 minutes (1200 seconds)
Confirm purchase order	Clerk	10 minutes (600 seconds)
Modify purchase order	Clerk	20 minutes (1200 seconds)
Order not approved	-	These tasks are needed only for generating log events.
Order approved	-	
Order timed out	-	

Table 4 - BPMN tasks simulation data.

Element	Simulation data
XOR gateway	Order not approved – 30% Order approved – 70%
Event-based gateway	Confirmation message – 80% Modifications message – 20%
Confirmation message	Received in uniform distribution between 0 and 1.5 hours
Modification message	

Table 5 - Control flow simulation data.

```
<SimulationData>
  <SimulationProfile>
    <StartTime>2000,1,1,0,0,0</StartTime>
    <EndTime>2001,1,1,0,0,0</EndTime>
    <TokensInBundle>1</TokensInBundle>
    <TotalTokens>100</TotalTokens>
    <CostPerToken>0</CostPerToken>
    <TimeBetweenBundles>600</TimeBetweenBundles>
  </SimulationProfile>
  <Tasks>
    <Task>
      <Id>53</Id>
      <ProcessingTime>1200</ProcessingTime>
      <ResourceType>CLERK</ResourceType>
    </Task>
                    …
  </Tasks>
  <Gateways>
    <Gateway>
      <Id>48</Id>
      <GateRefs>
        <GateRef>
          <IdRef>58</IdRef>
          <Probability>20</Probability>
        </GateRef>
                    …
      </GateRefs>
    </Gateway>
                    …
  </Gateways>
  <MessageEvents>
  <MessageEvent>
  <Id>64</Id>
  <Time>5400</Time>
  </MessageEvent>
                    …
  </MessageEvents>
  <Resources>
    <Resource>
      <ResourceType>CLERK</ResourceType>
      <ResourceAmount>4</ResourceAmount>
    </Resource>
  </Resources>
</SimulationData>
```

Figure 33 – Simulation data in XML file.

6.2.   Converting the process

To convert our process into CPN representation, we used our prototype converter. It is possible to run the conversion directly from the Java code. To do this we have to define the

location of the input simulation process file and simulation data files created previously. Doing this from Java code is straightforward and an example of this can be seen on Figure 34. To do the same from a precompiled Java jar file, we would have to execute the following command: *java –jar cpnConverter.jar process.xpdl output.cpn simulation_data.xml*.

```java
File xpdlFile = new File("process.xpdl");
File simDataFile = new File("simulation_data.xml");

Parser p = new Parser();
p.setElementFactory(new BPMNRelayFactory(p));
p.setElementParser(new XPDL2ElementParser(xpdlFile));
p.setSimDataParser(new KBSimDataParser(simDataFile));
p.parse();
p.save("output.cpn", true);
```

Figure 34 – Part of the Java code to start the conversion process.

6.3.   *Simulating the process*

There are different options to run the simulation on the generated CPN. We did this by using CPN Tools graphical environment because if the simulation is to be executed manually it is easier to use GUI than to use the same simulator from a plain system console environment. Short introduction to CPN Tools can also be seen in appendix 0. CPN Tools generates a separate log file for each executed process instance. This log consists of audit trail entries that are generated in each BPMN task. It is possible to generate log entries from every element but the entries from tasks are the most important ones. An example of a log entry can be seen on Figure 35.

49

```
<AuditTrailEntry>
  <Data>
    <Attribute name="WaitingTime">0</Attribute>
    <Attribute name="WaitTimeCost">0</Attribute>
    <Attribute name="ModelTimeStamp">0</Attribute>
  </Data>
  <WorkflowModelElement>Approve purchase order</WorkflowModelElement>
 <EventType>start</EventType>
  <Timestamp>2000-01-01T00:00:00.000+01:00</Timestamp>
  <Originator>CLERK</Originator>
</AuditTrailEntry>
<AuditTrailEntry>
  <Data>
    <Attribute name="ProcessingTime">1200</Attribute>
    <Attribute name="NoOfResources">1</Attribute>
    <Attribute name="ModelTimeStamp">1200</Attribute>
  </Data>
  <WorkflowModelElement>Approve purchase order</WorkflowModelElement>
  <EventType>complete</EventType>
  <Timestamp>2000-01-01T00:20:00.000+01:00</Timestamp>
  <Originator>CLERK</Originator>
</AuditTrailEntry>
```

Figure 35 – Simulation log trail entries.

As a final result we only need on MXML standard simulation log file. To convert all these log files generated from CPN tools simulator, we used ProM Import utility. A screenshot of this tool can be seen on Figure 36.

50
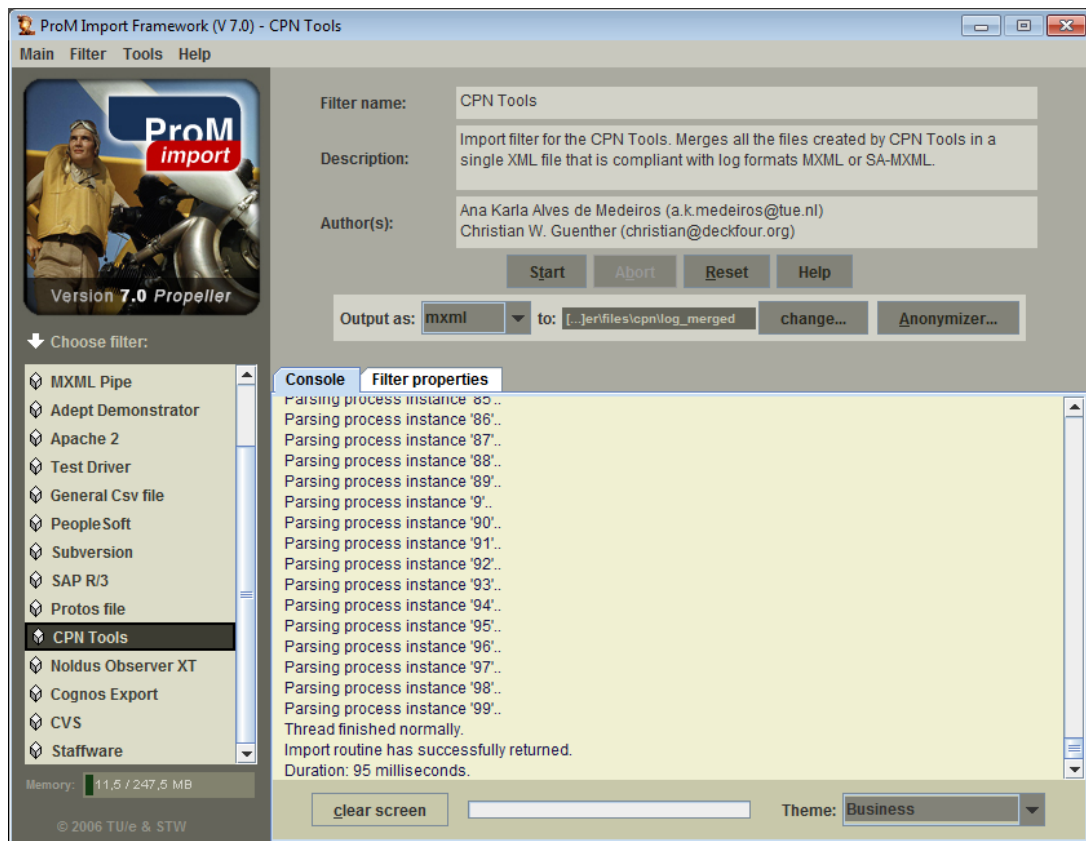
Figure 36 - Screenshot of ProM Import tool.

This utility asks for a directory where the process instance log files are and then generates a new combined log file as an output.

6.4. Simulation result analysis

As a result of running the simulation and merging the log files we now have one standard MXML log file that can be analyzed with ProM. This tool allows us to open the log file and do different analysis with it. First thing when you open the log file is a dashboard where you can see the summary of key data in your log file. This is also presented on Figure 37. From there we can see the number of cases that were executed and how many events these cases generated in total.

To see the basic information about how the order process ended, we can use log summary analysis. It finds all the end events and shows their occurrence probabilities. An example of this can be seen on Figure 38.
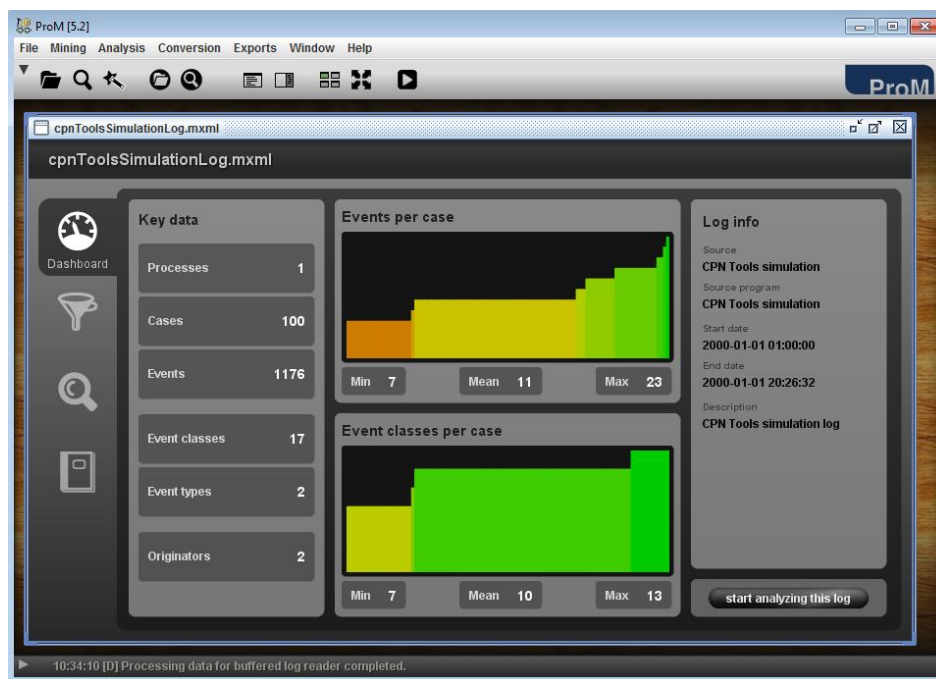
Figure 37 - ProM dashboard.

Figure 38 - Log summary in ProM.

6.5.   Conclusions

In this chapter we demonstrated our process converter architecture and showed how it can be part of an end to end simulation environment to support advanced simulation constructs. First a process model was designed in SketchpadBPMN and exported into XPDL file format. This file format is supported our prototype and it converted our model to a CPN simulation ready process model. Simulation data was added separately from another XML file. The converted CPN file was then simulated in CPN Tools environment which produced an output in multiple log files. Then ProM Import framework was used to combine these log files to one log file that we could analyze later in ProM process mining tool.

7. Conclusion and future work

In this thesis we analyzed several general purpose simulation tools and we concluded that most of these products provide only a fix set of simulation capabilities and that they cover only a subset of constructs of the underlying modelling notation. They mostly support de-facto standard process modelling notation BPMN, but lack the support for its simulation. Some of the tools allow to simulate only the minimum set of BPMN elements (tasks, gateways, start event, end event) and do not provide simulation support for other elements like intermediate events. Also the simulation data that can be added to the supported elements is usually very limited and in case there is a need to construct and simulate a process model in some specific domain, it can turn out to be impossible. The second problem with these tools is that besides the limited support for process model simulation, they also do not provide any extensibility mechanisms to add the needed functionality.

To overcome these problems our contribution is two-fold. We defined an open and extensible architecture for process converter and a set of advanced BPMN to CPN mappings. Process conversion to a CPN model allows the process to be extended with a comprehensive simulation support. Some of these mappings have also been discussed before in different papers (e.g. [6], [11] and [20]). Our main contribution in this part of the thesis is the mappings of boundary and intermediate events for tasks and sub-processes.

The process converter architecture we implemented is able to solve the problems that are seen in current tools. It makes use of the powerful CPN modelling language and CPN Tools simulator to handle complex simulation models. Upon this architecture it is possible to build a domain specific or general purpose process simulation framework. The CPN mappings we provided in the first part of this thesis are also used in a prototype built on our process converter architecture. This prototype already supports many BPMN elements like: tasks, start events, end events, different gateways, intermediate message and timer events, sub-processes and sub-process boundary events. All of these elements also have support for simulation.

Firstly future research should be done to extend the list of BPMN element to CPN mappings to gain full support for BPMN elements. Also the upcoming BPMN 2.0 should be considered as it adds some new modelling constructs (e.g. non-interrupting events) and extends the notation modelling capabilities.

The simulation schema that we used in our converter prototype supports only the main simulation data elements (e.g. task duration, branching probabilities, simulation profiles, etc). Implementing an extensive support for simulation data was not part of this thesis and is considered an important future work. Our architectural solution is built to be extendable also for adding simulation data and it can be extended based on the work [6] for example.

Thirdly the implemented prototype can be extended in the future to work as a RESTful service, or even Software as a Service. This allows it to be integrated into a full-fledged simulation environment to support end-to-end process simulation.

Avatud ja laiendatav äriprotsesside simulator

Magistritöö (30 EAP)
Karl Blum

Äriprotsesside haldamise üheks väga oluliseks osaks on protsesside simuleerimine. Simuleerimine annab hea võimaluse kontrollida protsesside toimimist ning leida muutmist vajavaid kitsaskohti. Käesolevas magistritöös vaatleme olemasolevaid protsesside simuleerimise keskkondi, mida peetakse hetkel selle valdkonna tippudeks (näiteks TIBCO, IBM WebSphere jt), ning uurime nendes esinevaid probleeme seoses funktsionaalsuse ja laiendatavuse puudumisega. Praktiliselt kõikidel uuritud vahenditel oli probleeme keerukama ülesehitusega protsesside simuleerimisel ning puudub võimalus töövahendit äridomeeni spetsiifikast lähtuvalt vajadusel muuta või täiendada.

Magistritöö põhiosa on jagatud kaheks. Esimeses osas toome välja protsessielementide teisendused *defacto* standard modelleerimisnotatsioonist BPMN, madalama taseme modelleerimiselementideks CPN keeles. Teisenduse kasulikkus seisneb selles, et CPN keeles olev protsess on simuleeritav vahendiga CPN Tools, ning konverteerimist on võimalik kohendada vastavalt vajadustele. Näiteks on võimalus lisada ärispetsiifilisi simulatsiooniandmeid või kasutada mittestandardseid tööde jaotamise mustreid.

Magistritöö teises osas töötame välja täiesti uue protsesside konverteerimise arhitektuuri, mis on kättesaadav avatud lähtekoodina, ning on kergesti laiendatav. Siinkohal tähendab laiendatavus seda, et välja töötatud arhitektuuri on võimalik kasutada erinevatest modelleerimiskeeltest protsesside konverteerimiseks CPN keelde. Näidisena oleme loonud ka prototüübi, mis on suuteline teisendama enamlevinud BPMN elemente simuleerimisvalmidusega CPN elementideks. Edasiste töödena näeme me võimalust laiendada olemasolevat prototüüpi toetamaks kõiki BPMN elemente ning keerulisi simulatsiooniandmeid. Prototüüpi on võimalik ka edasi arendada simuleerimise veebiteenuseks, mida saaksid edukalt kasutada erinevad modelleerimiskeskkonnad simulatsioonide läbiviimiseks.

References

1. W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21-66, 1998.

2. Business Process Cycle: Analyze Phase. http://www.sdn.sap.com/irj/bpx/analyze, (last visited: 06.06.2010).

3. M. Weske. Business Process Management: Concepts, Languages, Architectures. *Springer*, 2007.

4. Object Management Group, Business Process Model and Notation (BPMN) – Version 1.2. 2009.

5. M.H. Jansen-Vullers, M. Netjes. Business Process Simulation - A Tool Survey. *In Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN, University of Aarhus, Denmark*, 2006.

6. M. Zäuram. Business Process Simulation Using Coloured Petri Nets. Master Thesis, *Insitute of Computer Science, University of Tartu*, June 2010.

7. K. Jensen, L. M. Kristensen, L. Wells. Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. *International Journal on Software Tools for Technology Transfer (STTT)*, 9(3):213-254, 2007.

8. M. E. Porter. Competitive Advantage: Creating and Sustaining Superior Performance. *Free Press*, 1985.

9. V.Bosilj-Vuksic, V. Hlupic. Criteria for the Evaluation of business Process Simulation Tools. *Interdisciplinary Journal of Information, Knowledge, and Management*, 2:73-88 2007.

10. Extensibility. http://en.wikipedia.org/wiki/Extensibility (last visited: 06.06.2010).

11. R. M. Dijkman, M. Dumas, C. Ouyang, Formal Semantics and Automated Analysis of BPMN Process Models, 2007.

12. M. zur Muehlen, J. Recker. How Much Language is Enough? Theoretical and Practical Use of the Business Process Modeling Notation. *Proceedings of the 20th international conference on Advanced Information Systems Engineering*, *LNCS*, 5074:465-479, 2008.

13. M. W. Barnett. Modeling & Simulation in Business Process Management. *BP Trends Newsletter, White Papers & Technical Briefs*, 10(1), 2003.

14. Petri net. http://www.scholarpedia.org/article/Petri_net, (last visited: 06.06.2010).

15. ProM – Framework for Process Mining. http://sourceforge.net/projects/prom/, (last visited: 12.06.2010).

16. F. Gottschalk, W.M.P. van der Aalst, M.H. Jansen-Vullers, H.M.W. Verbeek. Protos2CPN: Using Colored Petri Nets for Configuring and Testing Business Processes. *Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN, University of Aarhus, Denmark* 2006.

17. Prototype CPN converter. https://code.google.com/p/xpdl-to-cpn/, (last visited: 12.06.2010).

18. Sketchpad BPMN. http://sourceforge.net/projects/sketchpadbpmn/, (last visited: 06.06.2010).

19. C. Wolf, P. Harmon. The State of Business Process Management 2010. *BPTrends,* 2010.

20. L. G. Bañuelos, M. Dumas. Towards an Open and Extensible Business Process Simulation Engine, *Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN, University of Aarhus, Denmark*, 2009.

21. V. V. Bulitko, D. C. Wilkins. Using Petri Net to Represent Context in Blackboard Scheduling. *Proceedings of the American Association for Artificial Intelligence (AAAI) Workshop on Reasoning in Context for AI Applications*, 1999.

22. What is CPN Tools? http://wiki.daimi.au.dk/cpntools/what_is_cpn_tools.wiki, (last visited: 06.06.2010).

23. W.M.P van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. *Workflow Patterns. Distributed and Parallel Databases*, 14(3):5-51, July 2003.

Apendices

A. BPMN

BPMN will provide businesses with the capability of understanding their internal business procedures in a graphical notation and will give organizations the ability to communicate these procedures in a standard manner. There are scores of process modelling tools and methodologies. Given that individuals will move from one company to another and that companies will merge and diverge, it is likely that business analysts are required to understand multiple representations of business processes – potentially different representations of the same process as it moves through its lifecycle of development, implementation, execution, monitoring, and analysis. Therefore, a standard graphical notation will facilitate the understanding of the performance collaborations and business transactions within and between the organizations [4]. BPMN will follow the tradition of flowcharting notations for readability; yet still provide a mapping to the executable constructs.

BPMN contains more than 50 modelling constructs, but the core set of constructs is used most frequently [12]. These constructs are shown on Figure 39.
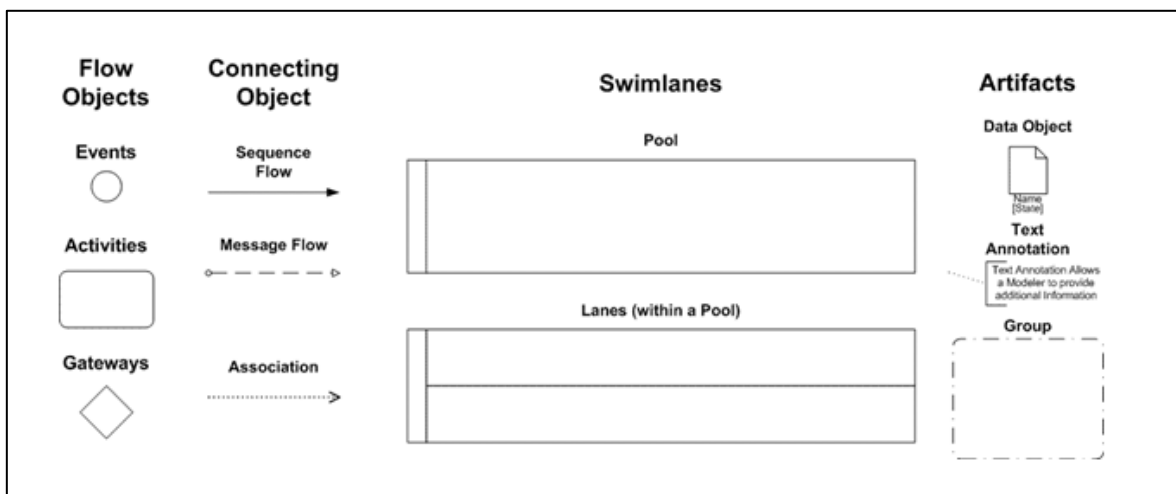

Figure 39 - Basic BPMN elements.

There are different types of events available. Start and end events specify the overall process starting (start event) and end point (end event). Intermediate events are used in the process flow to model the arrival of a message event (intermediate message event), occurrence of an error (intermediate error event) or a certain time being reached

(intermediate timer event). An activity is work that is performed within a business process. An activity can be atomic or non-atomic (compound). The types of activities that are a part of a Business Process Diagram are: Process, Sub-Process, and Task. [4] Task is a typical workflow activity that corresponds to some kind of work that has to be done in the real world (e.g. check validity of application, collect money, deliver goods). Process is a collection of these activities and the activities in a process can be grouped into sub-processes. Gateways are modelling elements that are used to control how Sequence Flow interacts as they converge and diverge within a Process. If the flow does not need to be controlled, then a Gateway is not needed. The term "Gateway" implies that there is a gating mechanism that either allows or disallows passage through the Gateway--that is, as Tokens arrive at a Gateway, they can be merged together on input and/or split apart on output as the Gateway mechanisms are invoked. To be more descriptive, a Gateway is actually a collection of "Gates" [4]. Gateways are represented as a diamond and can have different markings inside depending on the behaviour of the gateway. Gateways can define different sequence flow behaviour for flow joining and branching. Different types of gateways can be seen on Figure 40.
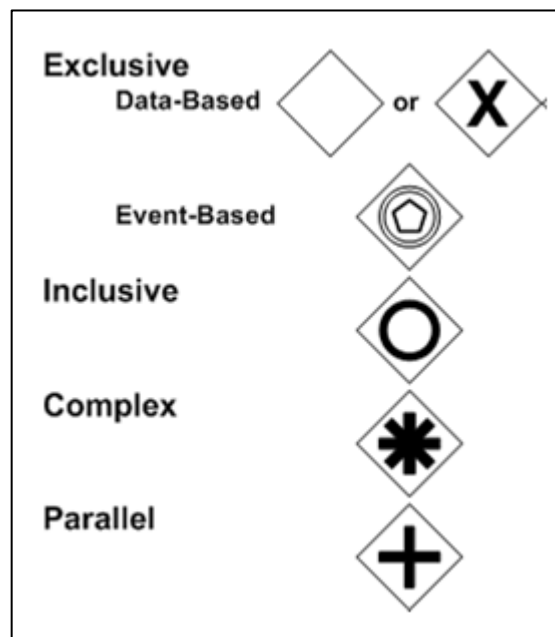

Figure 40 - BPMN gateway types.

B. Coloured Petri Nets

A Petri net is a graphical tool for the description and analysis of concurrent processes which arise in systems with many components (distributed systems). The graphics, together with the rules for their coarsening and refinement, were invented in August 1939 by the German Carl Adam Petri - at the age of 13 [14].

Petri net is a directed bipartite graph with two node types called places and transitions. The nodes are connected via directed arcs. Connections between two nodes of the same type are not allowed [1]. Places are represented by circles and transitions by rectangles. An example of a simple Petri net can be seen on Figure 41.
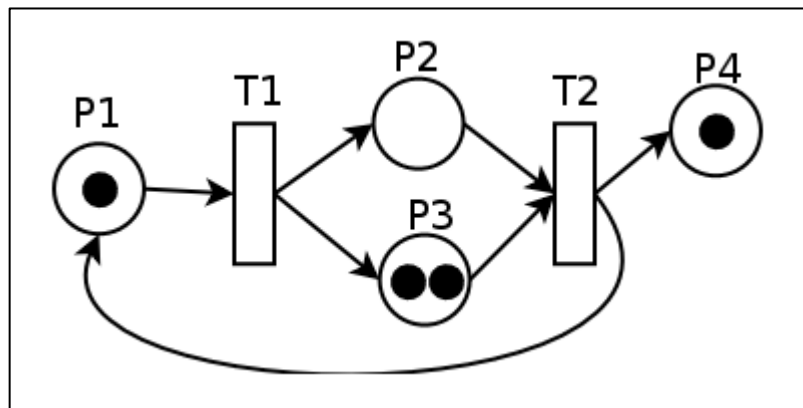
Figure 41 - Example of a Petri net.

Each place can have one or more tokens in it. The placing of tokens represents some certain state in the process. The purpose of transitions is to move tokens from one place to another place or places. The transition is able to fire if and only if there is a token available in each place that is connected to the transition via an incoming arc. After the transition fires, it consumes these tokens and generates a new token to each place connected with an outgoing arc. In our example only transition T1 can fire and after firing it will consume the token in place P1 and produce two tokens – one token to place P2 and one to place P3.

While being a powerful and attractive modelling tool, the classical Petri nets lack several features helpful for modelling complex real-time systems and processes [21]. Some of these shortfalls when modelling business processes according to [21] are:

- Classical PNs are essentially propositional (i.e. they have no variables or functions);

60

- There is no uncertainty support in classical PNs;
- Every time a transition fires all the enabling tokens are withdrawn from the enabling places.

Also when considering business processes, there is no support for timing information in classical Petri net. To overcome these problems there are various higher level Petri net versions available. One of them is Coloured Petri Net.

Coloured Petri Net (CP-net or CPN) is a graphical language for constructing models of concurrent systems and analyzing their properties. CP-net is a discrete-event modelling language combining Petri nets and the functional programming language CPN ML which is based on Standard Markup Language (SML). The CPN modelling language is a general purpose modelling language, i.e., it is not focused on modelling a specific class of systems, but aimed towards a very broad class of systems that can be characterized as concurrent systems [7].

A CPN model of a system describes the states of the system and the events (transitions) that can cause the system to change state. By making simulations of the CPN model, it is possible to investigate different scenarios and explore the behaviours of the system. Very often, the goal of simulation is to debug and investigate the system design. CP-nets can be simulated interactively or automatically [7]. An example of a CPN model can be seen on Figure 42.
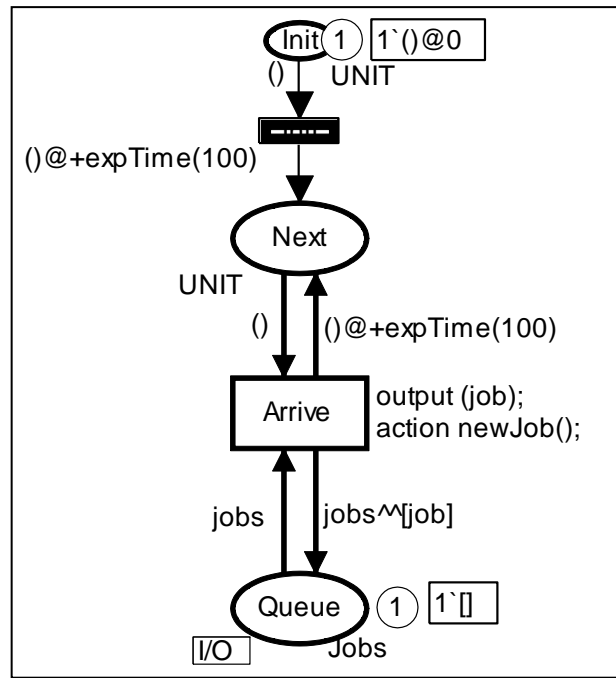
Figure 42 - Example of a CPN model.

CPN has also the ability to extend the model with time concept that is essential in business process simulation. The core building blocks in CPN are the same as in Petri Nets, but these elements behaviour can be extended by adding CPN ML functions and parameters. It is even possible to execute complex function when a certain transition is fired. CPN ML as an extension from SML is also used on place and arc inscriptions to describe initial markings and guards. It also provides a way to implement internal algorithms that can calculate the enabling and occurrence of bindings.

C. CPN Tools

CPN Tools is a tool for managing CPN models. It is possible to edit and simulate these models. The GUI is based on advanced interaction techniques, such as toolglasses, marking menus, and bi-manual interaction. Feedback facilities provide contextual error messages and indicate dependency relationships between net elements. The tool features incremental syntax checking and code generation which take place while a net is being constructed. A fast simulator efficiently handles both untimed and timed nets. Full and partial state spaces can be generated and analyzed, and a standard state space report contains information such as boundedness properties and liveness properties [22]. Example screenshot of this tool can be seen on Figure 43.
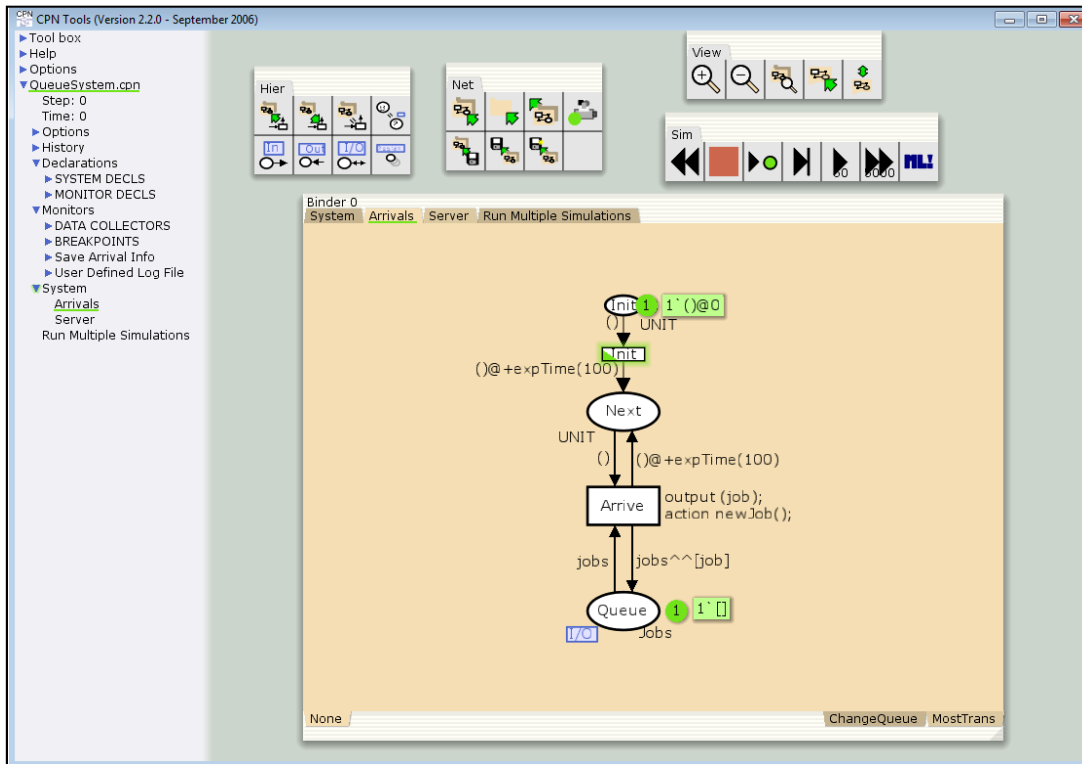

Figure 43 - CPN Tools editor.

CPN Tools also includes a simulation engine that is part of the tool itself but can handle inputs from operating system console. This gives us the ability to run simulations without even using the CPN Tools GUI. This will become important, if we want to use some automated simulation engine and we do not want to interact with the CPN process modelling tool.

D. Simulation schema

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:jaxb="http://java.sun.com/xml/ns/jaxb" elementFormDefault="qualified" jaxb:version="1.0">
        <xs:import namespace="http://www.w3.org/XML/1998/namespace"/>
        <xs:complexType name="Task">
                <xs:sequence>
                        <xs:element name="Id" type="xs:string"/>
                        <xs:element name="ProcessingTime" type="xs:string"/>
                        <xs:element name="ResourceType" type="xs:string"/>
                </xs:sequence>
        </xs:complexType>
        <xs:element name="Task" type="Task"/>
        <xs:complexType name="Tasks">
                <xs:sequence>
                        <xs:element name="Task" type="Task" maxOccurs="unbounded"/>
                </xs:sequence>
        </xs:complexType>
        <xs:element name="Tasks" type="Tasks"/>
        <xs:complexType name="Gateway">
                <xs:sequence>
                        <xs:element name="Id" type="xs:string"/>
                        <xs:element name="GateRefs" type="GateRefs"/>
                </xs:sequence>
        </xs:complexType>
        <xs:element name="Gateway" type="Gateway"/>
        <xs:complexType name="Gateways">
                <xs:sequence>
                        <xs:element name="Gateway" type="Gateway" maxOccurs="unbounded"/>
                </xs:sequence>
        </xs:complexType>
    <xs:element name="Gateways" type="Gateways"/>
        <xs:complexType name="GateRefs">
                <xs:sequence>
                         <xs:element name="GateRef" type="GateRef" maxOccurs="unbounded"/>
                </xs:sequence>
        </xs:complexType>
```

```xml
<xs:element name="GateRefs" type="GateRefs"/>
<xs:complexType name="GateRef">
        <xs:sequence>
                <xs:element name="IdRef" type="xs:string"/>
                <xs:element name="Probability" type="xs:int"/>
        </xs:sequence>
</xs:complexType>
<xs:element name="GateRef" type="GateRef"/>
<xs:element name="SimulationData" type="SimulationData"/>
<xs:complexType name="SimulationData">
        <xs:sequence>
                <xs:element name="SimulationProfile" type="SimulationProfile"
minOccurs="1" maxOccurs="1"/>
                <xs:element name="Tasks" type="Tasks" minOccurs="0"/>
                <xs:element name="Gateways" type="Gateways" minOccurs="0"/>
                <xs:element name="Resources" type="Resources" minOccurs="0"/>
                <xs:element name="MessageEvents" type="MessageEvents"
minOccurs="0"/>
        </xs:sequence>
</xs:complexType>
<xs:element name="SimulationProfile"/>
<xs:complexType name="SimulationProfile">
        <xs:sequence>
                <xs:element name="StartTime" type="xs:string"/>
                <xs:element name="EndTime" type="xs:string"/>
                <xs:element name="TokensInBundle" type="xs:int"/>
                <xs:element name="TotalTokens" type="xs:int"/>
                <xs:element name="CostPerToken" type="xs:int"/>
                <xs:element name="TimeBetweenBundles" type="xs:int"/>
                <xs:element name="Resources" type="Resources"/>
        </xs:sequence>
</xs:complexType>
<xs:element name="Resources"/>
<xs:complexType name="Resources">
        <xs:sequence>
                <xs:element name="Resource" type="Resource"
maxOccurs="unbounded"/>
        </xs:sequence>
</xs:complexType>
```

```
            <xs:complexType name="Resource">
                    <xs:sequence>
                            <xs:element name="ResourceType" type="xs:string"/>
                            <xs:element name="ResourceAmount" type="xs:int"/>
                    </xs:sequence>
            </xs:complexType>
            <xs:element name="MessageEvents"/>
                    <xs:complexType name="MessageEvents">
                    <xs:sequence>
                            <xs:element name="MessageEvent" type="MessageEvent"
maxOccurs="unbounded"/>
                    </xs:sequence>
            </xs:complexType>
            <xs:complexType name="MessageEvent">
                    <xs:sequence>
                            <xs:element name="Id" type="xs:string"/>
                            <xs:element name="Time" type="xs:string"/>
                            <xs:element name="Probability" type="xs:string"/>
                    </xs:sequence>
            </xs:complexType>
</xs:schema>
```

E. CD Contents

The contents of the accompanied CD-ROM are described in the following list:

1. /Source - Contains the source code for prototype CPN converter;
2. /Binary - Contains compiled version of prototype CPN converter;
3. /Examples - Some example XPDL models for testing the prototype.