UNIVERSITY OF TARTU

FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

Institute of Computer Science

**Marek Zäuram**

# Business Process Simulation Using Coloured Petri Nets

**Master's Thesis**

**(20 ap)**

Supervisor: Professor Marlon Dumas

Author: …………………………………….. "….." May 2010

Supervisor: ………………………………… "….." May 2010

TARTU 2010

# Contents

# Introduction

Business process management (BPM) is a management approach that promotes business effectiveness and efficiency while striving for innovation, flexibility and integration with technology. Business process management encompasses a range of activities such as designing, modelling, executing, monitoring, and optimizing business processes [1]. One of the tenets of BPM is that business processes should be placed in a continuous improvement cycle in which design and redesign play an important role. Various possibilities to change a process are considered each time and the best alternative design should replace the current process. Making a wrong choice when selecting a process design may lead to unpleasant surprises and lower process performance instead of yielding the expected gains [2]. Accordingly, it is crucial to analyze alternative business process designs in a rigorous manner.

The modelling of business processes has a significant role in the business process management. Sometimes business process modelling and business process management activities are confound with each other as they share the same abbreviation (BPM)[4]. In other words, business process management focus on overall process management and the business process modelling is a sub activity and concentrates on visualizing the process. The graphical representation of business process information has proven to be more effective for presenting it to business stakeholders, including business analysts and system developers. Business Process Modelling Notation (BPMN, [3]) and the Unified Modelling Language (UML) are two popular graphical representation standards for specifying business processes in a work-flow.

Business process modelling tools provide business users with the ability to model their business processes, implement and execute those models, and refine the models based on as-executed data. As a result, business process modelling tools can provide transparency into business processes, as well as the centralization of corporate business process models and execution metrics. Also modelling and simulation functionality allows for pre-execution “what-if” modelling and simulation. Post-execution optimization is available based on the analysis of actual as-performed metrics [4]. Simulation is one of the techniques suitable for the support of redesign. The simulation of business processes helps in understanding, analyzing, and designing processes. With the use of simulation the (re)designed processes can be evaluated and compared. Simulation provides quantitative estimates of the impact that a process design is likely to have on process performance and a quantitatively supported choice for the best design can be made [2].

Simulating business processes is, to a large extent, overlapping with the simulation of other discrete event systems. Regarding the simulation of business processes a number of steps can be distinguished [2]. There are essentially four steps in doing business process simulations: 1) building a model, 2) running a model, 3) analyzing the performance measures, and 4) evaluating alternative scenarios [5].

First the business process is mapped onto a process model, possibly supplemented with process documentation facilities. Then the sub processes and activities are identified. The control flow definition is created by identifying the entities that flow through the system and describing the connectors that link the different parts of the process. Lastly, the resources are identified and assigned to the activities where they are necessary. The process model should be verified to ensure that the model does not contain errors [2].

Before simulation of a business process, the performance characteristics, such as throughput time and resource utilization, need to be included. For statistically valid simulation results a simulation run should consist of multiple sub runs and each of these sub runs should have a sufficient run length. During the simulation, the simulation clock advances. The simulation tool may show an animated picture of the process flow or real-time fluctuations in the key performance measures. When the simulation has been finished, the simulation results can be analyzed. To draw useful and correct conclusions from these results, statistical input and output data analysis is performed [2].

BPMN is a de facto standard for modelling business processes on a conceptual level i.e. to provide simple and understandable graphical notation to analyze "as-is" models, to find the bottlenecks and to remodel "to-be" processes. Simulating the processes has proven to be very effective way to seek problems from "as-is" processes and give the important input to improve the process. However BPMN standard does not support any kind of simulation capabilities. Lots of modelling tools of business processes provide simulation of the models but usually the simulation engine is hidden inside of the tool and thus only models created with the tools themselves can be simulated. Also it is hard to add new features to the engine or modify the outcome parameters of the results.

This paper presents a body of work aimed at designing a simulator of BPMN process models. Instead of advocating for building a process simulator from scratch, it proposes to reuse an existing discrete event simulator based on Coloured Petri Nets (CPNs). The approach adopted in this work is the following. First, two of BPMN-based process simulation tools were reviewed and from this review, a meta-model of business process simulation that sits on top of BPMN is derived. Next, a mapping between this meta-model and CPNs is designed. In doing so, it focuses on the problem of mapping the simulation attributes in the model, because the problem of mapping plain BPMN models to Petri nets has already been addressed in previous work (see [6]). The proposed design has been validated through one fully-worked case study.

The paper is organized as follows. In the Chapter 1 describes process modelling and simulations using example of IBM WebSphere Modeler and ITP Commerce. Then an overview of Petri nets and Coloured Petri nets in Chapter 2 is given. Chapter 3 introduces a general meta-model of simulations and provides examples and templates how to transform this model to a Coloured Petri net and using CPN Tools for simulating it.

# Chapter 1

# Process Simulation

## 1.1. Brief Introduction to Process Modelling

As business process management is playing more and more important role in business then more companies are trying to satisfy the need of process management tools. Today the business processes and the forms of business are changing very quickly and the tools must apply changes and conform to new requirements. There are lots of different applications in the market. These business modelling tools can divide in different ways, but the most basic way is divide by the groups of users. For example, some are designed to support a specific group of users – say business managers – while others are intended for more technical users like business analysts or IT developers. Setting up a realistic simulation requires a significant amount of specialized skills in order to collect the required input data and to appropriately assign values to all simulation parameters. Consequently, it is difficult to produce a tool that makes it simple for managers to create effective simulations. Thus, there is a natural divide between general purpose process modelling tools that support minimal simulation features and those tools that are, in essence, designed for a more technical audience that understands more about simulation and therefore wants a more sophisticated simulation tool [7].

There are many definitions and requirements stated for business modelling tool. A very frequent requirement is the capability for simulating process models. In [2] are described three different categories of software tools that may be applicable for business process simulations: business process modelling tools; business process management tools; and general purpose simulation tools. The difference between process modelling and management tools is that modelling tools are more simple, just support for visualizing, describing and analyzing the business process (flow, resources, data etc). The process management tools are more complex and are broadened to support the whole process life-cycle.

Current popular process modelling languages and techniques are BPMN, YAWL, EPC and extensions of UML. None of these languages allow one to capture simulations directly. Accordingly, each tool adopts its own extension of these standards with additional data to meet the requirements for simulations. The most major tools like IBM WebSphere Modeler, Oracle BPA Suit etc. are using BPMN standard as graphical representation for workflow of business processes and for simulating the models they extend the standard.

The BPMN standard was created to provide a simple standard understandable for all business stakeholders (business managers, analyst, technical developers etc). The BPMN is created to support only the modelling of business processes, not data-flow or organisational structures for example.

There are four groups of elements in BPMN standard:

- Flow Objects (Events, Activities, Gateways)
- Connecting Objects (Sequence Flow, Message Flow, Association)
- Swimlanes (Pool, Lane)
- Artifacts (Data Object, Group, Annotation)

The most basic BPMN process diagram consists of events (circles), activities (rounded rectangles), gateways (diamonds) and connecting flow objects. An event denotes that something happens. There are 3 types of events: start, intermediate and end events. BPMN specification defines many types of events depending on the type of the trigger. Usually only two basic events, "*message*" and "*timer*" event are used. For example, a timer start event indicates that the process starts at a particular date and time or on a periodic date/time cycle. If timer event is used as an intermediate event then it is a delay to wait for a specified duration or until a specified date/time. The message events are triggered by the receipt of a message. In this paper only these two event trigger types are under consideration. Activities on the other hand denote work that must be done. Gateways are routing objects and are used for forking and merging of paths depending on the conditions expressed. There are 5 main types of gateways: AND-split (starts two or more parallel paths), AND-join (synchronizes/joins two or more parallel paths), XOR-split (decision point, only one path is selected based on condition), XOR-join (merges paths) and event-driven choice (only one path is selected based on first occurring event) gateways. Events, activities and gateways are collectively called *flow objects*. Flow objects are connected to other by means of flows (edges).

Figure 1 shows a simplified model of credit card application in the format of BPMN. It has one start and end events, multiple intermediate events ("Notify acceptance", "Request info" etc), tasks ("Check for completeness", "Perform checks") and gateways ("Decide", "complete?").

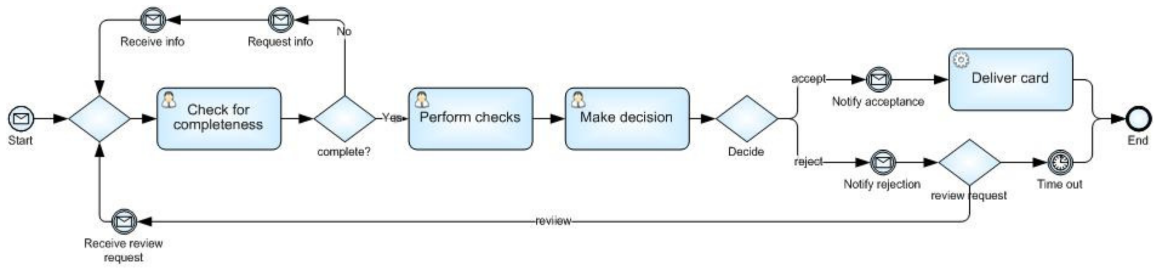


**Figure 1: The process of credit card application in BPMN standard**

## 1.2. Process Simulation in IBM WebSphere Modeler

IBM WebSphere Business Modeler (WebSphere) is a comprehensive tool for business process modelling, analysis and process deployment focused on supporting business process improvement. 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, artifacts, organizations, resources, timetables and locations.

The WebSphere simulation engine enables simulation of the dynamic behaviour of processes and analysis of workloads and bottlenecks. It is possible view analyses on the process, resources, activities and queues in real time during simulation or after completing the simulation. Also it includes report templates and a designer for customized reports. Built in reports can give a precious guidance and present a detailed analysis helping validate and optimize business processes [8].

WebSphere supports viewing and composing models using the Business Process Management Notation (BPMN) diagram style. The BPM notation of WebSphere is customize to support the process simulations and its meta-data, so it is not exactly correspond to BPMN standard. A process model in WebSphere is composed of tasks (rectangles) and connectors (lozenges or solid bars). Tasks correspond to work that needs to be done, while connectors serve to route the flow of control between tasks. For example, the process model in Figure 2 contains one decision connector which routes the flow of control either along the acceptance branch or the rejection branch.

For illustrating the capabilities of WebSphere a simple credit card application process is modelled (see Figure 2). The process will start after customer's credit card application has arrived. The clerk makes preliminary checks (i.e., all required data is filled correctly in the applications) and then registers the application to the system. This action may take approximately 30 minutes. After registering application starts the processing of the application - the clerk performs different checks (e.g. asking additional information from the customer, making credit checks etc.). This task may take 2 hours. After all checks are done then the application moves to the hands of manager. The manage recheck all the information and makes decision (takes another 2 hours). In the case of approval, the customer is notified and the credit card is delivered to the customer. Otherwise the customer is notified about the rejection of the application and he/she can start process from the start. The notifications and the delivery of card are in this case system tasks - these are sub-processes and not monitored in this case.

This example contains three human tasks ("Check for completeness", "Perform checks", "Make decision"), one decision/fork ("Decision"), two notifications ("Notify acceptance", "Notify rejection"), one system task ("Deliver card"), and one start and two end points. For modelling tasks it is needed a define resources at first. Resource can be individual or bulk. Resources represent the people, equipment, or material used to perform a project or a task. Individual resources are resources where a specific instance is required, whereas bulk resources are resources where any instance of a type of resource from a pool can be used. Individual resources include people and computers, and bulk resources include power and water [9].

Bulk resources can represent the material used to perform a project or a task. They can be non-consumable (such as employees, vehicles, or equipment) or consumable (such as fuel or printer paper). Consumable resources are diminished, or perhaps even used up, during the process [9].

Resources may have specified periods when they are available. To specify availability, it is possible to use timetables to indicate the periods of availability. If the timetable is not specified then it is assumed that the resource is always available. Also it is possible to add costs and qualifications to resources. The qualifications are specific roles that this resource must fulfil [9].

When modelling a human task, it is possible to specify the primary owner of the task i.e. the individual/bulk resource or the role that are required to complete the task. If the task requires more than one resource then additional resource requirement can be attached to the task. The current example of credit card application requires two types of roles that can be assigned to resources: clerks, who register applications and making checks and managers, who making decisions. The resource requirement is not only attribute of the task, there lots of more, e.g. the tasks inputs and outputs, costs and duration, required resources, associated locations and organization units, and a name and description can be additionally described. The only difference between a human and a non-human task is that a non-human task does not have a primary owner.

In credit card application model in the Figure 2 has one simple decision which splits the flow into two paths: the acceptance and the rejection path. The decisions in WebSphere can be probabilistic i.e. it is possible to add a probability to each branch of a decision to indicate the probability of that branch running at any given time or based on the defined expressions. WebSphere supports besides the simple decision (two output branches and one is selected) also a multiple-choice decisions. A multiple-choice decision can be exclusive (i.e. only the first path whose condition is true will be taken) or inclusive (i.e. all paths whose conditions are true will be taken) [9].

After the model is created the simulations can be run on it. For running the simulation the snapshot of simulation is required. The creating of simulation snapshot includes the choosing of the resources required for simulation. In simulation snapshot settings different attributes, for example the requirement of resources, the parameters of input trigger etc can be configured. After configuring the simulation snapshot it is possible to run simulations and perform different analysis based on the data generated from the simulations. WebSphere supports also versioning and it is easy to change some parameters of the model and compare the results.

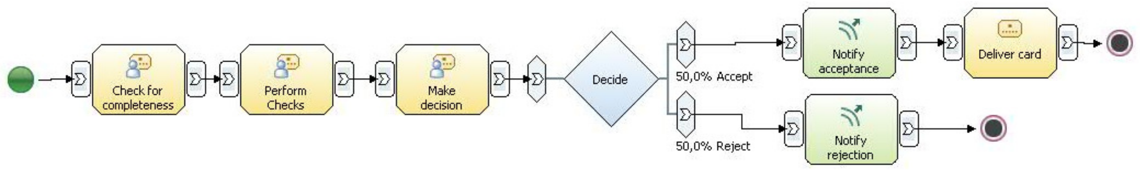


**Figure 2: Credit card application modeled in IBM WebSphere modeller**

WebSphere has a built-in two sorts of analytical reports: static and dynamic. Reports provide a way to view, share, and print information derived from the models. WebSphere provides a variety of predefined report templates that can be used to generate reports based on models. Also it is possible to create report templates or customize the predefined. The static analysis reports are used to visualise the result of analysing process models and the static elements of the project, for example, activity cost and duration report, resource cost and roles report, role availability report etc. Dynamic reports are used to carry out dynamic analysis based on one or more simulation results. Basically it is for extracting and aggregating information form simulation results and performing comparative analysis on the results of simulations. There are more than twenty types of dynamic analysis. The following illustrate some of the ways the dynamic analysis can be used to show aggregate results of process activities, results related to specific process instances created during simulation, process results based on all process instances in a simulated process, and comparative analysis of the process results of two different simulations [7]:

- Activity cost analysis — displays the average costs for the activity instances for each activity used in a process, including the average revenue, execution cost, idle cost, allocated resource cost, total cost, and profit.
- Process instance resource allocation analysis — displays the task instances involved in a specific process instance, including the resources allocated per task instance, the allocation duration, shortage, and cost.
- Process cost analysis — displays a list of process cases (alternative processing paths for the simulated process) and the average revenue, average execution cost, average idle cost, average allocated resources cost, and average profit for the process instances that match each case. This analysis also displays a weighted average of the analyzed costs across the various process cases. The relative weighting for each process case depends on its probability of occurrence.
- Processes cost comparison analysis - displays the process costs for two processes, including the revenue, execution cost, idle cost, allocated resource cost, total cost, and profit for each process. This analysis also shows the difference between the corresponding values of the two processes.

## 1.3. Other process simulation tools

For comparing WebShpere to other tools, the ITP Commerce Process Modeller for Microsoft Visio was tried out. It is more lightweight than WebSphere and some other tools in the market. It claims to support 100% of the BPMN standard and it is possible to export models in BPEL (Business Process Execution Language) format, with some restrictions. It allows creating different resource scenarios and it supports human and non-human resources and has a grouping capability. The grouping is just grouping of the resources and the tool does not support roles as WebSphere (i.e. virtual resources). For the resources it is possible to assign cost per hour and per usage, also shifts are supported, but all calculations are made per hours (i.e. the time unit in calculations is one hour). The tasks have possibility to assign 3 pre-defined KPI (Cost, Wait and Active time) and resource (resource cost is calculated from the active time of the task). Also it is possible to assign user defined KPI parameters and use them on calculations or analysis of the simulations (the KPI parameters are monitored and logged for every activity instance). The log of the simulation is in format of Microsoft Excel and provides some pre-defined statistics and charts. Also raw data is provides and thus the possibility to define custom reports in MS Excel or some other tool. The input generation is quite similar to WebSphere: it is possible to define maximum number of processes, the arrival rate and quantity per time unit, and the duration of the simulation (start and end dates).

In [2] 6 more different tools (two of them from each different area: Protos and ARIS (process modelling); FLOWer and FileNet (process management); and Arena and CPN Tools (discrete event simulation)) are compared based on the capabilities of modelling, support of simulation and output analysis. More precisely the paper draws out following criterion:

- Modelling capabilities
    - ease of model building
    - formal semantics and verification of correctness
    - workflow patterns
    - resource and data perspective
    - level of detail, transparency and suitability for communication
- Simulation capabilities
    - performance dimensions (possibility to simulate several different time and/or cost aspects)
    - distributions
    - animation

    - scenarios (process stays the same, only different configurations)
- Output analyze capabilities
    - statistic
    - format
    - what-if analysis (comparison of results)
    - conclusion-making support

In conclusion, the paper states that both business process management tools fell short on their simulation capabilities; Flower did not support simulation at all (like most business process management tools) and Filenet did support simulation though without stochastic functions and statistical analysis. The process modelling tool Protos provides a simulation module based on the ExSpect simulation engine, but the interface between the two modules omits important details with respect to data and resources, thus making the outcome of a simulation unreliable. Flower, Filenet and Protos are considered to be unsuitable for solid business process simulation studies. The three remaining tools, ARIS, Arena and CPN Tools, all three qualify for business process simulation studies. These tools have different principles that determine the suitability of the tool for a particular simulation study. ARIS is based on the informal process modelling language of EPCs and has difficulty to model some common workflow patterns. However, its strong point is the suitability for communication with process owners, which frequently is an important condition in such simulation studies. Arena is a strong simulation tool that proved to be appropriate for BPS. The modelling with this tool is based on predefined building blocks, which can be adapted and extended if necessary. In this tool, it is important to have a profound knowledge about the building blocks that are available and about the exact mode of operation. Finally, CPN Tools is based on the formal modelling techniques of Petri Nets. This opens many possibilities for the formal verification of the simulation model. The price to be paid however is high. Like modelling in Arena, a profound knowledge is required on modelling Petri Nets, but CPN

Tools differs from Arena in that respect that the resulting models are hard to understand by general process owners who should be able to understand and validate the models [2].

Another paper ([10]) suggests using CPN Tools as simulation engine, especially the support of the state information. Majority of the researches on the business process and workflow field focus on verification of the models rather than simulations. Authors of [10] note that lots of workflow management systems are providing simulation component but none of these tools does have support loading historical and state data into the model. Especially using the state data will provide more valuable predictions in short-term simulations as it is with higher quality and easier to interpret and apply – if the simulation model start in an arbitrary initial state (without any cases in pipeline) and then simulation process must run for a long time to make it to steady-state.

Also models are created manually and this may cause design mistakes. To create a model that is constructed accurately as possible and based on observed data it is needed to merge following data into the simulation model [10]:

1) Design information. The workflow system has been configured based on an explicit process model describing control and data flows. Moreover, the workflow system uses organizational data, e.g., information about users, roles, groups, etc.
2) Historic information. The workflow system records all events that take place in 'event logs' from which the complete history of the process can be reconstructed. By analyzing historic data, probability distributions for workflow events and their timing can be extracted.
3) State information. At any point in time, the workflow process is in a particular state. The current state of each process instance is known and can be used to initialize the simulation model. Note that this current state information includes

the control flow state (i.e., ‘tokens’ in the process model), case data, and resource data (e.g., resource availability).

Tools and techniques to support this approach and prove the usefulness of it was presented. Especially the extraction of historic data to support correct models and simulating “as-is” scenarios was demonstrated. The historical data was in the format of MXML (Mining XML, [11]) and the ProM framework was used as mining tool to extract and analyze simulation relevant information from the historical data and the YAWL models with simulation relevant data were automatically generated. The YAWL modelling technique was used as it is possible to convert YAWL to CPN model [12] and use CPN Tools as simulation engine. Also CPN has a possibility to load state data and start simulation “in the middle of process”. The generated simulation models uses MXML format for simulation log and it provides the ability to use same tools (i.e. ProM framework) for analyzing the simulation log and the real data in unified manner, i.e., it adds the possibility of tracking both the history and the future of particular cases and makes possible to analyzing and observing differences between simulated and real-world processes.

# Chapter 2

# Coloured Petri nets

## 2.1. Petri nets

A Petri net (also known as a place/transition net or P/T net) is one of several mathematical modelling languages for the description of discrete event systems. A Petri net is a directed bipartite graph, in which the nodes represent transitions (i.e. discrete events that may occur), places (i.e. conditions), and directed arcs (that describe which places are pre- and/or post-conditions for which transitions) [13]. Petri nets were invented in early 1960s by Carl Adam Petri.

A classical Petri net consists of places and transitions. A place is represented as a circle and a transition as a square. In a Petri net places and transitions are connected with each other by means of directed arcs. There are two kinds of arcs: arcs which run from a place to transition and arcs which run from a transition to a place. Arcs between two places or two transitions are not allowed [14].

Places may contain any non-negative number of tokens. Such a token is represented by a black dot. A distribution of tokens over the places of a net is called a marking. The state of a classical Petri net is determined by the number of tokens present in each place [14].

For illustrating Petri nets the previously known credit card application is modelled in Petri nets (see Figure 3).

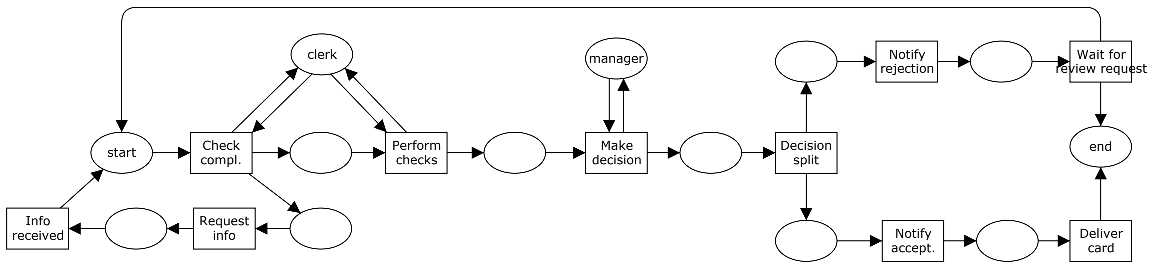


**Figure 3: Credit card application modelled in Petri nets**

The network structure of a Petri net is fixed. The distribution of tokens over the places, however, can be changed by transitions. This is called the firing of transition [14].

The firing of transition is bound to rules. A transition is only allowed to fire when there is a token in each of its input places. A place p is an input place of a transition *t* if there is an arc from *p* to *t*, likewise, a place *p* is an output place of a transition *t* if there is an arc from *t* to *p*. A transition which fires consumes one token from each of its input places and produces one token for each of its output places [14]. A firing is atomic, i.e., a single non-interruptible step [13]. If you see the example in the Figure 3 then only one transition is enabled and can be fired - the "*Check comp.*" transition as "*start*" and "*clerk*" places contains tokens.

The places in a Petri net are passive. They only depict the current state of a Petri net. The transitions of a Petri net, on the other hand, are active because they can change the state when they fire [14].

Execution of Petri nets is nondeterministic: when multiple transitions are enabled at the same time, any one of them may fire. If a transition is enabled, it may fire, but it doesn't have to.

Since firing is nondeterministic, and multiple tokens may be present anywhere in the net (even in the same place), Petri nets are well suited for modelling the concurrent behaviour of distributed systems [13].

## 2.2. Coloured Petri Nets

Coloured Petri Nets are an enhancement of Petri nets. The main extensions are: colour (data associated to tokens), time and expressions attached to arcs and transitions.

### 3.2.1. Colour

When modelling a system by means of a classical Petri net, elements of this system should represented as tokens, places, transitions and connections. Tokens can be used for modelling physical objects, information objects, collections of objects, states and conditions. A token can be used for modelling a car, for example. Often it is the case that, in a process model, it is desired to distinguish between cars of different brands (e.g. Saab, Volvo). Sometimes is more precise data required like model the registration number, the year of construction, the colour and the owner of the car. In classical Petri net, however, it is not possible to describe the attributes of the token. It is therefore natural to extend classical Petri nets in such a way that every token carries some data. A token which represents a certain car then has a value, from which we can derive for example the brand and the colour etc [14].

In a Coloured Petri net every place has a type and every token has a value and the value of the token is also called its colour. The value of a token can be used to keep up-to-date with information about the object represented by the token [14].

After adding colour dimension to Petri net, the markings are distinguishable and every place has a type (Figure 4). In Figure 4 has 3 places with tokens. The places "*clerk*" and "*manager*" has a type *resource* and the place "*start*" has a type *application*. Both types are so-called *record type*: `record Name:string * Type:string` and `record Name:string * ReceiveDate:string`. For example in the place "*clerk*", the value of a token thus is a record with two attribute values and all attribute values are strings. One token represents an employee John and another Joe, both are working as clerks.

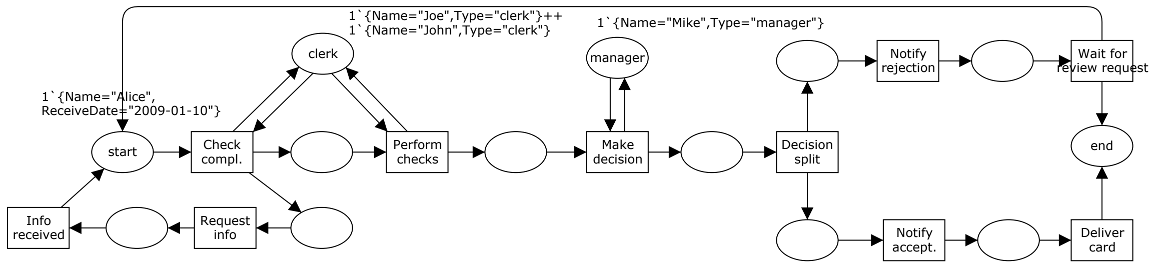


**Figure 4: The example of credit card application modelled in Petri net with colour**

### 3.2.2. Time

The classical Petri net cannot be used for the explicit modelling of time. It is easy to express that a certain action should take place before another one takes place, but it is impossible to indicate when actions take place and how long they take.

Because classical Petri nets cannot describe the temporal behaviour of a system, it is not possible to say anything about the performance of the modelled system. The performance of the system is often expressed in terms of processing time, waiting time and number of objects processed per time unit.

Coloured Petri nets incorporate two concepts that allow one to capture the temporal behaviour of a system: *Timestamps* and *delay expressions*. Timestamps attached to tokens and delay expressions capturing the availability of produced tokens, allow one to model when something happens and how long it takes [14].

With the extension of time in CPN, each token has a timestamp. The timestamp indicates from which moment the token is available for consumption (i.e. firing of the transition) and a token is available for consumption when its timestamp is smaller or equal to the present time of the model. In CPN the firing of transition consumes one token from every input place and produces new token for every output place. The timestamp of a produced token is equal to the firing time increased by a possible delay. The delay is determined by the transition that fires. The firing is timeless e.g. it does not take time and it is an indivisible action [14].

To illustrate time-stamps and delay expressions, the previously showed example of credit card application is extended (see Figure 5). The timestamp is attached only to application of credit card tokens (i.e. case). In the place *start* the timestamp of the token is 0 and it is denoted by @0. After the transition *Check compl.* consumes the token and produces a new token for place *compl.* which is assigned a new timestamp @1800 as the delay is 1800 seconds (i.e. 30 minutes; for simplicity and lack of other time units in CPN the second is used as time unit).

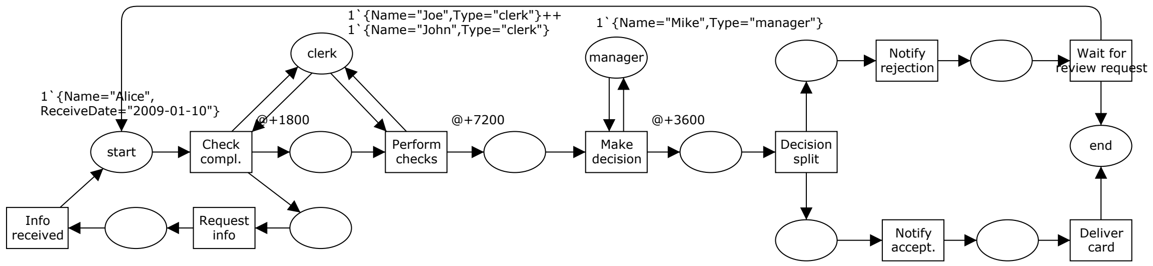


**Figure 5: Credit card application with time and delay**

### 3.2.3. Expressions

CPN allows one to attach data types to each place in a net, in order to constrain the types of values that tokens in that place may take. It also allows one to attach expressions to the arcs of the net, in order to manipulate the values attached to tokens produced or consumed by a transition. These two tasks are performed using a functional programming language, namely CPN-ML. CPN-ML is based on the Standard ML (SML) language. CPN inherits from SML the basic types, type constructors, basic functions, operators, and expressions form the ML language.

The basic types of CPN are: *int, real, string, bool* and *unit*.

- Type *int* is used to represent integers
- Type *real* represents reals (but CPN-ML does not allow one to use real as the type of color)
- Type *string* is used to represent strings. Strings are written between double quotes.
- Type *bool* (Boolean) has two values: *true* or *false*. This is used for logical expressions and variables.
- Type *unit* has only one value represented as an open bracket immediately followed by a closing bracket *( )*. This type can be used to define "black" token i.e., tokens that do not carry any information.

The operators are very similar to the operators of any other programming language (e.g. arithmetic operations, Boolean operations, string concatenation, etc,).

In CPN-ML, places are typed, i.e., all tokens on a place have a value of some common type. In CPN-terms this means that all tokens in a given place should belong to the same colour set. This implies that each place has a colour set (i.e., type). Colour sets are defined in using basic types, the subsets of basic types, creating new types by the explicit enumeration, or constructing new types using the type constructor *product*, *record*, and *list*.

Firing transitions not only changes the distribution of tokens over the places but also changes the colours (i.e. values) of the tokens flowing through the net. This requires a specification of the transitions involved. To specify the input/output behaviour of transitions, *arc inscriptions* (also called *arc expressions*) are used. The allocating a concrete value to each of the variables in input and output arcs is called binding. A binding is enabled if there are tokens matching the values of the arc inscriptions (i.e. the colour set of the arc expression must match the colour set of the place attached to the arc). If a binding is enabled, it can occur, i.e., the transition fires while consuming and producing the corresponding tokens. For example if the type of the place is INT then the type of variable in the arc inscription must be the same, otherwise it will not bind.

Transitions can have a *guard*. A guard is a Boolean expression (i.e., an expression which evaluates to either true or false) and it may have variables in exactly the same way that arc inscriptions have.

The purpose of a guard is to block a transition when it should not fire for some reason, i.e., the guard defines an additional constraint that must be fulfilled before the transition is enabled. More precisely: a binding element is only enabled if its corresponding guard evaluates to true. Another term for guard is precondition. Also functions can be used for more complex arc inscriptions or the guards of the transitions.

In CPN each transition may have an attached ML code segment. Code segments are executed when their parent transition occurs. Code segments may use CPN variables and may bind CPN variables located on output arcs that are not bound elsewhere. An input pattern is a tuple of CPN variables, preceded by the keyword input. The input pattern lists the CPN variables that can be used in the code action. The code action can use the values of these CPN variables but it cannot change them. An output pattern is a tuple of CPN variables, preceded by the keyword output. The output pattern lists the CPN variables to be changed as a result of the execution of the code action. The input and output is an optional part of code segment and the action part is mandatory. A code action is an ML expression, preceded by the keyword action. The code action cannot contain any declaration of colour sets, CPN variables, or reference variables. It can apply user-declared and pre-declared constants, operations, and functions and in addition it is possible to define new functions and constants can for local use. The code action is executed as a local declaration in an environment containing the CPN variables specified in the input pattern. This guarantees that the code action cannot directly change any CPN variables but only local copies of them. When the code action has been executed, its result is applied to bind the CPN variables in the output pattern. The code action, when evaluated in an environment containing the input pattern variables must yield a result of the same type as the output pattern. If no output pattern is given, its type is assumed to be unit [15].

After introducing the CPN the example of credit card application can be remodelled in CPN (see Figure 6). In this CPN, all resources are put into one place, namely *resource*. The resource place has type *RES*. It defines a record type with two attributes *Name* (string) and *Type* (string). The type is used to distinguish between clerks and managers. An alternative would be to have two places (*clerk* and *manager*). All other places have type CCA (a record type with two attributes: *Name* (string) and *ReceiveDate* (string)), it indicates the credit card application. Also the transitions *Check compl.*, *Perform checks* and *Make decision* have a guards (to ensure that only correct type of resource will be used for the action) and a delays (to indicate the time the process takes). Also very simple arc inscriptions are used: a variable *r* is used to bind the resource and variable *c* for credit card application.

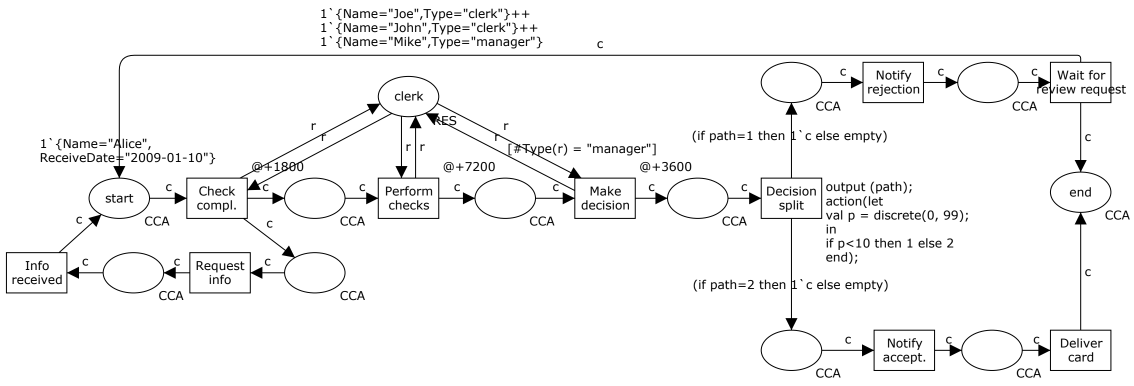


**Figure 6: Credit card application modelled in CPN**

## 2.3. CPN tools

CPN Tools is a widespread tool for editing, simulating and analyzing CP-nets. The GUI of CPN Tools was designed in cooperation with leading HCI experts and is based on advanced interaction techniques. Feedback facilities provide contextual error messages and indicate dependency relationships between model elements. The tool features incremental syntax checking and code generation which take place while a model is being constructed. This means that it is possible to simulate and analyze the parts of a model that are syntactically correct, while parts that are incomplete or have errors are ignored. It is also possible to modify a model during a simulation, and then continue the simulation after the modified parts of the model have been rechecked. 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. By means of a simple query language it is possible to specify and check system specific properties [16].

CPN Tools is using XML files for holding the models. The XML format of CPN Tools files is described using DTD (Document Type Defination). The DTD of CPN Tools can be found [17]. The format of XML file of CPN Tools is quite simple, for example in Figure 7 is described the definition of the place.

```
<place id="...">
  <posattr .../> <!-- position -->
  <fillattr .../> <!-- fill color -->
  <lineattr .../> <!-- line width and color -->
  <textattr .../> <!-- font and text color -->
  <text>...</text> <!-- name -->
  <ellipse .../> <!-- sets width and height -->
  <token .../> <!-- position of round no of tokens -->
  <type> <!-- color set -->
    <posattr .../> <!-- position -->
    <fillattr .../> <!-- fill color -->
    <lineattr .../> <!-- line width and color -->
    <textattr .../> <!-- font and text color -->
    <text>...</text>
  </type>
  <initmark> <!-- initial marking -->
    <!-- like type -->
  </initmark>
  <fusioninfo id="..." name="..."> <!-- omit if not fusion  place -->
    <posattr .../> <!-- position -->
    <fillattr .../> <!-- fill color -->
    <lineattr .../> <!-- line width and color -->
    <textattr .../> <!-- font and text color -->
  </fusioninfo>
</place>
```

**Figure 7: The example of place description in the format of CPN Tools**

# Chapter 3

# Process Simulation using CPNs

This chapter introduces a mapping from Business Process Simulation Models into CPNs. In order to define this mapping a formal meta-model for business process simulation models is introduced. This meta-model establishes what data entities compose a process simulation model and the relations between these entities. This meta-model is layered on top of the BPMN notation, meaning that it adds entities and associations on top of those supported in BPMN such as Task, Gateway, etc. Having defined such meta-model, an element-by-element mapping is then introduced that takes in a business process simulation model and produces a CPN model. The meta-model and the mapping are explained by means of the running example.

## 3.1. Meta-model

In this section a meta-model is outlined that captures the data required simulating a business process. The structure of the model is based on reviewing of the simulation tools of business processes, especially IBM WebSphere Modeler and ITP Commerce Process Modeller for Visio. The meta-model extends the meta-model of BPMN [6], essentially by attaching simulation parameters to tasks, events and gateways in BPMN models. This model and its description do not describe the elements and their attributes of the standard BPMN model.

The model is quite general and holds only the basic simulation objects. The data models used by the simulation modules of business process modelling suites are much more complicated and complex. To compare them, the model is rather incomplete and uses only the basic objects required to set up simulations.

In order to simulate processes in business process tools supporting simulations, the following steps must be carried out:

1) Model resources, timetables etc
2) Model the process
3) Attach simulation data to the tasks, probabilities to gateways
4) Create and set up simulation profile
5) Simulate the process and analyze the results

The meta-model is structured into three parts:

1) Classes to hold basic types, such as resources, timetables, roles etc.
2) Classes to attach metadata to tasks and gateways
3) Classes to hold simulation profiles

### 3.1.1. Modelling Resources

Figure 8 shows the fragment of the business process simulation meta-model that relates to resources. Resource related objects include timetables, roles, cost of resources, etc. Resource itself is divided into two objects: individual and bulk (i.e. non-human) resource. Individual resources represent the people or computers while bulk resources represent equipment or material used to perform a project or a task. Individual resources are resources where a specific instance is required, whereas bulk resources are resources where any instance of a type of resource from a pool can be used. Individual resources include people and computers, and bulk resources include power and water. Also bulk resource may be consumable and in this case, they may be exhausted during the simulation (i.e. they have quantity and each time they are used, this quantity is decremented). For example fuel or printer paper can be consumable bulk resources.

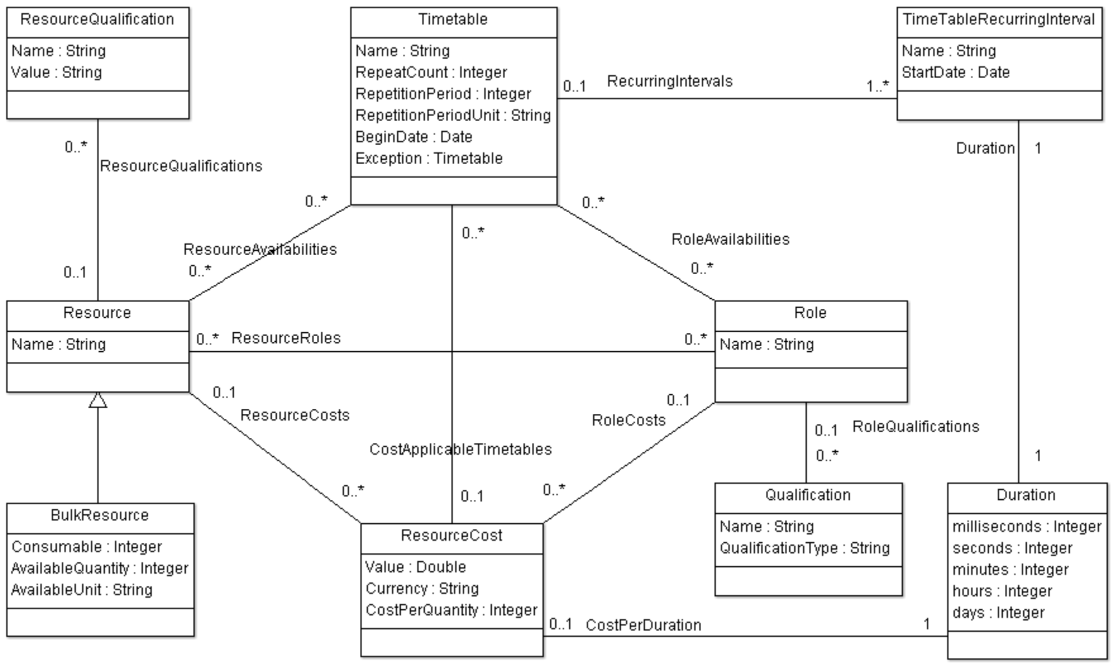


**Figure 8: Resource model**

The resources might not always be available. For example, a clerk might only be available from 8am to 5pm on weekdays excluding holidays. This information needs to be available in the simulation model since it affects the waiting time of a task. Timetables are defining the time when resource or some other object is available to use or is applicable. The definition of timetable requires the start date of the timetable and the repetition properties: how many times it will apply to and the cycle of the repetition (for what time period it will be available again). Timetables may have more than one recurring interval. The recurring interval defines the duration of the time when the timetable is applicable and the start time of the recurring interval. Also timetables may have exceptions, i.e., the time periods when this timetable is not applicable (also with the type of timetable). For example to model the availability of the clerk (from Monday to Friday from 8am to 4pm) a day shift timetable can be created. It has one recurring interval starting at 8am and with the duration of 8 hours. The length of repetition period is one day and it has two exceptions (Saturday and Sunday which are timetables and has a repeating period on every 7 days with the length of the duration of 1 day).

The purposes of simulating as-is or what-if scenarios are usually for observing the cost of the tasks through resources – how much cost to perform this task. Resource may have multiple cost at the same time, thus extra class is designed for that. Possible combinations of costs are:

- one time cost (Resource, Role, Bulk resource)
- per time unit cost (Resource, Role)
- per quantity (Bulk resource)
- per quantity for every time unit (Bulk resource)

Also the cost may depend on the time when the resource is used, thus it is needed to have a timetable associated it. The resource can be available according to one timetable but the costs are calculated using a different timetable. For example worker can be available from 8am to 8pm but 8am to 4pm the cost is calculated using one timetable and over hours by another.

Resources may have roles. A role is a group of resources with similar abilities and/or skills. For example the model of credit card application has two roles, *Manager* and *Clerk* and in the given simulation there will be many managers and clerks. Roles can be used to factor out data from individual resources and thus to prevent repeating the same data entry several times. For example, it is possible to assign cost and availability to role and then it can be used as virtual resource and the requirement of creating an individual resource is not needed. In case of the resource have roles and the cost or availability is also modelled to the roles, the resource properties are taking the priority, i.e., the cost and availability of the role are used only if no other cost or availability is associated with the resource.  Also a role can have qualifications which are used to model requirements. For example the “location” can be assigned as qualification to a role. Then it is possible to add the value of the “location” qualification to the individual resources in that role and afterwards it can be checked that only with specific values of “location” can perform the task.

### 3.1.2. Extensions to Tasks, Gateways and Events

Extending the BPMN basically means to extend *Task* class to attach extra data for simulation, such as resources performing the action, the costs of the task, the cost of waiting for the resource, the processing time of the task and the maximum time to wait for resource.

The proposed meta-model has three types of resources: roles, individual and bulk resources. They all have a common attribute – the time it takes to perform the tasks. Roles have extra attributes quantity (how many individual resources are needed) and the qualifications (the specific qualifications which are required for the individual resource in that role). Also bulk resources have a quantity attribute for specifying the amount of resources available at the beginning of the simulation (i.e., process). An example of a consumable resource is fuel such as gasoline.

Sometimes the tasks have no resource requirement. In this case the execution time and cost of performing a task cannot be calculated from the required resources. Still, these tasks are likely to have a time and cost. To cope with this situation, the simulation meta-model allows one to attach processing time and the cost directly to a task. In this respect, a task may have 3 types of costs: start up cost, processing cost and revenue (revenue is in fact the opposite of cost but it has the same data structure as cost). These costs do not depend on the time required for processing but the cost of waiting after resource depends on the time the task is waiting for resource. Note that it is possible to specify costs and processing time either as a deterministic value or as a probability distribution with certain parameters (e.g. normal with a given mean and standard deviation, exponential with a given rate, etc).

A workflow model is composed of a number of tasks which are connected in the form of a directed graph. A task corresponds to a single unit of work and each invocation of a task that executes is termed a work item. In general a work item is directed to a resource for execution and there are a variety of ways to do it [18]. So resources are required to invocations of tasks for specific cases.

The proposed meta-model specifies only which resource can do the job (i.e. authorization perspective). It does not support resource allocation model, i.e., it does not have structures to define how the resources are chosen for doing the task. However there are many possibilities to do it. For example resources can be chosen randomly, have a stacked/piled and picked as first in - first out, etc. In [18], the authors defined 7 categories of *resource patterns*. For example a detour pattern where pre-existing work allocations are interrupted either by the workflow system or at the instigation of the associated resource (e.g. delegation where a resource allocates a work item previously allocated to it to another resource or escalation where the workflow system attempts to progress a work item that has stalled by offering or allocating it to another resource, etc.) or auto-start patterns where the execution of work items is triggered by specific events in the lifecycle of the work item or the related process definition (e.g. piled execution, chained execution, etc.)

As the current simulation meta-model has no support for defining any of resource allocation patterns then one of the further possible directions of this work may be specifying and adding support for resource allocation models.

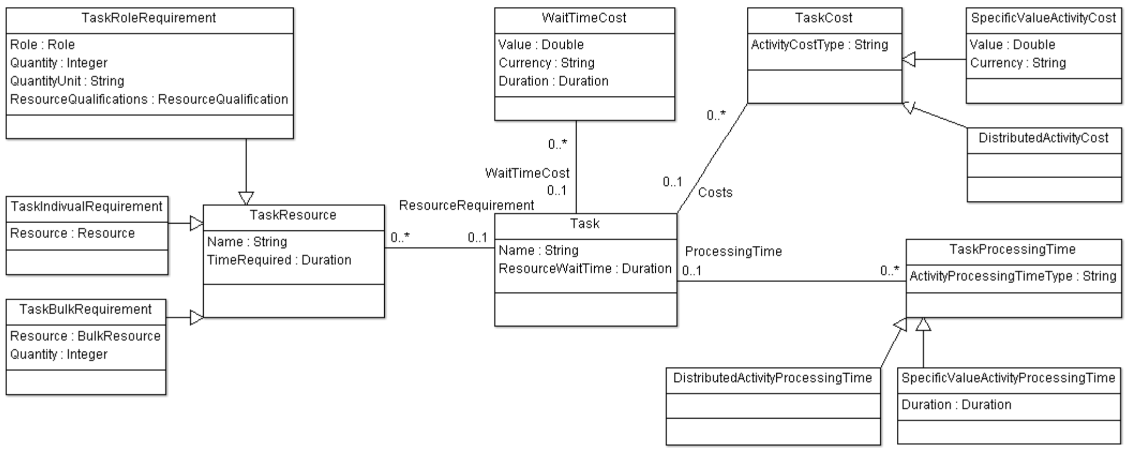


**Figure 9: BPMN extension model**

Gateways in BPMN models are used to converge or diverge the sequence flow of the process and to be more precise, gateway is a collection of *Gates*, i.e., paths. To add the support of the probabilities (i.e., the probabilities of selecting output paths), the BPMN gateway model must be extended with extra property (see Figure 10). Only two attributes are common to gateways in BPMN model: *GatewayType* and *Gates* and the *Gates* are representing paths, so the probability of selecting corresponding *Gate* is the property of the *Gate*. The type of the probability is *double* and the sum of the probabilities in one gateway must add up to 1 (i.e., 100 per cent).

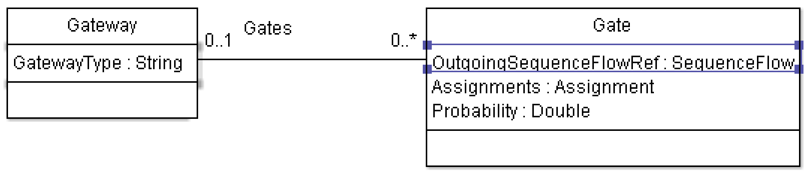


**Figure 10: Extending BPMN Gateway and Gate**

BPMN supports three event types: start, intermediate and stop events. As previously stated, events are triggers to a signal that something is happened. Events does not have resources attached to it and it does not do work. Events are designed to show how one process interacts with another (internal or external), so events can send or receive signals (i.e., it can be catching or throwing event). Also intermediate events can be bound to tasks. It means that if event is triggered the task is immediately aborted and the alternate or exception flow is chosen instead of main flow. From simulation context no additional properties is required – events are triggered when trigger is fired.

BPMN specification defines more than 10 different event triggers but if generalize then only two basic types of event triggers are really used: timer and message/signal. A timer start event indicates that the process starts at a particular date and time or on a periodic date/time cycle or used as an intermediate event then it means a delay, either wait for a specified duration or wait until a specified date/time. In Figure 11 the model of *Timer* trigger is presented.

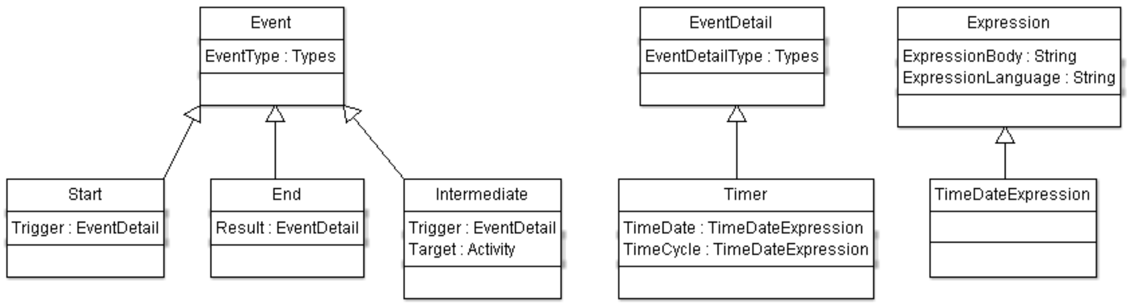


**Figure 11: Fragment of BPMN Event model**

Previously is noted that message events can receive or send signals. For that sending and receiving places must be connected. From the simulation perspective message event does not have any additional properties – message sending event is triggered when flow passes the event and receiving event is always waiting for event and cannot be occurred before received message trigger it. If more complex situation is required then *event-base gateway* should be used instead of plain events. The event-base gateway was not specially handled in the section of extending gateways because from the simulation view they do not require additional information as the flow path is chosen by the first event of the gateway is triggered. Another solution for handling event-based gateway is to manage it as normal gateway, i.e., adding probabilities to every outgoing path.

### 3.1.3. Simulation Profile

Setting up and running a simulation requires an extra data i.e. simulation profile. It contains start and end date of the simulation, the resource pool used for simulation and generation of cases i.e. input for simulation. The definition of a resource pool is quite simple: it consists of the available resources and their quantities. Only roles can have a quantity i.e. the amount of "virtual" individual resources created for the simulation.

Creating the input cases for the simulation requires to define total number of cases, the cost of creating a case, the number of cases per bundle (i.e. how many input cases are created at one time) and the time interval between new case arrives (i.e. input trigger). The input can be a message from some other part of the simulation or *Timer start event*. Current model does not extend BPMN Events and gives freedom to implementing the input trigger for simulation. The most frequents input triggers are timer with certain time interval or the time cycle depends on probability distribution with certain parameters (e.g. normal with a given mean and standard deviation, exponential with a given rate, etc). Also the timer can be represented as timetable. Creating new input cases after specific time requires only a start date and the time interval, for example it can create new input for every 30 minutes. Using the probability distribution trigger allows generate input a little bit stochastic way, for example it is possible to define new input using a normal distribution and generate new input for every 30 minutes +/- 5 minutes i.e. new case can arrive between every 25 to 35 minutes. The timetable trigger is using pre-defined timetable for generating input. For example timetable can define that new cases will arrive every day at 9am. All these timers can be presented as expressions in *Timer TimeCycle* field, for example normal distribution can be represented as "*mean=30 minutes; std=5 minutes*",

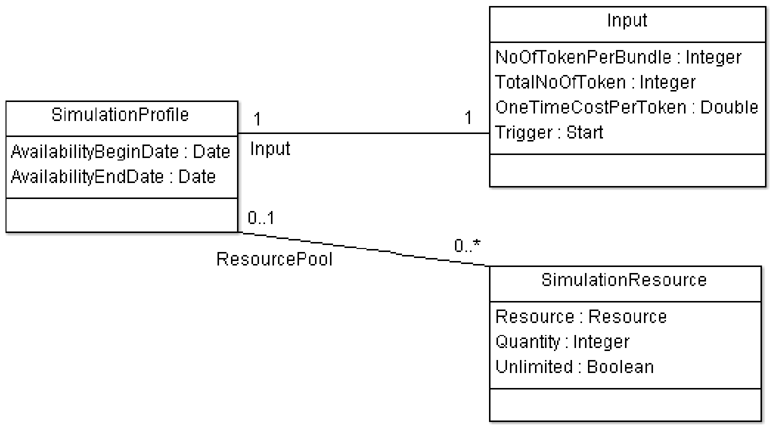


**Figure 12: Simulation profile model**

## 3.2. Mapping Simulation Model into CPN

This section explains how to transform an instance of the previously introduced extended BPMN meta-model into a CPN model, so that the resulting CPN model can be simulated using CPN Tools. To illustrate this transformation, the previously introduced credit card application is taken as an example and this section will define the building blocks for it.

### 3.2.1. Overview of the Mapping

As a starting point, Figure 13 describes a transformation of standard BPMN elements to plain Petri nets. This mapping was introduced in [6] as a part of the proposal of a BPMN formal semantics aimed at statically checking the semantic correctness of BPMN models. The mapping introduced in this section will extend the mapping of task, events and gateways given in [6] and will introduce a mechanism to produce simulation logs for further analysis.

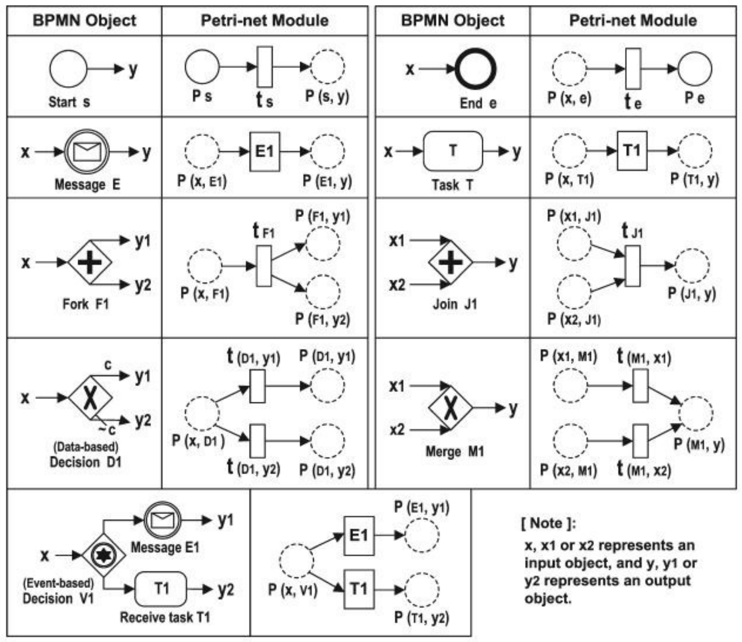


**Figure 13: Mapping task, events, and gateways from BPMN onto Petri-net modules [6].**

The proposed mapping is divided into 6 parts, which are summarized below and explained in details in the next six sub-sections. The first sub-section part deals with recording and reporting of the simulation in CPN. Section 3.2.2 gives an overview of different logging options in CPN tools and discussed their limitations and optional usage on simulation data collection. Among these options, the most flexible one appears to be the use of the ProM CPN Library [19] to generate simulation output in the standardised data mining XML format (MXML). Choosing this standardised format for representing the simulation output will broaden the amount of tools suitable for analysing the results of the simulation.

The second part of the mapping deals with the modelling of resources and resource pools in CPNs. In order to model resources in CPN, three types of structures must be defined (cf. Section 3.2.3):

- resource object, i.e., the properties/attributes of the resource, for example name of the resource, roles, etc.;
- resource requirement for performing a task, i.e., resource allocation and authorisation pattern;
- resource pool i.e. available resources.

In addition to these aspects, the modelling of resources also requires us to capture the cost of using resources, and the timetables at which resources are available.

The third part of the mapping (cf. Section 3.2.4) deals with mapping task and related data. One question that needs to be addressed n this part is that of which resource allocation policy is used. As the meta-model does not contain resource allocation information then it gives freedom to use any resource allocation policy. Random resource allocation pattern is used as an example. So all tasks which have a resource requirement are attached to a resource pool and only role and individual filtering of the resources is introduced.

Sometimes tasks do not have a resource requirement but instead they have a cost and execution time. These structures are rather trivial to capture in CPN.

Also, when mapping tasks, one needs to consider that each execution of a task/event needs to generate an entry in the simulation log. Accordingly, Section 3.2.4 also describes how to attach logging functionality to every task and which data should be saved.

The fourth part of the mapping deals with how to map different types of gateways (cf. Section 3.2.5).

Finally, the fifth part of the mapping deals with how input cases are generated (cf. Section 3.2.6). Here, it is important to distinguish between different types of triggers. Also setting up simulation requires additional data (also called a simulation profile) and part of the mapping characterises the CPN structures needed to capture simulation profiles.

In order to tie up all the parts of the mapping, Section 3.2.7 will summarise the steps required to model a business process simulation in CPN.

Appendix A shows the example of credit card application mapped to a 2-level hierarchical CPN model: it uses one top-level and one sub-level page. The sub-level page is for generating cases i.e. input cases to the simulation and the top-level is the process model itself. In this approach, the process model is flatten down i.e. all sub-processes are described in the main model and model has only one level. This is only one possibility to model the process and suitable for simple models. For example, another approach is to use sub-pages for every task or node of the BPMN model. Our approach is motivated by the fact that we do not intend that the CPN models will be read by users. Instead, the CPN will only serve to execute the simulation on the back-end and the users will only see the initial BPMN model and the results of the simulation. Therefore, we do not need to generate “modular” CPNs.

### 3.2.2. Simulation Logging

Analysing business processing using simulations requires one to collect data produced by the simulation. There are many possibilities to collect simulation data in CPN Tools. One of the options is to use built-in monitors. A monitor is a mechanism in CPN Tools that is used to observe, inspect, control, or modify a simulation of a CPN model. There are multiple types of the monitors: standard monitors (very easy to define and do not require users to write any code), parameterized monitors (more flexible than standard monitors and require some programming) and user-defined monitors (flexible but requires lots of programming). One model can contain more than one monitor and monitors can inspect both the markings of places and the occurring binding elements during a simulation. Also they can take appropriate actions based on the observations, for example they can stop the simulation when a particular place will come empty or some other condition is met. Generating reports from the data collected by monitors requires additional programming, for example GNUPlot scripts can be used to generate the reports from the results or transform the logs generated by CPN Tools to some other format and computing additional statistic with specialized software.

However monitors give a very small number of collectable data variables (i.e. fixed number of reports). Another option instead of using monitors is to extend CPN model and collect required information by means of this extension. One possibility is to extend model with the ProM CPN Library [19] to generate log files in MXML format [11]. The MXML (Mining XML) format is created for standardize the inputs of different mining tools and it is more extensible than output of monitors of the CPN Tools and more complex analyse can be done later. Using MXML format adds possibility to import simulation logs into the ProM framework which offers a wide range of tools related to process mining and process analysis. For example, ProM provides large set of mining algorithms, performance analysis with sophisticated reports for finding bottlenecks and tools for model correctness verification.

The ProM CPN extension is fairly simple to use. There are two steps necessary to create MXML logs using CPN Tools [19]:

1) Modify a CP-net to invoke the set of ML functions that will create logs for every case executed by the CP-net. This step involves modifying the declarations of the CP-net and the input/output/action transition inscriptions.
2) Use the CPN Tools plug-in, in the ProM import framework, to group the logs for the individual cases into a single MXML log.

For using this extension, it is needed to declare two constants FILE and FILE_EXTENSION. The constant FILE sets the location and the name prefix of the MXML files that the CP-net will create for every case it executes. The constant FILE EXTENSION sets the extension that these created files have. Also one needs to include the extension itself (it is suggested to use full path to the extension) and the extension will create a separate log file for every case in the format of FILE+CaseId+FILE_EXTENSION. The following code snippet shows how this set-up step is done:

```
val FILE = "simlog";
val FILE_EXTENSION = ".cpnxml";
use
"C:\\CPNToolsConverter\\MXMLlogs\\loggingFunctionsMultipleFiles.sml";
```

In the generation of input cases the result file is created by using the function *createCaseFile(int caseId)*. The log file is created for every case and this function must be executed only once per case and before the function *addATE* is invoked for this same case. Note that this function receives an integer (the case identifier) as input. The code segments of transitions must be modified to invoke the logging functionality, i.e., the function *addATE(caseId, transitionName, eventType, timestamp, originator, data)* to log the execution of a transition into the log of the case. The input parameters of the function *addATE* are case id as integer, the name of the task as string, event type as string (event type can be one from the list: schedule, assign, withdraw, reassign, start, suspend, resume, pi_abort, ate_abort, complete, autoskip, manualskip, unknown), timestamp as string (in the format of "YYYY-MM-DDThh:mm:ss"), the originator (i.e. resources) as string and extra data as list of strings in the format of [attributeName1, attributeValue1,..., attributeNameX, attributeValueX].

The MXML logs produced during the simulation can be imported into ProM using *ProM import* plugin for CPN Tools. After the logs have been imported into ProM, one can analyze them using the functionality available in ProM. For example, ProM immediately generates a summary report with basic statistics such as average execution time, number of traces, etc. Also ProM framework has wide range of data mining and analysis plugins, such as the performance plugin, which calculates additional statistics, etc. ProM framework supports also conversions between different modelling languages, such as CPN, YAWL, EPC, etc. So using ProM framework for analysing the results of the simulation gives a wide range of possibilities.

### 3.2.3. Resources

The process simulation meta-model has three types of objects dealing with resources: 1) resource object defining the attributes of the resource; 2) resource requirement for doing a task: and 3) resource pool.

In the model, *Roles* and *Resources* are distinguishable and different things: *Role* is one attribute of the *Resource* and shows only the capabilities of the resource. For example the individual resource inherits the availabilities and costs from role definition. Also role definition sets the required qualifications for the individual resource. When modelling the resource requirements of the tasks, then the roles, individual and bulk resources are the same – it is possible to specify the amount of time the resource are needed and the required quantity (only for role and bulk) of resource. For example, the task may require 30 minutes of one individual resource in the role of “clerk”. On the other hand, the amount of resources available is defined in the simulation profile. For example simulation has a hundred clerks and 10 managers available. In the simulation profile point of view the roles are just abstract resources and for the simulation it is needed to generate specified amount of anonymous individual resources from the roles defined in resource pool of simulation. This feature enables one to examine the effects of adjusting the availability of qualified resources without actually creating individual resources that are qualified for the role. To map the concept of resource to CPN, we only need the resource definition and not the role definition, since roles are merely an attribute of the resource and sets of roles are converted into a set of strings. Also the *RoleCosts* and *RoleAvailabilities* are merged into the *ResourceCosts* and *ResourceAvailablities* of individual resource from the specified roles of individual resource.

Transforming the meta-model object holding the resource definition is essentially a one-to-one conversion. The CPN colour set *RES* defined for resource contains the name of the resource, its costs, the list of roles (in case of conversion of role, it is only the name of role) and the availabilities (the structure of costs and availabilities are represented below).

```
colset RES = record Name:STRING * Costs:RCOSTS * Roles:SLIST *
availability:TIMETABLES;
```

After representing the colour set *RES* it is possible to create the resources using it. For example, the following code snippet captures two resources named “John Doe” and role Manager.

```
val manager = {Name="Manager", Costs=[costs_manager],
Roles=["manager"], Availability=[tt_week]};
val john_doe = {Name="John Doe", Costs=[costs_john_doe],
Roles=["clerk"], Availability=[tt_week]};
```

The process simulation meta-model also contains elements representing resource cost and the availabilities i.e. timetables. The object *ResourceCost* is represented as a tuple with a *Value* (in cents i.e. the smallest unit of base currency), a CostPer*Duration* property is seconds and a CostPerQuantity in the case of consumable resources. The *Currency* attribute could be captured as an enumerated type, but we choose to leave this out of the model to keep some simplicity. Multi-currency simulations would require some additional data structures to keep track of costs in different currencies, or to convert all costs in different currencies into an equivalent cost in a single currency.

```
colset RCOST = record Value:INT * CostPerDuration:INT *
CostPerQuantity:INT * CostApplicableTT:TIMETABLES;
colset RCOSTS = list RCOST;
val costs_manager = [{Value=1000, CostPerDuration=3600,
CostPerQuantity=0, CostApplicableTT=[]}];
```

The timetable structure in CPN remains pretty much the same as it is in simulation meta-model. Only problem is with the exceptions. Exceptions are time periods when a timetable is not applicable. Exceptions are modelled as timetables themselves. But CPN does not allow recursive type definitions whereby there is a reference from a colour-set to itself. This means that to captue exceptions, we must have an extra colour set *TTEXCEPTION.* It differs from timetable only with the lack of *TTException* parameter, so timetable exceptions cannot have exceptions. To simplify the calculations, the *RepetitionPeriod* and *RepetitionUnit* properties are converted to seconds and represent it as *RepetitionDuration* parameter in the colour set.

```
colset TTEXCEPTION = record Name:STRING * RepeatCount:INT *
RepetitionDuration:INT * BeginDate:STRING *
RecurringIntervals:RECURRINGINTERVALS;
colset TTEXCEPTIONS = list TTEXCEPTION;
colset TIMETABLE = record Name:STRING * RepeatCount:INT *
RepetitionDuration:INT * BeginDate:STRING *
RecurringIntervals:RECURRINGINTERVALS * TTExceptions:TTEXCEPTIONS;
colset TIMETABLES = list TIMETABLE;
```

Timetable must have at least one recurring interval. The recurring intervals are simplified in CPN approach as *StartTime* is a time in time units (i.e. the time unit of the model is one second) from the beginning or recurring period of timetable date. For example,

```
val recint_dayShift = {Name="Day_shift", StartTime=((60*60)*9),
Duration=((60*60)*9)}
```

represents a recurring interval starting 9 hours after the beginning of timetable and elapsing 9 hours.

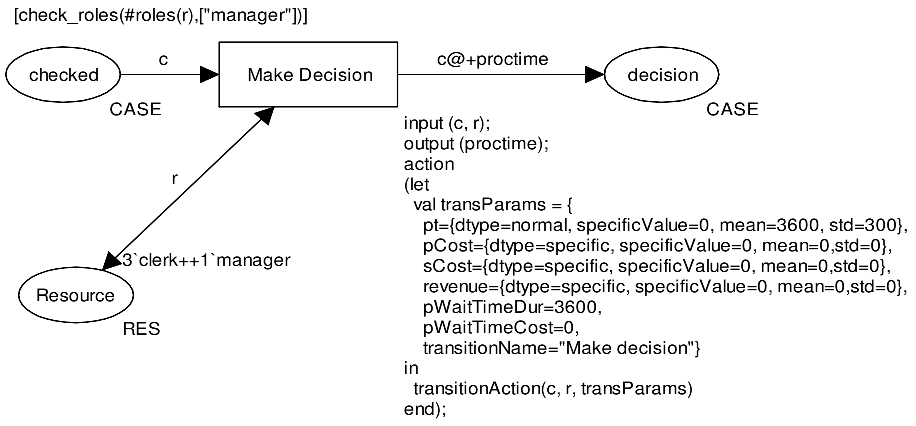
```
colset RECINT = record Name:STRING * StartTime:INT * Duration:INT;
colset RECURRINGINTERVALS = list RECINT;
```

The resource pool is simply one place where all available resources are hold as tokens. If the task requires a resource then it takes a resource and put it back after usage (in case of non-consumable resources). This simple resource management requires only generating specified amount of resource tokens from simulation profile and all required roles are converted to "*virtual*" individual resources.

### 3.2.4. Tasks

Tasks are the basic and most important building blocks of BPMN models. Simulating BPMN models need to extend tasks with extra data, such as cost and revenue, durations, resources etc. Figure 13 shows how to transform standard BPMN task to Petri-net module. As a result of extra data it is required to extend that mapping. The previous chapter explained that the transition in Colour Petri nets may have guards, delays and code segments. The current proposal is to take advantage of them to store extra data and perform calculations required to execute simulation.

Figure 14 provides an example of task with data required for simulation in CPN model. The task has two input and two output arcs – one is for the process flow and second to allocating and putting back the resource.



```
input (c, r);
output (proctime);
action
(let
  val transParams = {
    pt={dtype=normal, specificValue=0, mean=3600, std=300},
    pCost={dtype=specific, specificValue=0, mean=0,std=0},
    sCost={dtype=specific, specificValue=0, mean=0,std=0},
    revenue={dtype=specific, specificValue=0, mean=0,std=0},
    pWaitTimeDur=3600,
    pWaitTimeCost=0,
    transitionName="Make decision"}
in
  transitionAction(c, r, transParams)
end);
```

**Figure 14: Task modelled in CPN**

The code segment of the task has two input parameters in case of task is using resources and other way one input parameter (the case and the resource object). The code segment has one output (the processing time of the task) parameter. The processing time of the task is used as output parameter because then it can be calculated in the transition action (the duration of task may depend on multiple task parameters) and add it to the timestamp of the *CASE* object in the delay inscription of arc. The *CASE* object is quite simple, it holds the *ID* of the case (i.e. process) generated by the input of the simulation, the arrival timestamp (*ats*) and the timestamp updated after every task (*ts*) – it helps to calculate the waiting time after resources between tasks.

```
colset CASE = record ID:INT * ats:INT * ts:INT timed;
```

The action part of the code segment has a variable containing the data required for calculating results for simulation reports and to call the function *transitionAction* or *transitionActionWOR* (in case of task is not using the resources). The *transactionAction* takes 3 input parameters:

- object of current case (colour set *CASE*)
- resource (colour set *RES)*
- a parameter defining the data required for calculating simulation output (colour set *TRANSPARAMS)*

```
colset TRANSPARAMS  = record transitionName:STRING * pt: DISTRIBUTION *
pCost:DISTRIBUTION * sCost:DISTRIBUTION * revenue:DISTRIBUTION *
pWaitTimeDur:INT * pWaitTimeCost:INT * NoOfResources:INT;
```

The *TRANSPARAMS* colour set is constructed by following attributes:

1) transitionName – transition name in the simulation log
2) pt – processing time. It is time required to perform the action.
3) pCost – processing cost. The value of cost can be exact or calculated using some probability distribution.
4) sCost – start up cost. The value of cost can be exact or calculated using some probability distribution.
5) revenue – The value of revenue can be exact or calculated using some probability distribution. Revenue is positive cost.
6) pWaitTimeDur – the duration of time used for calculating the cost of waiting time after the resource.

7) pWaitTimeCost – the cost per unit of time (*pWaitTimeDur*) waiting after the resource
8) NoOfResources – number of resources used for calculations. This is meaningful only for consumable bulk resources. For human tasks it is always 1 and tasks which do not require resources, it is 0.

Some values, for example costs, processing times and durations etc., can be represented as exact values or by means of probability distributions (e.g. a normal distribution with a given mean and standard deviation, a negative exponential distribution with a given mean, etc.). In the working example exact values or normal distributions are used. For supporting that two colour sets, *DTYPE* and *DISRIBUTION* are constructed. The first one is an enumerated colour set and defines the supported distribution schemes (currently only two, exact value and standard deviation). The second defines the record holding required data, the *dtype* defines the distribution scheme, the *exactValue* is used if *dtype* is "*exact*" and *mean* and *std* is used otherwise. This unified record helps the calculations, i.e. it is possible to define the record and call function *calcDisValue* to calculate the standard deviation or returning the specific value depending on the type of record.

```
colset DTYPE = with specific | normal;
colset DISTRIBUTION = record dtype:DTYPE * exactValue:INT * mean:INT *
std:INT;
```

The example CPN model uses a very simple resource management approach – all resources required to the simulation is in one place (called the global resource pool) and every transition (i.e. task) which requires a resource to perform task is attached to a place holding available resources. As the transitions required a resource cannot perform their task with any arbitrary resource available then transition guards for specifying the type of required resource are used. The function *check_roles* with two parameters is used as a guard: first parameter is the list of roles (or qualifications) of resource and the second is the roles (or qualifications) required to execute task. If there is a need to construct the usage of individual resource then extra check should be added to the guard for checking the name of the individual resource.

If an activity requires more than one different resource to complete the task then the activity must be divided into smaller parts (as many as the requirements of the different resources, only exception is consumable bulk resource) and if the splitting to smaller tasks is not possible then more complex modelling technique should be used – for example divide the task into several (as many resources) parallel path where every resource is doing the task simultaneously. And if the bulk resource is consumable then the quantity of the resource should be generated into the resource pool and into resource allocation - the amount of resource is taken from the pool and not put it back to pool. This means if model has 50 litres of fuel as consumable bulk resource "Fuel" then in resource pool must contain 50 tokens to represent fuel. For other resources, the transitions can intake only one resource and if the task requires more than one resource then it must be split into smaller tasks or several simultaneous parallel paths.

The purpose of the simulating process is to analyse and improve it. Analysing the simulation requires data logged by the simulation. Previously mentioned ProM CPN Library extension requires modifying the declaration of transition inscriptions to add logging support. The current solution uses already the transition inscriptions to calculate the process time of the task, the cost and revenue of the task, costs of the task, etc. This data is calculated using functions *transitionAction* and *transitionActionWOR*, so these functions must be modified with logging functionality. Also a creating new cases (function *createCase*) requires logging. In *transitionAction* and *transitionActionWOR* the „start" and „complete" events are logged, and in *createCase* only „complete" event (as the transition only creates a case and does not consume any time). The MXML format has a data field where all additional data can be saved. Different events and task are saving different kind of information:

1) *createCase* „complete“ event
    a. Cost as cost to genereate token (i.e. *OneTimeCostPerToken*)
2) *transitionAction* „start“ event
    a. WaitingTime
    b. WaitTimeCost
3) *transitionAction* „complete“ event
    a. ProcessingTime
    b. Cost
    c. StartupCost
    d. ResourceCost
    e. Revenue
    f. NoOfResources
4) *transitionActionWOR* „start“ event I do not add any extra parameters
5) *transitionActionWOR* „complete“ event
    a. ProcessingTime
    b. Cost
    c. StartupCost
    d. Revenue

Note that the structure of the log file and recorded data is adjustable and depend on requirements and modelling templates.

### 3.2.5. Gateways

Modelling split and join gateways in Petri net is quite straightforward (see Figure 13). In this thesis, we consider three types of splits and 2 type of joins – fork (AND split), datadriven decision (XOR split), event-driven decision, join (AND join) and merge (XOR join).

Modelling a fork (i.e. AND split) in CPN needs the required amount of paths as places and output arcs from transition representing split to these places. From the basic of Petri-net we know that transition puts (i.e. clones) token to each output place connected to transition. Modelling the XOR split i.e. choosing the path based on normal distribution or data requires a simple modification on the template. Current examples do not cover data/expression-based decisions as they are more complex and requires expression engine (however, BPMN supports decisions based on data or conditional expressions). In Figure 16 transition "Decision split" has a code fragment for calculating a random value between 0 and 99, and based on the value selecting a path as an output parameter (i.e. path 1 is selected 90% of the cases and path 2 for remaining 10%). In the arc inscription the path parameter is checked and the token is only forwarded when the condition of path is matched, otherwise it will forward an empty token i.e. end the path (places with empty marking are omitted).

Joining different path into one is quite straightforward. In case of AND join (see Figure 17) the join point is transition and to ensure that it join two copies of same token the guard is added to the transition (`[#ID c =(#ID c1)]` where ID is the identification number of the token/case). The merge (XOR join, Figure 18) is much simpler and the join point is a place. As it does not need to distinguish the path then every token arriving to join place will enable the main path.

The event-based gateway mapping into Petri nets is shown in Figure 13 is very simplified. Modelling an event-based exclusive gateway has two options:

- The flow path is selected in the same way as for the XOR split (Figure 16), i.e., every path has a probability and the path is chosen according to this probability, or
- By mapping message events and timers to corresponding CPN transitions with the corresponding behaviour and letting these transitions create a race condition.

Below we explain the second type of mapping – since the first one has already been introduced in the context of XOR-splits. Figure 19 shows an event-based gateway in BPMN while Figure 20 shows the same gateway modelled in CPN. The gateway is attached to two message events and one timer (i.e., timeout) event, which are represented by transitions in the CPN. As only one path can be chosen, there is a place where case waits to see which events occur first. Every event ("yes", "no" and "timeout") has two input places and one of them ("waiting for response") is shared by all of them. This ensures that only one event can occur. The message events are enabled when they receive messages from the environment and timeout is enabled when model time has reached the gateway access time increased by delay. The transition representing the listeners must have guards to ensure that the message and the waiting case are the same (for example, checking the *ID*).

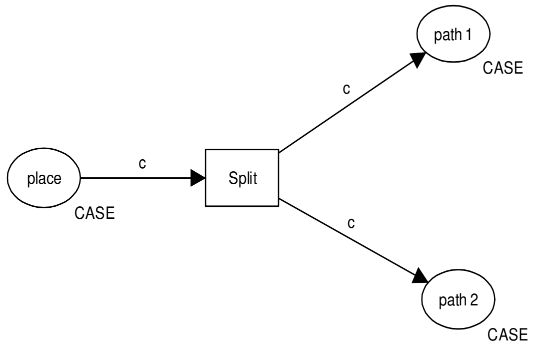
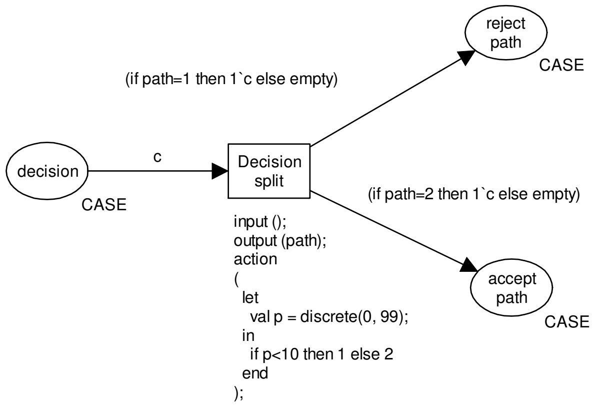


**Figure 15: AND split**


decision
CASE
c
Decision split
(if path=1 then 1`c else empty)
reject path
CASE
(if path=2 then 1`c else empty)
accept path
CASE

```
input ();
output (path);
action
(
 let
   val p = discrete(0, 99);
 in
   if p<10 then 1 else 2
 end
);
```

**Figure 16: XOR gateway/split template**

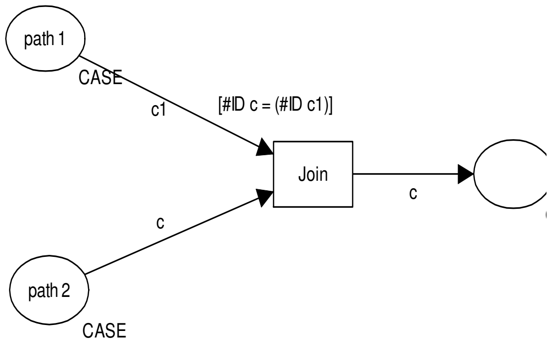


**Figure 17: AND join**

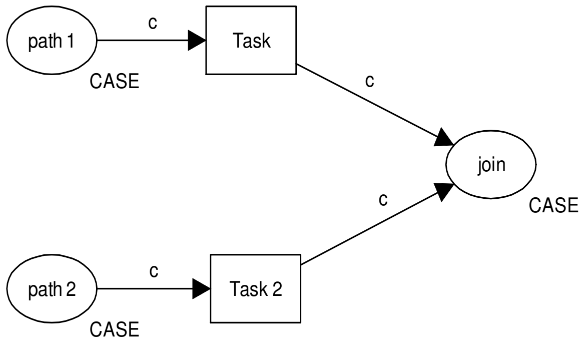
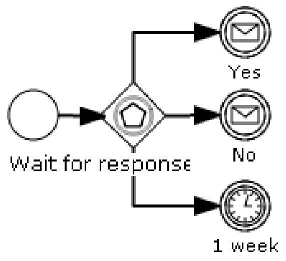


**Figure 18: XOR join**



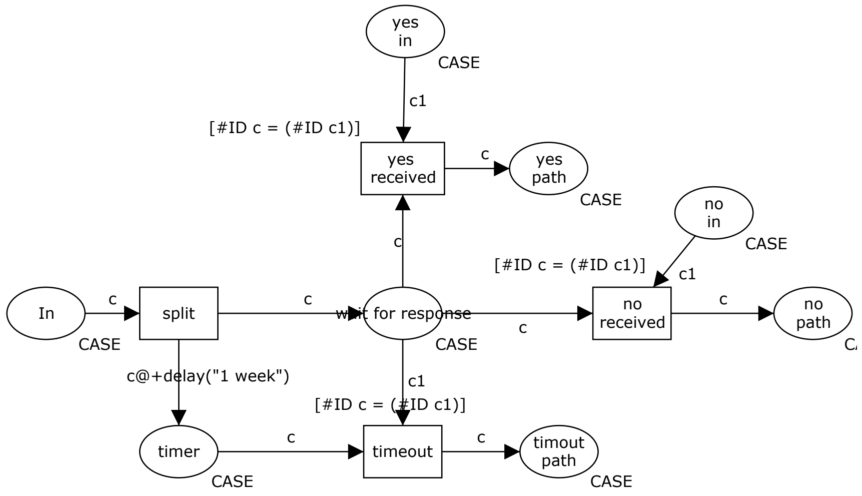
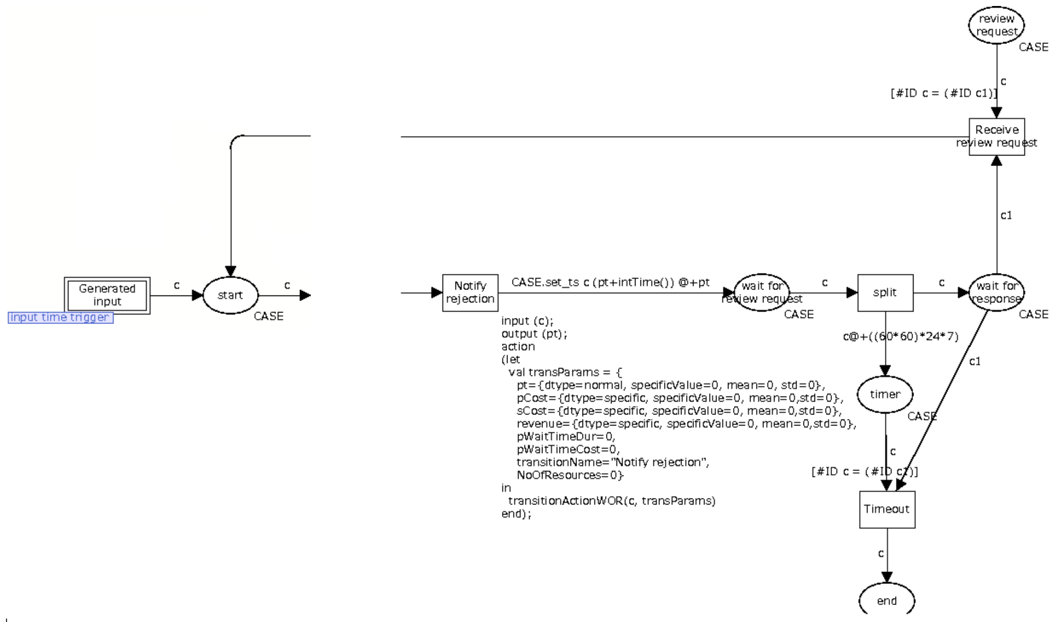


**Figure 19: Event-based gateway in BPMN**

**Figure 20: Example of event-based gateway in CPN**

The running example (i.e., credit card application in the Figure 1) has only one event-based gateway ("*review request*" gateway). It is simpler than the example in Figure 19 and its according model in CPN in Figure 20. It has two outgoing path, one is waiting for message ("Receive review request") and second one is timeout ("Time out"). In the Figure 21 the fragment is modelled in CPN.



```
input (c);
output (pt);
action
(let
  val transParams = {
    pt={dtype=normal, specificValue=0, mean=0, std=0},
    pCost={dtype=specific, specificValue=0, mean=0,std=0},
    sCost={dtype=specific, specificValue=0, mean=0,std=0},
    revenue={dtype=specific, specificValue=0, mean=0,std=0},
    pWaitTimeDur=0,
    pWaitTimeCost=0,
    transitionName="Notify rejection",
    NoOfResources=0}
in
  transitionActionWOR(c, transParams)
end);
```

**Figure 21: Example of event based gateway in credit card application.**

## 3.2.6. Generation of simulation input cases

In order to run a simulation, we need to generate a certain number of cases. These cases are generated according to a certain arrival rate. To capture the creation of cases, we use a sub-page which basically generates input tokens (i.e. cases) which are fed to the main part of the CPN model.  The structure of the sub-page is very simple, it has 2 places („counter“ and „start“) and one transition („generator“). The place „counter“ holds the identification number of the case and is initiated with one token „0@0“ (the *ID* of the first case is 0 and the timestamp 0, i.e., the clock of the model is 0 before the simulation). The place „start“ holds the type of *CASE* and its input is an output for the transition „generator“ and its output will be the input of the model. The transition „generator“ takes a value of type of *ID* and outputs a generated case to the place “start” (the case is created by the function *generateCase* in code segment), also it has a guard (function *generatorGuard*) to stop the generation of input cases if the limits are met, i.e., the total amount of the generated cases or the end date of the simulation. In the output arc to the place "counter" the *ID* is incremented by 1 and time delay/stamp is added to the token. The delay depends on the type of input trigger. Here we adopt a constant inter-arrival time which is captured by a variable *timeBetweenBundles*. At each arrival, we need to create a number of cases, which is determined by the variable *noOfTokensPerBundle*. Also the model start and end dates are defined by constant variables *startDate* and *endDate* accordingly. Both are the type of the ML Date [20]. The generating input or receiving new cases may require some spending and it can be defined by constant variable of *OneTimeCostPerToken*. The colour set of *OneTimeCostPerToken* is *DISTRIBUTION*.

```
val startDate
val endDate
val timeBetweenBundles = {dtype=specific, specificValue=3600, mean=0,
std=0};
val totalNoOfToken = 7;
val noOfTokensPerBundle = 2;
val OneTimeCostPerToken = {dtype=specific, specificValue=0, mean=0,
std=0};;
colset ID = INT timed;
var i:ID;
```

Figure 22 is the example of generating cases for the credit card application.

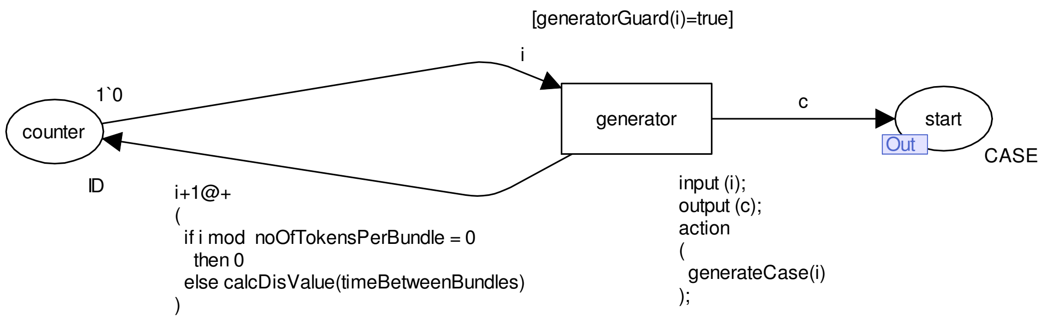


**Figure 22: Example of input generation in CPN**

Note that there are other more sophisticated models for generating input cases. A very common one is to use negative exponential inter-arrival times (also known as a Poisson process). In this case, one would need to provide an alternative implementation of the function *calcDicValue*, which would take as input the mean inter-arrival time, and would produce a randomly generated value according to a negative exponential distribution with the given mean.

### 3.2.7. End-to-end Mapping Method

Given the elements introduced above, we sketch below a step-by-step method for mapping a business process simulation model into a CPN.

1) Create colour sets, help functions. For example,

   colset ID = INT timed; colset DTYPE = with specific | normal; etc.

2) Variables for binding values in inscriptions. For example,

   var r:RES for resources; var c:CASE for cases; var path:INT for decisions; etc.

3) Create value constant

   a. Generation constants like start and end date of the simulation, etc.

val startDate = Date.date{day = 1, hour = 0, minute = 0, month = Date.Jan, offset = NONE, second = 0, year = 2000}

    b. Recurring intervals for timetables
    c. Timetables
    d. Resources
    e. Logging

4) Create functions
5) Create input generation subpage
6) Create and initiate place holding resource pool
7) Create process flow
    a. Create flow object i.e. tasks and decisions/gateways
    b. Add incoming-outgoing arcs
    c. Guards
    d. Code segments

This end-to-end procedure provides the building blocks for implementing an automated transformation from process simulation models to CPNs. As stated in the Introduction, implementing this automated procedure is outside the scope of this thesis and would require additional refinement work.

# Conclusion

The BPMN standard has begun de facto standard in the field of modelling business process. But it does not provide any kind of simulation support. However, simulations of business processes are one of the most important building blocks for studying and improving the processes. Also most of the business modelling tools providing simulations does not provide possibility to extend the simulation engine or import simulation models created with other tools.

In this paper two of the BPMN-based business process modelling tools was reviewed and a meta-model is constructed to extend the BPMN meta-model with simulation attributes. Also the paper suggests a mapping BPMN models extended by described meta-model to CPN models and use simulation engine of CPN Tools. CPN Tools has built to create CPN models and carry out performance analyzes through simulations.

The paper describes how to create a business process models in CPN through one fully working example. Also it provides simple templates and ML code snippets to give an impression how to use CPN constructs and the features of CPN Tools to model and simulate business processes.

The current version of the meta-model is quite general and needs an improvement. There are lots of specific attributes of simulation which left out from the scope of this paper.

Most of the current business process modelling tools and their simulation engines supports very limited set of resource patterns and usually they focus on pull-based resource patterns based on roles. This paper describes only a role based resource pattern and one of the possible implementation in CPN. But usually real life situations are much complex and usually the simulations are used to find such bottlenecks in the resource allocations. So there is an open opportunity to extend existing process simulation technology to deal with more sophisticated workflow resource patterns. One of the possible directions in the future is to extend simulation meta-model to support more workflow resources patterns and provide templates for using them in the CPN simulation models.

A second possible direction of this work in the future is to extend simulation meta-model to support state data. One of the main purposes is to create models that accurately reflect real world processes and simulations are used to answer strategic questions and to make tactical and operational decisions. Usually simulation experiments are starting from an empty initial state and this cause a shortcoming in case of short-term decision making – to get sufficient data to analyze simulation we have to run it a very large number of cases to get average statistics. Also the data generated by simulation may be flawed or misleading as models are created manually and are only approximations of processes of real word (i.e. the interpretation of historic data is not accurate and input parameters are wrongly chosen). This direction has been explored in [10].

As CPN Tools accepted models defined in XML format then it is possible to generate models automatically. This paper describes some mappings form extended BPMN to CPN model and the author of this paper suggests developing an automatic converter from BPMN with simulation meta-data to CPN tools simulation model. The purpose of this thesis was to provide a design for a transformation from process simulation models to CPNs. Implementing this transformation is likely to require additional refinement.

# Resümee

## Äriprotsesside simulatsioonid kasutades CPN mudeleid

Magistritöö (20AP)

Marek Zäuram

Äriprotsesside juhtimine on tänapäeva maailmas muutunud järjest olulisemaks tänu suurenevatele nõudmistele protsesside tõhusamaks muutmisel. Protsesside juhtimisega üritatakse leida võimalikke kitsaskohti, tõsta tootlikust ja vähendada vajadust ressursside järjele. Paljud protsesside modelleerimise vahendid toetavad lisaks protsesside modelleerimisele mudelite juurutamist ning simulatsioone. Simulatsioonid on üks parimaid tehnikaid protsesside efektiivsemaks muutmisel.

Enamus äriprotsesside modelleerimise standardeid ei toeta simulatsioone (k.a. de facto standard BPMN). Mitmed protsesside modelleerimise vahendid võimaldavad simulatsioone, kuid simulatsioonimootor on peidetud nende vahendite sisse. See piirab simulatsioonimootorite laiendatavust ning protsesside simulatsioonides tuleb läbi ajada tarkvara vahenditega. Kuid modelleerimistarkvara võimalused on võrreldes reaalse maailma situatsioonidega väga piiratud.

Antud töö eesmärgiks on välja pakkuda eelpool mainitud probleemile üks võimalik lahendus – BPMN meta-mudeli laiendus simulatsiooni atribuutidega. Töös kasutatakse simulatsioonimootorit CPN Tools, mis baseerub Coloured Petri Nets (CPN) modelleerimis-keelel. CPN on loodud spetsiaalselt diskreetsete süsteemide simulatsiooni keeleks.

Esmalt tehakse ülevaade BPMN standardist ning kahest levinumast BPMN modelleerimise vahendist – IBM WebSphere Modeller ja IPT Commerce. Järgnevalt antakse lühiülevaade Petri Nets ja CPN mudelitest ning CPN Tools modelleerimisvahendist. Kasutades olemasolevate vahendite analüüsist saadud kogemusi, laiendatakse BPMN standardi meta-mudelit simulatsioonide meta-mudeliga. Seejärel kirjeldatakse kuidas saada simulatsiooni andmetega täiendatud BPMN mudelist CPN mudel ning demonstreeritakse erinevaid võimalusi CPN mudelite simuleerimiseks kasutades CPN Tools vahendeid.

Töös kirjeldatud simulatsiooni meta-mudel ja BPMN mudeli elementide teisendamine CPN mudeli konstruktsioonideks on esitatud üldiselt, sest eesmärgiks oli testida välja pakutava lahenduse võimalikkust. CPN Tools sai valitud simulatsioonimootoriks sellepärast, et tema mudelid on kirjeldatud kasutades XML-i. Töö edasiarendusena pakub autor välja BPMN mudelitest CPN mudelite automaatse konverteri loomist.

# Bibliography

[1] Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, Mathias Weske, “Business Process Management: A Survey”. In Proceedings of the Business Process Management Conference, Eindhoven, The Netherlands, September 2003, pp. 1-12.

[2] M.H. Jansen-Vullers and M. Netjes, “Business Process Simulation - A Tool Survey”, In Proc. of the Seventh Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools (CPM 2006), University of Aarhus, Denmark, October 2006.

[3] Object Management Group, Inc., “Business Process Model and Notation (BPMN)”. [Online] [Cited: March 25, 2010] http://www.omg.org/spec/BPMN/1.2/

[4] Stephen A. White, “Introduction to BPMN”, IBM, May 2004, Available at http://www.bpmn.org/

[5] K. Tumay, "Business process simulation", Winter Simulation Conference (WSC'95), pp.55-60, 1995

[6] Remco Dijkman, Marlon Dumas and Chn Ouyang, “Formal Semantics and Analysis of BPMN Process Models.” Information and Software Technology Vol. 50, No. 12, pp. 1281-1294, November 2008

[7] Business Process Trends, “The 2007 BPM Suites Report“, [Online] [Cited: March 25, 2010] http://www.bptrends.com/reports_toc_01.cfm

[8] IBM, Inc., “WebSphere Software”, [Online] [Cited: March 25, 2010] http://www-01.ibm.com/software/websphere/

[9] IBM, Inc., “IBM WebSphere Modeller Help”, [Online] [Cited: March 25, 2010] http://publib.boulder.ibm.com/infocenter/dmndhelp/v6rxmx/index.jsp

[10] A. Rozinat, M. T. Wynn, W. M. P. van der Aalst, A. H. M. ter Hofstede, and C. J. Fidge, “Workflow Simulation for Operational Decision Support Using Design, Historic and State Information”, Proceedings of the 6th International Conference on Business Process Management, Milan, Italy, September 02-04, 2008

[11] Process Mining Group, Math&CS department, Eindhoven University of Technology. “Mining XML (MXML)”. [Online] [Cited: March 25, 2010] http://is.tm.tue.nl/research/processmining/WorkflowLog.xsd

[12] Anne Rozinat, Moe Thandar Wynn, Wil M.P. van der Aalst, Arthur H.M. ter Hofstede, Colin J. Fidge: “Workflow simulation for operational decision support.” Data Knowledge Engineering Vol. 68, No. 9, pp. 834-850, 2009

[13] University of Hamburg, Germany, ”Petri Nets World”, [Online] [Cited: May 8, 2010] http://www.informatik.uni-hamburg.de/TGI/PetriNets/

[14] Kurt Jensen, Lars M. Kristensen, “Coloured Petri Nets. Modelling and Validation of Concurrent Systems”, Springer-Verlag, Berlin Heidelberg 2009, ISBN: 978-3-642-00283-0

[15] University of Aarhus, “Coloured Petri nets”, [Online] [Cited: March 25, 2010] http://www.daimi.au.dk/CPnets/

[16] L. Wells, “Performance Analysis using CPN Tool”, ACM International Conference Proceeding Series; Vol. 180 archive;  ISBN:1-59593-504-5

[17] University of Aarhus, “DTD for net files“, [Online] [Cited: March 25, 2010] http://wiki.daimi.au.dk/cpntools-help/dtd_for_net_files.wiki

[18] Nick Russell, Wil M.P. van der Aalst, Arthur H.M. ter Hofstede, and David Edmond, “Workflow Resource Patterns: Identification, Representation and Tool Support“. In: Advanced Information Systems Engineering: 17th International Conference, pp. 216-232, June 2005

[19] A. K. Alves De Medeiros and C. W. Günther, “Using CPN Tools to Create Test Logs for Mining Algorithms”, Proceedings of the Sixth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, pp. 177-190, 2005

[20] Standard ML, [Online] [Cited: March 25, 2010] http://www.standardml.org/

## Appendices

## Appendix A