UNIVERSITY OF TARTU
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
Institute of Computer Science

Madis Abel

Lightning Fast Business Process Simulator

Master Thesis (30 EAP)

Supervisors: Luciano García-Bañuelos, PhD
Marlon Dumas, Prof

Author: ..... " ..... " May 2011
Supervisor: ..... " ..... " May 2011
Supervisor: ..... " ..... " May 2011
Professor: ..... " ..... " May 2011

TARTU 2011

Contents

Introduction

Background

Business process management is a discipline to make an organization's workflow more efficient and more capable of adapting to changes in an ever-changing global environment. Making changes in real-life business processes could lead to undesired results if potential impact of change is not completely analyzed before the changes are applied. Business process simulation is a widely used technique for analyzing business process models with respect to performance metrics such as cycle time, cost and resource utilization before putting them to production.

Problem Statement

Many commercial state of the art business process modeling tools incorporate a simulation component, e.g. IBM Websphere Business Modeler [13], Savvion Process Modeler [14] and others. However, these process simulators are often slow, cannot simulate complex real-life business processes and sometimes cannot even deal with large-scale simulations. For example, it is not possible to simulate process models with sub-processes, intermediate events or inclusive merge gateways (Or-joins).

Objective

The objective is to build a lightning fast business process simulator engine which could also handle advanced constructions in the process models that are used to represent real-life processes. The simulator is designed and implemented from scratch in the Java programming language and it will support the simulation of business process models defined in the BPMN 2.0 [6] standard.

Contribution

This work presents a novel approach to business process simulation field by using the architecture of a scalable and high-performance business process simulation engine. The contribution of this thesis is a set of design principles, architecture supporting simulations of models containing advanced BPMN constructions like loops, sub-processes, intermediate events and Or-joins.

Document Organization

- Chapter 1 introduces the concept of business process management, modeling and simulation in particular; it gives a brief overview of related work done so far in this field and summarizes the shortcomings of the current research and commercial state of the art simulation tools.
- Chapter 2 describes the new approach in detail including proposals of how to handle advanced constructions available in BPMN 2.0.
- Chapter 3 describes the architecture using class diagrams of the simulator engine.
- Chapter 4 contains the analysis of achieved results, performance comparison with several process models and an overview of simulation speeds with existing commercial business process simulation tools.

1. Business Process Management

A business process is a collection of related, structured activities or tasks that produce a specific service or product. In general we can say that business processes exist in all companies that serve a particular goal, but very often those are not written down or defined formally. A goal of business analysts is to make such process as efficient as possible in order to gain higher customer satisfaction, product or service quality, delivery and time-to-market speed. This kind of goal can be achieved using a systematic approach called business process management (henceforth BPM for short).

Business process management is a discipline to make an organization's workflow more efficient and capable of adapting to changes in an ever-changing global environment. BPM is an ongoing set of action items as shown in Figure 1.1.

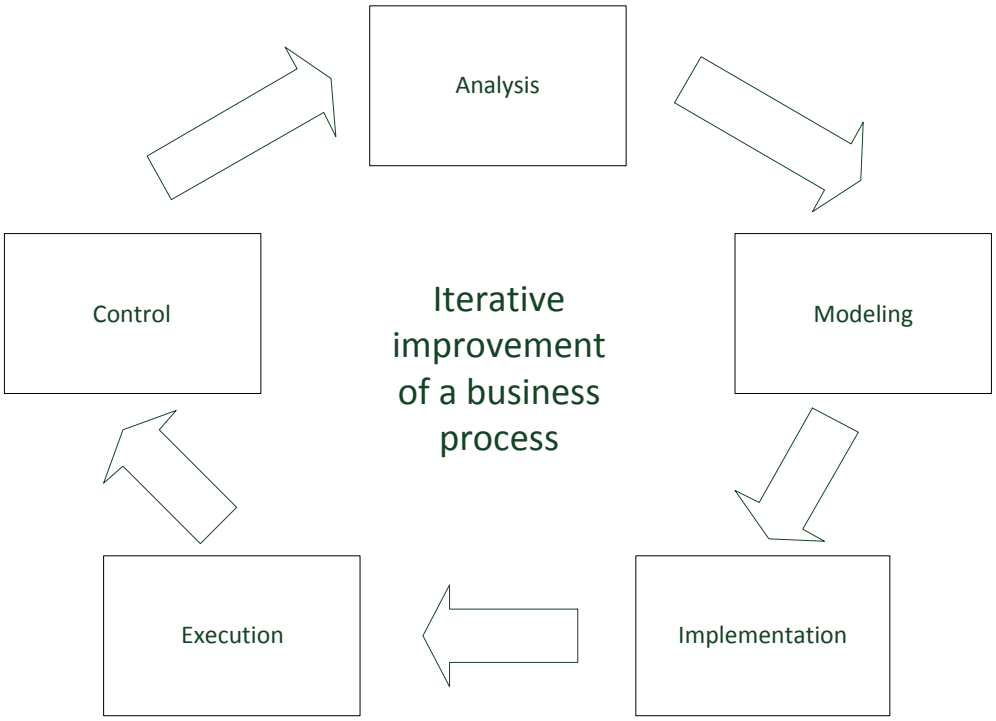

Figure 1.1. BPM Lifecycle

The first step to BPM is the analysis phase. The aim of the analysis phase is to determine what needs to be done, what are the key bottlenecks and weaknesses within a business process being managed. If the business process is being analyzed for the first time, the input information for this phase comes from the initial vision of the process. If the business process already exists, the data to be analyzed is obtained from the previous iteration(s)

and again, it is possible to determine the bottlenecks and weakness of the process to be improved.

In the modeling phase the business process will be written down, usually using some standardized format like Business Process Modeling Notation (BPMN) [6], Unified Modeling Language Activity Diagrams (UML) [15], Event-driven Process Chain (EPC) [16] or any other flowchart type diagram. If the model of a process does not exist yet, it will be developed. Otherwise, the model is iteratively revised in order to improve it using the data from the analysis phase. The goal of the modeling phase is to take an "as-is" version of the business process and to refine it to define a "to-be" version of it. In order to select the best alternative process model it would be useful if the new model could be tested out without having to apply it in the real world. It can be done by simulating the business process which in general is called the business process simulation (henceforth BPS for short).

The implementation phase is meant for preparing the current system, train the employees and do additional preparations so that the new model would be applicable to the real world business. The implementation may require some additional changes to be done before it can be applied. For example if the new model introduces a change which means that some part of the process will be automated using a service, then it would require additional work, probably by the IT department, to integrate the new service to the existing system to make it usable.

The execution step in the BPM lifecycle is to make the new "to-be" model effective in the real world business.

In the control phase, when the new business process is in use, process instances are being monitored and inspected. In this phase data will be collected for the next iteration of the BPM lifecycle.

The business process simulator, built as the result of this master thesis, helps the business analysts in the analysis and modeling phases of the BPM lifecycle. Business process simulation is an essential part of the BPM analysis and design phases. Redesigning a process always contains a risk of failure which could result in unnecessary reinvestments. BPS helps to develop new business processes and try them out before making the new model effective in the real business. Implementation, execution and control phases are applicable to the real world business process only and depend highly on the analysis and modeling phases which can be developed with the help of a business process simulator.

1.1. Key Performance Indicators

To measure the quality, performance, cost, duration, customer satisfaction and other indicators that represent the goodness of a business process, some measurements need to be defined. In the business process management domain such quantifiable measurements are called the key performance indicators (henceforth KPIs). KPIs could also be related to more generic company-wide business planning strategies which are the main drivers for BPM.

Process time is the time measured from the beginning to the end of a single process case. Process time for separate process instances could vary depending on how the process works. Usually unsuccessful process paths (e.g. order cancelled or rejected) take less time than successful process paths (e.g. order accepted, paid and shipped). Process time may be different even for processes with the same end result. This may be caused by resource allocation at some point in time or additional error handling that had to be done within one process instance and not within the other. For example there are two possible paths in the order management business process in Figure 1.3: one for rejected orders and other for successfully finished orders. Therefore average process time might not always be the most informative KPI, but it would give better results if processes with successful and unsuccessful activity paths are measured separately.

Cycle time is the sum of time spent on all possible process paths considering the probabilities of the path to be taken in a process instance. Therefore cycle time is not related to any specific process instance and can be calculated without the simulation.

Waiting time is the time measured from enabling a task to the time when task was actually started. For example it is the time that patient waits for in the queue of doctor's cabinet. Waiting time is caused by resources being busy and it introduces a queue of waiting process instances.

Processing time is the time spent on all activities without waiting and transfer times. Thus cycle time corresponds to the sum of processing and waiting times.

Process cost is the sum of all costs in a process instance. Costs may be related hourly wages paid for human resources performing activities, shipping, taxes or any other fees that need to be paid to perform an activity.

Resource utilization is an indicator that expresses the rate of allocated resources on average during the period that was inspected. This KPI depends on the activity execution time and the number of total resources available.

Also there are other KPIs, but those that were pointed out in this chapter are most common ones. For cost based KPIs there is always an economic trade-off between the cost of service and waiting as illustrated in Figure 1.2. The goal for a business analyst is always to maximize some and minimize the other KPIs to find the most optimal balance between them.

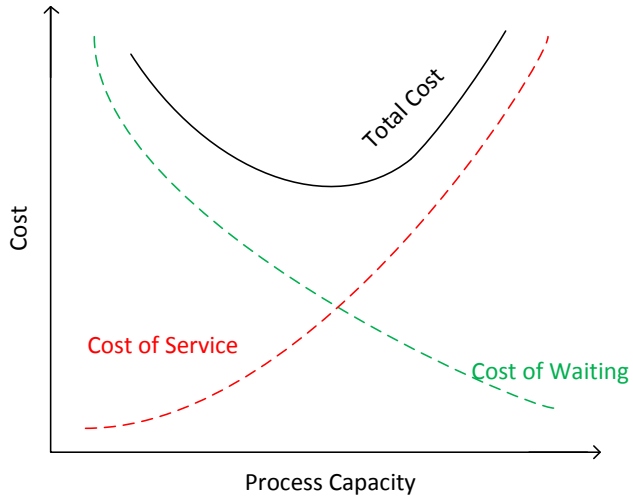

Figure 1.2. Economic Trade-off [27]

1.2. Business Process Modeling

There are several ways how to describe a business process. It could be just written down to a paper so that it can be understood or it can be visualized with a flowchart as a sequence of activities. Nowadays the Business Process Modeling Notation (BPMN) has become the *de facto* standard for process modeling. The last version of this notation, namely BPMN 2.0 [6], was released by the Object Management Group [17] in January 2011. It defines a graphical notation with a large set of elements, their semantics and a XML based serialization format.

The simulator built as the result of this work supports the business processes defined in the BPMN 2.0. A simple business process model diagram (henceforth BPD in short) of an order processing example in BPMN is shown in Figure 1.3. This BPD consists of nodes of three types: events (represented as circles), activities (represented as rectangles) and gate-

ways (represented as diamonds). Events denote things that happen at a particular point in time, activities denote work that needs to be performed, and gateways serve to route the flow of control along the branches of the BPD. Nodes are connected by means of directed edges called sequence flows. A sequence flow basically says that the flow of control can pass from the source node to the target node [26].

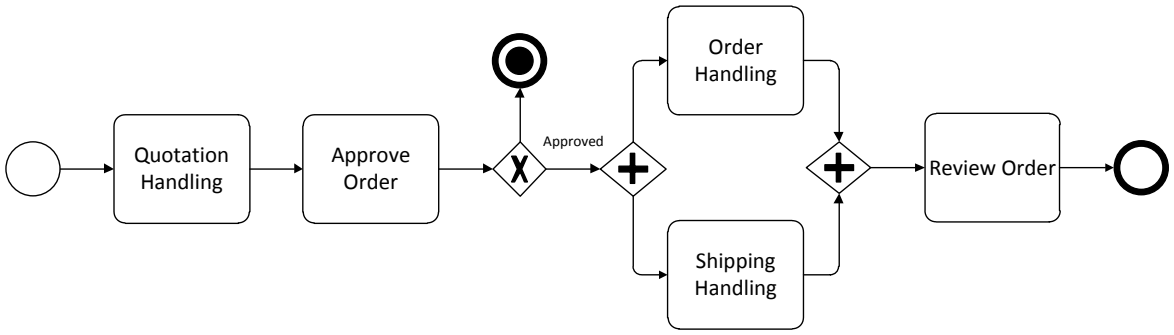

Figure 1.3. BPMN - Order Handling Example

1.3. Business Process Simulation

Business process simulation (BPS) is a widely used technique for analyzing business process models with respect to performance metrics described in section 1.1. BPS allows business analysts to understand how a business process actually works without putting the process to the production environment. With BPS it can be easily analyzed how the changes in the process, in resource management or in the process parameters such as arrival rate would affect the current KPIs. For example if an insurance company states in their insurance contracts that all cases would be checked within 7 days from the issue date, the company needs to make sure that they are actually capable of managing all claims in case of some disaster like earthquake or conflagration where number of incoming claims increases rapidly. In order to simulate that kind of scenarios, the existence of precise simulation information with a good tool are critical.

There are several possible outcomes of a business process simulator. In general a simulator takes a business process model diagram and additional simulation specific data and produces some type of output as shown in Figure 1.4. The output can be a set of process logs of each process instance that can be analyzed with a process mining framework, it could calculate only the KPIs or it could do some additional benchmarks and produce diagrams.

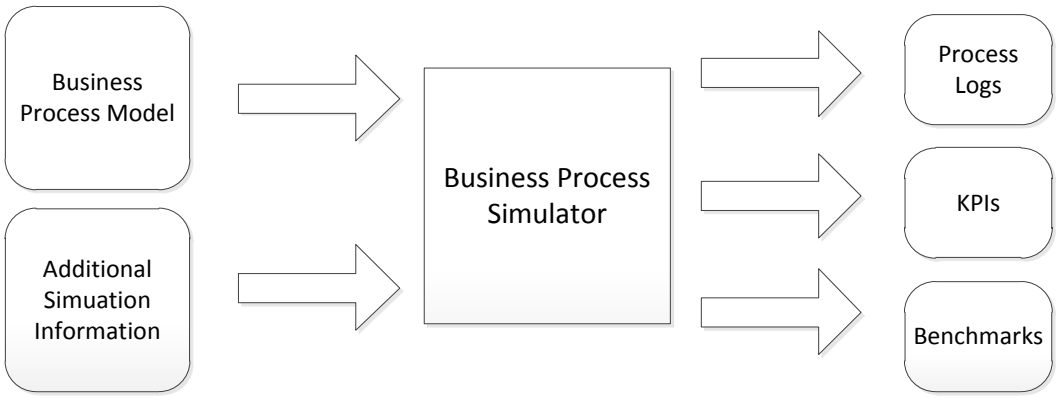

Figure 1.4. Business Process Simulator Input and Output

In the study presented in [2] the list of 70 evaluation criteria were provided for selecting the simulation software for a specific need. The main requirements are always the same, but those might vary depending on the goals set for the project. For example some tools provide support for animation of the simulation to dig into the process, but it would often slow down the simulation and visualization may not be needed at all if business analyst only wants KPIs and basic statistics of the process as fast as possible. The criteria presented in [2] also included items related to programming aspects like access to the source code and support; how the software can be integrated with other third party systems like DBMS or frameworks for statistical analysis. According to [3] there are three important categories of capabilities that must be considered when evaluating BPS tools: modeling, simulation and output analysis capabilities. Software supporting all these criteria would be too complex and often actually not required at all. Therefore it is important to select the simulation software according to the goals of the project. In the section 1.5 the capabilities of some research and commercial simulation tools are described which are prominent in the market nowadays.

1.4. Required Simulation Information

BPMN 2.0 does not specify how simulation related information has to be serialized, because the simulation is considered to be outside the scope of BPMN. Although some attempts have been made like up to now to overcome this problem (e.g. [1]), but at the moment there is no standard way of representing simulation information in BPMN.

In order to simulate a business process model additional simulation information has to be provided for:

1. Process instance initialization
2. Element execution
3. Resource allocation
4. Cost of activities
5. Branching probabilities

Process instance initialization data must contain the number of instances of the cases to create and the information about the arrival rate which defines the time interval of when the next instance started after the previous one. The process initialization data must contain information about which and how many resources (actors or roles for tasks) are available for the process instances defined in the model.

Element execution data has to be associated with all elements which last for some specified time and it must define how many time units it takes to complete the element in average.

Resource allocation data has to be associated also with each task and it must define which resource is responsible for and performs the task. Resource management is one of the key elements in the simulation – a task cannot be started if all resources are in use which results in a queue of waiting activities.

Cost of activity is a monetary value of how much does it cost to perform an activity. Usually the cost of activity includes the cost related to the duration (e.g. the hourly wage for human resources) and additional costs (e.g. the shipping cost, road usage fees) that are fixed.

Branching probabilities must be defined for the outgoing flows from the Xor- and Or-split gateways in the process model. Using the branching probability information, the simulator determines which path(s) will be taken in a particular process instance.

In the real world task execution times and process arrival rates in the most cases are not fixed values, but those can be defined by an average value and some distribution info. Besides the fixed amount of time units there are three commonly used distributions to describe a variety of time values: standard, uniform and exponential distribution.

1.5. Related Work

1.5.1. Research Prototypes

A BPS tool survey, which was carried out in [3], looked into the following projects: Protos, ARIS, FLOWer, FileNet, Arena and CPN Tools. In [3] modeling, simulation and output

analysis capabilities have been compared with the previously listed tools and it has been concluded that FLOWer, FileNet and Protos are unsuitable for BPS studies due to lack of simulation capabilities. The three remaining tools, ARIS, Arena and CPN Tools, can be considered for BPS studies in general. However, ARIS is based on informal process modeling language of EPCs and does not have the full capability for modeling workflow patterns. The modeling with Arena is based on predefined building blocks and it is important to have full knowledge of those building blocks and the exact mode of operation [3].

There have been several research projects on business process simulation. For example, in [23] a mapping from models in BPMN to Colored Petri nets (CPN) [25] has been provided and has been implemented in [4]. CPN Tools [25] is a powerful tool for editing, simulating and analyzing Colored Petri nets However, it is quite technical and not intuitive enough to model real-life business processes. A profound knowledge is required of modeling CPNs and the resulting models are difficult to understand by the general process owners who should be able to comprehend and validate models [3]. The mapping introduced in [23] and implementation in [4] help to overcome this shortcoming of the CPN Tools, but in the end the intermediate layer on behalf of the CPN Tools, which causes additional overhead, is still required.

In [24] an open source BPS tool has also been built, but this tool does not support process models in BPMN and, therefore, the set of available constructions which can be used in process models is limited.

1.5.2. Commercial Business Process Simulation Tools

There are several commercial BPS tools available nowadays. In this chapter, two commercial BPS tools on the market will be compared to find out about their capabilities: IBM WebSphere Business Modeler (WebSphere) and Savvion Process Modeler (Savvion). Some other tools (e.g. TIBCO Business Studio) have been discarded due to the lack of documentation or difficulties in setting them up.

The vision of the simulator to be built as the result of this work was to support as much as possible different BPMN constructs and to do it faster than any other available BPS tool. Therefore this section of the thesis concentrates mostly on the support of more advanced BPMN construct coverage and to the speed of the simulation.

WebSphere Business Modeler is one of the leading commercial business process analysis and simulation tools[1]. 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 [4]. Additionally, WebSphere supports different locations and timetables assigned to resources used in the process models. WebSphere is not fully BPMN compliant but the internal modeling language used there is inspired by it as seen in Figure 1.5. Modeling notation by WebSphere is inspired by BPMN, but it supports only a small subset of modeling elements like start events, end events; And- and Xor-gateways; loops; timers and others. With this subset of elements it is complicated or even impossible to model complex business processes to achieve a detailed simulation [4].

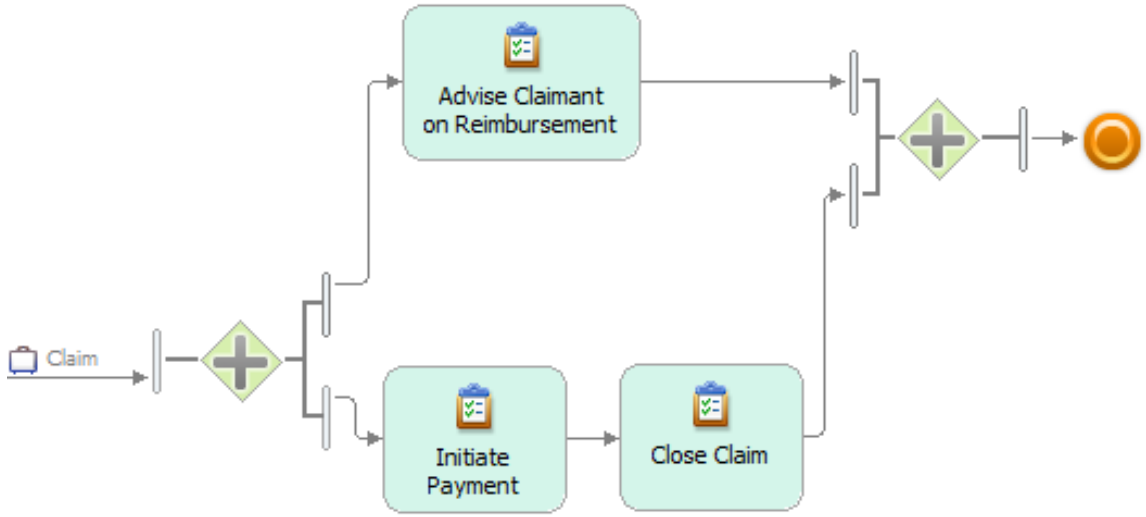

Figure 1.5. Modeling in IBM WebSphere Business Modeler

Savvion is another commercial BPS tool which supports business process modeling, analysis and optimization. Ease of use and simulation capabilities were criteria why this tool was chosen for comparison in this thesis. From the modeling perspective, Savvion uses BPMN as the modeling notation. Figure 1.6 shows a simple business process model in Savvion. Similarly to WebSphere only a small subset of BPMN elements is supported: start, end and message events; tasks; And-, Xor-split and And-, Xor- and Or-join gateways. However, the intermediate message events can be used only for modeling purposes and the Or-join gateway is not BPMN compliant. Or-join gateway in Savvion works basically like

[1] IBM WebSphere Business Modeler 7.0 (Advanced Edition) has been used.

Xor-join and does not wait for all its started preceding elements to be completed. In conclusion, we can say that the set of supported BPMN constructions for simulation is currently very limited and it is not possible to conduct simulations of more complex business processes.

Performance and simulation speed comparison with Websphere and Savvion BPS tools is described later in chapter 4.

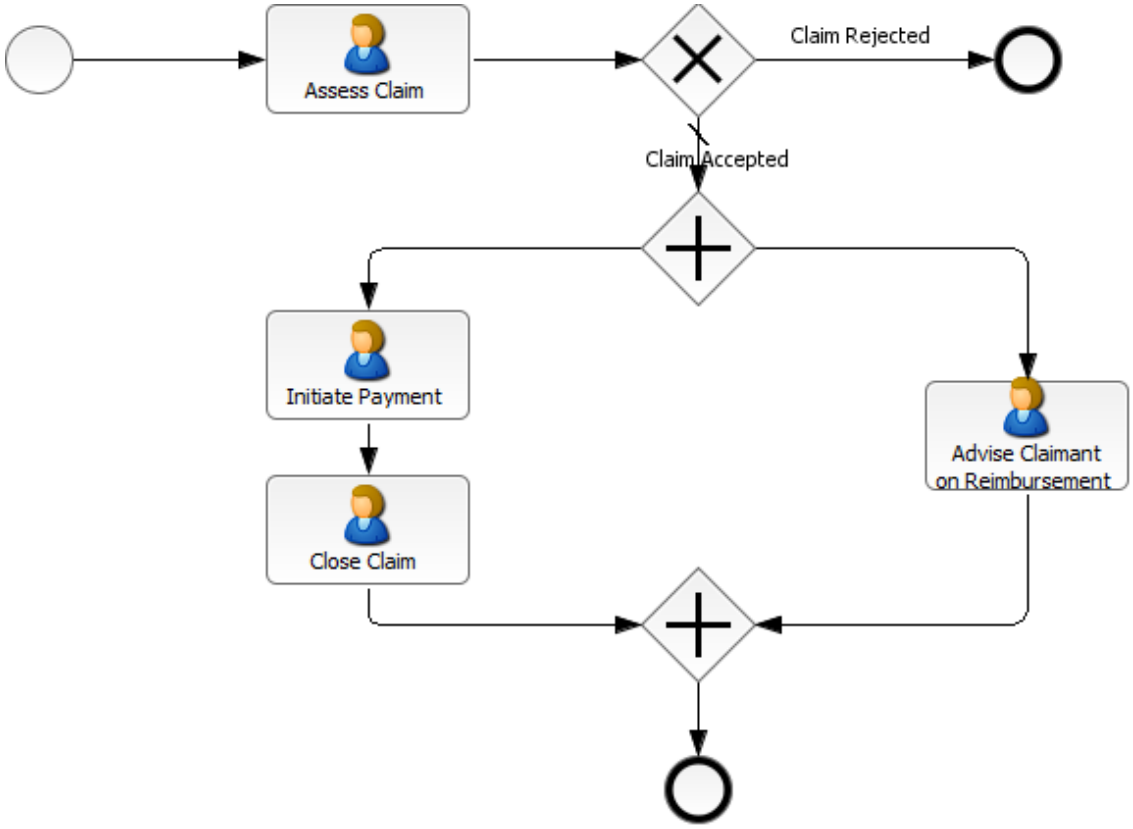

Figure 1.6. Modeling in Savvion Process Modeler

1.6. Summary

In this chapter it has been shown that the existing BPS tool research prototypes are not always suitable for simulations, require profound knowledge of the tool-specific concept and do not support the *de facto* standard modeling notation BPMN. It was concluded in the previous section that the existing commercial and research BPS tools lack the support of more advanced BPMN constructs such as intermediate, boundary and error events; event-based gateways and inclusive converging gateways. Also, the simulation process with these existing tools is rather slow and cannot deal well with sophisticated simulations containing a large amount of process instances to generate.

2. Business Process Simulator

Simulation is a complex process and there are some constraints for models that can be processed. This chapter describes which models can be simulated by the built engine, what exactly is required as input data and how the simulator works internally.

2.1. Requirements

The first and the most obvious requirement for the BPMN 2.0 model to be simulated is that it must contain only those BPMN 2.0 elements that are supported by the simulation engine.

Secondly, simulation-specific data has to be provided with business process models. BPMN 2.0 specification does not describe how simulation-specific data like task execution times and process instance arrival rate should be stored. The *Documentation* property of BPMN elements is used (in the built program) to store this kind of data in the JSON [18] format.

The third constraint is related to the so-called unsafe models. The notion of a safe and unsafe model is explained thoroughly later in this chapter.

2.1.1. Business Process Model in BPMN 2.0

BPMN 2.0 defines how a business process model in this notation has to be serialized in the XML format. As a result it is relatively easy to read the structure of a model programmatically. For example a start event, a task and a connector flow between these elements in BPMN 2.0 are serialized as follows:

```
...
<startEvent name="Order Received" id="start"/>
<task completionQuantity="1" startQuantity="1"
  isForCompensation="false" name="Review Order" id="review"/>
...
<sequenceFlow sourceRef="start" targetRef="review" id="1"/>
...
```

A simple order handling business process shown previously in Figure 1.3 serialized in BPMN 2.0 can be found in appendix A.

2.1.2. Simulation Data in the Process Model

For simplicity it was decided to include the additional data required for simulation entirely in the serialized BPMN 2.0 model. Alternative approach would have been to keep additional data in a separate file but that would have been difficult to maintain. Since all BPMN elements have the *Documentation* property which is serialized as a string, the decision was made to store simulation data there.

The start event must contain information about how many instances have to be created, what is the arrival rate including distribution info, and which and how many resources are available. It is described in more detail in the API [10] of the simulator. To give a brief example, the simulation data for the start event in JSON format can be as follows:

```
{
  "arrivalRateDistribution":
  {
    "type":"normal",
    "mean":100,
    "stdev":20
  },
  "instances":1000,
  "resources":
  {
     "Clerk":10,
     "Manager":3
  }
}
```

The example above describes the arrival rate by normal distribution with mean 100 time units and standard deviation of 20 time units. 1000 instances of a process will be created and there are two types of resources available: 10 clerks and 3 managers. The simulator supports also fixed and variable arrival rate using uniform or exponential distribution.

For each task or event in the business process model that takes time to execute or uses some kind of resource, data for simulation has to be provided. The format has been described in detail in the documentation [10]. To give a brief example, a task that will be assigned to a resource “Clerk” and which takes 600 time units to complete using exponential distribution can be defined as follows:

```
{
  "durationDistribution":
  {
    "type":"exponential",
```

```
    "mean":600
  },
  "resource":"Clerk"
}
```

The resource has to be assigned for tasks only, but duration information is required for all catch events that are not thrown from the model itself. For example, the duration distribution info for a timer event that will be fired after 100 units of time has passed since the event was enabled, can be defined as follows:

```
{
  "durationDistribution":
  {
    "type":"fixed",
    "value":100
  }
}
```

2.2. Core of the Simulator

2.2.1. Process State

During the simulation, each element in the business process model has its own state per each process instance. The elements representing tasks can be *enabled*, *started*, *withdrawn* or *completed*. The element becomes *enabled* when it is discovered by the simulator and determined to be handled. When a resource is assigned to a task, it is said that the task has been *started*. After that "if" element gets completed without interruptions, the element is said to be *completed*, otherwise it got *withdrawn*, as shown in Figure 2.1.

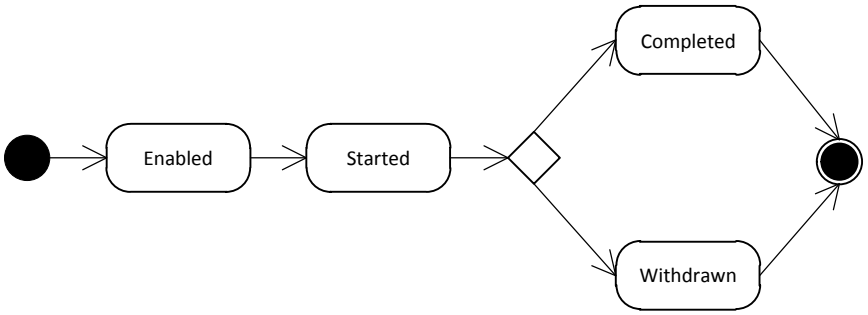

Figure 2.1. State Diagram of a Task in a Process Instance

Elements other than tasks do not have the state *started* because the only difference between the *enabled* and *started* states is whether a resource has been assigned or not. Resources

can be assigned only to tasks, and other elements can be *completed* after they are *enabled*. Figure 2.2 shows a state diagram of other elements than tasks in a process instance.

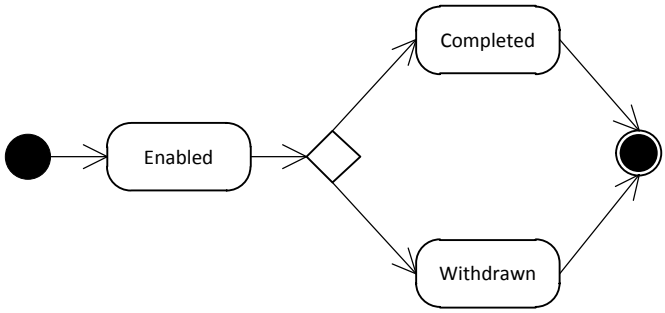

Figure 2.2. State Diagram of Other Elements than Tasks in a Process Instance

Every process instance that is being simulated has a state. Similarly to Petri Nets [19], enabling and completing elements can be described as a token game, where every completion of an element consumes tokens from its incoming flows and produces tokens to outgoing flows depending on the type of the element. Tokens will be passed along flows in the process model until all of them have been consumed, indicating that the whole process instance has been finished. To illustrate the concepts, let us consider the order handling process presented in Figure 2.3.

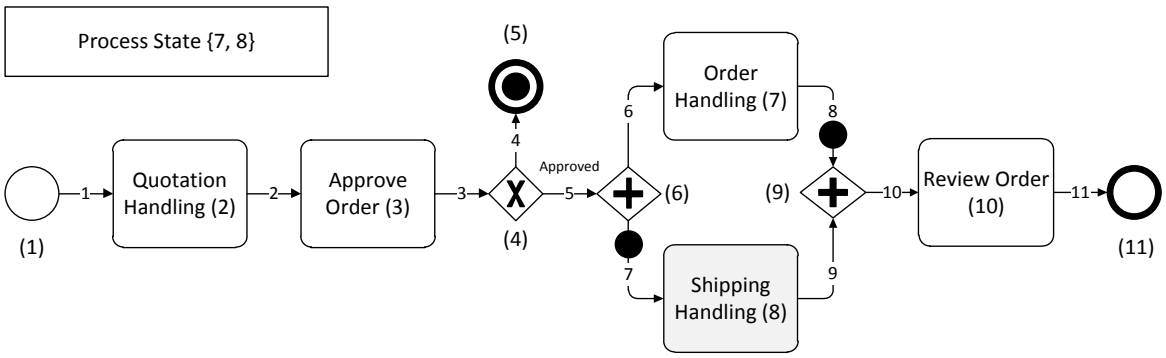

Figure 2.3. Order Handling Completed, Shipping Handling Started. State {7, 8}

Figure 2.3 shows a state of tasks "Order Handling" completed and "Shipping Handing" either enabled or already started. To ease the explanation, elements and flows have a numeric label. With these labels the process state can be denoted as {7, 8}, which means that a token is present on flow #7 and other one on flow #8.

If the task "Shipping Handling" takes much more time to complete than "Order Handling", then the state {7, 8} shown in Figure 2.3 will eventually occur. After an element in the model has been completed, the simulator has to determine the set of elements that get

enabled. In the state {7, 8} simulator cannot place a token on flow #10, because it has to wait for a token to arrive at flow #9. When "Shipping Handling" gets completed, the token is consumed from flow #7 and produced on flow #9. This results in the state {8, 9} and the And-join gateway (9) becomes enabled as seen in Figure 2.4. To detect efficiently which of the elements are the candidates for becoming enabled and which can actually be enabled, the simulator uses two auxiliary structures: the pre- and post-condition tables.

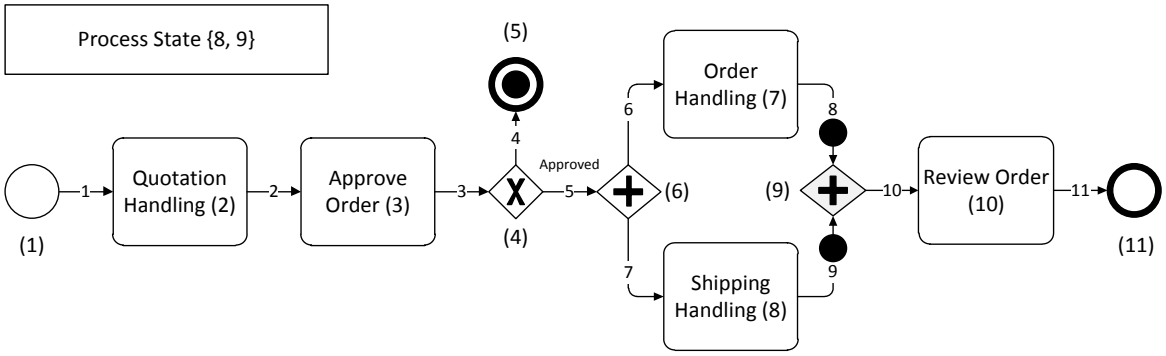

Figure 2.4. Order and Shipping Handling Completed. State {8, 9}

2.2.2. Post-Condition Table

The post-condition table is an association from one element to the set of its successor flows in the process model. In other words, the post-condition table is used to get all outgoing flows from an element that was just completed. In Figure 2.3 and Figure 2.4 all elements and flows were assigned numeric labels. Later unique numbers associated with elements and flows are referred to as indexes. The post-condition table for the process model in Figure 2.3 is provided in Table 2.1 below.

Element # (index)	Post-Condition (flows)
1	{1}
2	{2}
3	{3}
4	{4, 5}
5	{ }
6	{6, 7}
7	{8}
8	{9}
9	{10}
10	{11}
11	{ }

Table 2.1. Post-Condition Table

From the post-condition table it can be easily detected in which flows the tokens could be produced after an element has been completed. The tokens will be put into those flows to represent the new state of the process and the associated target elements need to be tested to see if they can be enabled. The pre-condition table introduced in the next section contains information about which process state is required to enable any element in the model.

2.2.3. Pre-Condition Table

The pre-condition table maps the element to the required process state to enable the element. The pre-condition table is built by traversing the business process model using the depth first search at the pre-processing phase of the simulation. When the depth first search reaches an element, the required state is updated by including the index of the discovered flow in the required state. This kind of approach produces similar pre-conditions for all types of elements, including gateways. The gateway type is checked later when determining if the element can be enabled by requiring the tokens on all incoming flows for And-joins; one token on any incoming flow for Xor-joins, split gateways, activities and events. Inclusive converging gateway (Or-join) is handled separately by another component and is thoroughly explained later in section 2.4.5. The pre-condition table for the process model in Figure 2.3 is provided in Table 2.2 below.

Element # (index)	Pre-Condition (flows)
1	{ }
2	{1}
3	{2}
4	{3}
5	{4}
6	{5}
7	{6}
8	{7}
9	{8, 9}
10	{10}
11	{11}

Table 2.2. Pre-Condition Table

In conclusion, the combination of pre- and post-condition tables allows determine efficiently which successor elements can be enabled given a state of a process instance and an element that has been completed.

2.2.4. Process Instance Creation

When the pre- and post-condition tables have been constructed, the instances of the business process can be created. Each process instance has a unique identifier; the state which will be updated during the execution; start and end timestamps; and at the end the trace of elements that were completed.

The first step of the simulation is process instance creation and enabling the start event for each generated case. The total number of cases to generate has to be provided as additional simulation-specific data which also contains arrival parameters. Arrival parameters (also known as arrival rate) specify the interval between two started process instances. For example, if the arrival rate is a constant c and the first instance is started at time t, then the second instance will be started at time $t + c$, the third instance at time $t + c + c$, and so on. Of course, the arrival rate is usually random based on some duration info but this does not change the logic of the completion time calculation.

The generated start events with their start times will be added to the priority queue [20] of elements to be completed, which will be discussed in the next section of the thesis.

2.2.5. Queue of Elements to Complete

All started tasks and other enabled elements will be added in the global event queue for completion. Each element in the queue will have a completion timestamp that means priority for the queue. When an element is added to the "to be completed" queue, the completion time has to be provided as well. The completion time is calculated based on the current type of the system and the execution time (or duration) of the element.

The event processor component of the simulator takes the item with the highest priority from the queue. The item selected by priority will be the next to be completed in the chronological order. Processing an item from the event queue might cause new elements to be added to the queue or a process instance completion or withdrawal. The queue will be processed until it gets empty meaning that all cases have been finished and simulation is completed. The core components and the main interactions between them are presented in Figure 2.5.

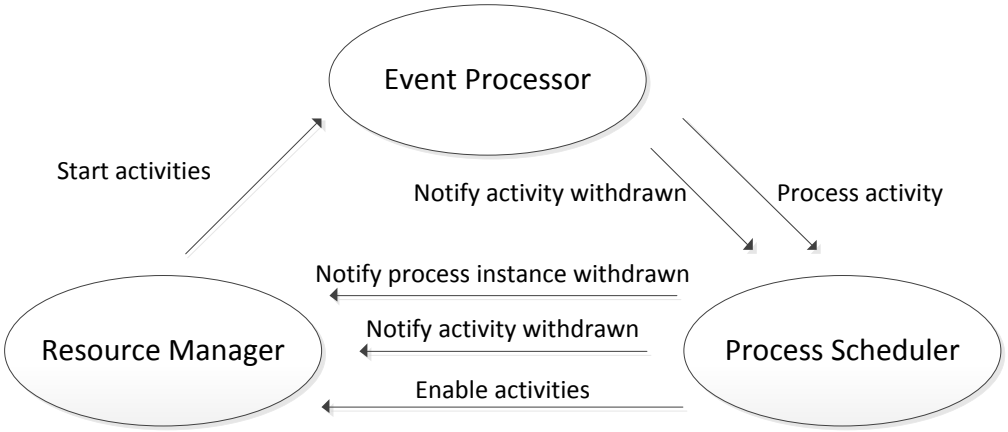

Figure 2.5. Main Interactions between the Core Components

2.3. Processing Elements

Element processing is done by the process scheduler component of the simulator. Element processing starts when the event is taken from the priority queue and submitted for completion which, in general consists of the following steps:

1. Updating the state of the process
2. Handing resources that became available
3. Doing additional handling based on the type of element
4. Discovering the elements to be enabled
5. Enabling the discovered elements or completing them immediately

In the first step all tokens on the incoming flows of the element being processed are consumed (cleared from the state) and tokens will be placed on all outgoing flows of the element.

If resources were allocated by the task element, those resources must be made available at this point of time. Resource management is handled separately by the resource manager component. Resource management is discussed in general in section 2.5.

Some elements of BPMN require additional handling besides just completion which depends on the type of element being completed. For example, a sub-process contains another independent process in it, which has to be started separately, and when the sub-process gets completed, its container element in the parent process can be completed. The elements that require special handling are called advanced constructions of BPMN and those are explained later in section 2.4.

The last step in the process is the new element discovery using the pre- and post-condition tables explained in section 2.2. All discovered elements will be enabled or completed. All gateways discovered can be completed immediately because gateways do not use any resources and their duration is always zero. All tasks are usually assigned to a resource, their execution takes some time, and events usually occur after some amount of time. Therefore, tasks and events need resource management and queuing logic to be applied.

The BPMN elements are divided into two categories based on the complexity of handling them in the simulator: basic and advanced constructions. The following elements have been considered as basic ones: start and end event; task; exclusive (Xor) and parallel (And) join gateways. When the simulator reaches any of these activities, no additional or special handling is required. All elements supported by the simulator are presented in Figure 2.6.

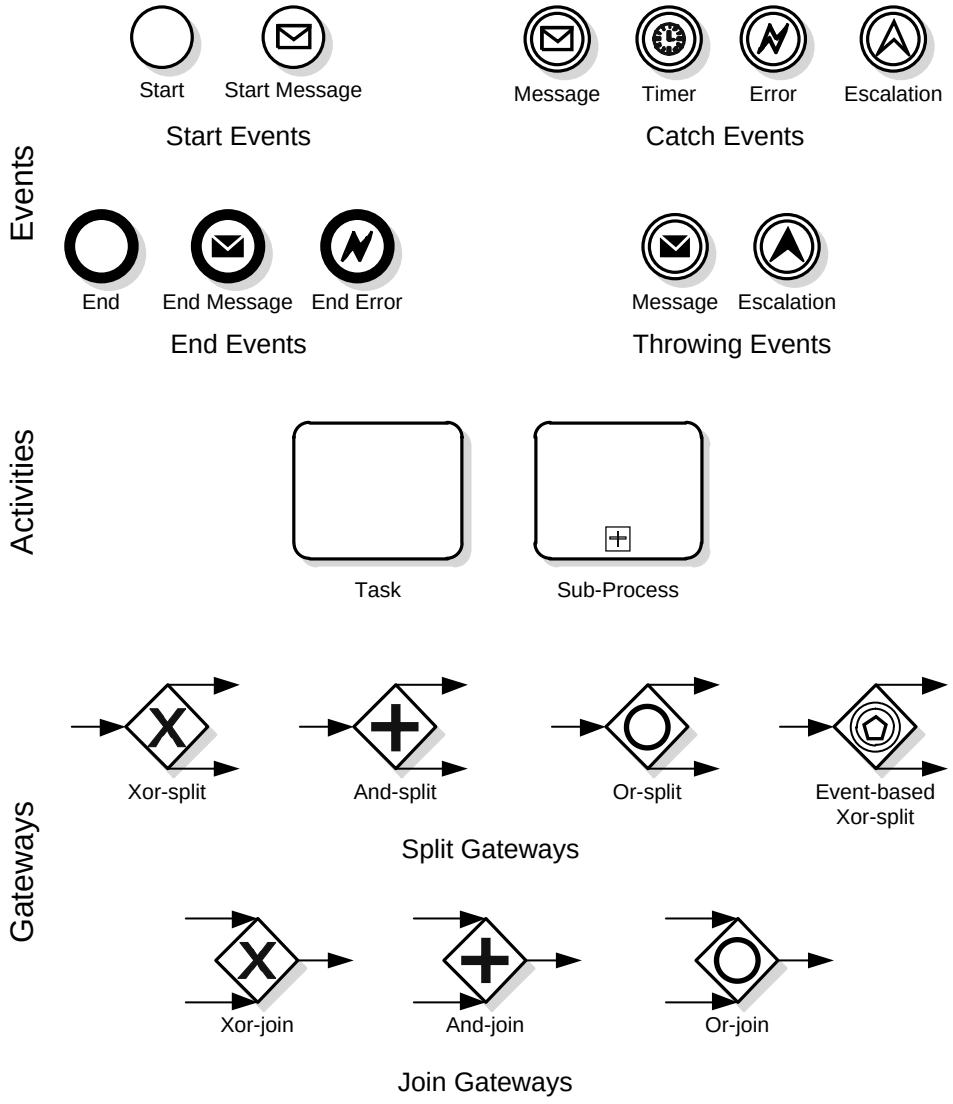

Figure 2.6. Supported Elements of BPMN

Detecting whether any of the basic elements can be enabled after completing the previous element is relatively easy as well. Let the process state be S_p and the pre-condition of an element i P_i. Any basic or advanced element other than And-join gateway can be enabled if $S_p \cap P_i \neq \emptyset$ and And-join can be enabled if $S_p \cap P_i = P_i$. In other words, And-join can be enabled if the process state contains tokens on all incoming flows and any other element is enabled if at least one token exists on any of its incoming flows. Elements which can be enabled in a given state represented by tokens on flows are shown in Figure 2.7. However, the Or-join handling is more complex and described later in section 2.4.5.

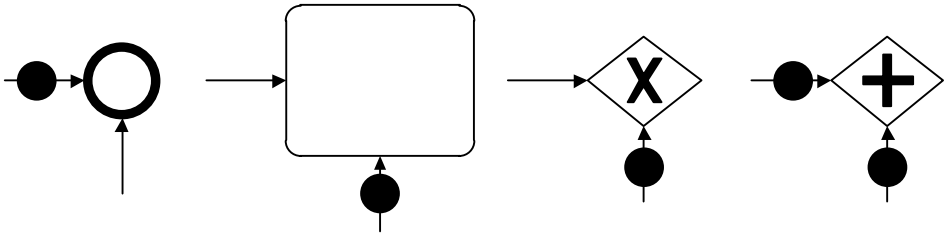

Figure 2.7. Enabled Elements

2.4. Processing Advanced Constructions of BPMN

2.4.1. Branching with Exclusive and Inclusive Split Gateways

Depending on the gateway type either all, only one or any number of paths will be selected and activated in a process instance. It means that each outgoing flow of an exclusive (Xor) or an inclusive (Or) gateway must be assigned a probability from 0% to 100% of taking this path in a process instance. Branching probabilities are not applicable for parallel (And) split gateways that always activate all of their outgoing flows.

The simulator generates random numbers and then checks the branching probabilities assigned to flows. Exactly one outgoing flow will be selected for Xor-splits and each flow from an Or-split is evaluated separately. Due to randomness it might happen that the Or-split does not select any outgoing flows. In order to solve this issue, outgoing flows for an Or-split in a particular process instance would be determined iteratively until at least one of them is selected. Or-joins in particular are described in section 2.4.5.

2.4.2. Sub-Processes

In BPMN there are two types of sub-process elements (Figure 2.8): collapsed and expanded sub-processes. In process modeling the sub-processes are often used to make larger

models easier to understand by grouping the inner detailed process together and hiding the details from the parent process. However, in the simulator the collapsed and expanded sub-processes are handled similarly provided that all sub-processes are included in the BPMN 2.0 file.

For both types of sub-processes the parent process contains a single element which corresponds to the child-process. The elements of the child process form a new process model which will be linked to the parent container (the respective sub-process activity in the parent process). Actually, in BPMN 2.0 an element of a collapsed sub-process just refers to another process model by its identifier and the element representing an expanded child process is a container (parent node in the concept of XML) for all its child elements. More information related to that can be found from the BPMN 2.0 specification [6]. As already mentioned, the simulator handles both cases similarly and the two constructs are distinguished only in the model parsing phase.

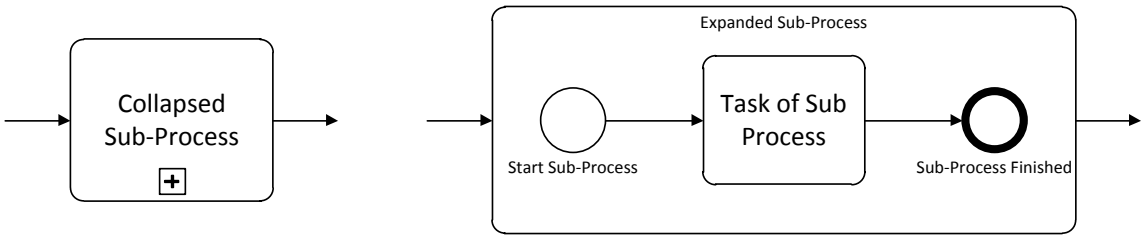

Figure 2.8. Collapsed and Expanded Sub-Processes

In the parsing phase, a separate process model will be constructed from all elements in the sub-process and linked to the parent sub-process element. When a sub-process element is enabled, a new process instance with the chain of elements in the sub-process will be created and the start event of it will be added to the event queue for completion. The rest of the elements in the sub-process will be processed on a regular basis. When a child process gets completed or withdrawn, the parent container element will be notified of completion and the parent process can proceed. Withdrawal of a child-process causes all its started elements withdrawn as well.

2.4.3. Intermediate Events

The intermediate event in a business process model indicates where something happens in the middle of the process. According to the BPMN 2.0 specification [6] intermediate events can be used to:

- Show where the messages are expected or sent within the process;
- Show where delays are expected within the process;
- Disrupt the *normal flow* through *exception*;
- Show extra work needed for *compensation.*

There are twelve types of intermediate events in total available in BPMN 2.0 [6], but within the scope of this project the following are supported: *none*, message, timer, escalation, and error event. The intermediate events can be thrown from a normal flow (e.g. send a message to another process); or caught in a normal flow (e.g. receive a message from another process or wait for some time to represent a delay) or by a boundary catch event (described in section 2.4.4) of an activity. A simplified process of the news life cycle containing two intermediate events (circles with double thin lines) in a normal flow of a fictional news agency is shown in Figure 2.9. The "More information received" event represents a message being waited for and the "30 Days" represents a time delay of 30 days that has to pass after the process can move on. Boundary event handling is discussed later in section 2.4.4.

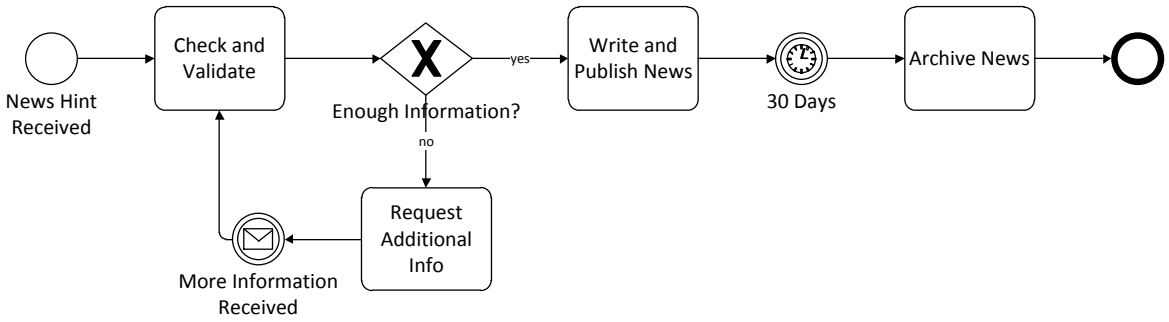

Figure 2.9. Intermediate Message and Timer Events in a Normal Flow

Catching intermediate events in a normal flow without exclusive event gateway can be handled easily by the simulator. When an intermediate catch event in a normal flow is enabled, a message will be generated and added to the queue with the completion time generated based on the additional simulation data of the event.

Throwing intermediate events is explained along with boundary events in section 2.4.4. However, often there is the need to model a scenario where multiple events need to be triggered, but only the first of them has to be completed. In BPMN, situations like this can be handled with exclusive event-based gateways.

To illustrate the concept of exclusive event-based gateway, the timeout handling is added to the “More information received” event (a star symbol inside a diamond) as seen in Figure 2.10. As the name of the gateways already states, events followed directly by the gateway are exclusive, meaning that only one of them has to be completed without waiting for the rest to be triggered. When the simulator reaches the gateway, subsequent events with generated execution time will be added to the events to be completed queue. When one of those events gets completed, others must be withdrawn. Simulator handles this by keeping the reference to other exclusive events created by the same preceding gateway. When the first one is completed, others will be marked as withdrawn to be skipped later. Withdrawn events are still kept in the queue because the event-based gateway handling does not end at this point yet.

Now the loop handling with event gateways is considered in the context of the same news agency process model. When the event gateway was enabled, two exclusive events to be completed were generated. For example, it is assumed that for a particular process instance the “More information event” will arrive exactly in 2 days and 5 minutes (generated randomly based on the duration distribution information) and the “Send Reminder” event takes exactly 10 minutes to complete. In this case, both elements would be added to the queue and processed in the following order: the “2 days” timer event, the “More information received” message event (withdrawn though and skipped this time) and the “Send reminder” task. While the simulator was handling the timeout, a message was actually received. When the “Send reminder” task gets completed, the event gateway will be enabled again. Now, it would not be correct to register the new message to be received again, since this message actually has already arrived. To overcome this problem, simulator uses a registry of arrived messages. In the first iteration, when the “More information received” event was skipped, it gets added to the registry. In next iteration, if a message to be requested already exists in the registry for the same process instance, then the message will be “restored” and removed from the registry. Completion time of the message event will be set to the current system time which results the event to be completed immediately. In other words, when considering the same example, duration of the “More information received” message event was 0 in the second iteration and it was completed at the time when the event gateway was enabled.

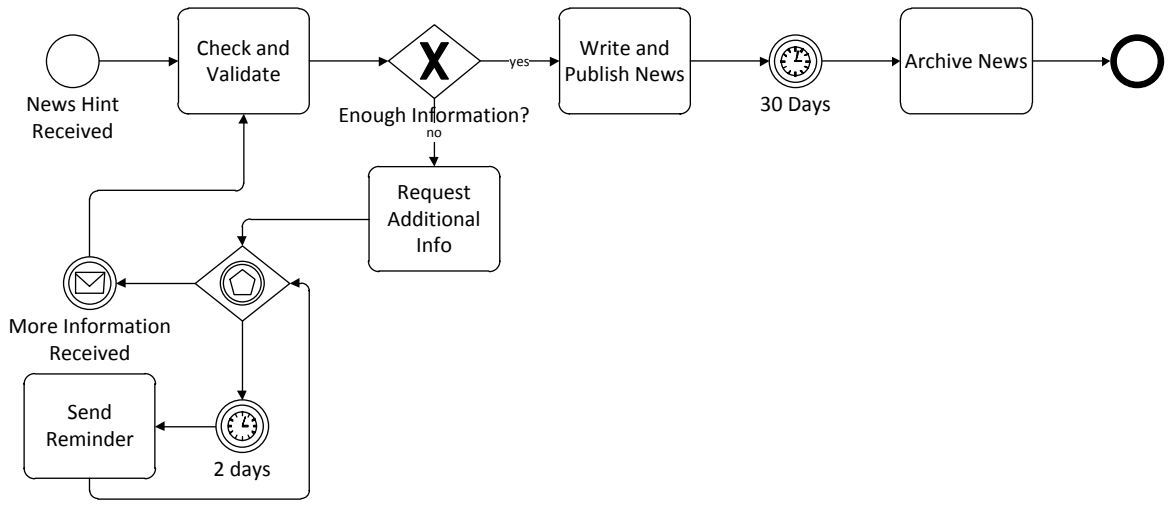

Figure 2.10. Intermediate Events with Exclusive Event-based Gateway

2.4.4. Boundary Catch Events

Intermediate events can be caught by the events attached to the boundary of an activity. Such events are called the boundary catch events and they are used for exception or message handling. For example, the timer event on the boundary of event would be triggered if the duration of a task is longer than the specified timer. Another common usage of the boundary event is with sub-process error handling. If a sub-process ends with some error (exception), this error could be caught by a boundary catch event and the exception flow would take place instead of the normal flow. See Figure 2.11 of an article procurement sub-process which may throw two intermediate events and Figure 2.12 of an enhanced order fulfillment process inspired by [7].

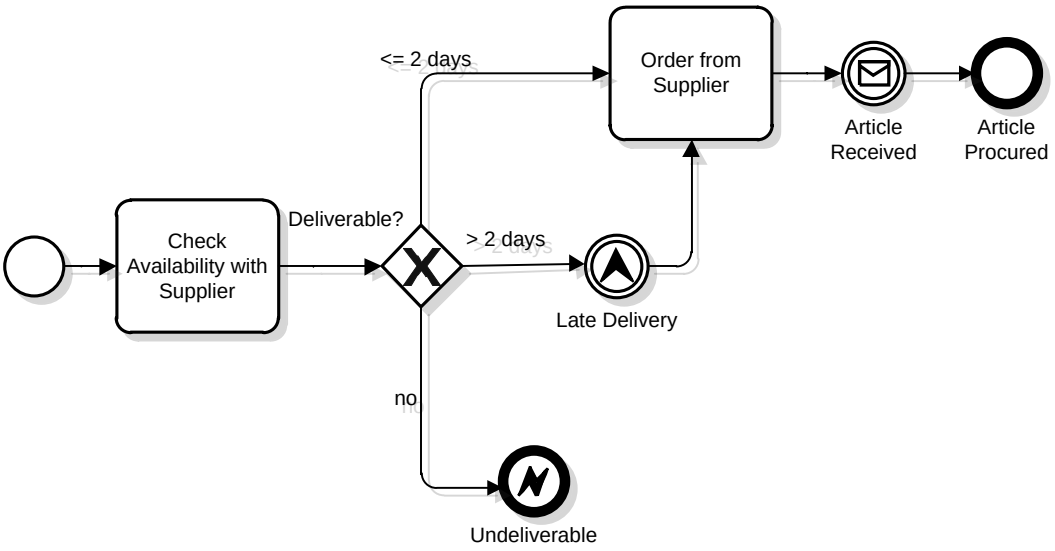

Figure 2.11. Procurement Business Process Model

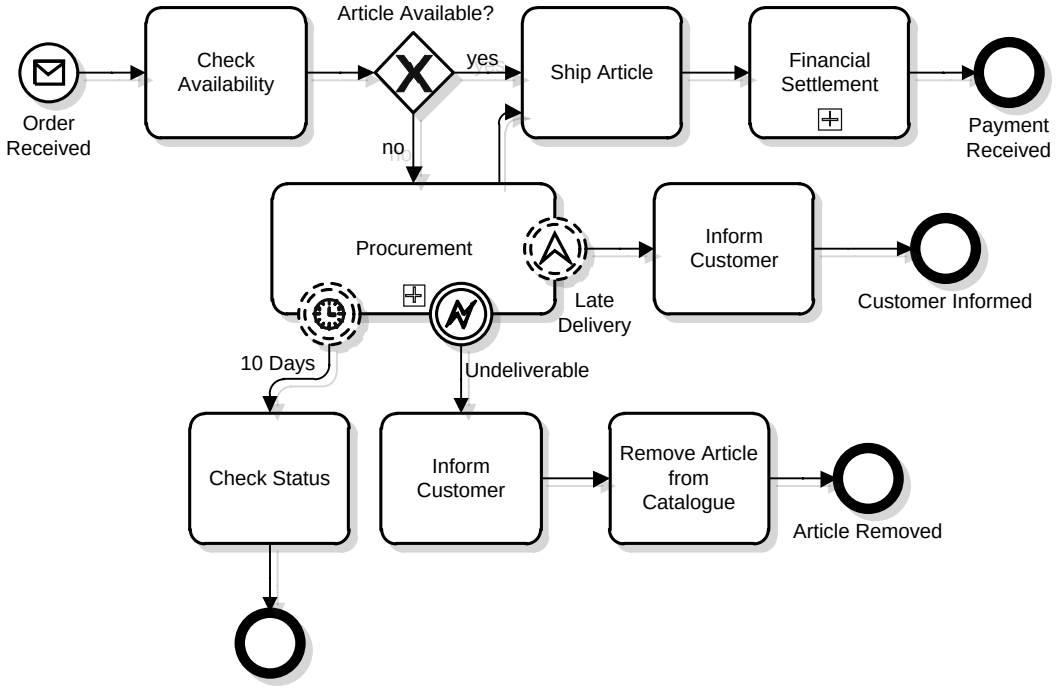

Figure 2.12. Order Fulfillment Business Process Model

In Figure 2.11 there are two new elements: an error end event “Undeliverable” and a throwing escalation event “Late delivery”. The error end event is a special type of end event which has to have a name assigned to it (“Undeliverable” in this example). The name of the event can be used in the boundary error catch event for specifying the exception to be handled. An escalation identifies a business situation that a process might need to react

to [6]. Similarly to the error events, escalations can be identified and handled by the event name. However, throwing an escalation event like in Figure 2.11 does not end the process.

In Figure 2.12 three types of boundary events have been defined for the "Procurement" sub-process: timer, error and escalation. The boundary events can be either interrupting or non-interrupting. Interrupting events have a solid and non-interrupting events have a dashed boundary. The difference is that if a non-interrupting boundary catch event is executed, it is handled in parallel with the normal process flow. On the other hand, interrupting catch events are executed instead of the normal flow which also results the associated activity being withdrawn. The error event in particular always interrupts the common path of the process model.

When an element with the boundary timer event gets enabled, the timer event with their durations would be placed to the event queue. If the attached timer event is marked as interrupting, then similarly to the event-based gateways the activity and the timer events must be mutually exclusive. When one of interrupting timer events fires or the associated element gets completed, others will be marked as withdrawn so they will not be processed.

If an intermediate event is thrown from a process instance, the simulator has to check whether there is a catch event defined in any of the parent processes. The simulator maintains a list of catch events by their name. Now, when an event is thrown, parent processes will be checked for the catch event with the same name and if it exists, the catch event will be completed immediately.

2.4.5. Inclusive Converging Gateways (Or-joins)

An inclusive converging gateway (Or-join) is an element in the process where several paths merge. While exclusive join gateway (Xor-join) waits for one token to arrive from any of its incoming paths and parallel join gateway (And-join) waits for tokens to arrive from all incoming flows, Or-join waits for all those branches which contain tokens. If at some point in time it is clear that no tokens will arrive along the given path, the gateway will not wait for that branch anymore. Figure 2.13 presents one process model with an Or-join from [8]. Unlike other routing constructs, the Or-join has a non-local semantics: in order to determine whether or not an Or-join is enabled, it is not sufficient to examine the presence of tokens in its immediate vicinity. Instead, enabling an Or-join may depend on the presence or absence of tokens in places far away in the model [8].

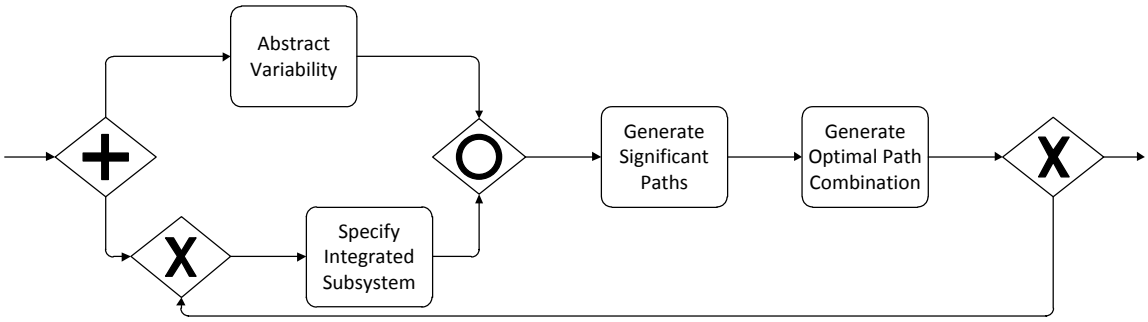

Figure 2.13. Process Fragment with an Or-join from [8]

The first time the fragment of a process in Figure 2.13 gets executed, the parallel gateway will result in the "Abstract variability" and the "Specify integrated subsystem" tasks being enabled and the Or-join has to wait for both of them to be completed. After that the tasks "Generate significant paths" and "Generate optimal path combination" are completed in sequence. If the Xor-split takes the bottom path which enables the "Specify integrated subsystem" task, then the second time around, the consecutive Or-join has to wait only for this latter task to be completed.

Or-join handling in the simulator is particularly inspired by an approach introduced in [8]. In short, when the first token arrives on an incoming flow of an Or-join, it needs to be checked if there are tokens in any other incoming path(s). If there are not, then the Or-join can be enabled, but if there is at least one token, then the Or-join has to wait for it. It might happen that if the only token that an Or-join was waiting for moves out from the Or-join branch (e.g. caused by following Xor-split), then the Or-join needs to be enabled at this point. To achieve this, a registry of Or-joins waiting for elements to be completed is introduced. If at some point in time an element which has been registered in the Or-join registry in a process instance gets completed, the Or-joins waiting for this element will be re-evaluated again. The Or-join re-evaluation in this phase might result in the Or-join being enabled or the Or-join waiting for the next element to complete. The challenge at this point is to detect efficiently if the tokens exists in incoming branches and for which elements the Or-join has to wait to be re-evaluated for enabling.

The process state is retained as a token on active flows. For all Or-joins and their incoming flows similar states (masks) can be generated that contain only the flows from incoming branch in the pre-processing phase of the simulation. Such a mask can be easily compared to the process states to determine the presence or absence of tokens in Or-join branches. If there is at least one common flow, the Or-join has to wait for the destination element of the

flow to be completed. Figure 2.14 shows two incoming branches for the Or-join: branch A containing flows 2-3 and branch B containing flows 4-10.

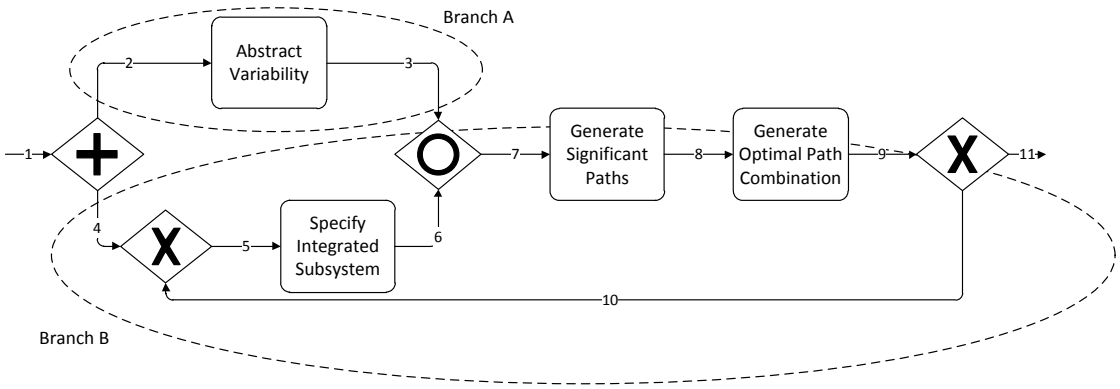

Figure 2.14. Branches of an Or-join

The remaining part of Or-join handling is generating the masks for incoming branches. In this project the following algorithm is used:

1. For all coming flows to the Or-join O_i do:
    a. Let the mask of branch *i* for Or-join O be $Prec_i$
    b. Traverse path back to the start event or to the Or-join O_i along unvisited incoming flows and include flow being visited in $Prec_i$
2. Update discovered masks: remove all flows which are common for all branches

Step 2 is required to support the vicious circles in well-structured process models described in [5] and [8]. In short, a vicious circle is a situation where the two Or-joins are waiting for each other and a *deadlock* occurs. A vicious circle in a well-structured model from [8] in Figure 2.15 can be resolved by the proposed algorithm. The approach requires the business process model to be *safe*. Notion of safeness is explained in section 2.6.

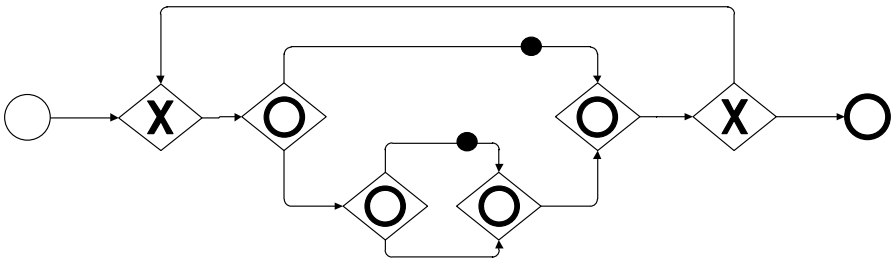

Figure 2.15. A Vicious Circle in a Well-Structured Graph

2.5. Resource Management

For the simulation purposes additional information about resource availability and assignment has to be provided. For example, if an insurance company has 5 clerks and at some point in time 6 cases come in together, then only 5 of those can be processed in parallel. The remaining case will be queued until any of started tasks gets completed and a clerk becomes available. As discussed in section 2.1.2, each available type of resource with quantity and resource usage for tasks has to be defined as additional simulation-specific data. If resources have been assigned to tasks in the business process model, the simulator can do resource management that involves:

- Resource assignment for instances of tasks being processed;
- Resource availability check with queuing of tasks if desired resources are busy;
- Making available the associated resource when a task gets either completed or withdrawn;
- Making available all resources used by a process instance which gets withdrawn;
- Assigning a resource to the instances of tasks in the waiting queue when the desired resource becomes available.

The current version of the simulator supports simplified resource management. Currently, only one resource can be assigned to a task and the completion time does not depend on the assigned resource. For example, in a business process a task can be assigned to either a junior or a senior clerk depending on their availability. If a senior clerk was assigned, the duration of the task is 50% less compared to a junior clerk. From the design perspective this is not a limitation and advanced resource management can be implemented in the future.

Resources in the simulator are managed by the Resource Manager component. At first, for each defined resource it is maintained how many instances are available at any point in time. When the Process Scheduler component discovers a task to enable, this task will be submitted for resource management. If the defined resource for this task is available, one instance of the desired resource will be assigned and the activity will be enabled. Otherwise, if defined resources are busy, the task will be added to the “first in first out” (FIFO) queue [9] of the resource. If a resource becomes available, the first task from its waiting queue will be enabled. In order to maintain available resources and waiting queues and in

order to enable queued tasks, the resource manager always has to be notified when a task gets either completed or withdrawn; or a process instance gets withdrawn.

2.6. Safe Models

According to [5] a process state is *unsafe* or has a *lack of synchronization* if there is a flow which carries more than one token, otherwise it is *safe*. If a business process model may cause an *unsafe* state, the model is *unsafe* and otherwise it is *safe*. An example of an *unsafe* model is shown in Figure 2.16.

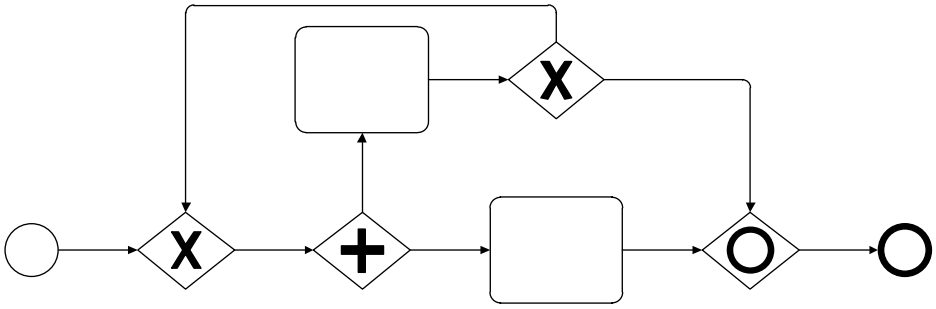

Figure 2.16. *Unsafe* Model

Only *safe* models can be processed by the simulator. Semantics provided for the Or-join in [5, 8] have the same requirement for business process models. The constructs similar to the one shown in Figure 2.16 can be sorted out by static analysis as a modeling error.

2.7. Summary

In this chapter the entire simulation process has been described thoroughly starting from the requirements for models that can be simulated, how a simulation is conducted, how resources are managed and how various BPMN elements are handled. It has been shown that even advanced constructions of BPMN like sub-processes, several types of intermediate events and Or-joins can be supported by the simulator engine using the new proposed approach.

3. Architecture of the Simulator

In this chapter the architecture of the simulator is described and illustrated with some class diagrams. The classes and their relations are introduced in the order of when they are used in the simulation process – starting from the `BPSimulator`, a class that initializes BPMN 2.0 parser and other components used later in the simulation process. Section 3.2 describes the core components of the simulation and in the last section of the chapter, the helper components are presented that only encapsulate some very specific business logic (e.g. gateway path selection and random number generation).

3.1. Initialization of a Simulation

3.1.1. Creation of Objects Representing the Model

The entry point to the simulation process is the `BPSimulator` class. `BPMN2Helper` contains the BPMN 2.0 parsing logic and provides the methods for the `BPSimulator` to access the business process model. From all nodes, representing BPMN elements, and all token flows, the pre- and post-condition tables will be built like described in section 2.2; the list of `Activity` objects is created; and the component for Or-join handling is initialized based on the business process model. The main components in the initialization phase are shown in Figure 3.1.

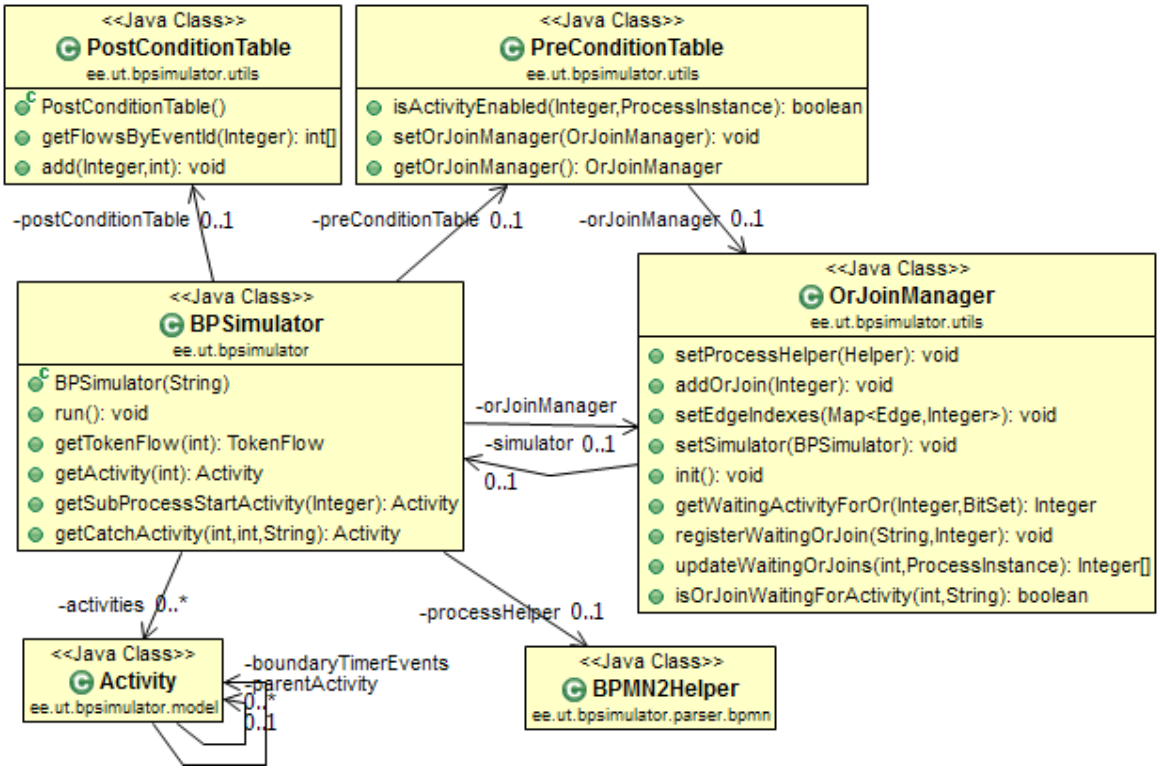

Figure 3.1. Classes Used at the Initialization Phase

The instance of an `Activity` represents an element of the business process model. An `Activity` can be a type of task, a sub-process, a gateway or an event. In addition, the type of the gateway or event will be assigned to the instance depending on the type of the element. As shown in Figure 3.1, `Activity` has two self-relations. Both of them are used in the boundary event handling so that one of them defines the parent activity in case of a boundary event and the second one defines boundary events in case of a task or a sub-process. Also, an instance of a `Resource` may be assigned for each activity. Figure 3.2 shows the relations of the `Activity` class.

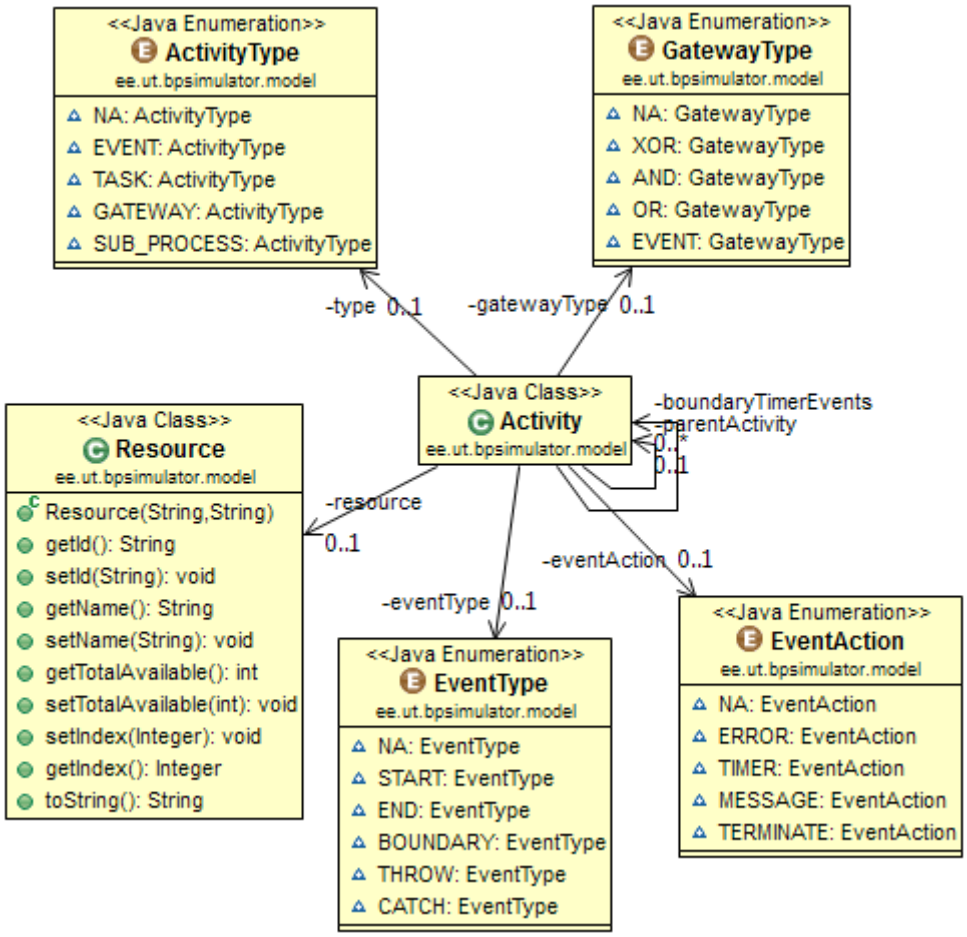

Figure 3.2. `Activity` Class with Relations

3.1.2. Case Creation

When the internal structure representing the business process model has been built, the process instances will be generated and the start events of those will be put to the event queue for processing as discussed previously in section 2.2.5. An instance of a process (case) is represented by a `ProcessInstance` class and an instance of a BPMN element in particular process instance is represented by a `ProcessActivity` object. Figure 3.3 shows relations with the `ProcessInstance` class. Similarly to the `Activity` class, `ProcessActivity` has two self-relations for the boundary event handling. The `processInstance` relation indicates the instance of a process which owns a particular activity. If the `ProcessActivity` object represents a sub-process in BPMN, then, as discussed in section 2.4.2, the sub-process will be handled in a separate process instance which will be referenced by the relation `handlingProcessInstance`.

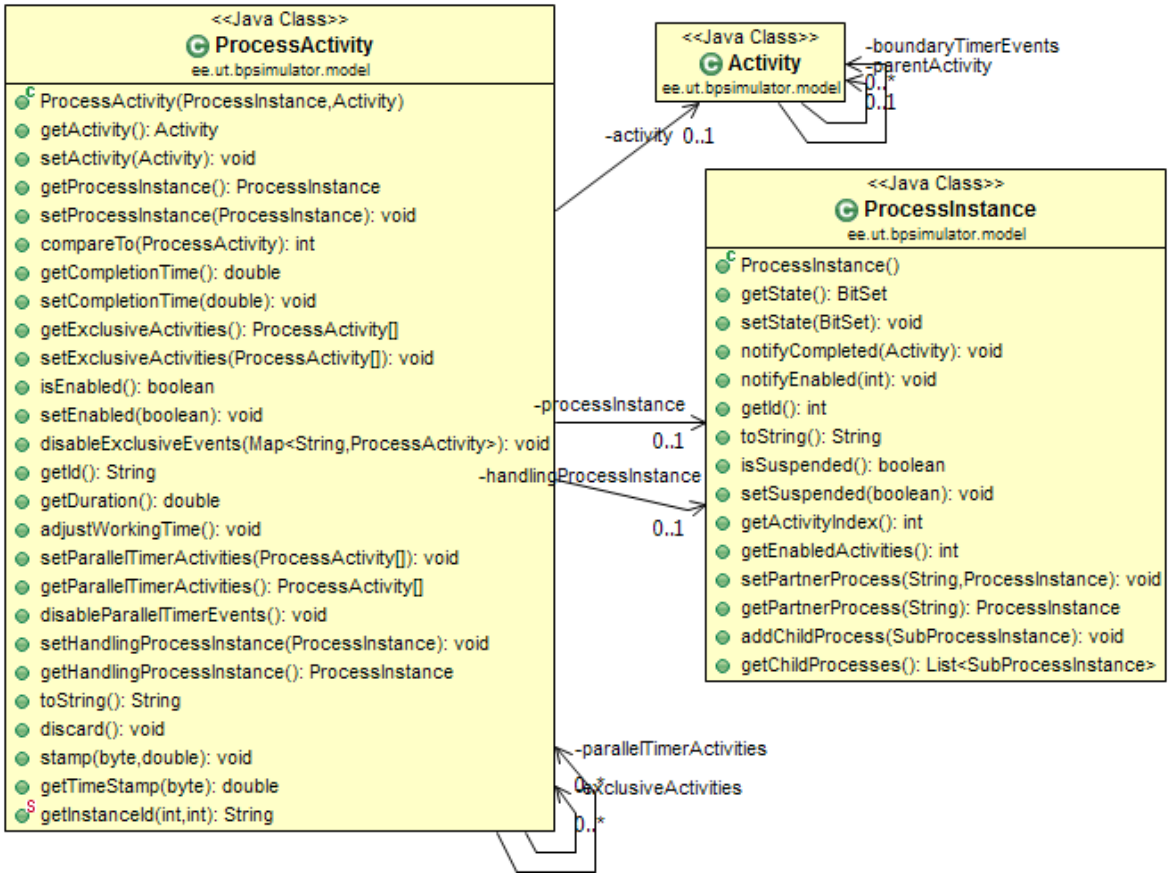

Figure 3.3. `Activity`, `ProcessActivity` and `ProcessInstance` Classes

3.2. Core Components

Before the simulation process starts, the process instances have been created and the start events of those have been added to the event queue for processing. The classes and interfaces which conduct the simulation process by processing the event queue or completing, enabling and starting the activities are considered as the core components. In this thesis those components are interfaces `IEventProcessor`, `IResourceManager`, `IProcessScheduler` and classes implementing these correspondingly `EventProcessor`, `ResourceManager` and `ProcessScheduler`. Figure 3.4 shows the main interactions between those components. An object of a `ProcessActivity` class is passed through core components. At first, a `ProcessActivity` (like a start event or a task) to be completed next is taken from the event queue by the event processor. Then the element will be given to the process scheduler component for completion. The process scheduler will update the process state and discover consecutive elements in the business process model that have to be enabled. After that an array of elements to be enabled is passed to the resource manager. The resource manager will check if the resources are required and available for

each given task. Finally, the tasks, for which resources are not available at this point of time, are queued and the remaining elements are passed to the event processor, which adds those to the event queue. This is the lifecycle of a task instance in the simulator. After that the event processor takes the next element to complete from the event queue and the whole process is repeated until the event queue is empty.

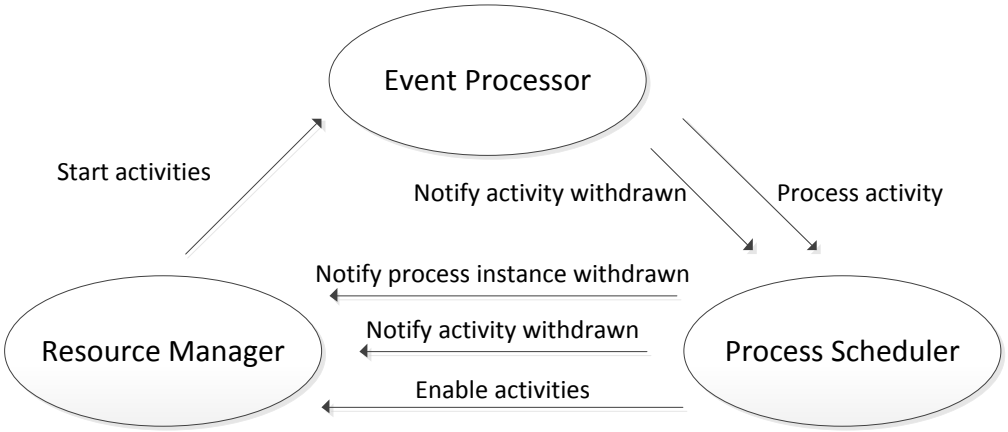

Figure 3.4. Main Interactions between the Core Components

The previously described lifecycle handles only the basic flow of activities. As shown in Figure 3.4 there are actually more interactions between the core components. Other interactions occur mostly while processing advanced BPMN 2.0 constructs discussed in section 2.4. For example, if an interrupting boundary timer event of a sub-process activity is completed, the associated process instance, including its started activities has to be withdrawn. Withdrawal of an activity may result in withdrawal of other activities or starting them in case where resources become available.

Figure 3.5 shows the class diagram of the main components. Additionally, it is shown how the core components are related to the `BPSimulator` class, which is the simulation entry point as explained in section 3.1. Also, while the activities are being enabled, started and completed, those actions must be tracked in order to create the audit trail entry logs for process mining framework for calculating key performance indicators or for simply showing the status of the simulation. Such a tracking component can be built by implementing the `IProcessLogger` interface whose relations have been shown in Figure 3.6. In this thesis we have implemented the mining XML (MXML) and console loggers using `MxmlLogger` and `ConsoleLogger` classes.

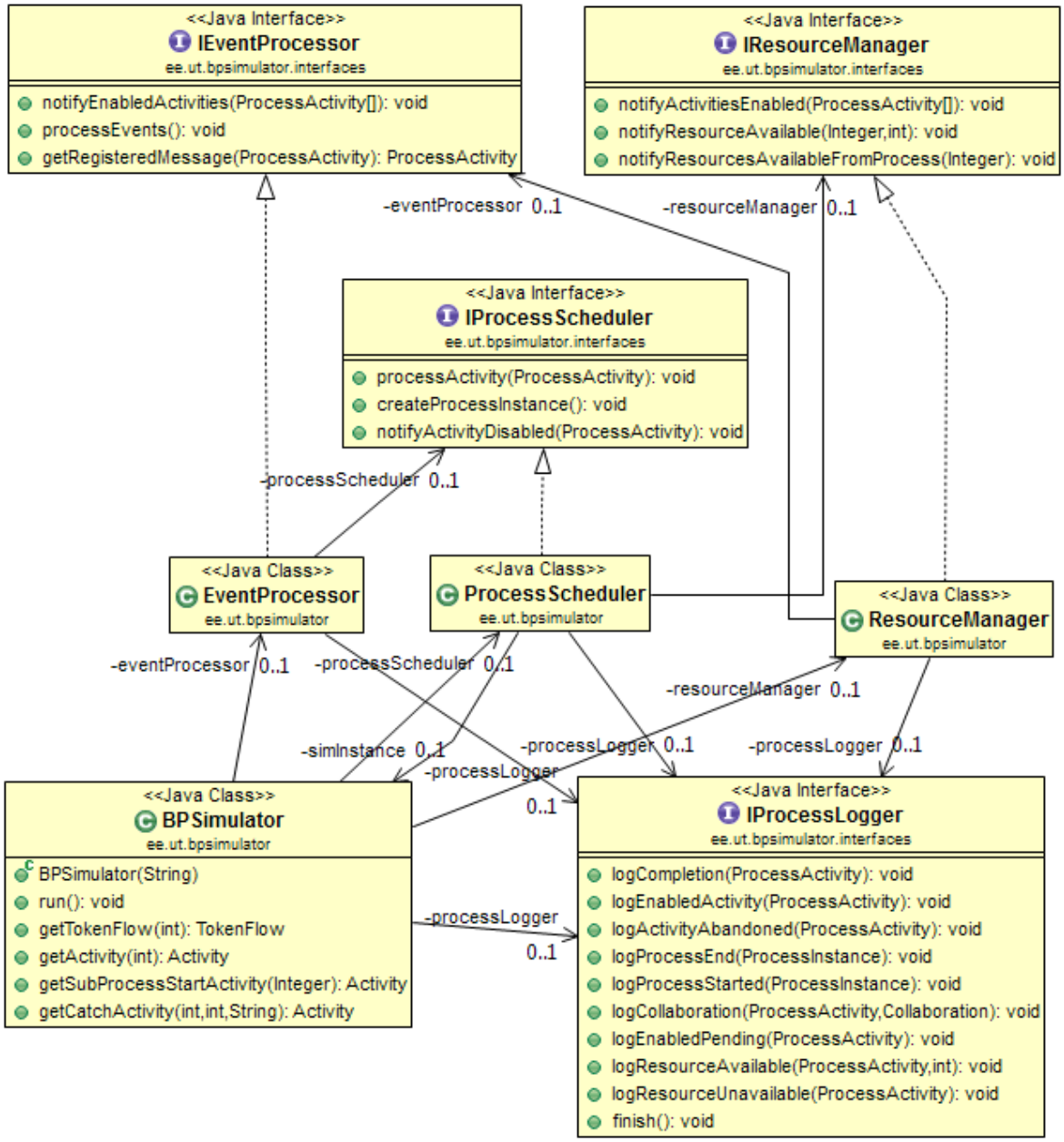

Figure 3.5. Class Diagram of Core Simulator Components

3.2.1. Process Scheduler

The process scheduler is the component which implements most of the simulation business logic and is represented by the `ProcessScheduler` class. As already discussed, the corresponding class is closely connected to other core components. The main purpose of the component is to:

- Complete activities;
- Maintain states of process instances;
- Discover activities to enable using the pre- and post-condition tables;

- After the completion of an element, provide a list of enabled activities to the resource manager.

3.2.2. Resource Manager

Resource management was already introduced in section 2.5. The corresponding class in the simulator is the `ResourceManager` whose relations to other components can be seen in Figure 3.6. The resource manager is responsible for starting all activities by providing the list of them to the event processor.

There are three types of triggers in the simulation process that may indicate that some activities can be started when:

- A task has resources available or an event is enabled;
- A started task has been withdrawn (e.g. by an interrupting intermediate boundary event);
- A process instance with started activities has been withdrawn (e.g. an instance of a sub-process).

The triggers listed above can also be seen in the interaction diagram in Figure 3.4.

3.2.3. Event Processor

The event processor is represented by the class `EventProcessor`. The main purpose of the component is to maintain the queue of events to be processed. In addition, the class has to handle the cases where one or more events are mutually exclusive and one of them fires. This is a common scenario for events connected by an event-based gateway or for interrupting boundary events. If an event to be processed is withdrawn in the queue, then it must have been caused by the completion of a previous event.

Events to start are provided by the resource manager and the event to be processed next is passed to the process logger. Additionally, events to be added to the queue can be "discovered" by the component itself. This happens if an activity with boundary timer events is started. In this case the boundary timer events must be started as well and those which have been marked as interrupting ones must be set as mutually exclusive. The simulation process ends when the queue of events becomes empty.

According to the intermediate message event handling discussed in section 2.4.3, all withdrawn events representing arrived messages should be still recorded as being potentially is-

sued and requested later. Therefore, the component is also responsible for maintaining the registry of arrived messages.

Moreover, the event processor also maintains the global system clock. Since the events are processed in the ascending order of their completion times, global system clock is changed to the completion time of the next event to complete.

3.2.4. Process Logger

Loggers implementing the `IProcessLogger` interface can be plugged into the simulator to track actions in the simulation process. The logger is notified of actions by the event processor, the resource manager and the process scheduler as shown in Figure 3.5. Figure 3.6 below shows the three classes that currently implement the interface:

- `ConsoleLogger` – logs actions to the console;
- `MxmlLogger` – logs actions in the mining XML (MXML) format to file;
- `ComplexLogger` – can consist of multiple logger objects and delegates actions to inner objects.

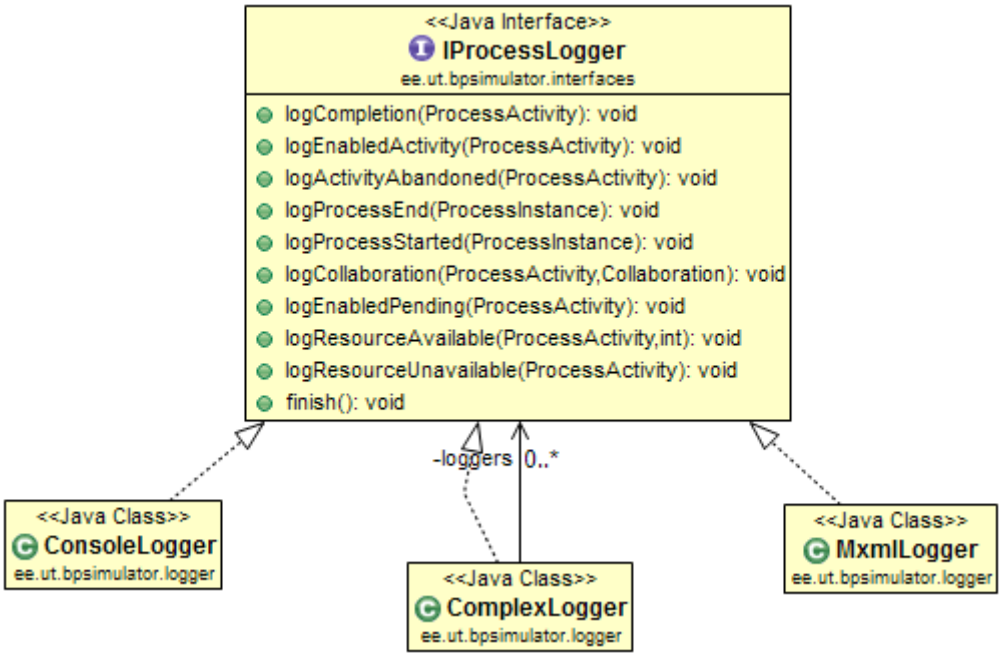

Figure 3.6. Class Diagram of Loggers in the Simulator

In addition to the currently implemented loggers, more of them can be easily added in the future to store logs in database or in any format supported by tools for statistic analysis, for example. Moreover, KPIs could be calculated by implementing the logger interface.

3.3. Helper Components

Smaller classes used to encapsulate some very specific logic for simulation process are briefly described in this section of the thesis. Figure 3.7 shows four classes that have been used by the main components of the simulator.

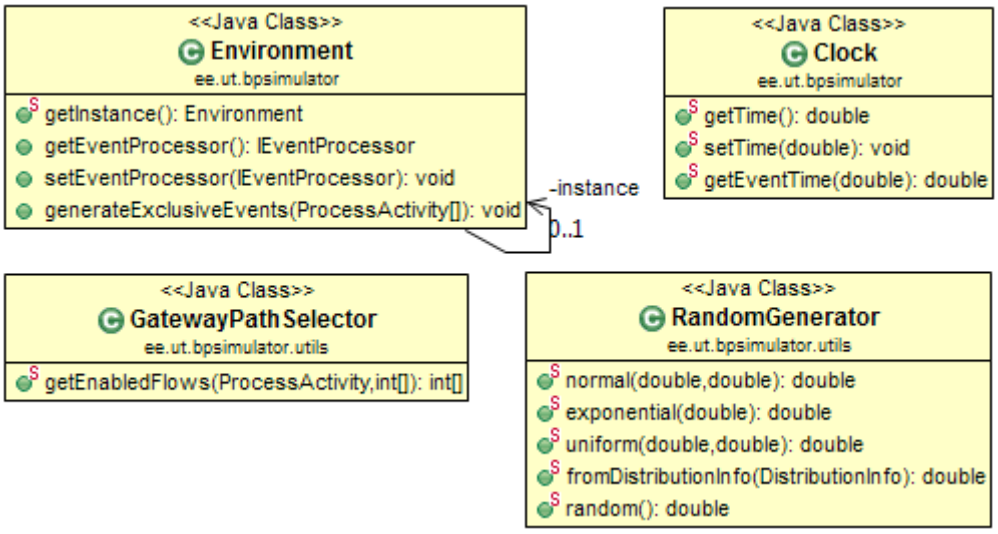

Figure 3.7. Helper Components

The class `Environment` is used for exclusive event generation. Its method `generateExclusiveEvents` takes an array of events to be set as exclusive. The processing time will be calculated for the generated timer events.

The `Clock` class was created for managing the global system clock. All other components including loggers can get the current system from this class. Time handling was encapsulated to a separate class to simplify the potential improvements related to complex clock management (e.g. run simulation on workdays only). Currently, time is stored in the simulator in a variable type called *double*, but in the future it could be easily replaced with some other data type supporting a greater range of values.

The `GatewayPathSelector`, as its name indicates, is responsible for path selections in split gateways. It contains only the public method for which an element representing the gateway and an array of all outgoing flow indexes has to be given. Depending on the gateway type either all, only one or any number of flows will be selected and returned. For each outgoing flow of a gateway, the probability of taking this path in a process instance has to be provided. Although any number between 0 and the count of all outgoing flows could be selected for Or-split gateways, the simulation requires at least one path to be se-

lected. In order to solve this problem with Or-splits, the `GatewayPathSelector` generates outgoing flows until at least one of them is selected.

All random numbers during the simulation process are generated by the `RandomGenerator` class. This class can generate random numbers with normal, uniform and exponential distributions. The implementation of the component uses the Colt library [11], which is a set of open source libraries for high performance scientific and technical computing for Java, for random number generation.

Actually, there are more components used by the simulator, but those are not of great importance to the simulation. The list and descriptions of all used classes, their attributes and methods can be found from the simulator API documentation [10].

3.4. Summary

The architecture of the simulator including relations between different components used was explained in this chapter. The components were divided into two categories by their purpose and roles in simulation process: the core and helper components. It was shown which components are engaged in handling the simulation of basic and advanced constructions available in BPMN in relation to the approach discussed previously in chapter 2. Additionally, the class diagrams of all introduced classes were provided.

4. Analysis of the Results

As it turned out in section 1.5.1, state of the art commercial business process simulation tools lack simulation support for advanced constructions available in the BPMN 2.0 like sub-processes; intermediate events and event-based gateways; error handling with boundary catch events; and inclusive converging gateways (Or-joins). It has been shown that using the new approach introduced in the chapter 2 with the architecture described in the chapter 3 it is possible to implement the simulator engine which can handle most of the advanced BPMN 2.0 constructs.

In this chapter the performance of the BPS tool built as the result of this thesis will be compared to the commercial BPS tools and an overview is given of the performance of business process models containing advanced constructions of the BPMN.

4.1. Performance Comparison with Commercial BPS Tools

Firstly, a series of experiments has been conducted to compare the performance the simulation engine with those of WebSphere and Savvion tools. To that end, the insurance claim handling process (taken from [12]) was specified in BPMN, as shown in Figure 4.1. In short, there are two call centers with 90 call center agents and one back-office with 150 claim handlers. The process model is relatively simple and contains only those elements which are supported by both BPS tools. Both call centers receive a given amount of calls during a week and they register the claims for the back-office. For ease of simulation, the constant call arrival rate of 11 seconds will be used. Full process model in BPMN is shown in Figure 4.1. The experiments were conducted in an environment with Intel Core Duo T5470 (1.60 GHZ) processor, 2 GB of memory (RAM) and 32bit Microsoft Windows 7 operating system.

In WebSphere it is possible to turn on and off several settings that have an impact on the performance of the simulation. For example, animations, displaying statistics during simulation and storing the simulation result can be turned off entirely. Turning all those options off is not reasonable for real-world analysis because no reports or statistics would be produced about the simulation. However, for the purposes of this comparison all these settings are turned off to measure the duration of the simulation only and have comparable data. In this configuration the simulation takes from about 5 to 51 seconds to complete 10 000 to 100 000 process instances as seen in Figure 4.2.

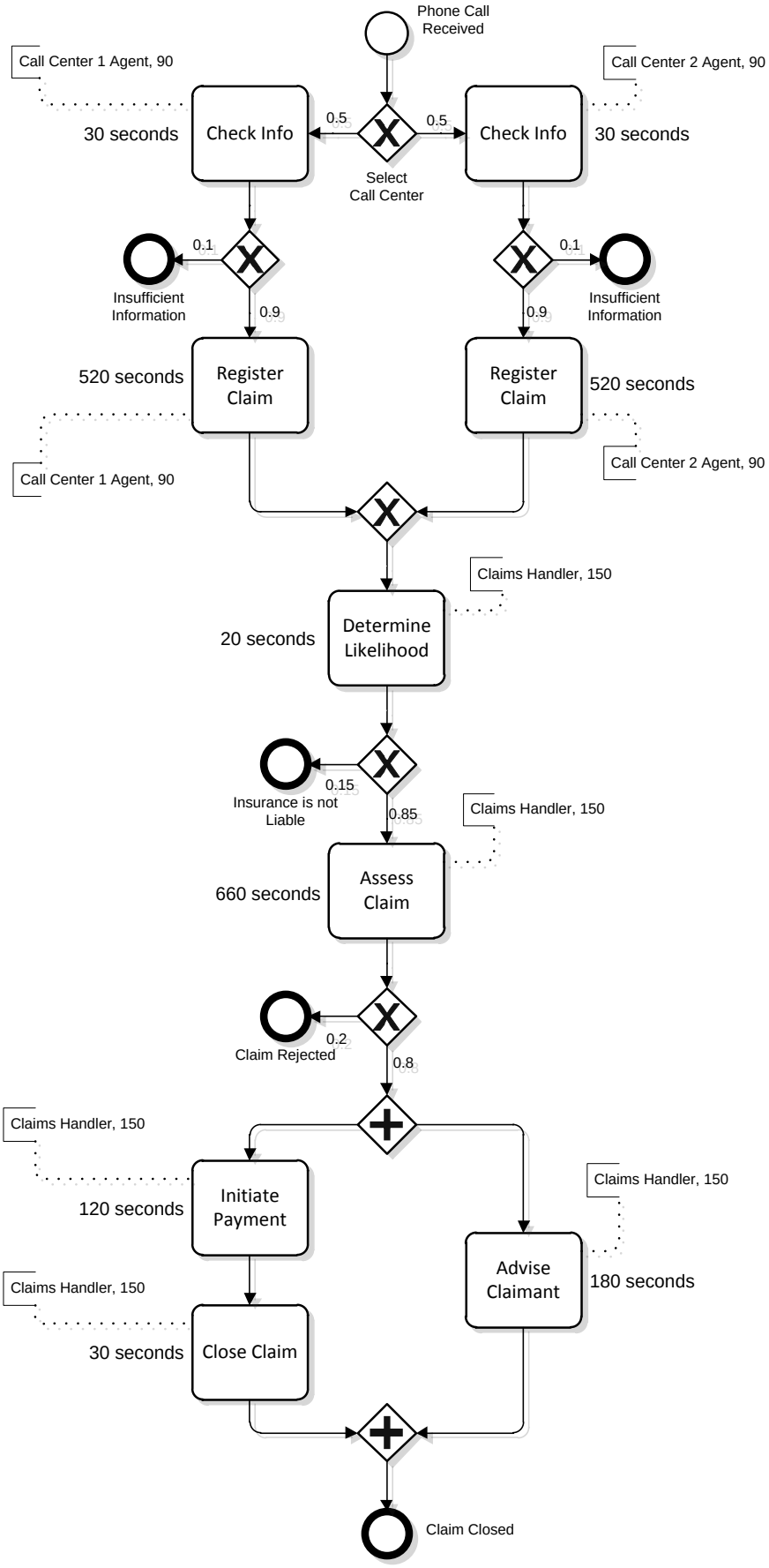

Figure 4.1. Insurance Claim Handling Process Model in BPMN from [12]

For Savvion such simulation configuration options are not available and the animation with simulation-time statistics cannot be turned off. The simulation of 18 000 process instances (case presented in [12]) lasts for about 2 minutes and 20 seconds using the given configuration. As the Savvion simulator does not allow turn the user interaction off, it does not make sense to compare the simulation times with those of WebSphere and our simulation tool. However, it results surprising that such optimization is not provided in state of the art BPS tools.

Running the same insurance claim handling process model in Figure 4.1 with our BPS tool gives the following results: from about 0.4 to 4.0 seconds. When comparing the results to the WebSphere, we can say that we have implemented a simulator which is more than 10 times faster.

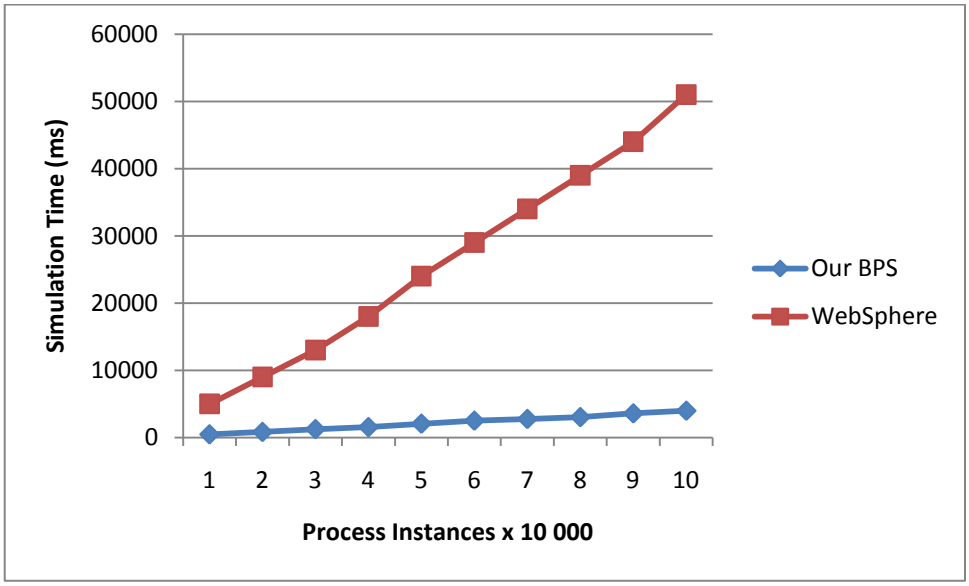

Figure 4.2. Insurance Claim Handling Process Simulation Performance

4.2. Performance of Complex Business Process Models

In this section it is discussed how the complexity of a business process model affects the performance of the simulation. Let us take four models containing various BPMN constructions as follows:

a) Simple insurance claim handling, taken from [12], Figure 4.1;
b) Loop with an Or-join, taken from [8], Figure 4.3;
c) Pizza ordering process with intermediate events, Figure 4.4;
d) Order fulfillment with sub-processes and boundary events, Figure 4.5.

Models b) and c) use one human resource with the total availability of 100. Both the main process and the "Procurement" sub-process in model d) have 100 human resources available for each process. For simplicity the arrival rate for models b) and c) is fixed to 100 and for model c) to 1000 time units correspondingly. Simulation-specific data for model a) was described previously and is shown in Figure 4.1.

The complexity of these four models can be seen in the following comparison: model a) is the simplest one involving only the basic control flow, Or-join has to be handled in model b), intermediate events are introduced in model c), and model d) adds additional complexity of sub-processes with event throwing and catching between the two processes.

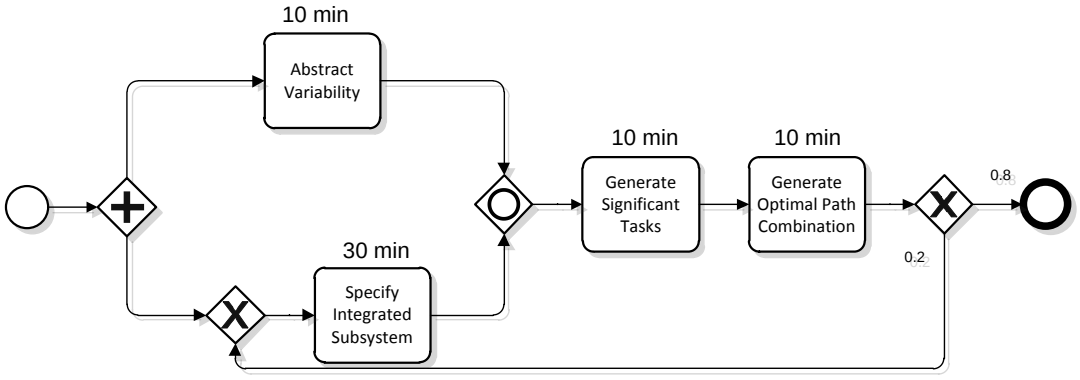

Figure 4.3. Model b) Loop and an Or-join, from [8]

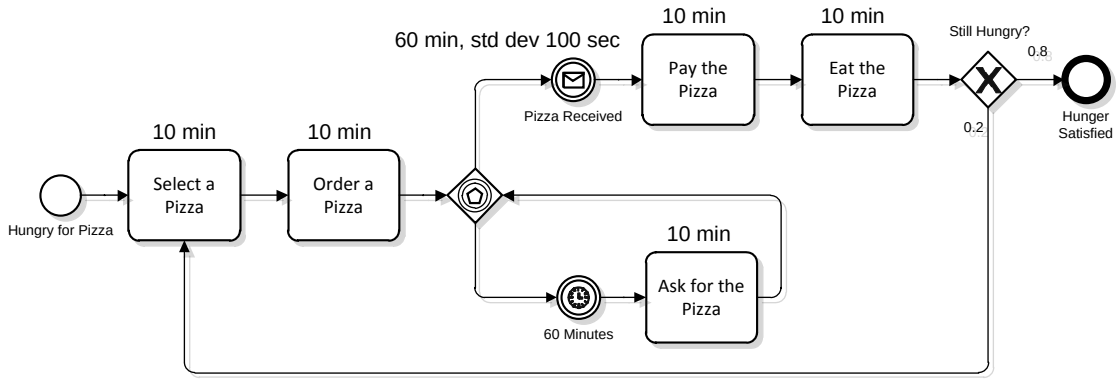

Figure 4.4. Model c) Pizza Ordering, Inspired from [7]

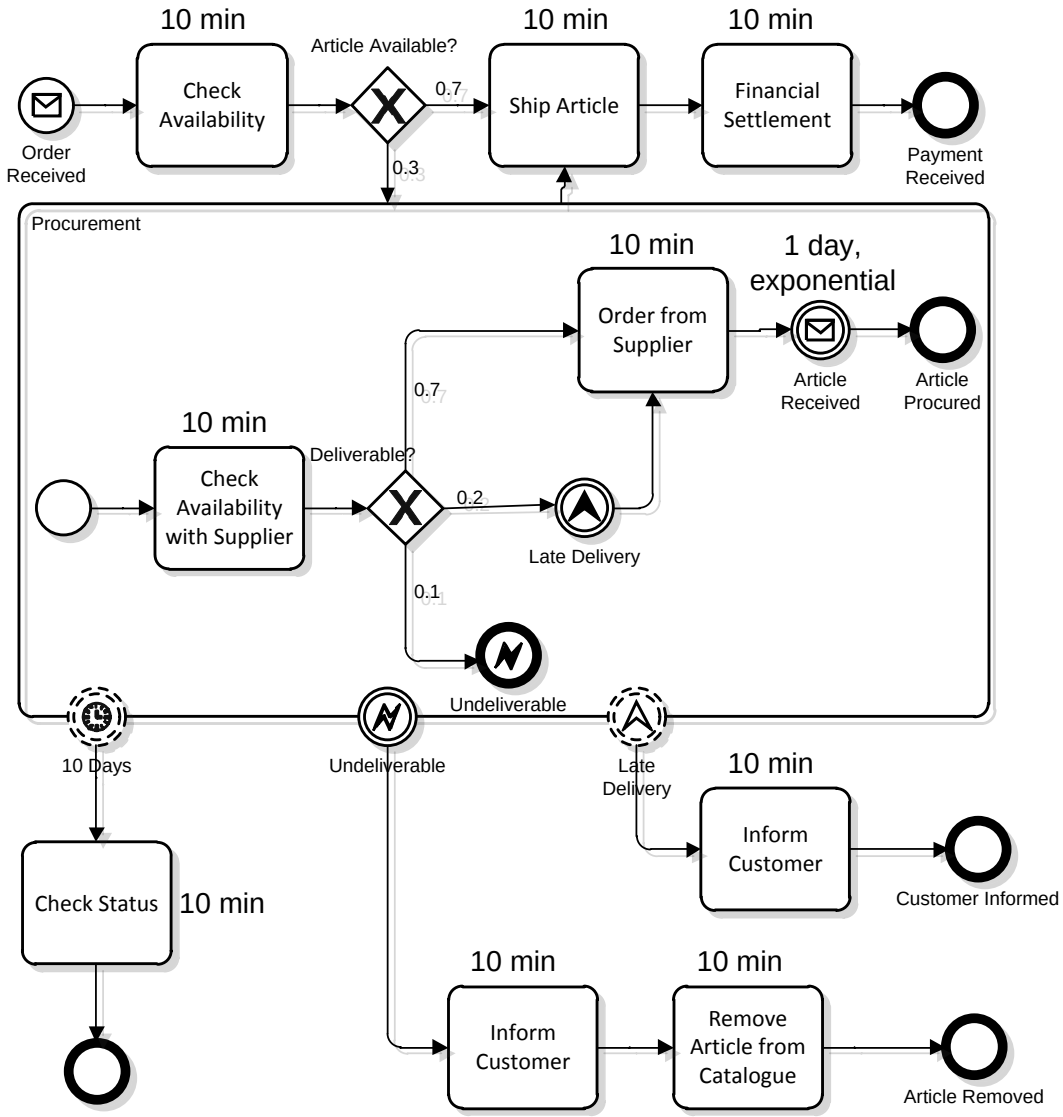

Figure 4.5. Model c) Order Fulfillment with a Sub-Process and Boundary Events

The simulation was carried out from 10 000 to 100 000 process instances which resulted in simulation times from 0.4 to 4.0 seconds on average for model a), from 0.5 to 4.6 seconds for model b), from 0.4 to 3.4 for model c), and from 0.4 to 3.1 seconds on average for model d). The simulation times for the four models can be seen in Figure 4.6.

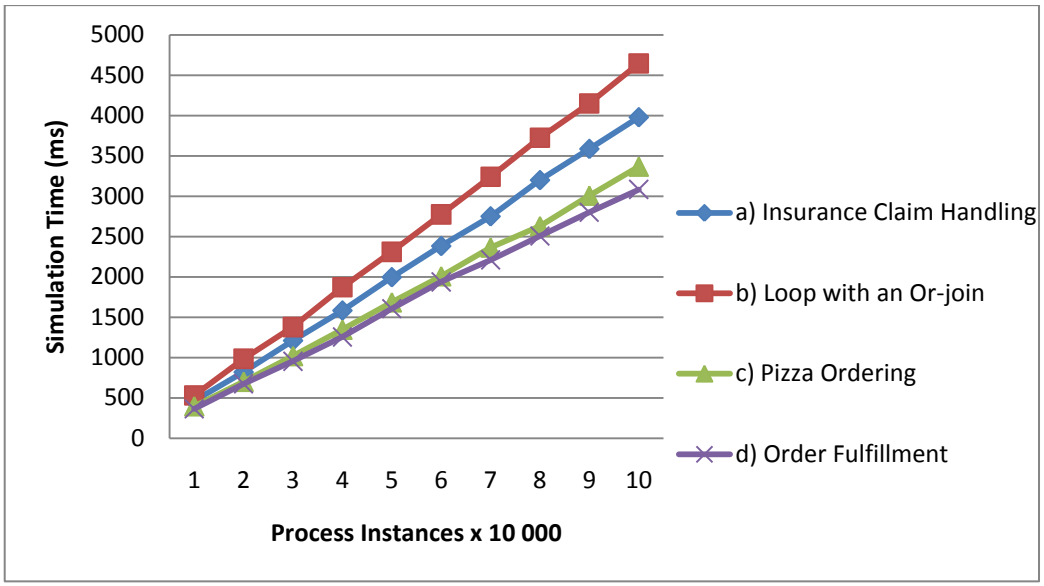

Figure 4.6. Simulation Times

The total number of processed elements in simulations has been counted as well and the results are illustrated in Figure 4.7. From there it can be seen that the simulator has actually processed a different amount of elements for each model. In order to make those results comparable, the measure elements processed per second will be calculated. The results of elements processed per second are shown in Figure 4.8. Figure 4.8 indicates that there exists a constant for each model which expresses the complexity of the model for the simulator. It can be noticed that the special handling which is required for Or-joins, intermediate and boundary events and sub-processes slows down the simulation process only a little and the impact to the simulation process is not considerable. In Figure 4.8 it can be seen that from 10 000 to 30 000 process instances the simulation speed is not as good as with greater amount of instances. This is caused by the fact that the measured time includes the model pre-processing which is constant per business process model, does not depend on the number of process instances and, therefore, has more effect on small-scale simulations.

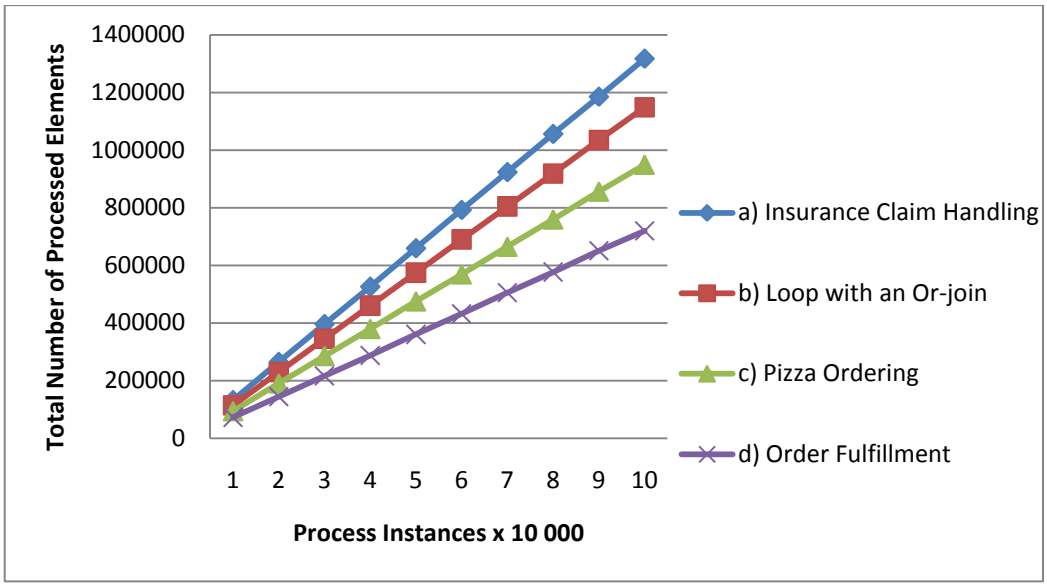

Figure 4.7. Total Number of Processed Elements

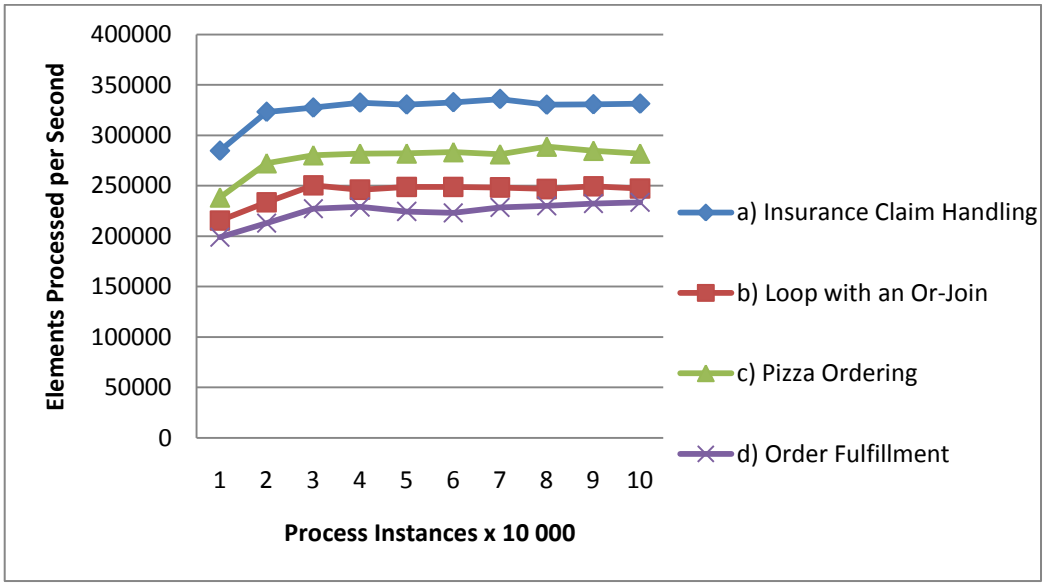

Figure 4.8. Elements Processed per Second

4.3. Complexity of the Or-join

So far the simulation performance of several business process models with various set of BPMN elements has been compared. In addition to that, a further comparison will be made between the speed of simulation in the process models from Figure 4.3 containing an Or-join and the equivalent process model without the Or-join shown in Figure 4.9. Model in Figure 4.9 is basically an enhanced version of model b) from the previous section where the Or-join has been eliminated by duplicating the "Specify Integrated Subsystem" task when loop occurs in a process instance.

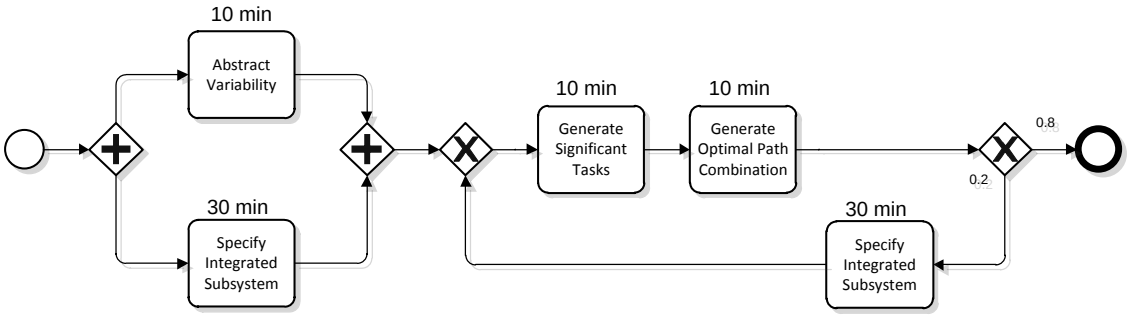

Figure 4.9. Loop without the Or-join

Figure 4.10 shows the simulation time differences and Figure 4.11 illustrates the processed elements per second for both models with and without the Or-join. As both models are equivalent to each other, the total numbers of processed elements are the same in all scenarios. The simulation times vary from about 0.5 to 4.7 seconds in case of the model with the Or-join and from about 0.4 to 3.5 seconds in case of the model without the Or-join. Approximately 320 000 elements in the particular model with and about 245 000 elements in the particular model without the Or-join can be handled per second. The difference is only about 23% and it can be concluded that the Or-join handling adds a little overhead to the simulation process which, however, is not significant for real-world simulations due to the fast simulation speed which can be measured in seconds.

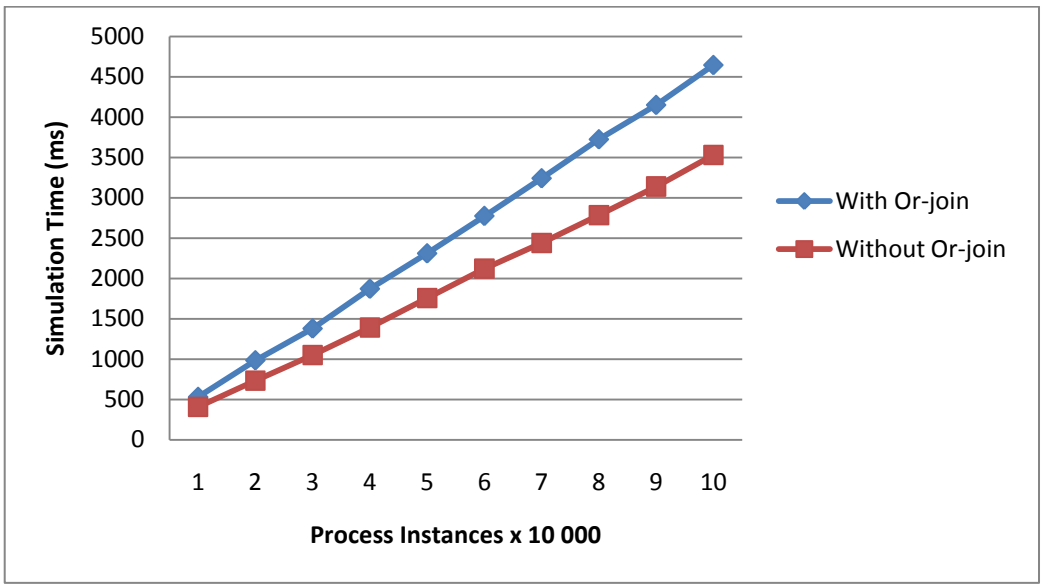

Figure 4.10. Simulation Times with and without Or-join

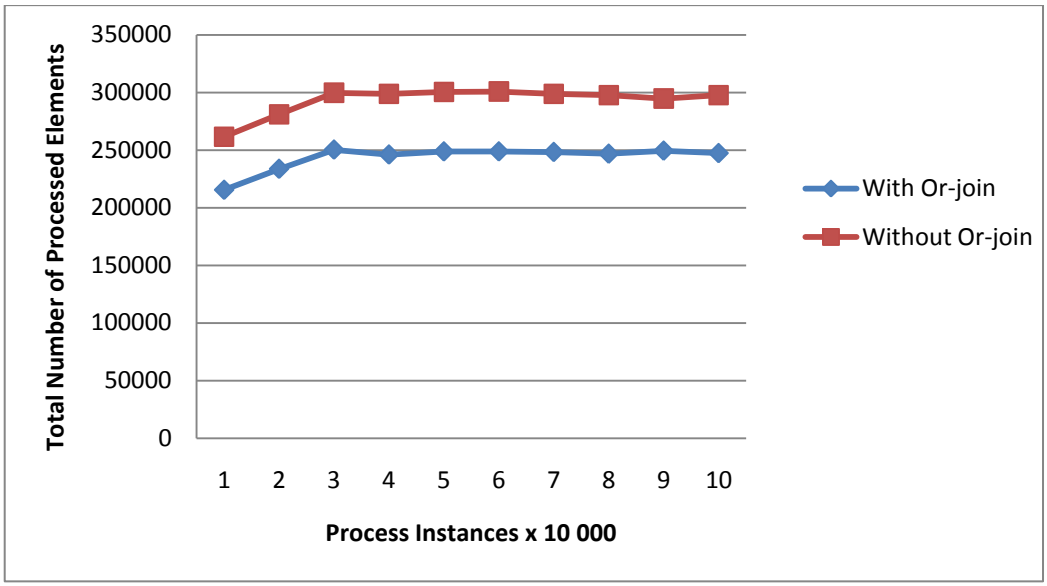

Figure 4.11. Processed Elements per Second with and without Or-join

4.4. Summary

In conclusion it can be said that the simulation time with the built simulator engine is always linear, relative to the number of process instances and elements in the model, and finally, affected by the complexity of the model which can be assessed by the usage of advanced BPMN constructions in the model. The complexity of the Or-join handling was analyzed separately and the results have indicated that this additional overhead has almost no effect on the speed of the simulation.

Although performance of the simulation is affected by the complexity of the model, it turns out that the difference in simulation speed between models is not significant and probably not noticed when simulating the real-world scenarios.

Conclusion and Future Work

In this thesis the concept of business process management as a discipline of making organizations more effective and productive was described. The thesis specifically focused on the two phases of the discipline which involve analysis and modeling and it was shown how the business process simulation technique can be used to try and make experiments with potential candidates of "to-be" business processes without having to put them into production. In the worst case, without thorough analysis, a new version of the business process could lead to even more degraded performance. Business process simulation tools have been used to make such experiments virtually and two state of the art tools have been described: IBM WebSphere Business Modeler and Savvion Process Modeler. It turns out that simulation tools available at the moment are often very slow, can handle only simple process models and cannot deal with large-scale simulations.

To overcome these problems a new untraditional approach was introduced. The proposed approach uses efficient lookup tables to detect which of the elements in the model can be enabled at any moment given a state of a process instance. The state of a process instance is represented by active flows in the model. By enumerating all flows in a model we can express each possible state as a set of bits and perform fast bitwise operations. It was shown that using such approach it is possible to simulate even more complex constructions of BPMN like loops, sub-processes, intermediate events with or without event-based gateways, error and timer handling with boundary catch events, and inclusive converging gateway (Or-join).

Moreover, the simulator was implemented using the proposed approach and the architecture of it is presented in this thesis. The core of the engine is split into separate components to encapsulate functionality. For instance, there are separate components for resource management, handler for queue of started events, event scheduler and Or-join management.

Experiments have been conducted to compare the performance and coverage of BPMN constructions with some commercial simulation tools. For example, the performance of simulation has been compared to the IBM WebSphere Business Modeler and it has been shown that the simulator using the proposed architecture and approach is more than 10 times faster. This experiment was conducted with rather simple business process model of insurance claim handling discussed in [12] due to the fact that more complex models with advanced constructions of BPMN could not be simulated at all using WebSphere. Internal-

ly, the performance has been analyzed using several process models of various complexities in the meaning of BPMN constructions used in these. It can be concluded that the complexity of the model affects the performance slightly, but in general the speed of the simulation is always linear and can be considered very fast.

Although the fast simulation engine has been implemented, it is not complete yet and should be improved in the future. At the moment it is possible to conduct the simulation without getting any statistics of key performance indicators like resource utilization, waiting times and cost directly from the simulator. There exists the logger component which creates audit trail entries in mining XML (MXML) format that can be analyzed with a process mining framework like ProM [21], but a new component for key performance indicator measurements could be implemented as well. Currently it has not been specified how to define costs, waiting or transfer times for tasks or working schedules for participants in BPMN 2.0, however, this is not a limitation from design perspective and can be easily defined and counted in the simulation process.

At the moment the simulator supports business process models only in BPMN 2.0 format, but it has been designed so that parsers for other standardized formats like XPDL [22] could be added and simulated afterwards.

Finally, the majority of constructions available in BPMN 2.0 have been covered, but there are still some of them remaining to implement to gain full coverage of BPMN 2.0. For example the multi instance activities with different combinations of their markers, transactions and compensations, signal and conditional events have not been implemented. Again, from the design perspective this is not a limitation and the support for remaining elements in BPMN 2.0 could be added using the proposed architecture and existing components.

Ülikiire äriprotsesside simulaator

Magistritöö (30 EAP)

Madis Abel

Äriprotsesside juhtimine on teatud hulk järjepidevalt korratavaid tegevusi alustades äriprotsessi analüüsist, millele järgneb modelleerimine, väljaarendamine, elluviimine ning jälgimine. Korrektne äriprotsesside juhtimine on aluseks efektiivsele ning produktiivsele ettevõttele ning võimaldab kiirelt muutuvas keskkonnas kohandada vastavalt ka ettevõtte äriprotsesse. Rutakalt, läbimõtlemata või –proovimata tehtud muudatused ettevõtte töövoo korralduses võivad halvemal juhul lõppeda veel ebaefektiivsemate tulemustega, mis põhjustavad oodatud kasu asemel hoopis kahju. Seetõttu on oluline tehtavaid muudatusi enne reaalset rakendamist põhjalikult analüüsida, mida omakorda saab teha läbi virtuaalse äriprotsesside simuleerimise. Protsesside simuleerimine on laialdaselt levinud metoodika katsetamaks kavandatavaid mudeleid ning analüüsimaks mõju erinevatele ettevõtte tulemuslikkuse näitajatele, mis tuleneb tehtud muudatustest.

Käesoleval ajahetkel on olemas erinevaid äriprotsesside simuleerimise rakendusi nii teadusliku kallakuga kui ka kommertslahendusi nagu näiteks IBM Websphere Business Modeler, Savvion Process Modeler ja teised. Osutub aga, et olemasolevad rakendused on tihtipeale väga aeglased, nendega ei saa modelleerida või simuleerida keerukamaid äriprotsesse või need ei tule toime suuremahulisemate simulatsioonidega.

Käesoleva magistritöö esimeses osas on räägitud üldiselt äriprotsesside juhtimisest, nende simuleerimisest ning olemasolevast tarkvarast. Seejärel esitletakse täiesti uut lahendust, kuidas ehitada äriprotsesside simulaator, mis toetab ka keerukamaid konstruktsioone äriprotsesside mudelite *de facto* esitusstandardist BPMN ning on kordi kiirem kui olemasolevad tasuliselt pakutavad simulatsioonitarkvarad. Kolmandas osas kirjeldatakse lähemalt loodud simulaatorit ja selle arhitektuuri ning viimases peatükis võrreldakse saavutatud tulemust eelpool nimetatud olemasolevate äriprotsesside simuleerimisrakendustega ja antakse ülevaade simulaatori jõudlusest üldiselt.

References

[1] R. M. Dijkman, M. Dumas, C. Ouyang. Formal Semantics and Automated Analysis of BPMN Process Models, *Queensland University of Technology*, April 2007.

[2] V. Bosilj-Vuksic, V. Hlupic. Criteria for the Evaluation of Business Process Simulation Tools. *Interdisciplinary Journal of Information, Knowledge, and Management, Volume 2*, 2007.

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

[4] K. Blum. Open and Extensible Business Process Simulator, *Master Thesis, Institute of Computer Science, University of Tartu*, 2010.

[5] H. Völzer. A New Semantics for the Inclusive Converging Gateway in Safe Processes, *IBM Research*, 2010.

[6] Object Management Group. Business Process Modeling Notation 2.0, January 2011.

[7] Object Management Group. BPMN 2.0 by Example. Version 1, June 2010. .

[8] M. Dumas, A. Grosskopf, T. Hettel, M. Wynn. Semantics of Standard Process Models with Or-joins. *In Proceedings of the International Conference on Cooperative Information Systems (CoopIS). Algarve, Portugal. Springer Verlag*, November 2007.

[9] FIFO Queue

http://en.wikipedia.org/wiki/FIFO (01.05.2010)

[10] Project Homepage in Google Project Including the Simulator API

http://code.google.com/p/ut-bpsimulator (21.05.2010)

[11] Colt Library

http://acs.lbl.gov/software/colt/ (03.05.2010)

[12] W. M. P. van der Aalst, M. Rosemann, and M. Dumas. Deadline-based Escalation in Process-Aware Information Systems. *Elsevier Science Publications*, March 2007.

[13] IBM WebSphere Business Modeler Advanced Version 7

http://www.ibm.com/software/integration/wbimodeler/advanced (08.05.2010)

[14] Savvion Process Modeler

http://web.progress.com/en/savvion/process-modeler.html (08.05.2010)

[15] UML Activity Diagram

http://en.wikipedia.org/wiki/Activity_diagram (08.05.2010)

[16] Event-driven Process Chain (EPC)

http://en.wikipedia.org/wiki/Event-driven_process_chain (08.05.2010)

[17] Object Management Group (OMG)

http://www.omg.org (08.05.2010)

[18] JavaScript Object Notation (JSON)

http://www.json.org (08.05.2010)

[19] Petri Nets

http://www.scholarpedia.org/article/Petri_net (08.05.2010)

[20] Priority Queue

http://en.wikipedia.org/wiki/Priority_queue (08.05.2010)

[21] ProM – Process Mining Framework

http://www.promtools.org/prom6/ (08.05.2010)

[22] XML Process Definition Language (XPDL)

http://en.wikipedia.org/wiki/XPDL (08.05.2010)

[23] M. Zäuram. Business Process Simulation Using Coloured Petri Nets, *Master Thesis, Institute of Computer Science, University of Tartu*, 2010.

[24] B. Rücker. Building an open source Business Process Simulation tool with JBoss jBPM, *Master Thesis, Stuttgart University of applied science*, 2008.

[25] Colored Petri Nets (CPN)

http://cpntools.org (17.05.2010)

[26] G. Decker, R. Dijkman, M. Dumas, L. G. Bañuelos. The Business Process Modeling Notation. In A. H. M. Hofstede, W. M. P. van der Aalst, M. Adams, N. Russell (eds). Modern Business Process Automation, pages 347-348, Springer, 2010.

[27] M. Laguna, J. Marklund. Business Process Modeling, Simulation and Design. Prentice Hall, 2004.

Appendices

A. Business Process Model Serialized in BPMN 2.0

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<semantic:definitions id="_1275486223307" targetNames-
pace="http://www.trisotech.com/definitions/_1275486223307"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:di="http://www.omg.org/spec/DD/20100524/DI"
xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI"
xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"
xmlns:semantic="http://www.omg.org/spec/BPMN/20100524/MODEL">
    <semantic:process isExecutable="false" id="_6">
        <semantic:startEvent name="" id="StartProcess">
            <semantic:outgoing>_6-468</semantic:outgoing>
        </semantic:startEvent>
        <semantic:task completionQuantity="1" isForCompensation="false"
startQuantity="1" name="Quotation Handling" id="TaskQuotationHandling">
            <semantic:incoming>_6-468</semantic:incoming>
            <semantic:outgoing>_6-470</semantic:outgoing>
        </semantic:task>
        <semantic:exclusiveGateway gatewayDirection="Diverging" name=""
id="GatewayOrderApprovedDecision">
            <semantic:incoming>_6-500</semantic:incoming>
            <semantic:outgoing>_6-502</semantic:outgoing>
            <semantic:outgoing>_6-552</semantic:outgoing>
        </semantic:exclusiveGateway>
        <semantic:task completionQuantity="1" isForCompensation="false"
startQuantity="1" name="Order  Handling" id="_6-190">
            <semantic:incoming>_6-504</semantic:incoming>
            <semantic:outgoing>_6-508</semantic:outgoing>
        </semantic:task>
        <semantic:task completionQuantity="1" isForCompensation="false"
startQuantity="1" name="Shipping Handling" id="_6-241">
            <semantic:incoming>_6-506</semantic:incoming>
            <semantic:outgoing>_6-532</semantic:outgoing>
        </semantic:task>
        <semantic:userTask implementation="##unspecified" completionQuan-
tity="1" isForCompensation="false" startQuantity="1" name="Review Order"
id="TaskReviewOrder">
            <semantic:incoming>_6-534</semantic:incoming>
            <semantic:outgoing>_6-536</semantic:outgoing>
        </semantic:userTask>
        <semantic:endEvent name="" id="EndProcess">
            <semantic:incoming>_6-536</semantic:incoming>
        </semantic:endEvent>
        <semantic:parallelGateway gatewayDirection="Diverging" name=""
id="ParaSplitOrderAndShipment">
            <semantic:incoming>_6-502</semantic:incoming>
            <semantic:outgoing>_6-504</semantic:outgoing>
            <semantic:outgoing>_6-506</semantic:outgoing>
        </semantic:parallelGateway>
        <semantic:parallelGateway gatewayDirection="Converging" name=""
id="ParaJoinOderAndShipment">
            <semantic:incoming>_6-508</semantic:incoming>
            <semantic:incoming>_6-532</semantic:incoming>
            <semantic:outgoing>_6-534</semantic:outgoing>
```

```
        </semantic:parallelGateway>
        <semantic:endEvent name="" id="TerminateProcess">
            <semantic:incoming>_6-552</semantic:incoming>
            <semantic:terminateEventDefinition/>
        </semantic:endEvent>
        <semantic:userTask implementation="##unspecified" completionQuan-
tity="1" isForCompensation="false" startQuantity="1" name="Approve Order"
id="TaskApproveOrder">
            <semantic:incoming>_6-470</semantic:incoming>
            <semantic:outgoing>_6-500</semantic:outgoing>
        </semantic:userTask>
        <semantic:sequenceFlow sourceRef="StartProcess" targe-
tRef="TaskQuotationHandling" name="" id="_6-468"/>
        <semantic:sequenceFlow sourceRef="TaskQuotationHandling" targe-
tRef="TaskApproveOrder" name="" id="_6-470"/>
        <semantic:sequenceFlow sourceRef="TaskApproveOrder" targe-
tRef="GatewayOrderApprovedDecision" name="" id="_6-500"/>
        <semantic:sequenceFlow sourceRef="GatewayOrderApprovedDecision"
targetRef="ParaSplitOrderAndShipment" name="Approved" id="_6-502"/>
        <semantic:sequenceFlow sourceRef="ParaSplitOrderAndShipment" tar-
getRef="_6-190" name="" id="_6-504"/>
        <semantic:sequenceFlow sourceRef="ParaSplitOrderAndShipment" tar-
getRef="_6-241" name="" id="_6-506"/>
        <semantic:sequenceFlow sourceRef="_6-190" targe-
tRef="ParaJoinOderAndShipment" name="" id="_6-508"/>
        <semantic:sequenceFlow sourceRef="_6-241" targe-
tRef="ParaJoinOderAndShipment" name="" id="_6-532"/>
        <semantic:sequenceFlow sourceRef="ParaJoinOderAndShipment" targe-
tRef="TaskReviewOrder" name="" id="_6-534"/>
        <semantic:sequenceFlow sourceRef="TaskReviewOrder" targe-
tRef="EndProcess" name="" id="_6-536"/>
        <semantic:sequenceFlow sourceRef="GatewayOrderApprovedDecision"
targetRef="TerminateProcess" name="" id="_6-552"/>
    </semantic:process>
    <bpmndi:BPMNDiagram documentation="" id="Trisotech.Visio-_6"
name="Order Process" resolution="96.00000267028808">
        <bpmndi:BPMNPlane bpmnElement="_6">
            <bpmndi:BPMNShape bpmnElement="StartProcess"
id="Trisotech.Visio__6_StartProcess">
                <dc:Bounds height="30.0" width="30.0" x="120.0"
y="393.0"/>
                <bpmndi:BPMNLabel/>
            </bpmndi:BPMNShape>
            <bpmndi:BPMNShape bpmnElement="TaskQuotationHandling"
id="Trisotech.Visio__6_TaskQuotationHandling">
                <dc:Bounds height="68.0" width="83.0" x="175.0"
y="374.0"/>
                <bpmndi:BPMNLabel/>
            </bpmndi:BPMNShape>
            <bpmndi:BPMNShape bpmnElement="GatewayOrderApprovedDecision"
isMarkerVisible="false"
id="Trisotech.Visio__6_GatewayOrderApprovedDecision">
                <dc:Bounds height="42.0" width="42.0" x="419.0"
y="387.0"/>
                <bpmndi:BPMNLabel/>
            </bpmndi:BPMNShape>
            <bpmndi:BPMNShape bpmnElement="_6-190"
id="Trisotech.Visio__6__6-190">
                <dc:Bounds height="68.0" width="83.0" x="578.0"
y="312.0"/>
                <bpmndi:BPMNLabel/>
```

```
            </bpmndi:BPMNShape>
            <bpmndi:BPMNShape bpmnElement="_6-241"
id="Trisotech.Visio__6__6-241">
                <dc:Bounds height="68.0" width="83.0" x="578.0"
y="442.0"/>
                <bpmndi:BPMNLabel/>
            </bpmndi:BPMNShape>
            <bpmndi:BPMNShape bpmnElement="TaskReviewOrder"
id="Trisotech.Visio__6_TaskReviewOrder">
                <dc:Bounds height="68.0" width="83.0" x="746.0"
y="374.0"/>
                <bpmndi:BPMNLabel/>
            </bpmndi:BPMNShape>
            <bpmndi:BPMNShape bpmnElement="EndProcess"
id="Trisotech.Visio__6_EndProcess">
                <dc:Bounds height="32.0" width="32.0" x="860.0"
y="392.0"/>
                <bpmndi:BPMNLabel/>
            </bpmndi:BPMNShape>
            <bpmndi:BPMNShape bpmnElement="ParaSplitOrderAndShipment"
id="Trisotech.Visio__6_ParaSplitOrderAndShipment">
                <dc:Bounds height="42.0" width="42.0" x="527.0"
y="387.0"/>
                <bpmndi:BPMNLabel/>
            </bpmndi:BPMNShape>
            <bpmndi:BPMNShape bpmnElement="ParaJoinOderAndShipment"
id="Trisotech.Visio__6_ParaJoinOderAndShipment">
                <dc:Bounds height="42.0" width="42.0" x="668.0"
y="387.0"/>
                <bpmndi:BPMNLabel/>
            </bpmndi:BPMNShape>
            <bpmndi:BPMNShape bpmnElement="TerminateProcess"
id="Trisotech.Visio__6_TerminateProcess">
                <dc:Bounds height="32.0" width="32.0" x="424.0"
y="320.0"/>
                <bpmndi:BPMNLabel/>
            </bpmndi:BPMNShape>
            <bpmndi:BPMNShape bpmnElement="TaskApproveOrder"
id="Trisotech.Visio__6_TaskApproveOrder">
                <dc:Bounds height="68.0" width="83.0" x="294.0"
y="374.0"/>
                <bpmndi:BPMNLabel/>
            </bpmndi:BPMNShape>
            <bpmndi:BPMNEdge bpmnElement="_6-500"
id="Trisotech.Visio__6__6-500">
                <di:waypoint x="377.0" y="408.0"/>
                <di:waypoint x="419.0" y="408.0"/>
                <bpmndi:BPMNLabel/>
            </bpmndi:BPMNEdge>
            <bpmndi:BPMNEdge bpmnElement="_6-536"
id="Trisotech.Visio__6__6-536">
                <di:waypoint x="830.0" y="408.0"/>
                <di:waypoint x="848.0" y="408.0"/>
                <di:waypoint x="860.0" y="408.0"/>
                <bpmndi:BPMNLabel/>
            </bpmndi:BPMNEdge>
            <bpmndi:BPMNEdge bpmnElement="_6-534"
id="Trisotech.Visio__6__6-534">
                <di:waypoint x="710.0" y="408.0"/>
                <di:waypoint x="728.0" y="408.0"/>
                <di:waypoint x="746.0" y="408.0"/>
```

```
                <bpmndi:BPMNLabel/>
            </bpmndi:BPMNEdge>
            <bpmndi:BPMNEdge bpmnElement="_6-532"
id="Trisotech.Visio__6__6-532">
                <di:waypoint x="662.0" y="476.0"/>
                <di:waypoint x="689.0" y="476.0"/>
                <di:waypoint x="689.0" y="429.0"/>
                <bpmndi:BPMNLabel/>
            </bpmndi:BPMNEdge>
            <bpmndi:BPMNEdge bpmnElement="_6-470"
id="Trisotech.Visio__6__6-470">
                <di:waypoint x="258.0" y="408.0"/>
                <di:waypoint x="294.0" y="408.0"/>
                <bpmndi:BPMNLabel/>
            </bpmndi:BPMNEdge>
            <bpmndi:BPMNEdge bpmnElement="_6-552"
id="Trisotech.Visio__6__6-552">
                <di:waypoint x="440.0" y="387.0"/>
                <di:waypoint x="440.0" y="352.0"/>
                <bpmndi:BPMNLabel/>
            </bpmndi:BPMNEdge>
            <bpmndi:BPMNEdge bpmnElement="_6-508"
id="Trisotech.Visio__6__6-508">
                <di:waypoint x="662.0" y="346.0"/>
                <di:waypoint x="689.0" y="346.0"/>
                <di:waypoint x="689.0" y="387.0"/>
                <bpmndi:BPMNLabel/>
            </bpmndi:BPMNEdge>
            <bpmndi:BPMNEdge bpmnElement="_6-506"
id="Trisotech.Visio__6__6-506">
                <di:waypoint x="548.0" y="429.0"/>
                <di:waypoint x="548.0" y="476.0"/>
                <di:waypoint x="578.0" y="476.0"/>
                <bpmndi:BPMNLabel/>
            </bpmndi:BPMNEdge>
            <bpmndi:BPMNEdge bpmnElement="_6-504"
id="Trisotech.Visio__6__6-504">
                <di:waypoint x="548.0" y="387.0"/>
                <di:waypoint x="548.0" y="346.0"/>
                <di:waypoint x="578.0" y="346.0"/>
                <bpmndi:BPMNLabel/>
            </bpmndi:BPMNEdge>
            <bpmndi:BPMNEdge bpmnElement="_6-468"
id="Trisotech.Visio__6__6-468">
                <di:waypoint x="150.0" y="408.0"/>
                <di:waypoint x="164.0" y="408.0"/>
                <di:waypoint x="175.0" y="408.0"/>
                <bpmndi:BPMNLabel/>
            </bpmndi:BPMNEdge>
            <bpmndi:BPMNEdge bpmnElement="_6-502"
id="Trisotech.Visio__6__6-502">
                <di:waypoint x="461.0" y="408.0"/>
                <di:waypoint x="527.0" y="408.0"/>
                <bpmndi:BPMNLabel/>
            </bpmndi:BPMNEdge>
        </bpmndi:BPMNPlane>
    </bpmndi:BPMNDiagram>
</semantic:definitions>
```

B. CD Content

The accompanied CD-ROM contains the following content:

1. /samples – example business process models in BPMN 2.0 that have been tested
2. /src – source code of the simulator
3. /utbpsimulator_lib – used libraries
4. Readme.txt – instructions to simulate the example process models
5. utbpsimulator.jar – executable Java Archive File to run the simulator