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

Viljar Kärgenberg

Online Business Process Model Simulator

Bachelor's thesis (6 ECTS)

Supervisors: prof. Marlon Dumas
Meelis Kull

Author: ..... "....." May 2012
Supervisor: ..... "....." May 2012
Supervisor: ..... "....." May 2012

Allowed to defend
Professor: ..... "....." May 2012

Tartu 2012

Table of Contents

1. Introduction

Business process modelling is the art and science of documenting real-life processes in order to analyse their performance and to identify ways of improving them. For example, a process for handling quotes and purchase orders is an important process in almost every company because customers often judge the quality of company and it's services based on how quickly and effectively they perform this process. A company that takes hours to respond to requests for quotes or purchase orders will certainly be preferred by customers compared to a company that takes several days or weeks.

In real-life, business processes are very complex because there are many details and exceptions that need to be handled. Even something as apparently simple as providing a quote can be very tricky and time-consuming in some companies. For example, a construction company needs to provide very detailed quotes, and giving such quotes sometimes requires that workers from the company visit the customer premises one or several times in order to take measurements and to understand what the customer wants and what they can afford to obtain. Sometimes, several technicians or sub-contractors may be involved in providing a quote for a construction project. The coordination of these technicians and sub-contractors can be very complex. Also, many exceptions can occur during a process, for example when the customer does not accept the first quote and wants to obtain a new quote with new requirements.

By representing the processes in a graphical and reusable form, business process models allow business analysts to get a better grasp of what the process does and to abstract away from the numerous details that real-life business processes bring. In addition, business process modelling allows businesses to analyse the performance of their processes in details, in order to determine how much time these processes take, how busy are the workers who contribute to the process, or how much it costs to run the process. Business process simulation is a popular technique for analyzing the performance of processes based on process models.

Business process modelling and also business process simulation are important components of Business Process Management (BPM) [1]. Business process simulation is a widely used technique for analyzing business process models in terms of performance measures such as time, cost and resource utilisation [2]. Companies are improving their performance by a constant evaluation of the value added in all parts of their processes. Business processes are in a continuous improvement cycle in which design and redesign play an important role [3].

Problem statement

As a part of his master's thesis Madis Abel developed a business process simulator named BIMP - a scalable and high-performance business process simulation engine capable of handling advanced constructions in the process models that are used to represent real-life processes [4]. The simulation tool was only usable from command line and additional simulation specific metadata had to be added manually to the model. From a user's perspective this solution was cumbersome and constituted a barrier to the adoption of BIMP in a wider setting.

During a team-based software project, an initial prototype of an online interface for BIMP was developed [5]. The project was short-term – 3 months. As a result, the prototype produced from this project was insufficiently tested, had performance issues, was not integrated with other modelling tools, and was not designed for extensibility. This thesis aims at addressing these shortcomings.

Contributions

The contributions of this thesis are the following:

- A detailed documentation of the BIMP web-application architecture;
- An automated testing infrastructure to support future development of BIMP;
- Integration of BIMP with a commercial process modelling tool;
- Several improvements and bug-fixes to BIMP based on feedback from automated testing and user testing;

Thesis organization

- Chapter 2 gives an overview of the process modelling and simulation in general. Also the BIMP simulator itself is introduced.
- Chapter 3 describes the BIMP web application's architecture and key components.
- Chapter 4 introduces the testing and improvements made to the online interface of BIMP.
- Chapter 5 contains the summary of the thesis and points out possible improvements.

2. Background and related work

2.1 Process modelling

Business process modelling is an activity of representing real life processes. The representation could be just written on paper or it could be visualised with a diagram consisting of activities and ohter types of nodes. The Business Process Modelling Notation (BPMN) is considered as standard for business process modelling. BPMN defines a graphical notation of comprehensive set of elements, their semantics and XML-based serialization format. The latest version of BPMN was released in January 2011 by the Object Management Group [6]. The version is named BPMN 2.0 [7].

A sample business process diagram (BPD) can be seen in Figure 1. This BPD is a simple one and consists of three types of objects: activities (rectangles), events (circles) and gateways (diamonds). All the objects are connected by directed edges called sequence flows. A diamond with "x" represents an exclusive gateway (XOR gateway). It routes the sequence flow to exactly one of the outgoing branches based on conditions. A diamond shape with "+" represents a parallel gateway. When it is used to split the sequence flow, all the outgoing branches are activated simultaneously. When merging parallel branches, it waits for all the incoming branches to be completed before triggering the outgoing flow [4].

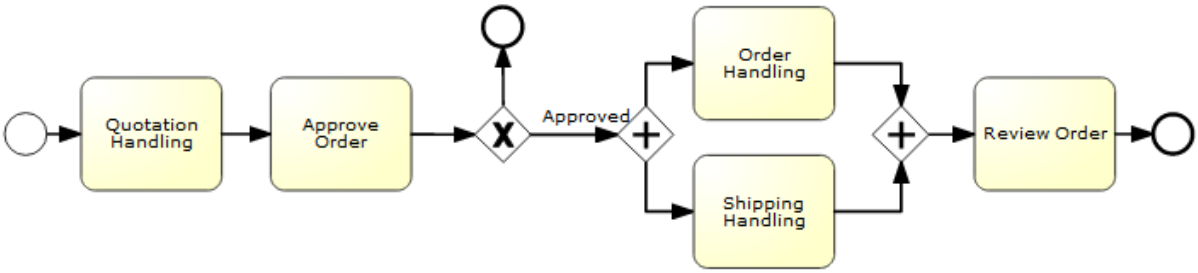

Figure 1. A sample business process model diagram in BPMN notation.

2.2 Process simulation

Business process simulation (BPS) is a widely used technique for analyzing business process models in terms of performance measures such as time, cost and resource utilisation. Simulation allows business analysts to see how the business process is working without having the process in use in real life. BPS makes it possible to see how changes in some parts of the process or changes in resource management are affecting the overall process. For example an insurance company can use business process simulation to make sure, that during rush seasons, when the amount of claims is higher, they are able to process all of them in the allowed timeframe. If the current process does not scale up to the needs, then the simulation makes it possible to understand the reasons and find a possible solution. For example if claim handlers are overloaded with work, then hiring additional personnel could help with overcoming the additional load.

2.3 BIMP

BIMP is an online simulator of business process models captured in the BPMN process modelling notation (Figure 2). It is usable with all modern browsers on the following url: http://bimp.cs.ut.ee. The application enables users to provide a business process model in BPMN 2.0 notation. A sample Business process model in BPMN 2.0 notation with simulation specific metadata is available in Appendix C.

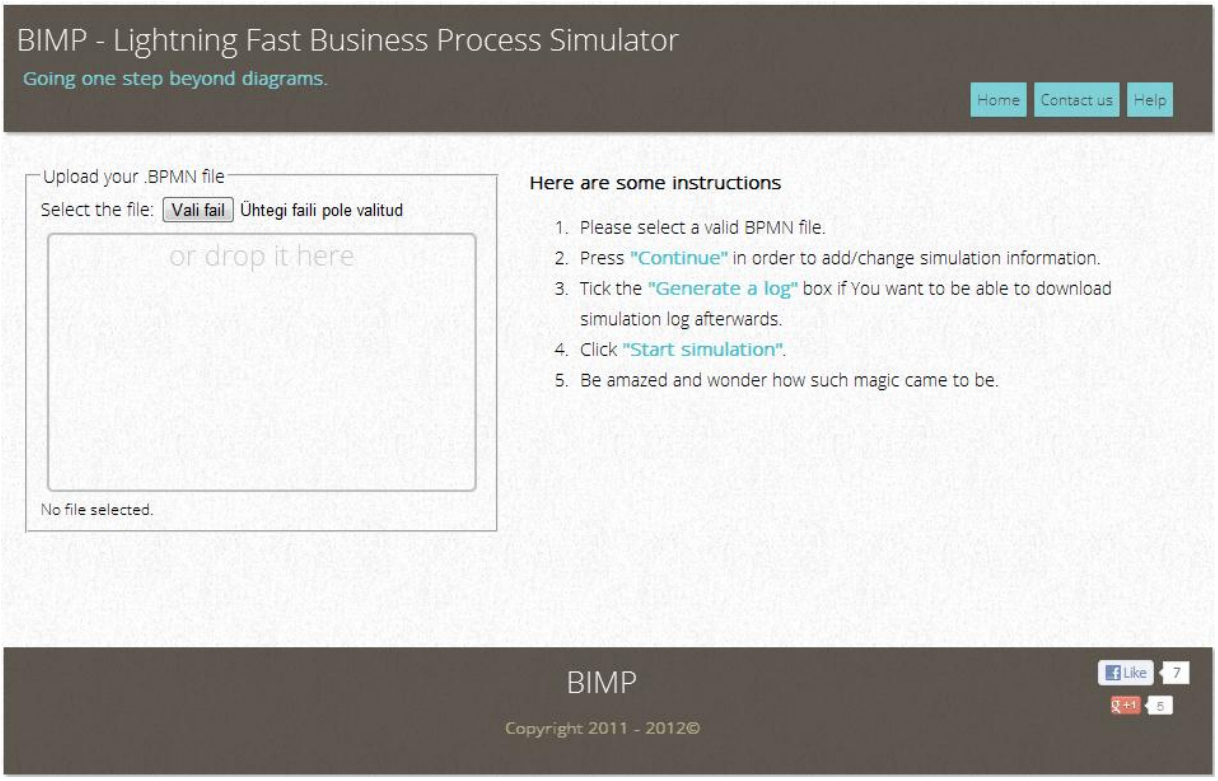

Figure 2. A screenshot of the BIMP's web-interface, the file upload page.

BIMP analyses the model and allows users to enter or edit simulation specific metadata related to the contents of the model and simulation process. The form for providing the simulation specific metadata can be seen in Figure 3. The figure displays the result after the sample model, whose diagram we saw in Figure 1, was loaded to the BIMP. The content of the editable fields (text fields and dropdowns) are not part of the process model and are provided by user.

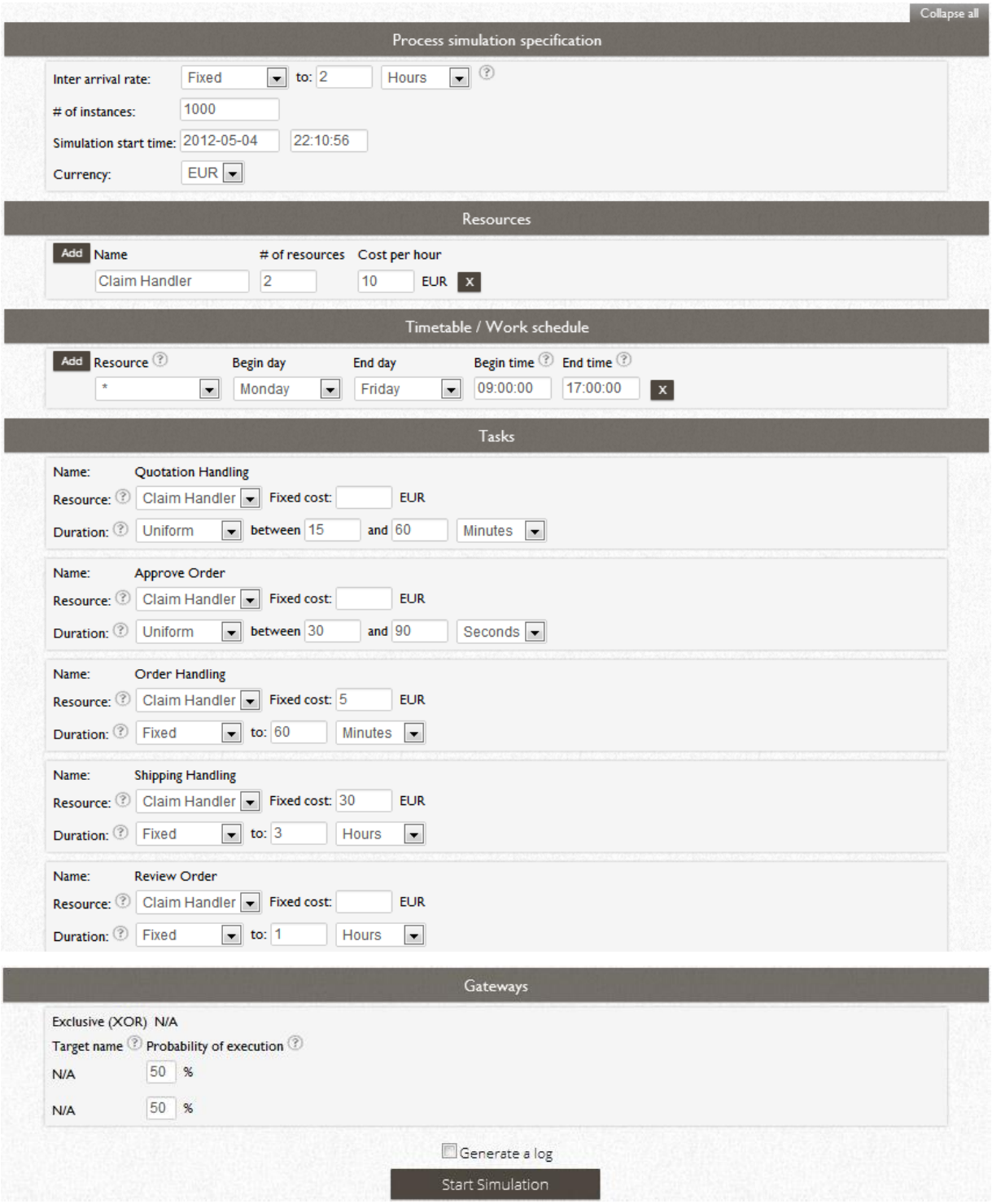

Figure 3. A screenshot of the BIMP's web-interface, the form for entering the simulation specific metadata.

The following information needs to be specified for each task in the process model in order to simulate it:

1. Probability distribution for the processing time of each task.
2. Cost of the task (besides the cost of the resources used by the task).
3. The set of resources that are able to perform the task. This set is usually called a resource pool. For example, a possible resource pool could be the “Claim Handlers”, “Clerks” or “Managers"”.

Separately, the analyst needs to specify for each resource pool the number of resources in this pool (e.g. the number of clerks) and the cost (e.g. the hourly cost of a clerk).

Common probability distributions for task durations in the context of process simulation include:

1. Fixed - always the same value.
2. Exponential distribution - is generally used when the processing time of the task is most often around a given mean value, but such that in some cases, the processing time is much longer, potentially order of magnitudes longer.
3. Normal distribution - is used when the processing time of the task is around a given average, and the deviation around this value is symmetric, meaning that the actual processing time can be above or below the mean with the same probability.

When assigning an exponential distribution to a task duration, the user has to specify the mean, while in case of normal distribution both mean and standard deviation are needed. In addition to the above "per-task" simulation data, a branching probability needs to be specified for every outgoing arc of a decision gateway. Finally, in order to run a simulation, the inter-arrival rate of cases also needs to be specified. Inter-arrival rate is time interval describing how frequently a new case arrives.

After the metadata is provided, users can submit the model for simulation. A status window with a progress bar is shown, displaying the simulation’s progress.

The simulation engine used to run the simulation is developed as a result of Madis Abel’s master thesis. As described in [4], the engine uses an untraditional approach, where the engine is split into separate components which use efficient pre- and post-condition lookup tables of

activities in the model. The state of a process instance is represented by active flows in the model. By enumerating all flows in a model it is possible to express each possible state as a set of bits and perform fast bitwise operations. The components use the state of process instance in order to simulate all process instances from the beginning to the end. The lookup tables are used to detect which are the next possible states of the process. Randomness within the predefined bounds is used to simulate probabilities for gateways and time consumption of activities.

After the simulation has ended, the user is presented with the results page. The page contains histograms showing process durations, process costs, average waiting times and resource utilisation (Figure 4).

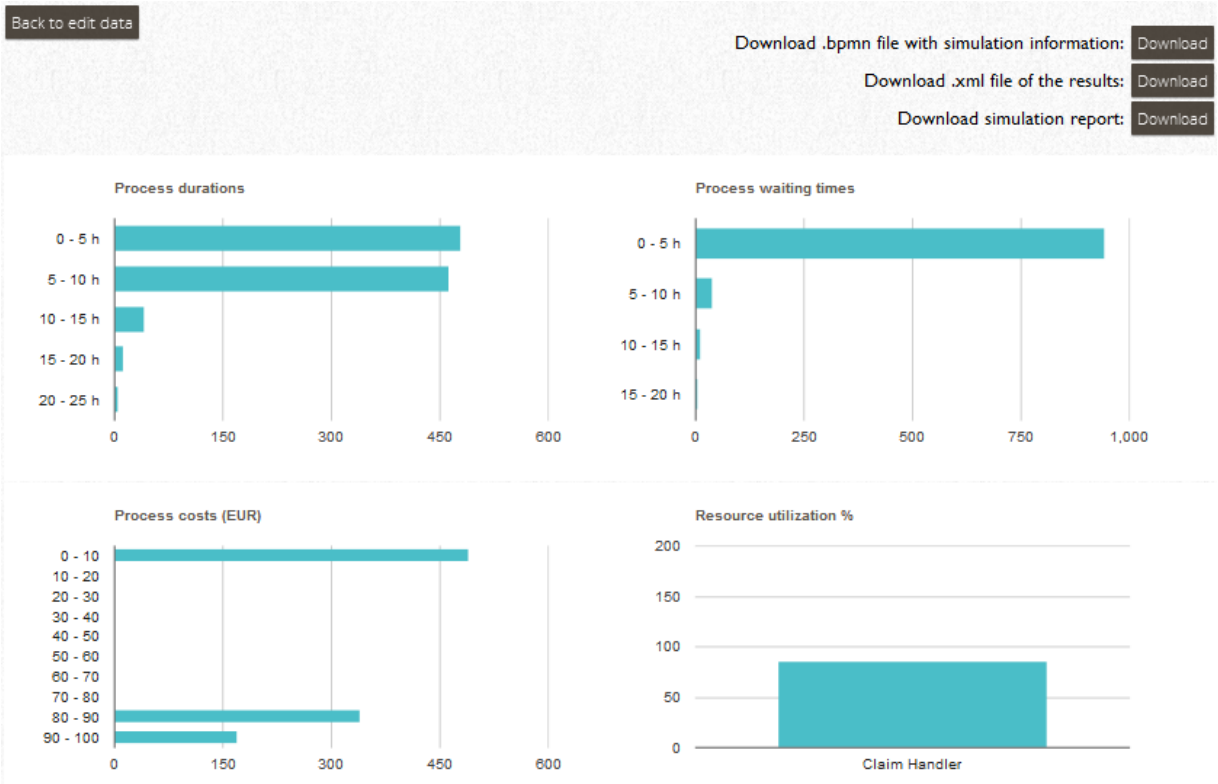

Figure 4. A screenshot of the BIMP's web-interface, the histograms showing the simulation result's summary.

Also statistics of key performance indicators like resource utilization, waiting times and cost are presented (Figure 5). If the user is interested in more detailed information, it is possible to

select a “Generate a log” checkbox before starting the simulation. It results in a detailed simulation log, that will be downloadable on the results page. Specifically, simulation log is represented using the Mining XML (MXML) format, which makes it possible to import it into the ProM framework for further analysis [8]. The summary about the simulation is also downloadable in CSV and XML formats. The process model with metadata is available for download, too (Figure 4). This creates the possibility to continue working with the model in the future. The application understands the metadata included in the file and pre-fills the fields when loaded.

Completed elements	Completed process instances		
7550	1000		
Minimum process cost	**Maximum process cost**	**Total cost**	**Average cost**
2.65 EUR	95.24 EUR	49796.71 EUR	49.8 EUR
Minimum process duration	**Maximum process duration**	**Total duration**	**Average duration**
16.1 min	21.3 h	194.1 days	279.5 min

Task description	*Average cost (EUR)*	*Average waiting time (s)*
Approve Order	0.17	1178.58
Order Handling	15	710.66
Quotation Handling	6.28	1524.41
Review Order	10	1651.95
Shipping Handling	60	2668.91

Figure 5. A screenshot of the BIMP’s web-interface, the summary of the simulation results.

3. BIMP web application

3.1 High-level architecture

The BIMP web application is divided the following way:

1. Client side application
2. Server side application
3. Simulator

Client side application

The client side application is a web page. It is written using HTML, CSS, JavaScript (JS) and jQuery [9] and is usable with all the modern browsers. The client side application is responsible for parsing the model, editing the metadata and displaying simulation results.

The communication between the client and server is achieved with AJAX [10] calls. The user is not actually navigated away from the page after the file has been uploaded to the client side application. The content is loaded and changed dynamically. This design serves several purposes. Firstly, it is fast, because the JavaScript files are loaded and initialised only once. Secondly, while navigating the page, the created JavaScript objects will be gone, and since the parsing and working with the model is all written in JavaScript and resulting data is being held in JS objects, then the navigation can not happen until the metadata is added to the file and the file is submitted for simulation. Thirdly, to enable the quick editing of the model right after the simulation, the input file and all provided metadata are kept in JS objects until the browser is navigated away from the page.

Server side application

On the server side there is a servlet based application using Spring MVC [11]. The application is written in Java and mainly consists of different controllers carrying out various activities like starting the simulation, checking simulation status, providing the simulation summary and

log files, generating the test-report. A sequence diagram explaining the components and information exchange between them can be seen in Figure 6.

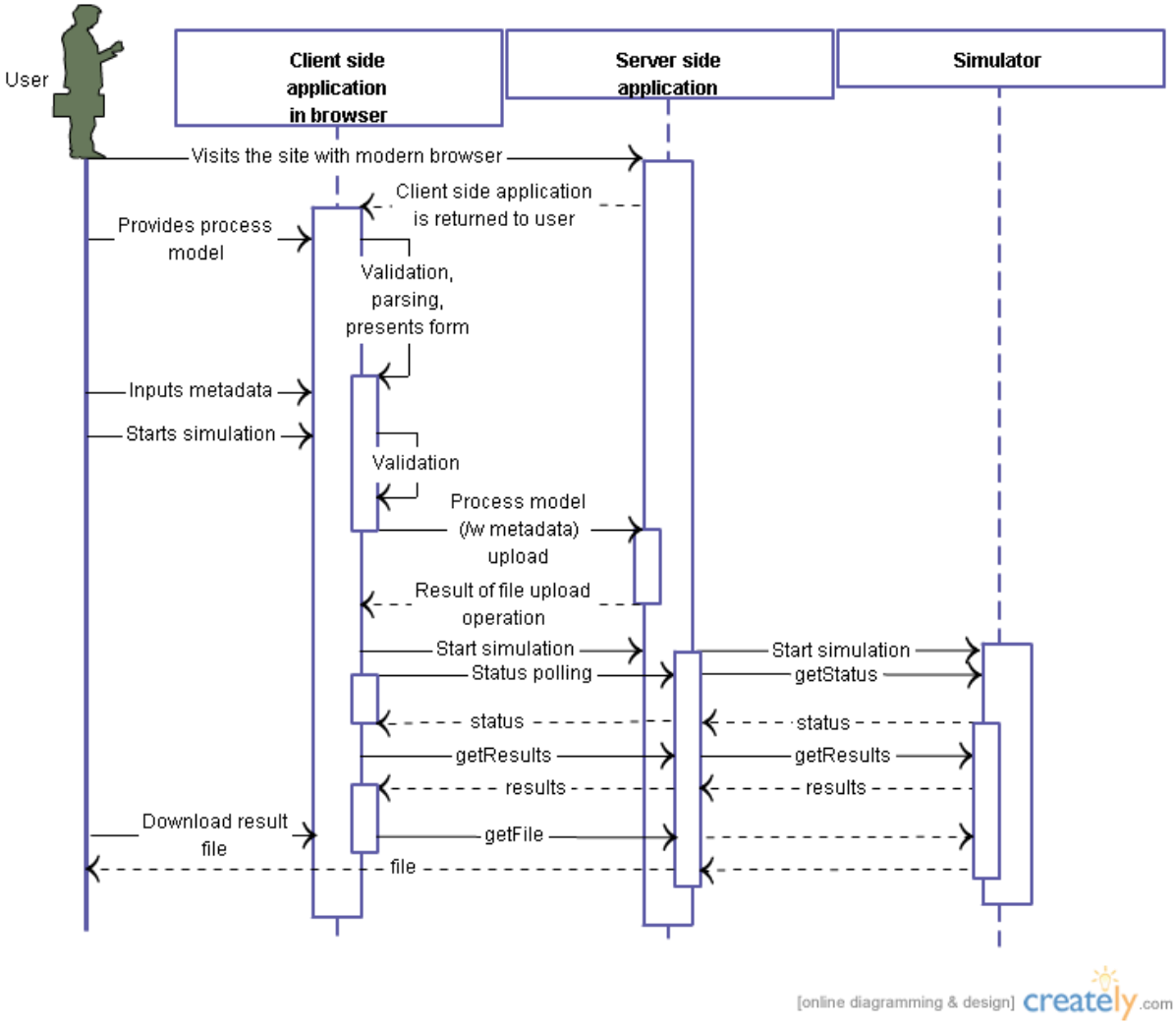

Figure 6. Sequence diagram showing the application's components and flow.

Simulator

The simulator is packaged into a Java library and is ebedded in the server-side application. The package provides a set of classes to initiate and observe the simulation in the server-side

application. Additionally, classes for retrieving the MXML-log file and statistics of key performance indicators are included in the library. The simulator library is available at [12].

3.2 Low-level architecture

Given that the application relies on JavaScript and jQuery and a lot of the key functionality is happening on the client side, the main JavaScript components (Figure 7) of the online application will now be introduced in detail.

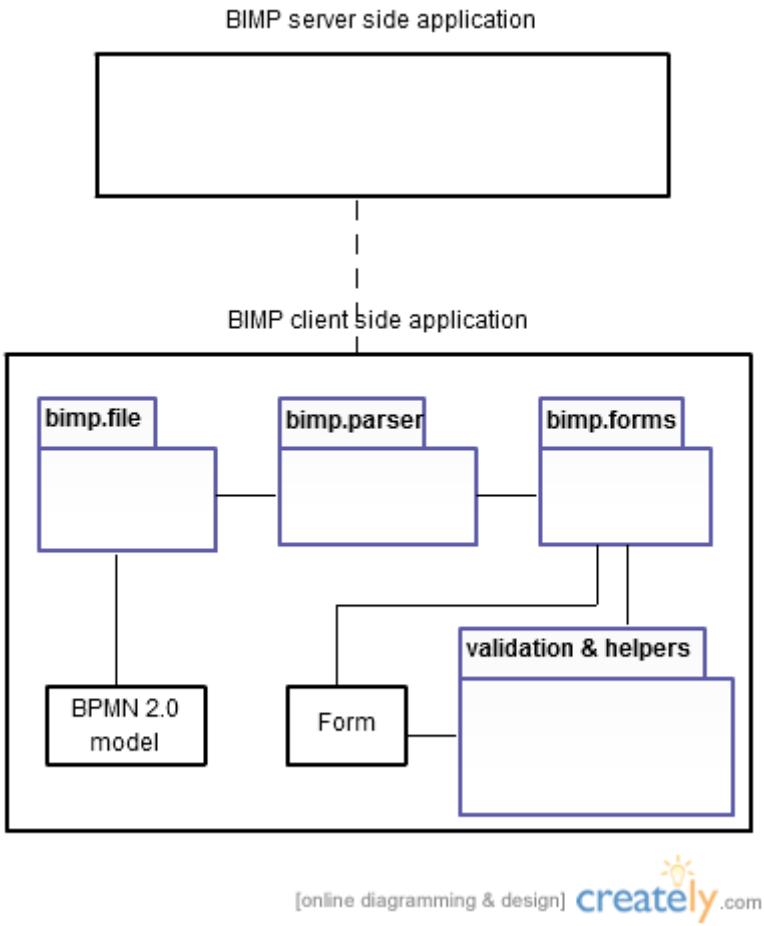

Figure 7. Overview of the client-side application components.

File handler object - bimp.file

The file handler, named bimp.file, contains a collection of functions that are related to the handling of the provided process model file. The key functions are as follows:

1. parseFile(file) - this function is initialised when file selection handler is called, which happens when the user selects the file or drops it into the drop-zone. The function reads the input file as a text and if it is successful, a more specific function readTextToDocument(text) is called.
2. readTextToDocument(text) - this function parses the text into a XML document, using the jQuery.parseXML(). This function also identifies and handles the XML namespaces and does the file validation check, by seeing if the "startEvent" node is present in the XML document.
3. uploadFile() - function is responsible for sending the business process model to the controller that is mapped to "/uploadjson" URL, the file is sent with an AJAX POST request, where the file is passed as "fileData" parameter. When file upload is successful a modal window displaying the simulation progress is displayed.
4. updateFile() - this function is for updating the user provided process model with the metadata that was entered in the form. In this function, initially, the documentation tags are added to the XML file if they are not present. The tags will hold the metadata about events, activities and gateways. After the documentation tags are added or they are already present, the tags will be updated with the metadata which are held as JavaScript objects in bimp.parser parent object.

File parser object - bimp.parser

The file parser object is responsible for finding the events, activities and gateways from the XML-encoded model. The parser object also holds all the simulation metadata in the respective objects or arrays. The metadata is mostly tied to the XML document by using the IDs of the nodes.

The BIMP simulator expects the metadata to be in JSON [13]. The following code snippet is the empty JavaScript object with helper functions representing the metadata to be included with startevent. The ID mapping can be seen in use in addResource() and addTimetable() functions:

```
startEvent : {
      arrivalRateDistribution : {
            type : "",
            mean : "0",
```

```
            value : "0",
            stdev : "0",
            min : "0",
            max : "0",
            timeUnit : ""
        },
        instances : "",
        resources : {},
        startAt : "",
        timetable : {},
        currency : "",
        addResource : function(id, name, costPerHour, amount) {
            this.resources[id] = {
                        "name" : name,
                        "costPerHour" : costPerHour,
                        "amount" : amount
            };
        },
        addTimetable : function(id, obj){
            timetable[id] = obj;
        }
    }
```

The simulator is expecting the JSON in a format where the objects are named as seen in the example snippet. This kind of solution is used also for all the other objects holding metadata. The reason to do this is the following. Since the JSON (JavaScript Object Notation) is language independent is based on a subset of the JavaScript programming language, it is really easy to convert the JavaScript object to JSON [13]. In fact, we need only one function - JSON.stringify(). To do it vice versa, we can use another function: JSON.parse(). This means that there is no need for any additional converting or mapping solution. JSON already present in the model can be parsed easily into objects and new metadata in objects can be easily converted to JSON and inserted to the XML document. An example of how the metadata in JSON is included to XML document can be seen below:

```
...
<startEvent name="Start Event" id="oryx_711D5154">
    <documentation id="oryx_c93289a6">
        {"arrivalRateDistribution":{"type":"exponential","mean":"8
        ","value":"0","stdev":"0","min":"0","max":"0","timeUnit":"
        seconds"},
        "instances":"18000",
        "resources":{"Agent1":{"name":"Call Center 1 Agent",
        "costPerHour":"10", "amount":"90"}},
```

```
            "startAt":"2011-08-08 08:00:00", "timetable":{"*":{"Mon-
            Fri":"09:00:00-17:00:00"},"Agent1":{"Mon-Fri":"10:00:00-
            16:00:00"}},
            "currency":"EUR"}
      </documentation>
</startEvent>
...
```

The main functions in the file parser object are dealing with the retrieval of the events, activities, gateways and their metadata. The XML document is searched for nodes with jQuery. For example, $(xmlFile).find("startEvent").find("documentation") function searches in the XML document named xmlFile for *documentation* nodes that are child elements of the *startEvent* nodes. The main functions of bimp.parser are described next.

1. readStartEvent() - as the name says, the function reads the *startEvent* node from the XML document, in this case, it means that the metadata is loaded to JavaScript object. There are some modelling tools that enable resources describing with lanes. This function tries to handle these kind of cases also, when the *documentation* node is not present, by looking through the lane nodes and updating the startevent's metadata object. If the *documentation* node is present, the lanes are ignored.
2. readTasks() - the tasks can be described in the model with 2 types of nodes, the first one is a common *task* node, the second one is an empty *subProcess* node, that should be handled as a task, too. In this function both of the nodes are searched from the XML document and respective metadata is retrieved.
3. readIntermediateCatchEvents() - this function is a basic one, the document is searched for *intermediateCatchEvent* node and if it is present, metadata is read and saved as JavaScript object.
4. readConditionExpressions() - this function is reading the data of the gateways from the model. It reads the execution probabilities of gateway's outgoing flows. It must be noted that in the models, there is no gateway node. Gateway consists of several *conditionExpression* nodes, that may be different types (i.e. OR and XOR) and that have execution probabilities. The nodes having the same *sequenceFlow* as a source form a gateway.

Form handler object - bimp.forms

Before it is possible to describe the functions, the HTML structure of the forms has to be characterized. The main idea behind the HTML structure is that the form elements' classes, field names and IDs have to follow the JavaScript object naming. For example, a *div* element that is representing a task, has a classname of *task* and the child element representing the resource assigned to the task has *resource* for a class name.

This concept enables to use a general function for displaying the data gathered from the model in the forms. Also, it eases the JavaScript object updating after the form has been filled with metadata. The general function iterates over the JavaScript object's fields, creating the jQuery selectors out of the names of the fields and sets the value for the corresponding DOM (Document Object Model) [14] element. Since the code is reusable for almost all the node types (events, tasks, gateways), it is easier to maintain it and add new functionalities.

Here are the main objects and functions located in the form handler object:

1. generate - is an object containing functions for creating or updating DOM elements. These functions mostly use the general object creation function that will be described next.
2. populateWithData(selector, obj, clone, htmlObj) - this function is a general function for updating the forms and adding new elements. The "selector" parameter is a string, that is used as a parameter for jQuery DOM search. The "selector" is a parent container's class, i.e ".task". The "obj" parameter is a JavaScript object, that holds the metadata about the model element. The function creates an iteration over all the child elements of the "obj". This serves the automation purpose that will guarantee that the correct value of an object ends up in the correct form element. The "clone" parameter is used together with the "htmlObj" parameter. The purpose of these is to define if the already existing element in DOM will be changed or a new additional instance of the DOM element is used and changed. The general logic on both cases is same, but in the first case a direct change to the DOM will be done. In the second case, a DOM object is cloned, changed, and after that appended to the DOM. It is designed this way

because initially all the possible elements are written once in HTML. If there are no instances of these elements, the elements are removed, if there are several elements of same type, then the new ones are created dynamically by cloning the existing element from DOM. The design has some drawbacks, but the main reason for using it is that HTML code is separated from the JavaScript code. It would be hard to maintain HTML chunks that are created from strings in JavaScript . Also, it would be harder to read JavaScript code.

3. read - it is an object that contains functions for reading the metadata from the forms to the JavaScript objects by using the universal data reading function. The data reading function will be described next.
4. readData(selector, obj) - this function is responsible for automatic object updating with data read from the DOM. The "selector" parameter here is defining the object that is looked through for the values of fields named in the "obj" parameter. The "obj" parameter itself is the JavaScript object that is holding (or will hold, if there was no metadata before) the metadata of the element.

General event handlers and helper functions

The user action handlers are located in javascript.js file and are applied on document load.
Here are the most important functions described in detail:

1. openLoadingModal() - this function is initialised after a successful file upload. The function will display a modal loading window on top of the form. The loading window displays the simulation's status and progress. The function makes a AJAX request to the "/simulate" URL, that will start the business process model simulation. After this a simulation status polling begins.
2. getStatus() - this function is making requests to "/getStatus" URL, the controller mapped to this URL replies to the request with the simulation's status and progress in JSON. Function also updates the loading modal window with the latest progress data. If the simulation is finished, then this function retrieves the results and displays them to the user.

The form values are being validated before they are added to the metadata objects and sent for simulation. Validation occurs when the user changes the field that is validated. The form validation rules and logic are located in validate.js file. General logic behind the JavaScript validation is the following:

- the form fields that are validated are described in the validate object
- the required attributes for the field validation object are the following
    - “class”, which has to be the selector of the field that will be validated, i.e. “.costPerHour”;
    - “required”, if this is set false, the field is not validated if the value is not provided, if it is true, then the validation will be done against the regular expression [15] or custom validation function;
- the following fields are optional:
    - “regexp” - this field holds a regular expression that is executed against the value of the field, if result of the execution is true, then the field is valid;
    - “msg” - this is the message string, which is displayed when the field is not valid;
    - “custom” - this is the custom validation function, if the value passes validation, the function must return true, otherwise false;
    - “customMsg” - this is a string that will be displayed if the custom validation fails;

A sample validation object is in the following code snippet:

```
amount: {
  "class": ".amount",
  required: true,
  regexp: "^[0-9]+$",
  msg: "This field has to be a positive
integer!"
}
```

The regular expression seen in the snippet enables the user to only enter numeric characters. If the user enters an unallowed character, then an error message with the aforementioned text will be shown.

Summary

The objects and functions described previously are the core components of the BIMP's front-end application. Most of the key logic was also addressed, for more detailed information, please see the source code of the client side application [16].

3.3 Integration with Signavio

During the practical work a discussion with Signavio, a company providing online business process modelling software [17], took place. The subject of the discussion was an integration possibility of the two tools. The discussions resulted in a RESTful [18] interface, that enables any third party application to upload a process model to the BIMP's front-end application.

Since the 8th of May 2012, BIMP is available as a simulation tool in Signavio Process Editor (Figure 8).

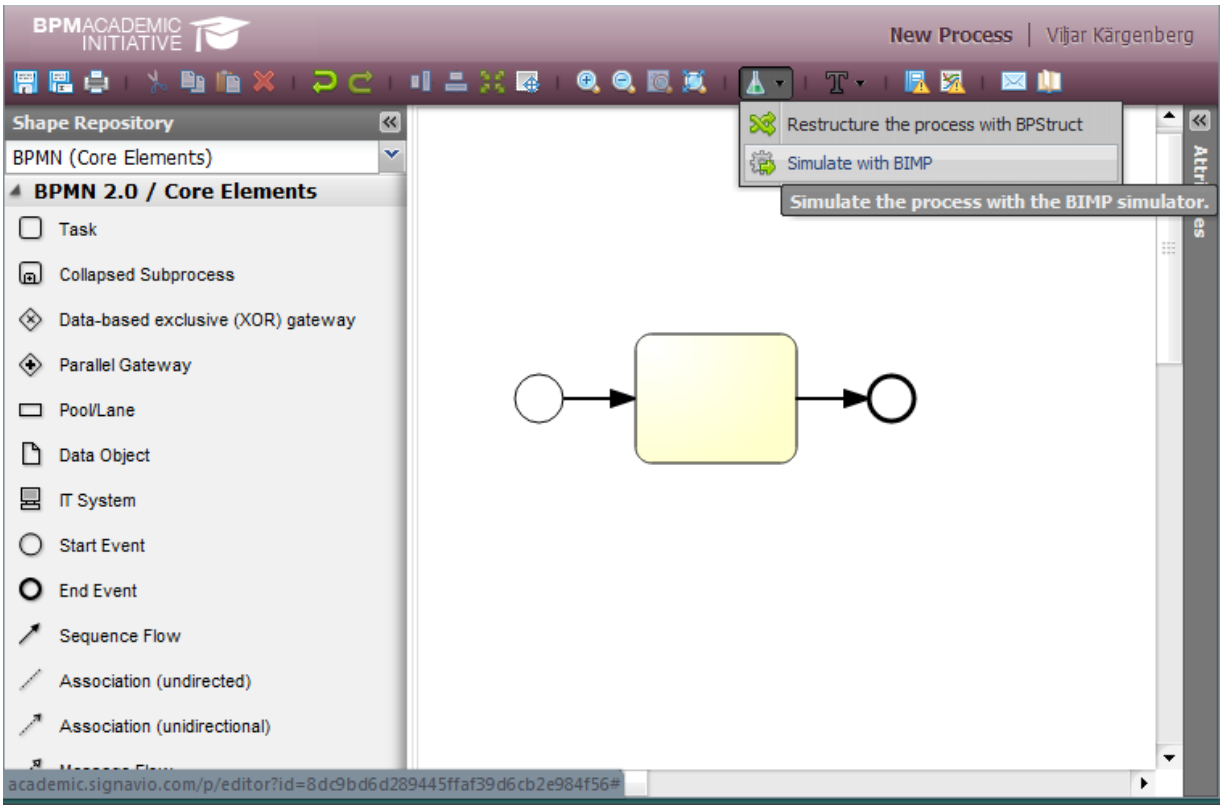

Figure 8. Screenshot of Signavio Process Editor, BIMP plugin.

The third-party integration solution was achieved with the following additions. A new controller, named SignavioController, was created and mapped to "/uploadsignavio" URL.

The controller is listening only POST requests and it reads the process model from the request's "file" parameter. Also, the referer is saved. The user will be redirected back to the referer URL if the model parsing failed for some reason. After the model is received, a code snippet with the sent file is added to the regular upload page. The code snippet starts the parsing process and if the model turns to be valid, the form will be displayed to the user. In case of invalid file, the user is notified with a popup and is directed back to the URL from where the request was made.

4. Testing

Testing is a really important ongoing process of any software project. When developing in a small team, it tends to be really limited. That is why it should be seamless and if possible, automated. These qualities help you save time and help you find possible problems quicker.

During the prototype development, the testing was quite limited. The test files available for the team were passing the parsing successfully, but once and then, when a new file, i.e. created with some other modelling tool, was parsed, they tended to fail because of reasons the developer team could not have foreseen. After a reasonable amount of fixes, it became tiresome to test all the files that were creating problems (regression testing), so the thought of automated tests arised. During the prototype development the automatic testing solution was not implemented, but as a part of the practical work of the thesis, the solution was developed. Next the used datasets, testing methods and fixed problems will be discussed and the testing solution described.

4.1 Datasets

There were 2 main datasets that were used during the automatic testing and bug-fixing. The first one, lets call it dataset A, consisted of 13 models, that were created with different modelling tools, some models were provided by Signavio. This dataset was mostly used during the development of the testing solution and early stages of functional testing. The second dataset, dataset B, consisted of 216 process models, that were part of the BIT process library, release 2009 [19]. Dataset B was used for functional and performance testing. Also, a part of it is now being used for continuous testing.

4.2 Functional testing

Process

The aim of the automated process is to simulate a regular use case with some exceptions. The actual user interaction is not simulated. The main idea behind the functional testing is to find

out, if a simulation with a model is possible. This means that the model has to be parsed, information extracted and displayed without any problems. Also, the simulation process itself should be successful. If any of these actions fail, as much information as possible should be gathered about the occurred error to make the problem identification and fixing as quick as possible. To carry out this design, a testing solution was developed, keeping the simplicity of problem identification as the main goal.

The functional testing also occurred during the Business Process Management course [20], where students were using BIMP to analyse the models they had created. Some models showed additional problems and these problems were also solved. Also employees from Signavio and University of Tartu contributed to the functional testing.

Testing solution

The testing solution consists of a JavaScript module and a backend controller. The JavaScript module is responsible for running the client application's processes, error details extracting and communicating with the controller. Controller, on the other hand, is providing the models for the client application that are going to be simulated, also controller is generating a simulation report, where the information about successful and unsuccessful phases of the simulations are separately displayed. For unsuccessful events, a stacktrace of the error is available. If the error happened in client application, then the stack-trace from there is displayed, but if the error is occurring when the model is simulated or the histogram data is being calculated, then the exception is retrieved and is also available. A sample simulation report can be seen on the Figure 9. The modal window displays an exception that happened during simulation of the model.

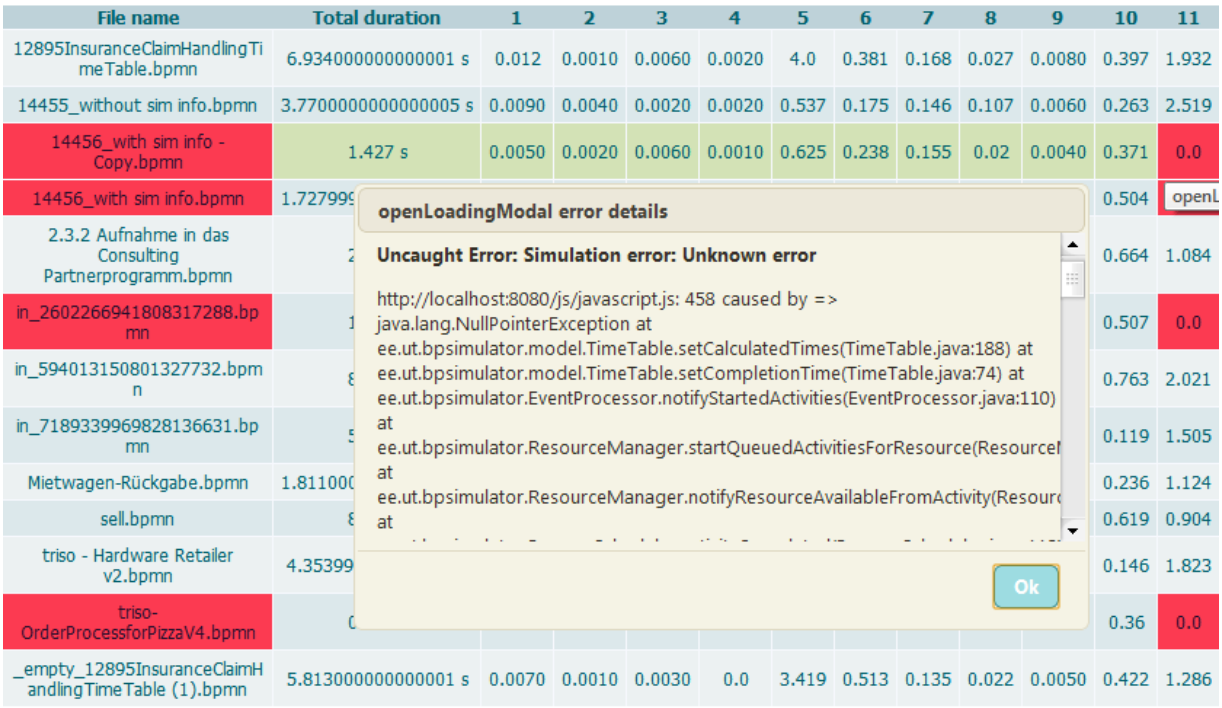

Figure 9. A sample simulation report showing a detailed view of an error.

Results

The functional testing revealed a number of issues. The issues in general and improvements made are described next. The list of improvements outlined here is representative and not exhaustive. Other bugs, not discussed here, were identified during the integration with Signavio and when BIMP was used in the context of the Business Process Management course.

1. Simulation failed due to NullPointerException when setting the completion time of the simulation. Solution: Bug fixed in the simulator library.
2. Simulation was not completed after several minutes and the simulator became unresponsive. Solution: Limit the maximum completed elements count to 3 millions.
3. No simulation result retrieved due to histogram values calculation. Solution: Bug fixes in the respective method.
4. Simulation results showing unrealistic waiting times. Solution: Fixes in simulator library.

5. Histograms that should display simulation summary were missing in several cases. Solution: Multiple bug fixes in the method calculating the histogram values and value intervals. Rewrote the logic to follow OOP principle.
6. The form validation failed when the XOR gateway had odd (excluding 1) count of the possible paths and default automatically calculated values were used. Solution: Value calculation was rewritten.
7. If only one instance was simulated, most of the histograms were not available on the results page. Solution: Histogram value calculations were overlooked and changes to fix the issue were made.
8. No simulation results displayed when re-submitting the file after "Back to edit data" button was used. Solution: Fix in respective function.
9. File parsing failed with models from dataset B. Solution: A fix was added to handle the case when BPMN elements like task were missing "name" attribute.
10. BIMP was not handling empty subprocesses as tasks. Solution: The web-application and simulation library were extended to fulfill the requirement.
11. The simulation fails when the simulation process takes less than 1ms. Solution: To be provided.

4.3 Performance testing

During the BIMP front-end prototype development no attention was paid to the performance of the client side application, because the focus was on functionality and development speed. The JavaScript performance issue first revealed itself when larger process models were used for testing. The application became unresponsive for seconds when the file was being parsed and analysed, but at that point, this delay was acceptable and no further effort regarding the issue was made.

When developing the testing solution a decision was made to measure the time spent on the key subtasks of the regular use case. This enabled to see how the different parts of application are performing. The test results with the dataset B were considerably bad. Summary of the test results can be seen in Table 1. With some files, the process took 3 minutes, which was unacceptable. On average, the completion time for one file was 20.4 seconds, and only 5% of

that time was spent on the actual simulation process. Subtask 5 took 90% of total simulation time. These results were surprising and showed, that the front-end parsing is a real bottleneck of the whole application. For more detailed overview please see the Test report 1 (Dataset B) in the Appendix.

Subtask	Avg time spent on subtask (seconds)	% of total
1	0,0	0,0
2	0,0	0,0
3	0,0	0,0
4	0,0	0,0
5	18,5	90,4
6	0,2	0,8
7	0,1	0,6
8	0,0	0,1
9	0,0	0,0
10	0,6	2,8
11	1,0	5,1
All	20,4	100,0

Table 1. Summary of the test results with dataset B before performance fixes.

As seen from the table above, the most problematic was subtask 5. The subtask 5 is readConditionExpressions() function, which was discussed in chapter 2.2. High time consumption was the result of too frequent XML document (of simulation model) accessing, when parsing the gateways. To solve the issue, the need for XML document access was minimized and also a caching solution for the most occurred searches was implemented. As the result of these changes, the performance improvement was remarkable. For detailed overview please see the Test report 2 (Dataset B) in the Appendix. The summary of the test results with the dataset B after the changes is seen in Table 2. The subtask 5, which previously

took 18,5 seconds, took only 0.2 seconds after changes. This is about 92 times faster. The average completion time for one file dropped from 20.4 seconds to 1.7 seconds, which is 12 times faster. A graph, representing the difference between the execution times, can be seen in Figure 10.

Subtask	**Avg time spent on subtask (seconds)**	**% of total**
1	0,0	0,3
2	0,0	0,1
3	0,0	0,2
4	0,0	0,0
5	0,2	10,8
6	0,1	7,2
7	0,1	5,4
8	0,0	1,0
9	0,0	0,4
10	0,4	21,3
11	0,9	53,2
All	1,7	100,0

Table 2. Summary of the test results with dataset B after performance fixes.

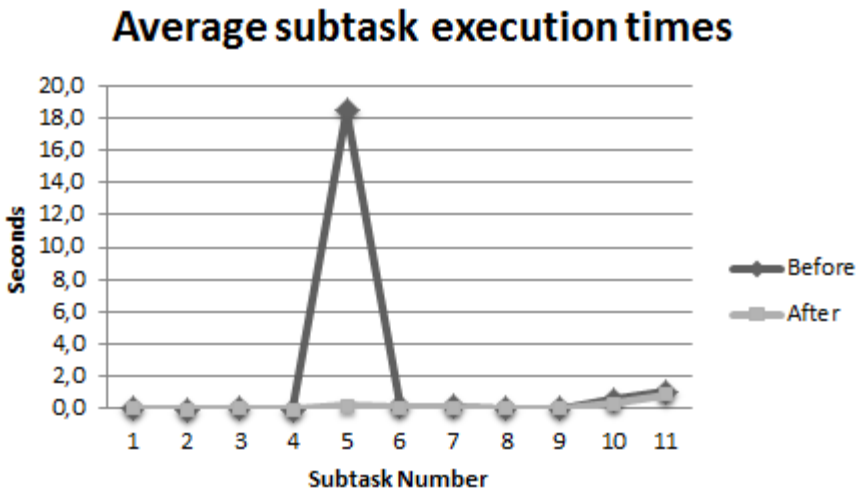

Figure 10. Average subtask execution times before and after performance fixes.

4.4 Summary

Testing revealed several bugs in the web-application, but also in the simulation library. In total, 11 issues were found and fixed (grouping all the histogram limit calculation issues into one). Out of these, 4 were from the simulator library. In order to continue the testing and find the issues as soon as possible, the continuous testing solution was set up. The solution consists of the automated testing solution, that was integrated into a simple Selenium [21] powered test. When Selenium test is run, it initializes the automated testing solution and waits for it to complete. When completed, the test downloads the simulation report and saves it on the project's workspace. For the process to be as seamless as possible, the project is built and tests are run nightly. For this, a continuous integration (CI) service provided by CloudBees [22] is used. The CI service is extended to use the SauceLabs OnDemand [23] service, that enables to run Selenium tests in cloud. As the result of this combination, when build is successful and test solution finishes, a nightly test report is available, showing the simulation status of 50 models from dataset B. Also a video recording of the browser's behaviour is provided by the SauceLabs OnDemand service. These outputs enable to keep an eye on new changes in the codebase, that might affect or break the overall simulation process. It also reduces the possibility of buggy solution ending up in live environment, because if build or tests fail, then the owner of the CI service is notified by email.

5. Conclusion

In this thesis the concepts of business process modelling and simulation were described. Specifically, the thesis focused on BIMP - an online simulator of business process models captured in the BPMN process modelling notation. An initial prototype of the online interface of BIMP was developed during a team-based software project. The project was short-term – 3 months. As a result, the prototype produced from this project was insufficiently tested, had performance issues, was not integrated with other modelling tools, and was not designed for extensibility.

To address these problems, this thesis provides the following contributions:

- A detailed documentation of the BIMP web-application architecture;
- An automated testing infrastructure to support future development of BIMP;
- Integration of BIMP with a commercial process modelling tool;
- Several improvements and bug-fixes to BIMP based on feedback from automated testing and user testing.

Although BIMP was tested with several datasets, there are still modelling tools whose process models were not included to the datasets and hence may create problems for the application. At the moment the majority of constructions available in BPMN 2.0 are supported, but there are still some of them remaining to implement to gain full coverage of BPMN 2.0 [4]. Also, currently only browsers, having the support for HTML5's File API, are capable of using the application without problems. A support for older browsers could be added by providing some workaround for client side file parsing, which is currently done with the functionality provided by the File API. In addition, the form for entering metadata could be more interactive and could resemble the business process diagram more, so that the environment would be more familiar for the user.

The source code of the BIMP online simulator (excluding the source code of the simulation engine) is available as an open-source project from [16].

Onlain äriprotsessimudelite simulaator

Bakalaureusetöö (6 EAP)
Viljar Kärgenberg
Resümee

Äriprotsesside modelleerimine ning nende simuleerimine on äriprotsesside juhtimise tähtsad komponendid. Äriprotsesside simulatsioon on laialdaselt kasutatav viis analüüsimaks äriprotsesside mudeleid, pöörates tähelepanu just jõudluse ning efektiivsuse mõõtmetele nagu aeg, maksumus ning ressursside hõivatus. Ettevõtted parandavad oma jõudlust, analüüsides regulaarselt väärtust ning panust, mida äriprotsessi erinevates osades lisatakse. Äriprotsessid on pidevates täiustustsüklites, milles protsessi disain ning rekonstruktsioon mängivad olulist rolli.

BIMP on äriprotsessimudelite simulaator, mille onlain liidese prototüüp valmis meeskonnapõhise tarkvaraprojekti raames. Kuna antud projekt oli lühiajaline, siis valminud prototüüp oli vähesel määral testitud, oli jõudlusprobleemidega ning polnud integreeritud modelleerimistööriistadega. Antud bakalaureusetöö keskendubki BIMP onlain äriprotsessi-mudelite simulaatori arhitektuuri kirjeldamisele, rakenduse testimisele ning täiendamisele.

Käesoleva bakalaureusetöö esimeses osas kirjeldadakse äriprotsesse, nende modelleerimist ning simuleerimist. Töö teises osas antakse ülevaade BIMP'i arhitektuurist ning veebirakenduse põhikomponentidest, sealjuures on põhirõhk kasutajapoolse rakenduse ehituse kirjeldusel. Lisaks kirjeldatakse ka integratsioonilahendust Signavio äriprotsessimudelite modelleerimistarkvaraga. Bakalaureusetöö kolmandas osas tutvustatakse BIMP simulaatori testimisprotsessi ning valminud automaattestide jooksutamise lahendust. Lisaks käsitletakse ka leitud vigu ning tehtud parandusi. Samuti tuuakse välja kliendipoolse rakenduse jõudlustestide tulemused ning tehtud täiendused.

Antud bakalaureusetöö tulemid on järgmised:

1. BIMP veebirakenduse detailne dokumentatsioon;
2. Automaattestimist võimaldav infrastruktuur, toetamaks BIMP'i arendust tulevikus;

3. BIMP’i integratsioon kommertstarkvaraga, mis võimaldab äriprotsesside modelleerimist;
4. Mitmed täiendused ning veaparandused BIMP’i veebirakenduses, tuginedes automaattestide ning kasutajate testide tulemustele.

References

[1] Business Process Management
http://en.wikipedia.org/wiki/Business_Process_Management (14.05.2012)

[2] L. Garcia-Banuelos and M. Dumas. "Towards an Open and Extensible Business Process Simulation Engine". *In Proceedings of the 10th Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, University of Aarhus, Denmark*, 2009.

[3] M.H. Jansen-Vullers and M. Netjes. "Business Process Simulation - A Tool Survey". *In Proceedings of the 7th Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, University of Aarhus, Denmark*, 2006.

[4] M. Abel. "Lightning Fast Business Process Simulator". Master Thesis, *Institute of Computer Science, University of Tartu*, June 2011.

[5] University of Tartu, MTAT.03.138 Software Project, Team 1
http://courses.cs.ut.ee/2011/tvp/Teams/Team1 (14.05.2012)

[6] Object Management Group (OMG),
http://www.omg.org (14.05.2012)

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

[8] ProM Framework
http://www.promtools.org/prom6 (14.05.2012)

[9] jQuery JavaScript Library
http://jquery.com (14.05.2012)

[10] AJAX
http://en.wikipedia.org/wiki/Ajax_%28programming%29 (14.05.2012)

[11] Spring MVC
http://www.springsource.org (14.05.2012)

[12] BIMP Simulation Library
http://code.google.com/p/ut-bpsimulator/downloads/list (14.05.2012)

[13] JSON
http://www.json.org (14.05.2012)

[14] Document Object Model

http://en.wikipedia.org/wiki/Document_Object_Model (14.05.2012)

[15] Regular Expression

http://en.wikipedia.org/wiki/Regular_expression (14.05.2012)

[16] BIMP simulator Homepage in Google Project

http://code.google.com/p/bimp-simulator (14.05.2012)

[17] Signavio

http://www.signavio.com/en.html (14.05.2012)

[18] Representational State Transfer (REST)

http://en.wikipedia.org/wiki/Representational_state_transfer (14.05.2012)

[19] BIT process library, release 2009

http://www.zurich.ibm.com/csc/bit/la1.html (14.05.2012)

[20] University of Tartu, MTAT.03.231 Business Process Management

http://courses.cs.ut.ee/2012/bpm (14.05.2012)

[21] Selenium

http://seleniumhq.org (14.05.2012)

[22] CloudBees Platform

http://www.cloudbees.com (14.05.2012)

[23] SauceLabs OnDemand

https://saucelabs.com/ondemand (14.05.2012)

Appendices

A. Test report 1 (Dataset B)

http://bimp-simulator.googlecode.com/files/1-200_IBM.htm (14.05.2012)

B. Test report 2 (Dataset B)

http://bimp-simulator.googlecode.com/files/1-200_IBM_gateway_caching_performance_fixes_no_duplicate_files.htm (14.05.2012)

C. A sample Business process model in BPMN 2.0 notation


```
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI"
xmlns:omgdc="http://www.omg.org/spec/DD/20100524/DC"
xmlns:omgdi="http://www.omg.org/spec/DD/20100524/DI"
xmlns:signavio="http://www.signavio.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" exporter="Signavio Process
Editor, http://www.signavio.com" exporterVersion="5.4.1"
expressionLanguage="http://www.w3.org/1999/XPath" id="sid-3e2a76e7-2201-4919-b3c8-
c61afaa84158" targetNamespace="http://www.signavio.com/bpmn20"
typeLanguage="http://www.w3.org/2001/XMLSchema"
xsi:schemaLocation="http://www.omg.org/spec/BPMN/20100524/MODEL
http://www.omg.org/spec/BPMN/2.0/20100501/BPMN20.xsd">
   <process id="sid-22a65ac6-268e-4adb-8155-b55076a364df" isExecutable="false">
      <startEvent id="sid-B11F91C0-9B86-48EB-9DBC-E0C1E64CADA8" name="">
         <extensionElements>
            <signavio:signavioMetaData metaKey="bgcolor" metaValue="#ffffff"/>
         </extensionElements>
         <outgoing>sid-762623E0-1CA1-42A5-A6A3-365C7C8C903D</outgoing>
      <documentation id="bbdb0055-7c86-8a1d-73f3-
050e4c08c873">{"arrivalRateDistribution":{"type":"fixed","mean":0,"value":7200,"stde
v":0,"min":0,"max":0,"timeUnit":"hours"},"instances":"1000","resources":{"ClaimHandl
er":{"name":"Claim Handler","costPerHour":"10","amount":"2"}},"startAt":"2012-05-04
22:10:56","timetable":{"*":{"Mon-Fri":"09:00:00-
17:00:00"}},"currency":"EUR"}</documentation></startEvent>
      <task completionQuantity="1" id="sid-16B5CA0F-58DD-42EE-9F86-A6252BAA7493"
isForCompensation="false" name="Quotation Handling" startQuantity="1">
         <extensionElements>
            <signavio:signavioMetaData metaKey="bgcolor" metaValue="#ffffcc"/>
            <signavio:signavioMetaData metaKey="risklevel" metaValue=""/>
            <signavio:signavioMetaData metaKey="externaldocuments" metaValue=""/>
         </extensionElements>
         <incoming>sid-762623E0-1CA1-42A5-A6A3-365C7C8C903D</incoming>
         <outgoing>sid-B0EC7F00-1A64-447B-9DBA-4570F2287E1E</outgoing>
      <documentation id="57a42c1e-da6c-1512-b9d1-
565049604b9e">{"durationDistribution":{"type":"uniform","mean":0,"value":1800,"stdev
```

```
":0,"min":900,"max":3600,"timeUnit":"minutes"},"resource":"ClaimHandler","fixedCost"
:"0","name":"Quotation Handling"}</documentation></task>
      <task completionQuantity="1" id="sid-2DCAD416-6A01-458B-98A7-C2ADEBA0101C"
isForCompensation="false" name="Approve  Order" startQuantity="1">
         <extensionElements>
            <signavio:signavioMetaData metaKey="bgcolor" metaValue="#ffffcc"/>
            <signavio:signavioMetaData metaKey="risklevel" metaValue=""/>
            <signavio:signavioMetaData metaKey="externaldocuments" metaValue=""/>
         </extensionElements>
         <incoming>sid-B0EC7F00-1A64-447B-9DBA-4570F2287E1E</incoming>
         <outgoing>sid-F1DFF9B4-22EA-47F4-963D-27655F80C27B</outgoing>
      <documentation id="f41cc190-0755-48cc-d980-
e49595f6ba1b">{"durationDistribution":{"type":"uniform","mean":"0","value":"0","stde
v":"0","min":"30","max":"90","timeUnit":"seconds"},"resource":"ClaimHandler","fixedC
ost":"0","name":"Approve  Order"}</documentation></task>
      <exclusiveGateway gatewayDirection="Diverging" id="sid-EE140FDB-39D2-4E18-
B183-6893B7616DA0" name="">
         <extensionElements>
            <signavio:signavioMetaData metaKey="bgcolor" metaValue="#ffffff"/>
         </extensionElements>
         <incoming>sid-F1DFF9B4-22EA-47F4-963D-27655F80C27B</incoming>
         <outgoing>sid-98F6546B-FE0D-42F2-86D9-11A99411708A</outgoing>
         <outgoing>sid-D9ECE5EA-922E-4507-AB4D-E9C74764A18F</outgoing>
      </exclusiveGateway>
      <endEvent id="sid-097B7E00-28C5-498F-902D-5F8B3EE8A732" name="">
         <extensionElements>
            <signavio:signavioMetaData metaKey="bgcolor" metaValue="#ffffff"/>
         </extensionElements>
         <incoming>sid-98F6546B-FE0D-42F2-86D9-11A99411708A</incoming>
      </endEvent>
      <parallelGateway gatewayDirection="Diverging" id="sid-290BB7BE-E7EF-485A-B756-
11D22CF7B11E" name="">
         <extensionElements>
            <signavio:signavioMetaData metaKey="bgcolor" metaValue="#ffffff"/>
         </extensionElements>
         <incoming>sid-D9ECE5EA-922E-4507-AB4D-E9C74764A18F</incoming>
         <outgoing>sid-2DA95B6B-8456-4D26-9CCC-4DDBBE28789F</outgoing>
         <outgoing>sid-3E03B54A-553A-4020-90F8-C6FBBCDD4E65</outgoing>
      </parallelGateway>
      <task completionQuantity="1" id="sid-6E878226-A985-483D-831A-DECA413B597C"
isForCompensation="false" name="Order  Handling" startQuantity="1">
         <extensionElements>
            <signavio:signavioMetaData metaKey="bgcolor" metaValue="#ffffcc"/>
            <signavio:signavioMetaData metaKey="risklevel" metaValue=""/>
            <signavio:signavioMetaData metaKey="externaldocuments" metaValue=""/>
         </extensionElements>
         <incoming>sid-2DA95B6B-8456-4D26-9CCC-4DDBBE28789F</incoming>
         <outgoing>sid-2DCE9070-C9FC-42A6-BC15-FA18A2C8C577</outgoing>
      <documentation id="f89bcd7d-5a3b-511e-a026-
997bcd9485bd">{"durationDistribution":{"type":"fixed","mean":0,"value":3600,"stdev":
0,"min":0,"max":0,"timeUnit":"minutes"},"resource":"ClaimHandler","fixedCost":"5","n
ame":"Order  Handling"}</documentation></task>
      <parallelGateway gatewayDirection="Converging" id="sid-78635E96-C8C4-4042-
9236-298598C92021" name="">
         <extensionElements>
            <signavio:signavioMetaData metaKey="bgcolor" metaValue="#ffffff"/>
         </extensionElements>
         <incoming>sid-2DCE9070-C9FC-42A6-BC15-FA18A2C8C577</incoming>
         <incoming>sid-6540F36A-3DBA-4ACE-B57C-02B0BEBA7942</incoming>
         <outgoing>sid-EF413DE7-814F-4158-89E1-83C579732D74</outgoing>
      </parallelGateway>
      <task completionQuantity="1" id="sid-95C3C2E4-0C4F-45DA-A5C0-7509B7EFE76A"
isForCompensation="false" name="Shipping Handling" startQuantity="1">
```

```
         <extensionElements>
            <signavio:signavioMetaData metaKey="bgcolor" metaValue="#ffffcc"/>
            <signavio:signavioMetaData metaKey="risklevel" metaValue=""/>
            <signavio:signavioMetaData metaKey="externaldocuments" metaValue=""/>
         </extensionElements>
         <incoming>sid-3E03B54A-553A-4020-90F8-C6FBBCDD4E65</incoming>
         <outgoing>sid-6540F36A-3DBA-4ACE-B57C-02B0BEBA7942</outgoing>
      <documentation id="65e09381-5f4f-18ed-dc4f-
50ab133d27eb">{"durationDistribution":{"type":"fixed","mean":0,"value":10800,"stdev"
:0,"min":0,"max":0,"timeUnit":"hours"},"resource":"ClaimHandler","fixedCost":"30","n
ame":"Shipping Handling"}</documentation></task>
      <task completionQuantity="1" id="sid-A0D467C2-FE53-4EED-96FD-FD1BE0C921E5"
isForCompensation="false" name="Review Order" startQuantity="1">
         <extensionElements>
            <signavio:signavioMetaData metaKey="bgcolor" metaValue="#ffffcc"/>
            <signavio:signavioMetaData metaKey="risklevel" metaValue=""/>
            <signavio:signavioMetaData metaKey="externaldocuments" metaValue=""/>
         </extensionElements>
         <incoming>sid-EF413DE7-814F-4158-89E1-83C579732D74</incoming>
         <outgoing>sid-5E2ED906-B60A-4A84-A4D8-BE109BD997EE</outgoing>
      <documentation id="f92ead1b-aa5a-c74d-bfed-
318b52a5cf7f">{"durationDistribution":{"type":"fixed","mean":0,"value":3600,"stdev":
0,"min":0,"max":0,"timeUnit":"hours"},"resource":"ClaimHandler","fixedCost":"0","nam
e":"Review Order"}</documentation></task>
      <endEvent id="sid-2AC97580-58DC-42B4-93B6-A434390AD2D9" name="">
         <extensionElements>
            <signavio:signavioMetaData metaKey="bgcolor" metaValue="#ffffff"/>
         </extensionElements>
         <incoming>sid-5E2ED906-B60A-4A84-A4D8-BE109BD997EE</incoming>
      </endEvent>
      <sequenceFlow id="sid-762623E0-1CA1-42A5-A6A3-365C7C8C903D" name=""
sourceRef="sid-B11F91C0-9B86-48EB-9DBC-E0C1E64CADA8" targetRef="sid-16B5CA0F-58DD-
42EE-9F86-A6252BAA7493"/>
      <sequenceFlow id="sid-B0EC7F00-1A64-447B-9DBA-4570F2287E1E" name=""
sourceRef="sid-16B5CA0F-58DD-42EE-9F86-A6252BAA7493" targetRef="sid-2DCAD416-6A01-
458B-98A7-C2ADEBA0101C"/>
      <sequenceFlow id="sid-F1DFF9B4-22EA-47F4-963D-27655F80C27B" name=""
sourceRef="sid-2DCAD416-6A01-458B-98A7-C2ADEBA0101C" targetRef="sid-EE140FDB-39D2-
4E18-B183-6893B7616DA0"/>
      <sequenceFlow id="sid-98F6546B-FE0D-42F2-86D9-11A99411708A" name=""
sourceRef="sid-EE140FDB-39D2-4E18-B183-6893B7616DA0" targetRef="sid-097B7E00-28C5-
498F-902D-5F8B3EE8A732"><conditionExpression xsi:type="tFormalExpression"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">0.5</conditionExpression></sequenceFlow>
      <sequenceFlow id="sid-D9ECE5EA-922E-4507-AB4D-E9C74764A18F" name="Approved"
sourceRef="sid-EE140FDB-39D2-4E18-B183-6893B7616DA0" targetRef="sid-290BB7BE-E7EF-
485A-B756-11D22CF7B11E"><conditionExpression xsi:type="tFormalExpression"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">0.5</conditionExpression></sequenceFlow>
      <sequenceFlow id="sid-2DA95B6B-8456-4D26-9CCC-4DDBBE28789F" name=""
sourceRef="sid-290BB7BE-E7EF-485A-B756-11D22CF7B11E" targetRef="sid-6E878226-A985-
483D-831A-DECA413B597C"/>
      <sequenceFlow id="sid-3E03B54A-553A-4020-90F8-C6FBBCDD4E65" name=""
sourceRef="sid-290BB7BE-E7EF-485A-B756-11D22CF7B11E" targetRef="sid-95C3C2E4-0C4F-
45DA-A5C0-7509B7EFE76A"/>
      <sequenceFlow id="sid-2DCE9070-C9FC-42A6-BC15-FA18A2C8C577" name=""
sourceRef="sid-6E878226-A985-483D-831A-DECA413B597C" targetRef="sid-78635E96-C8C4-
4042-9236-298598C92021"/>
      <sequenceFlow id="sid-6540F36A-3DBA-4ACE-B57C-02B0BEBA7942" name=""
sourceRef="sid-95C3C2E4-0C4F-45DA-A5C0-7509B7EFE76A" targetRef="sid-78635E96-C8C4-
4042-9236-298598C92021"/>
```

```
      <sequenceFlow id="sid-EF413DE7-814F-4158-89E1-83C579732D74" name=""
sourceRef="sid-78635E96-C8C4-4042-9236-298598C92021" targetRef="sid-A0D467C2-FE53-
4EED-96FD-FD1BE0C921E5"/>
      <sequenceFlow id="sid-5E2ED906-B60A-4A84-A4D8-BE109BD997EE" name=""
sourceRef="sid-A0D467C2-FE53-4EED-96FD-FD1BE0C921E5" targetRef="sid-2AC97580-58DC-
42B4-93B6-A434390AD2D9"/>
   </process>
</definitions>
```