BUSINESS PROCESS SIMULATION
WITH R

Aantal woorden/ Word count: 6742

Yoren Van Couwenberghe
Stamnummer/ Student number: 01205551

Promotor/ Supervisor: Prof. dr. Frederik Gailly

Masterproef voorgedragen tot het bekomen van de graad van:
Master's Dissertation submitted to obtain the degree of:

Master of Science in Business Engineering

Academiejaar/ Academic year: 2016 - 2017
BUSINESS PROCESS SIMULATION
WITH R

Aantal woorden/ Word count: 6742

Yoren Van Couwenberghe
Stamnummer/ Student number : 01205551

Promotor/ Supervisor: Prof. dr. Frederik Gailly

Masterproef voorgedragen tot het bekomen van de graad van:
Master’s Dissertation submitted to obtain the degree of:

Master of Science in Business Engineering

Academiejaar/ Academic year: 2016 - 2017
PERMISSION

I declare that the content of this Master's Dissertation may be consulted and/or reproduced, provided that the source is referenced.

Naam student: Yoren Van Couwenberghe

Signature:
**Foreword**

This thesis, handles the subject ‘business process simulation in R’, that I wrote to graduate as a Business Engineer (in Data Analytics) at the University of Ghent. The idea of the subject came from Professor Gailly, professor in the Management Information Systems department of the University of Ghent. Later, I refined the objective of this thesis in collaboration with Professor Gailly to create a process model simulator in R. The whole process of creating this process model simulator and writing this thesis took almost 2 years from September 2015 until May 2017, where the last 6 months were the most intensive.

This thesis contributed to my skill set as a university student in 2 ways. Firstly, it improved my R-programming knowledge, which is a valuable competence nowadays. Thanks to this thesis, I could extend my prior R-programming knowledge from my data analytics courses and learn how to contribute to the R language myself by creating packages. Secondly, I learned how to write a scientific paper, which is a required skill for each university graduate.

To end, I want to thank Professor Gailly and his team to offer me the opportunity to work on this subject, as well as for their support during the process. The result as it is today would not be achieved without his guidance and ideas.

I’ll hope you will enjoy reading my thesis.

Yoren Van Couwenberghe

May 2017
Table of Contents

Foreword .............................................................................................................. I

Table of Contents .............................................................................................. III

List of abbreviations ........................................................................................ VI

List of tables ...................................................................................................... VI

List of figures ..................................................................................................... VI

List of R codes .................................................................................................. VI

1. Introduction .................................................................................................. 1

2. Background information about the simmer-package .................................. 3

3. Development Process ................................................................................ 5

   3.1 Dissociating the input of process model information from the input of additional information 6

   3.2 Automatically import information from a visual BPMN representation .......... 8

   3.3 Finalization of the R-package ................................................................ 9

4. Demonstration .............................................................................................. 11

5. Conclusion ................................................................................................... 21

6. References .................................................................................................. I

List of annexes ................................................................................................... I
1. Installing R & RStudio

2. Installing the BPMNsimulator-package

3. Complete code of demonstration (section 5)

4. Defining probabilities to follow alternative paths with loops

5. Importing BPMN models with subprocesses
List of abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPM</td>
<td>Business process management</td>
</tr>
<tr>
<td>BPS</td>
<td>Business process simulation</td>
</tr>
<tr>
<td>BPMN</td>
<td>Business process modelling and notation</td>
</tr>
<tr>
<td>EPC</td>
<td>Event-driven Process Chain</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical user interface</td>
</tr>
</tbody>
</table>

List of tables

Table 1: overview of the functions to initialize the storing system ........................................ 8
Table 2: Work Schedules in the claim handling department of the acquirer. ........................................ 12
Table 3: Interarrival times of the processes ................................................................. 12
Table 4: Activity durations and resource responsible for acquirer’s process .................................. 12
Table 5: Probabilities to follow alternative paths for acquirer’s process .................................. 13
Table 6: Activity durations and resource responsible for acquiree’s process ................................ 13
Table 7: Probabilities to follow alternative paths for acquiree’s process .................................. 14

List of figures

Figure 1: Link between trajectory object and simulation environment in the simmer-package ............... 5
Figure 2: Visual describing the information required for each BPMN element. ........................................ 6
Figure 3: Visual representation of the storage system in the BPMNsimulator package .................. 7
Figure 4: Interaction between the functions in the BPMNsimulator package .................................. 11
Figure 5: Acquirer’s BPMN model of the claim-handling process .................................................... 12
Figure 6: Acquiree’s BPMN model of the claim-handling process .................................................... 13
Figure 7: Part of the global environment after executing the import_BPMN()-function .................. 14
Figure 8: Part of dataframe received by the get_mon_arrivals()-function ........................................ 18

List of R codes

Code Snippet 1: Importing the BPMN-file .................................................................................. 14
Code Snippet 2: Specifying the activity duration of the ‘close claim’ activity. ........................................ 15
Code Snippet 3: Specifying the resource responsible for the ‘close claim’ activity. .............................. 15
Code Snippet 4: Specifying the probabilities after the ‘Information Complete?’ XOR-split .................. 15
Code Snippet 5: Calling the transform_BPMN()-function() in an efficient manner ...................... 16
Code Snippet 6: Add arrivals to a trajectory. .................................................................................. 16
Code Snippet 7: Define schedules and resources in your system. .................................................... 17
Code Snippet 8: Run the simulation for 3600 minutes .................................................................. 17
Code Snippet 9: Analyzing the results using the get()-functions ................................................... 18
Code Snippet 10: Plot()-functions returning quick visual overviews ............................................. 19
1. Introduction

A business process is a chain of tasks that should be executed in a certain order by resources of an organization to deliver a good or service. Last years, there is a gaining interest in the management of these business processes in organizations. This so called business process management (BPM) is responsible for visualizing, analysing, redesigning & monitoring existing business processes as well as designing, analysing and implementing new business processes (Dumas et al., 2013).

Business process simulation (BPS) is a process analysis method that allows the simulator to investigate potential business process changes in a computer simulation environment. BPS is among the most popular techniques for business process analysis due to its capability to assess modifications of a business process before implementing them (Bocciarelli et al., 2014). Analyzing modifications without real-life implementation results in a low-cost & risk-averse approach.

The rise of BPM has led to a necessity for representing business processes, achieved in the form of process models (e.g. BPMN models, EPC models, …). A process model is using a predefined syntax to visually describe the workflow of a business process (Kopp et al, 2015). Although this gives a detailed view of the specifics of a business process, it is also missing crucial information necessary for a simulation. Based on the Business Process Simulation model of Martin et al. (2014) and the research of Pereira & Freitas (2016) for BPMN models specifically, we conclude that the following information is necessary for a simulation but not present in a process model of a business process:

- The interarrival time: The time between 2 arrivals, frequently described by a function.
- Activity duration: The time it takes to execute an activity, frequently described by a function.
- The resource in the organization responsible for executing an activity in more detail
- Probability of following an alternative path
- Number of resources available in the system

In the early days of BPS, discrete event simulators were used to simulate business processes. These discrete event simulators are stand-alone applications with their own syntax (e.g. Arena). Even though a discrete event simulator gives a lot of modelling freedom for the user, this tool-specific syntax is a severe drawback for BPS for two reasons. First of all the necessary skills for using this tool should be present in the organization. Secondly, due to the rise of BPM, business processes are often represented by a process model (Kopp et al, 2015), leading to an inevitable mapping from the
process model syntax to the tool specific syntax which is increasing the effort necessary for setting up a simulation. Both drawbacks reduce the accessibility of a discrete event simulator as a business process simulator.

More recently, Business Process Modelling tools are expanding their services with build-in simulation capabilities (Pereira et al., 2016) offering organizations the possibility to simulate their business processes starting from their process model. Such a process model simulator is typically extracting the workflow information automatically from the business process model. The missing information (e.g. activity durations, ...) mentioned earlier can be inserted by making use of Graphical User Interfaces (GUIs). This approach of the process model simulators will be later referred to as the process model simulator method. By omitting the syntax mapping and using GUIs, these process model simulators are increasing the usability and accessibility of BPS. On the other hand, we still observe some shortcomings in these process model simulators.

Based on the research of Pereira et al. (2016) for the specific case of BPMN-simulators (process model simulators that are supporting the BPMN language), we conclude that the general drawbacks for process model simulators are:

- Inability to simulate multiple related business processes in one integrated simulation
- Limited functions available to define activity durations and interarrival times
- Not supporting work schedules for resources
- Limited result analysis possible

In the remainder of this thesis, we will investigate the potential of R as a business process simulator and how it can be used to solve the drawbacks of the currently available discrete event simulators and process model simulators. R is a statistical programming language that is often used in the context of data analysis projects. In this era of a growing amount of data streams and an increased interest in the analysis of these data streams, the popularity of R has quickly raised. Nowadays, many organizations employ people that master R programming and the people educated in R are expected to increase in the future. This together with the fact that R contains a Discrete Event Simulation package, motivated us to investigate the possibilities of R as a business process simulator.

The addressed research question is: ‘How can we create a business process simulator in R with a higher accessibility than discrete event simulators that increases the depth of analysis of the current process model simulators by eliminating their drawbacks, while not reducing on their usability and
accessibility? A higher accessibility than discrete event simulators means that syntax-mapping starting from a business process model needs to be avoided. Not reducing on the usability of a process model simulator means that the difference in required effort to set up a simulation is limited. Not reducing on the accessibility of a process model simulator means that using our simulator only requires basic R knowledge and someone without prior R or programming experience only needs a short learning period.

To answer this research question, a three-part methodology is used. Firstly, the existing capabilities of R and the business process model simulators to conduct a BPS are explored. In R, The simmer-package allows its users to conduct Discrete Event Simulations. However, as it is a Discrete Event Simulator, the user-friendliness and accessibility are too limited to comply with the requirements of our research question. The current process model simulators are indeed offering a higher usability and accessibility by automatically extracting the work flow of a business process from its process model but the previously listed drawbacks are limiting the scope of analysis.

Secondly, a tool in R is designed to reduce the usability and accessibility problem of the simmer-package. This tool should allow the user to set up a simulation via the process model simulator method, while extending the features of the current process model simulators to eliminate their shortcomings. The development of this tool consisted of 3 steps. As a first step, the moment information from the process model is inputted needed to be dissociated from the moment additional information is inputted. Secondly, a feature should be created that is able to automatically extract the information from a visual process model. Thirdly, the tool should be finalized so that it can be exposed to a larger audience.

As a last step, the newly created tool is demonstrated on a fictive case to illustrate its usability and accessibility, as well as its improvements over the existing process model simulators.

2. Background information about the simmer-package

To understand the remainder of this thesis, the reader should have a basic knowledge about the structure of the simmer-package, because it will be used as the underlying package of the created tool. The simmer-package has a 2 pieced-structure, a trajectory definition part and a simulation
environment definition part. In this section, the focus is on how these 2 parts are integrated with each other.

When setting up a simulation with the simmer-package, defining a trajectory object in R is the first step. The whole path an instance of the simulation should follow is defined in this trajectory object. However, this trajectory object is not limited to the sequence flow that an instance should follow. It also simultaneously integrates information about the possibilities to follow alternative paths, the time to execute the different steps in the flow and the resources responsible for the execution. Also important to mention is that modifying a trajectory object after its creation is impossible.

When the trajectory object is created, the simulation environment should be defined. Defining this simulation environment should always be done with the trajectory object in mind. First of all, all the resources, which are responsible for executing tasks in the trajectory object, should be defined in the simulation environment. Information about the capacity and the queue size of the resources can be specified as well. Secondly, a generator should be added in the simulation environment. A generator is responsible for generating arrivals to a trajectory object, so it is essential to specify that trajectory object. A nice feature of the simulation environment is the possibility to add more than 1 generator, resulting in the feasibility of simulating multiple trajectory objects in 1 simulation environment. After creating the simulation environment, functions are available in the simmer-package to run the simulation and analyze the results afterwards. Figure 1 summarizes the linkage between the 2 parts.

The following characteristics of the simmer-package help to solve the drawbacks of the current process model simulators:

- The ability to define multiple generators in the simulation environment opens the possibility to simulate multiple dependent BPMN models in 1 environment.
- All possible R-functions as well as user-defined functions can be used to define activity durations and interarrival times.
- Schedules for resources can be integrated in the simulation environment.
- Detailed data about the instances and resources of the simulation can be retrieved to analyze the results
The most important takeaways of this phase in the development process are:

- The trajectory object simultaneously uses information that can be extracted from a process model and additional information.
- Once a trajectory object is created, it cannot be modified.
- By using the simmer-package as underlying package for our created tool, it is possible to eliminate the current drawbacks of the process model simulators.

3. Development Process

The current capabilities of R, in the form of the simmer-package, are too limited to conduct business process simulations in an efficient manner (see section 2). This was the incentive to create a process model simulator in R. The business process modelling language that will be supported by our simulator will be the Business Process Modelling and Notation (BPMN) language, which is seen as the industry-standard by Kunze & Weske (2016).

The created tool, which was later called the BPMNsimulator-package, is a wrapper around the existing simmer-package. This means that the package is relying on the simmer-package to conduct the actual simulation but is offering its user an alternative way to set up the simulation. By using the simmer-package as underlying package, the drawbacks of the current process model simulators are eliminated (see section 2). This section covers the development of the BPMNsimulator-package.
3.1 Dissociating the input of process model information from the input of additional information

A typical BPMN-simulator automatically extracts all information from a BPMN representation and enhances it with additional information in a later phase by requesting input from the user. This feature of a BPMN-simulator combined with the characteristics of the simmer-trajectory object created a need for an external storage system in R. It was infeasible to store the extracted BPMN information in a trajectory object and enrich it in a later phase with additional information (see most important takeaways in section 2).

Figure 2 represents the required information for each type of BPMN element to set up a simmer trajectory-object. The information that can be extracted from the BPMN representation is indicated in red, the additional information is indicated in black. Note that not all BPMN elements are represented. BPMN elements that are not important for a simulation model are omitted (e.g. data objects, ...) as well as gateways for which it is hard to define the probabilities of following the subsequent paths (e.g., or-splits, event-based-splits, ...). As each business process can only contain 1 start event, there is no need to keep track of the start event.

After analyzing Figure 2, the decision was made to use R-list objects as the storage system. The characteristics of an R-list are perfectly suited for our application:

- It is a data object in which it is possible to store multiple elements
- These different elements should not be of the same type, meaning that we can store all types of information in the same list
- Elements can be added, deleted and modified after the creation
- It is possible to assign a custom made class to an R-list, which can reduce the program effort required when setting up a simulation (see demonstration).

The idea behind the storage system is to create an R-list object for each element in the BPMN model. Dependent on the type of the BPMN-element (see Figure 2 for the types supported by our package), the R-list object will have different type specific elements. Figure 3 is giving a visual overview of the storage system. All BPMN-types will have 3 elements in common (name, prev_element and type), the other elements are type specific elements.

![Figure 3: Visual representation of the storage system in the BPMNsimulator package](image)

With the storage system created, the next step was to create functions to help us initialize the storage system. Due to the properties of a BPMN-simulator, the moment when information from the BPMN model is fed to the system should be separated from the moment that additional information is added.

Dissociating these 2 moments is achieved by creating 2 types of functions. The first type of functions are the *add-functions*, which are used to store extracted information from a BPMN model. For each BPMN-element type a different *add-function* is constructed. In the arguments of these *add-functions*, the name of the BPMN element should be specified as well as the information about the sequence flow by specifying the previous element in the BPMN model. Note that a join-type should have more than 1 previous element in the BPMN model, these should be all specified as a character vector as well as the split to which that join belongs. When an *add-function* is called, it creates an R-list object that is initialized with the correct elements dependent on the BPMN-element type. The name of that R-list will be the same as the name argument of the *add-function*.

Functions, that can be used to input additional information, are the *set-functions*. For a BPMN-element of the activity type, a simulation should know the activity duration and the resource
responsible for executing that activity. The `set_activity_duration()` and `set_resource_to_activity()`-functions can be used for that purpose. In the arguments of these functions the R-list object of the activity as well as the activity duration or the resource should be specified. These functions will enhance this R-list object with the correct information. A similar approach is used by the other set-functions to enhance the storage system with the intermediate event durations and the probabilities after a XOR-split.

<table>
<thead>
<tr>
<th>EXTRACT INFORMATION FROM BPMN MODEL</th>
<th>INPUT ADDITIONAL INFORMATION</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>add_activity(name, prev_element)</code></td>
<td><code>set_activity_duration</code></td>
</tr>
<tr>
<td><code>add_intermediate_event(name, prev_element)</code></td>
<td><code>set_resource_to_activity</code></td>
</tr>
<tr>
<td><code>add_stop_event(name, prev_element)</code></td>
<td><code>set_intermediate_event_duration</code></td>
</tr>
<tr>
<td><code>add_AND_split(name, prev_element)</code></td>
<td><code>set_probabilities_to_XOR_split</code></td>
</tr>
<tr>
<td><code>add_AND_join(name, prev_element, of_split)</code></td>
<td></td>
</tr>
<tr>
<td><code>add_XOR_split(name, prev_element)</code></td>
<td></td>
</tr>
<tr>
<td><code>add_XOR_join(name, prev_element, of_split)</code></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: overview of the functions to initialize the storing system

Now that the storage system with the necessary functions to populate and enrich it is in place, all information for a simmer-trajectory object can be stored. The next step was to construct a support function that takes the storage system (an R-list object for each BPMN element) as input and delivers the correct simmer-trajectory as output. This phase of the research project required the biggest programming effort. The result is a `transform_BPMN()`-function that takes all the R-lists of our storage system as arguments and returns the corresponding simmer-trajectory object.

### 3.2 Automatically import information from a visual BPMN representation

In the current state of our BPMNsimulator-package, the user still needs to initialize the storage system by using the `add-functions` presented in section 3.1. Even though these `add-functions` are closely related to a visual BPMN representation, a mapping is still required. In other process model simulators this mapping is not necessary, the user can directly import a process model. To ensure that the accessibility and usability requirement of the research question is met, the current features of the tool need to be extended with a function that is able to directly initialize the storage system based on a visual BPMN model.

Most Business Process Modelling tools offer the option to export a BPMN model as an XML-file, allowing users to save the model locally. R as a data analysis tool supports a lot of extensions to
import data, among them are XML-files. An `import_BPMN()`-function is created, which accepts the file path of the BPMN model’s XML-file as input and returns an initialized storage system. As a result, translating a BPMN model to a simmer trajectory object can now be done by making use of only 6 functions. This `import_BPMN()`-function, the 4 `set-functions` and the `transform_BPMN()`-function presented in section 3.1.

### 3.3 Finalization of the R-package

As R is an open-source project, users can contribute to the expansion of the language by writing and distributing packages. An R-package typically consists of functions created by the package writer. Other users can download these packages and use its functions. Putting our functions in an R-package is the ideal way to distribute them.

Until now, our tool offers an alternative easier manner to define a simmer-trajectory starting from a BPMN model. To increase the accessibility even further, the features of the tool are extended with the `create-functions` to ease the set up of the simulation environment. In a simmer simulation environment, generators are added to create arrivals and the available resources responsible for executing the processes of these generators are added as well. Revisiting the list of prerequisite additional information for process model simulators from the introduction, these 2 prerequisites were still left wanting.

First of all, The `create_arrivals()`-function was developed to generate arrivals by defining the interarrival time. With the `create_resource()`-function the user can now add the available resources to the system. The first time one of these 2 functions is called, a new variable called `simulation_environment` is created, which stores the information of that `create-function`. The next time the user calls one of these 2 functions, the new information is added to the existing `simulation_environment` variable.

The reason that the BPMNCsimulator-package is using the same structure as the simmer-package with regards to the trajectory and simulation environment split is twofold. First of all, this offers the user the ability to simulate related business processes in one simulation exercise (see section 4 for an illustration), which is a characteristic the current process model simulators are lacking. Secondly, it allows the user to define the simulation environment using the simmer-package functions. Even though these functions are a little more complex, they offer more options that can be useful in specific simulation exercises (e.g. setting priorities).
To increase the usability of the tool, the functions were enhanced with error-throwing capabilities. These errors prohibit the user to input values in the arguments of the functions that will cause an error in a later phase of the simulation process. By writing a manual the usability was increased even further. A manual is describing how each function should be used and can be retrieved easily from within the tool itself when help is required.

The BPMNsimulator-package in R is the final result of this research project. The structure of the package is visualised in figure 4. The import_BPMN()-function can be used to automatically map a visual BPMN model to our internal storage system. With the 4 set-functions, it is possible to enhance this internal storage system with additional information. A next step is to map this internal storage system to the simmer trajectory object by making use of the transform_BPMN()-function. The last phase is to link this trajectory object to a simulation environment object by making use of the create_arrivals()- and create_resource()-functions.

BPMN models that can be imported with the import_BPMN()-function should comply with the following conditions:

- BPMN model should be modelled using the BPMN 2.0 standard.
- All BPMN-elements should have unique names.
- Activities, stop events and intermediate events can only have 1 incoming and 1 outgoing sequence flow. This means that:
  - Loops should be modelled using XOR-splits and XOR-joins
  - AND-gates should be modelled using AND-splits and AND-joins
- The BPMN model can only have 1 start event.
- The BPMN model cannot contain OR-gateways or event based-gateways.

Once a simmer simulation environment object is created, the simulation can be executed and analyzed using the functions of the simmer-package (see demonstration).
4. Demonstration

Demonstrating the package will be conducted by using BPMN models of business processes modelled in Signavio and using RStudio as Integrated Development Environment (IDE) for R (see appendix for the installation). The case that will be simulated is described below, note that the case is no real-life case and is invented for demonstration purposes based on a case in the book of Dumas et al. (2013).

An insurance company has recently done an acquisition of a smaller competitor. One of the cost reduction measures after the take-over was to integrate both claim-handling departments of the 2 companies. Both companies had their own claim-handling process, the BPMN models of these processes can be found in figure 5 and figure 6. As a first step in the integration, both processes would remain unchanged but would be both handled by the acquirer’s claim-handling department. The management wanted to assess whether the current capacity of the acquirer’s claim-handling department was sufficient to handle both processes by conducting a simulation. More information about the activity durations, probabilities to follow alternative paths, resources responsible to execute the activities, interarrival rates and the capacity of the department is introduced in the tables 2, 3, 4, 5, 6 and 7.

Description of the illustrative case
Resource Schedule

<table>
<thead>
<tr>
<th>Resource</th>
<th>08:00-12:00</th>
<th>12:00-16:00</th>
<th>16:00-20:00</th>
</tr>
</thead>
<tbody>
<tr>
<td>Call centre 1</td>
<td>40</td>
<td>55</td>
<td>25</td>
</tr>
<tr>
<td>Call centre 2</td>
<td>40</td>
<td>55</td>
<td>25</td>
</tr>
<tr>
<td>Claim handling office</td>
<td>150</td>
<td>150</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2: Work Schedules in the claim handling department of the acquirer.

<table>
<thead>
<tr>
<th>Process</th>
<th>Interarrival time (in minutes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acquirer’s process</td>
<td>Equal chance between 0.1, 0.3 and 0.8</td>
</tr>
<tr>
<td>Acquiree’s process</td>
<td>70 % chance for 0.2 and 30 % chance for 0.4</td>
</tr>
</tbody>
</table>

Table 3: Interarrival times of the processes

![BPMN model of the claim-handling process]

Figure 5: Acquirer’s BPMN model of the claim-handling process

<table>
<thead>
<tr>
<th>Activity</th>
<th>Activity Duration (in minutes)</th>
<th>Resource responsible</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check if customer has...</td>
<td>Exponential distribution: Mean = 1</td>
<td>Call centre 1</td>
</tr>
<tr>
<td>Register claim</td>
<td>Exponential distribution: Mean = 9</td>
<td>Call centre 1</td>
</tr>
<tr>
<td>Check if customer has..._2</td>
<td>Exponential distribution: Mean = 1</td>
<td>Call centre 2</td>
</tr>
<tr>
<td>Register claim_2</td>
<td>Exponential distribution: Mean = 9</td>
<td>Call centre 2</td>
</tr>
<tr>
<td>Determine likelihood of the claim</td>
<td>Exponential distribution: Mean = 2</td>
<td>Claims handling office</td>
</tr>
<tr>
<td>Assess claim</td>
<td>Exponential distribution: Mean = 20</td>
<td>Claims handling office</td>
</tr>
<tr>
<td>Initiate payment</td>
<td>Exponential distribution: Mean = 2</td>
<td>Claims handling office</td>
</tr>
<tr>
<td>Advise claimant on ...</td>
<td>Exponential distribution: Mean = 4</td>
<td>Claims handling office</td>
</tr>
<tr>
<td>Close claim</td>
<td>Exponential distribution: Mean = 1</td>
<td>Claims handling office</td>
</tr>
</tbody>
</table>

Table 4: Activity durations and resource responsible for acquirer’s process
XOR-split name | Probabilities to follow alternative paths
---|---
Which centre? | Centre 1: 50% Centre 2: 50%
Information Complete? | Yes: 90% No: 10%
Information Complete? | Yes: 90% No: 10%
Insured liable? | Yes: 85% No: 15%
Claim Rejected? | Yes: 20% No: 80%

Table 5: Probabilities to follow alternative paths for acquirer’s process

Figure 6: Acquiree’s BPMN model of the claim-handling process

<table>
<thead>
<tr>
<th>Activity</th>
<th>Activity Duration</th>
<th>Resource responsible</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check if customer has ...</td>
<td>Exponential distribution: Mean = 2</td>
<td>Call centre 1</td>
</tr>
<tr>
<td>Add incomplete information</td>
<td>Exponential distribution: Mean = 7</td>
<td>Call centre 1</td>
</tr>
<tr>
<td>Register claim</td>
<td>Exponential distribution: Mean = 2</td>
<td>Call centre 2</td>
</tr>
<tr>
<td>Check if customer has..._2</td>
<td>Exponential distribution: Mean = 2</td>
<td>Call centre 2</td>
</tr>
<tr>
<td>Add incomplete ..._2</td>
<td>Exponential distribution: Mean = 7</td>
<td>Call centre 2</td>
</tr>
<tr>
<td>Register claim_2</td>
<td>Exponential distribution: Mean = 7</td>
<td>Call centre 2</td>
</tr>
<tr>
<td>Determine likelihood of ...</td>
<td>Exponential distribution: Mean = 4</td>
<td>Claims handling office</td>
</tr>
<tr>
<td>Assess claim</td>
<td>Exponential distribution: Mean = 26</td>
<td>Claims handling office</td>
</tr>
<tr>
<td>Initiate payment</td>
<td>Exponential distribution: Mean = 2</td>
<td>Claims handling office</td>
</tr>
<tr>
<td>Advise claimant on ...</td>
<td>Exponential distribution: Mean = 4</td>
<td>Claims handling office</td>
</tr>
<tr>
<td>Close claim</td>
<td>Exponential distribution: Mean = 1</td>
<td>Claims handling office</td>
</tr>
</tbody>
</table>

Table 6: Activity durations and resource responsible for acquiree’s process
As a first step, the BPMN-file of the business process needs to be imported. In the described case, 2 different BPMN models need to be imported. To avoid mistakes in the program, the user should first complete the whole set up of 1 BPMN model before setting up a new one. By set up is meant the whole process of importing the BPMN model, enriching it with additional information and translating it to a simmer-trajectory object. In what follows, part of the set up of the acquirer’s claim handling process will be showed, the full set up of the acquirer’s and acquiree’s process can be found in appendix.

Importing a BPMN-file can be done by the `import_BPMN()-function`. In the arguments is specified where the BPMN-file is saved locally as an XML-file. Note that the function will throw errors when the user tries to input a BPMN model that does not comply with the conditions of section 4. Once the function is executed, an R-list will appear in the global environment for each BPMN-element (see figure 7). All these R-lists will be of the custom class ‘bpmn-element’.

###step 1: import the bpmn's xml file

```r
import_BPMN(filepath = 'C:/Users/Yoren/Downloads/acquirer.bpmn')
```

The next step is to enrich these R-lists with additional information that cannot be extracted from the BPMN-model but that is necessary for setting up a simmer-trajectory object. More concretely this means that the activity durations, the probabilities to follow alternative paths and the resources responsible for executing the activity should be added. This is done by the `set-functions` of the BPMNsimulator package.
Specifying the activity duration is done by the `set_activity_duration()`-function. The activity and duration should be included in the function’s arguments. Important to mention is that the duration should be defined as a function (see red box in code snippet 2). In code snippet 2, the activity duration for the ‘close claim’ activity is specified. The duration parameter accepts all functions returning numbers, even user defined functions. If a negative number is returned by that function, the absolute value is used to define the duration. Note that the `set_intermediate_event_duration()`-function of the package works identically.

###step 2: Add additional information not available in a BPMN

**#ACTIVITY DURATIONS**

```r
set_activity_duration("Close Claim", duration = function() rexp(1, rate = 1))
```

*Code Snippet 2: Specifying the activity duration of the ‘close claim’ activity.*

Next, it should be specified that a member of the claim handling office is responsible for executing the ‘close claim’ activity. In the function’s arguments the activity and the responsible resource are required. The amount-parameter can be used to indicate that more than one member of the resource type is required to execute the activity. However, 1 member of the claim handling office is sufficient to close the claim in this example.

**#RESOURCES RESPONSIBLE FOR ACTIVITY**

```r
set_resource_to_activity("Close Claim", resource = 'claim_handler', amount = 1)
```

*Code Snippet 3: Specifying the resource responsible for the ‘close claim’ activity.*

Adding probabilities to follow alternative paths after a XOR-split can be achieved by using the `set_probabilities_to_XOR_split()`-function. The R-list of the XOR-split element should be indicated in the arguments of the function as well as the probabilities of following an alternative path and the first elements of each path. In code snippet 4, an example is given for the ‘Information complete?’ XOR-split. The probabilities vector and the first_elements vector should be in the same sequence meaning that the probability of flowing through the path that starts with ‘Call Ended’ is 0.1 and the probability of flowing through the other path is 0.9.

**#PROBABILITIES TO XOR-SPLIT**

```r
set_probabilities_to_XOR_split("Information Complete?",
                               first_elements = c("Call ended", "Register Claim"),
                               probabilities = c(0.1, 0.9))
```

*Code Snippet 4: Specifying the probabilities after the ‘Information Complete?’ XOR-split*

After adding all activity durations, probabilities and resources responsible, the R-lists should be mapped to a simmer-trajecory object. This is achieved by the `transform_BPMN()`-function. In the
arguments of this function all R-lists describing a BPMN-element should be included. In the case of the acquirer’s BPMN model, this means that 22 arguments need to be included. Omitting one will result in a wrong trajectory. Making use of the power of R and the custom made class of the R-lists, it is possible to reduce the programming effort as well as the error sensitivity of this step.

###step 3: Create the simmer-trajectory object

#Put all elements created by the import_BPMN function on a list
bpmn_elements <- Filter(function(x) is(x, "bpmn_element"), mget(ls()))
#Now we can use the do.call function (of base R)
traj_acquirer <- do.call(transform_BPMN, args = bpmn_elements)

In code snippet 5, the `Filter()`-function first selects all R-lists in the global environment of the class ‘bpmn_element’ and stores them as elements in a list called `bpmn_elements`. The `do.call()`-function executes the `transform_BPMN()`-function, it will use the elements of the `bpmn_elements` list as the arguments of that function. The `transform_BPMN()`-function stores a simmer trajectory object in the `traj_acquirer` variable and deletes the R-lists passed to it as arguments. In a similar fashion, the acquiree’s trajectory can be defined (see appendix).

After creating the trajectories of both business processes, the interarrival time of the processes and resources in the system still need to be defined. The `create()`-functions of the package are designed for this. The first time a `create()`-function is executed, the variable `simulation_environment` will be created in the global environment. All the subsequent calls of a `create()`-function will alter the information stored in this `simulation_environment` variable. With the `create_arrivals()`-function, the interarrival time can be added to a previously defined trajectory. The `sample()`-function in code snippet 6 is used to define the interarrival times listed in table 3. All functions returning positive numbers can be used as interarrival rate, even user defined functions.

###step 4: Define the simulation environment

#first delete simulation_environment if it already exists from a previous simulation
rm(simulation_environment)

#INTERARRIVAL_TIME
#Acquirer's process
create_arrivals(traj_acquirer, function() sample(x=c(0.1,0.3,0.8), size = 1))
#Acquiree's process
create_arrivals(traj_acquiree, function() sample(x=c(0.2,0.4), size = 1, prob = c(0.7,0.3)))

Code Snippet 6: Add arrivals to a trajectory.
The `create_resource()`-function is defining the resources in your system. Implementing work schedules can be achieved by making use of the schedule function of the simmer-package. More information about this function can be found in the comments of code snippet 7 and in the documentation of the simmer-package. Important to mention is that the same time unit should be used in the entire simulation set-up, more concretely this means that the schedules of table 2 should be recalculated to minutes. These schedules can be called in the arguments of the `create_resource()`-function afterwards.

```r
##RESOURCES AVAILABLE
#create schedule: Everything should be defined in same time unit: minutes
#the first 4 hours of the day (from minute 0 until 240): 40 Operators in call Centre
#the next 4 hours of the day (from minute 240 until 480): 55 Operators in call Centre
#the last 4 hours of the day (from minute 480 until 720): 25 Operators in call Centre
sched_CallCentre < - schedule(c(0, 240, 480), c(40, 55, 25), period=720)
#the first 8 hours of the day (from minute 0 until 480): 150 claim handlers in office
#the last 4 hours of the day (from minute 480 until 720): no claim handlers in office
sched_Claimhandler < - schedule(c(0, 480), c(150, 0), period=720)
create_resource(resource = 'operator_CC1', schedule = sched_CallCentre)
create_resource(resource = 'operator_CC2', schedule = sched_CallCentre)
create_resource(resource = 'claim handler', schedule = sched_Claimhandler)

Code Snippet 7: Define schedules and resources in your system.
```

At this point in time, The BPMN model and all additional information is inputted. Running the simulation and analyzing the results is the last step, for which is totally relied upon the simmer-package. The simulation is executed by the `run()`-function, in the arguments the run-time of the simulation can be specified. In the example case, the simulation runs 5 consecutive working days of 12 hours, which means that it should run for 3600 minutes. After executing the statement in code snippet 8, the simulation_environment variable is enhanced with the specifics of the simulation run and these specifics can be retrieved by the result-analyzing functions of the simmer-package.

```r
###step 5: run the simulation (using simmer package functions)
#run for 1 week: 5 days of 12 hours ==> run for 3600 minutes
run(simulation_environment, until = 3601)

Code Snippet 8: Run the simulation for 3600 minutes
```

For analyzing the results of a simulation, the simmer-package provides 2 types of functions. A first type are the `get()`-functions. The `get_mon_arrivals()`-function returns a dataframe containing information about the start time, end time and activity time of each instance. As mentioned in the
comments of code snippet 9, this dataframe should be transformed when and-structures appear in one of the processes in the simulation. The `transform_get_mon_arrivals()`-function is developed in the BPMNSimulator-package to take care of this transformation. To distinguish which instance originates from which process the name-column of the dataframe is used. The name of an instance is the name of the process plus a unique number, this mean that an instance with the name ‘traj_acquiree0’ flowed through the acquiree’s process.

![Figure 8: part of dataframe received by the get_mon_arrivals() function](image)

The `get_mon_resources()`-function returns a dataframe about the number of instances in the queue and in the server of each resource at each point in time. After retrieving the results with this `get()`-functions, basic R coding is used to analyze the result. An easy example is showed in code snippet 9, where the average waiting time and activity time are calculated. However working in R and receiving the information in the form of dataframes opens the possibility for much deeper analysis, as R is a data analysis programming language.

```r
###step 6: retrieve results (using simmer package functions)

#dataframe containing information about arrivals
arrivals <- get_mon_arrivals(simulation_environment)
#returns a dataframe containing information about:
#start_time, end_time and activity_time per arrival
#When having AND-gate structures in one of your trajectories
#You need to transform this arrivals dataframe first due to
#the technical implementation of this structure in simmer-package
#A function is added to the BPMNSimulator-package to do this
#use the transform_get_mon_arrivals()-function
transf <- transform_get_mon_arrivals(arrivals)

#calculate waiting time
#waiting_time = (end_time - start_time) - activity_time
transf$waiting_time = (transf$end_time - transf$start_time) - transf$activity_time

#Calculate average waiting time and activity time of instance
mean(transf$waiting_time) #62.73 minutes
mean(transf$activity_time) #23.61 minutes

#dataframe containing information about resources
resources <- get_mon_resources(simulation_environment)
```

*Code Snippet 9: Analyzing the results using the get()-functions*
A second type of result analyzing functions are the plot()-functions. These functions give a quick and visual overview of the results of the simulation. To receive an overview of the resource usage, the plot_resource_usage() and plot_resource_utilization()-function can be used. The plot_evolution_arrival_times()-function returns an overview about the instances.

```r
# plots containing information
plot_resource_usage(simulation_environment, resource_name = 'operator_CC1')
plot_resource_utilization(simulation_environment, c('operator_CC1'))
plot_evolution_arrival_times(simulation_environment, type = 'flow time')
```

In appendix more information can be found about defining loops and importing a BPMN containing subprocesses.

The goal of this research project was to develop a business process simulator in R with a higher accessibility than the discrete event simulators on the one hand and that increases the depth of analysis of process model simulators, while not reducing on their usability and accessibility on the other hand. To evaluate our research project properly, the BPMNSimulator-package should be compared with discrete event simulators and process model simulators.

An higher accessibility than discrete event simulators is ensured by using the process model simulator method. The import_BPMN()-function ensures that the information present in a BPMN model can be extracted automatically, omitting a syntax-mapping.

To not reduce on the accessibility and usability of the process model simulators, the decision was made to create a BPMN-simulator (a process model simulator that is supporting the BPMN language) in R with enhanced capabilities. The usability is preserved by making use of simple set- and create-functions to input additional information. Working in a programming environment decreases the accessibility for programming laymen compared to the GUI-based process model simulators. To mitigate this decreased accessibility, the design of the package is kept simple to keep the learning curve short.

Increasing the depth of analysis of the current process model simulators is achieved by eliminating their drawbacks listed in the introduction. The first drawback that is addressed is the inability of simulating related business processes. Most process model simulators limit the user to simulate 1 business process model in isolation, this means no integration is possible between related business processes. The BPMNSimulator-package offers the possibility to simulate multiple business processes...
in 1 simulation environment as is shown in the demonstration. The ability to let multiple business processes seize the same resources will ensure that the definition of the simulation will be closer to reality, as in real-life situations resources are responsible for multiple business processes (Van Der Aalst et al., 2010).

Secondly, the function-types available to define the activity durations and interarrival time in BPMN-simulators are limited. Some BPMN-simulators only offer normal distributions, while others offer more than 10 different function-types (Pereira et al, 2016). However, by creating our BPMNsimulator-package in a data analysis programming language like R, the possibilities to define activity durations and interarrival times are endless. By offering the possibility to use user-defined functions as activity durations and interarrival time, the opportunity exists to write functions that extract activity durations and interarrival times from the existing event logs of the process. Again bringing the definition of the simulation closer to reality. Using event logs to set up a simulation model is also researched by martin et al. (2015 & 2016).

Thirdly, the integration of work schedules for the available resources is not supported by all process model simulators (Pereira et al, 2016) (Salvidar et al, 2016). The BPMNsimulator-package includes this option by making use of the schedule()-function in the simmer-package.

Lastly, some process model simulators show the outcome of a simulation with some standard overview statistics about resource utilizations and cycle times, giving no possibility to conduct a custom analysis (Salvidar et al, 2016). The get-functions of the simmer-package offer the possibility to receive detailed information on the instance and resource level, giving the user the opportunity to conduct a detailed analysis on the complete or part of the simulation by making use of basic R. With the plot-functions some quick overview statistics can be retrieved. Although a more thorough analysis on resource and instance level can be conducted, an analysis on the activity level is missing. This is an inevitable disadvantage of the fact that the BPMNsimulator-package is using the simmer-package to execute and analyze the simulation, this simmer-package is a discrete event simulator which is not been build around activities as known in a BPMN model.
5. Conclusion

Observing the current landscape of business process simulators, the discrete event simulators are solutions that offer the possibility to conduct deep, specific and tailored analysis but have a reduced accessibility because of their tool-specific syntax. The process model simulators offer an accessible and user-friendly solution but the depth of analysis is limited.

In this research project, the goal was to create a new tool in R that preserves the usability and accessibility of the process model simulators and extends their depth of analysis. The result was the BPMNsimulator-package in R that is enhancing the current process model simulators by offering the possibility to simulate related business processes and by enlarging the set of functions to define activity durations and interarrival times. Also schedules of resources can be inputted and more detailed results can be retrieved on the instance and resource level.

Drawbacks of the tool are twofold. Firstly an analysis on the activity level is not possible, which can be important dependent on the goal of the simulation. Secondly, the consequence of working in a programming language is that the accessibility for programming-laymen is reduced compared to the GUI-based process model simulators. However we believe that the advantages are outperforming the disadvantages and that a process model simulator capable to simulate related business processes and offering a large variety of functions to define activity durations and interarrival times is a valuable expansion of the current business process simulators available.
6. References

List of annexes

1 Installing R & RStudio
2 Installing the BPMNsimulator-package
3 Complete code of demonstration (section 5)
4 Defining probabilities to follow alternative paths with loops
5 Importing BPMN models with subprocesses

1. Installing R & RStudio

Installing R is as easy as installing any other program. The installation wizard can be downloaded on the following link: https://cran.r-project.org/bin/windows/base/. Once you have downloaded the installation wizard, this wizard will help you through the installation process. If you don’t change the installation settings, the wizard will install both the 32-bit and 64-bit version. If your computer has a 32-bit CPU, you can only run the 32-bit version. A 64-bit CPU computer can run both versions. This can be checked in the system information of your computer.

RStudio is an integrated development environment (IDE) for R. RStudio will make your life easier when you are coding in R. An important feature is the clear distinction between your R-script (the code is written here) and your R-console (Once your code is run, it is passed to the R-console and executed here). Other advantages are that it gives the programmer an overview of the created variables, the installed packages and the plots created.

The RStudio installation wizard can be downloaded for free for personal use on the following link: https://www.rstudio.com/products/rstudio/download3/. The installation wizard will help you through the installation process. A requirement for running RStudio is that there should be an R-version installed on your PC.
2. Installing the BPMNsimulator-package

As the package is not available on the CRAN, the install.packages()-function of R won’t work to install the package. If you own the package as a .tar-file, RStudio offers an alternative way to install packages. Use the steps below to install the package.

1) Go to RStudio
2) Select Packages in the right bottom window of the standard Rstudio set-up (see figure 9)
3) Push Install (see red box in figure 9)
4) The GUI of figure 10 will appear, set the ‘Install From: ‘ setting on Package Archive File (As in figure 10)
5) At Package Archive, Browse in your local drive system to the location where your .tar-file is saved
6) Push install

Following these 6 steps the package will install it and can be used as any other R package and will be listed among your packages.

*Figure 9: screenshot from RStudio*
Figure 10: GUI after pushing install
3. Complete code of demonstration (section 5)

```r
evaluate_code(list = ls())

###load the required package
library('BPMNsimulator')

###IMPORTING ACQUIRER'S PROCESS###

###step 1: import the bpmn's xml file
import_BPMN(filepath = 'C:/Users/Yoren/Downloads/acquirer.bpmn')

###step 2: Add additional information not available in a BPMN

# ACTIVITY DURATIONS
set_activity_duration(`Check if customer has all required information`, duration = function() rexp(1, rate=1/1))
set_activity_duration(`Check if customer has all required information_2`, duration = function() rexp(1, rate=1/1))
set_activity_duration(`Register Claim`, duration = function() rexp(1, rate=1/9))
set_activity_duration(`Register Claim_2`, duration = function() rexp(1, rate=1/9))
set_activity_duration(`Determine likelihood of the claim`, duration = function() rexp(1, rate=1/2))
set_activity_duration(`Assess Claim`, duration = function() rexp(1, rate=1/20))
set_activity_duration(`Initiate Payment`, duration = function() rexp(1, rate=1/2))
set_activity_duration(`Advertise Claimant on Reimbursement`, duration = function() rexp(1, rate=1/4))
set_activity_duration(`Close Claim`, duration = function() rexp(1, rate=1/1))

# PROBABILITIES TO XOR-SPLIT
#Comment: no need to set probabilities if probability is equal between branches (this is the default)
set_probabilities_to_XOR_split(`Information Complete?`, first_elements = c("Call ended", "Register Claim"), probabilities = c(0.1, 0.9))
set_probabilities_to_XOR_split(`Information Complete?_2`, first_elements = c("Call ended_1", "Register Claim_2"), probabilities = c(0.1, 0.9))
set_probabilities_to_XOR_split(`Insured Liable?`, first_elements = c("case closed", "Assess Claim"), probabilities = c(0.15, 0.85))
set_probabilities_to_XOR_split(`Claim Rejected?`, first_elements = c("claim rejected", "default_gateway_3"), probabilities = c(0.2, 0.8))

# RESOURCES RESPONSIBLE FOR ACTIVITY
set_resource_to_activity(`Check if customer has all required information`, resource = 'operator_CC1', amount = 1)
set_resource_to_activity(`Check if customer has all required information_2`, resource = 'operator_CC2', amount = 1)
set_resource_to_activity(`Register Claim`, resource = 'operator_CC1', amount = 1)
set_resource_to_activity(`Register Claim_2`, resource = 'operator_CC2', amount = 1)
set_resource_to_activity(`Determine likelihood of the claim`, resource = 'claim_handler', amount = 1)
set_resource_to_activity(`Assess Claim`, resource = 'claim_handler', amount = 1)
```
```r
set_resource_to_activity('Initiate Payment', resource = 'claim_handler', amount = 1)
set_resource_to_activity('Advise Claimant on Reimbursement', resource = 'claim_handler', amount = 1)
set_resource_to_activity('Close Claim', resource = 'claim_handler', amount = 1)

###step 3: Create the simmer-trajectory object
#Put all elements created by the import_BPMN function on a list
bpmn_elements <- Filter(function(x) is(x, "bpmn_element"), mget(ls()))
#Now we can use the do.call function (of base R)
traj_acquirer <- do.call(transform_BPMN, args = bpmn_elements)

###IMPORTING ACQUIREE'S PROCESS###

###step 1: import the bpmn's xml file
import_BPMN(filepath = 'C:/Users/Yoren/Downloads/acquiree.bpmn')

###step 2: Add additional information not available in a BPMN
#ACTIVITY DURATIONS
set_activity_duration('Check if customer has all required information', duration = function() rexp(1, rate = 1/2))
set_activity_duration('Check if customer has all required information_2', duration = function() rexp(1, rate = 1/2))
set_activity_duration('Add Incomplete Information', duration = function() rexp(1, rate = 1/7))
set_activity_duration('Add Incomplete Information_2', duration = function() rexp(1, rate = 1/7))
set_activity_duration('Register Claim', duration = function() rexp(1, rate = 1/26))
set_activity_duration('Register Claim_2', duration = function() rexp(1, rate = 1/2))
set_activity_duration('Determine likelihood of the claim', duration = function() rexp(1, rate = 1/4))
set_activity_duration('Assess Claim', duration = function() rexp(1, rate = 1/26))
set_activity_duration('Initiate Payment', duration = function() rexp(1, rate = 1/4))
set_activity_duration('Advise Claimant on Reimbursement', duration = function() rexp(1, rate = 1/4))
set_activity_duration('Close Claim', duration = function() rexp(1, rate = 1/4))

#PROBABILITIES TO XOR-SPLIT
#Comment: no need to set probabilities if probability is equal between branches (this is the default)
set_probabilities_to_XOR_split('Information Complete?', first_elements = c("Add Incomplete Information", "join1"), probabilities = c(0.4, 0.6))
set_probabilities_to_XOR_split('Information Complete?_2', first_elements = c("Add Incomplete Information_2", "join2"), probabilities = c(0.4, 0.6))
set_probabilities_to_XOR_split('Insured Liable?', first_elements = c("case closed", "Assess Claim"), probabilities = c(0.15, 0.85))
set_probabilities_to_XOR_split('Claim Rejected?', first_elements = c("claim rejected", "Initiate Payment"), probabilities = c(0.2, 0.8))

#RESOURCES RESPONSIBLE FOR ACTIVITY
```
set_resource_to_activity('Check if customer has all required information', resource = 'operator_CC1', amount = 1)
set_resource_to_activity('Check if customer has all required information_2', resource = 'operator_CC2', amount = 1)
set_resource_to_activity('Add Incomplete Information', resource = 'operator_CC1', amount = 1)
set_resource_to_activity('Add Incomplete Information_2', resource = 'operator_CC2', amount = 1)
set_resource_to_activity('Check if customer has all required information', resource = 'operator_CC1', amount = 1)
set_resource_to_activity('Check if customer has all required information_2', resource = 'operator_CC2', amount = 1)
set_resource_to_activity('Register Claim', resource = 'operator_CC1', amount = 1)
set_resource_to_activity('Register Claim_2', resource = 'operator_CC2', amount = 1)
set_resource_to_activity('Determine likelihood of the claim', resource = 'claim_handler', amount = 1)
set_resource_to_activity('Assess Claim', resource = 'claim_handler', amount = 1)
set_resource_to_activity('Initiate Payment', resource = 'claim_handler', amount = 1)
set_resource_to_activity('Advise Claimant on Reimbursement', resource = 'claim_handler', amount = 1)
set_resource_to_activity('Close Claim', resource = 'claim_handler', amount = 1)

###step 3: Create the simmer-trajectory object
#Put all elements created by the import BPMN function on a list
bpmn_elements <- Filter(function(x) is(x, "bpmn_element"), mget(ls()))
#Now we can use the do.call function (of base R)
traj_acquiree <- do.call(transform_BPMN, args = bpmn_elements)

###step 4: Define the simulation environment
#first delete simulation_environment if it already exists from a previous simulation
rm(simulation_environment)

###INTERARRIVAL TIME
#Acquirer's process
create_arrivals(traj_acquirer, function() sample(x=c(0.1,0.3,0.8), size = 1))
#Acquiree's process
create_arrivals(traj_acquiree, function() sample(x=c(0.2,0.4), size = 1, prob = c(0.7,0.3)))

###RESOURCES AVAILABLE
#create schedule: Everything should be defined in same time unit: minutes
#the first 4 hours of the day (from minute 0 until 240): 40 Operators in call Centre
#the next 4 hours of the day (from minute 240 until 480): 55 Operators in call Centre
#the last 4 hours of the day (from minute 480 until 720): 25 Operators in call Centre
sched_CallCentre <- schedule(c(0, 240, 480), c(40, 55, 25), period=720)
#the first 8 hours of the day (from minute 0 until 480): 150 claim handlers in office
'### Step 5: Run the Simulation (Using Simmer Package Functions)
# Run for 1 Week: 5 Days of 12 Hours ==> Run for 3600 Minutes
run(simulation_environment, until = 3601)

### Step 6: Retrieve Results (Using Simmer Package Functions)

#### Dataframe Containing Information About Arrivals
arrivals <- get_mon_arrivals(simulation_environment)
# Returns a dataframe containing information about:
# Start Time, End Time, and Activity Time Per Arrival
# When Having AND-gate Structures in One of Your Trajectories
# You Need to Transform This Arrivals Dataframe First Due to
# the Technical Implementation of This Structure in Simmer-Package
# A Function Is Added to the BPMNsimulator-Package to Do This
# Use the Transform Get Mon Arrivals() - Function
transf <- transform_get_mon_arrivals(arrivals)

# Calculate Waiting Time
# Waiting Time = (End Time - Start Time) - Activity Time
transf$waiting_time = (transf$end_time - transf$start_time) - transf$activity_time

# Calculate Average Waiting Time and Activity Time of Instance
mean(transf$waiting_time) # 62.73 Minutes
mean(transf$activity_time) # 23.61 Minutes

#### Dataframe Containing Information About Resources
resources <- get_mon_resources(simulation_environment)

#### Plots Containing Information
plot_resource_usage(simulation_environment, resource_name = 'operator_CC1')
plot_resource_utilization(simulation_environment, c('operator_CC1'))
plot_evolution_arrival_times(simulation_environment, type = 'flow_time')
4. Defining probabilities to follow alternative paths with loops

The definition of probabilities to follow alternative paths with loops is dependent on the type of loop. The first type of loop is a loop where no additional activities should be conducted before returning (see figure 9 for the BPMN representation). In this case the `prob_to_continue` parameter of the `set_probabilities_to_XOR_split()` function should be used (see code snippet 11).

![Figure 9: Loop with no additional activities before returning](image)

### step 1: import the bpmn’s xml file
```
import_BPMN(filepath = 'C:/Users/Yoren/Downloads/Loop_option1.bpmn')
```

### step 2: Add Additional information
```
#probabilities to XOR-split
#70% chance of continuing without entering the loop
set_probabilities_to_XOR_split(Split1, prob_to_continue = 0.7)
```

---

A second type of loop is a loop where first some additional activities should be conducted before returning (see figure 10 for the BPMN representation). Setting the probabilities for this loop can be done using the ‘next_elements’ and ‘probabilities’ parameters of the `set_probabilities_to_XOR_split()` function (see code snippet 12).

![Figure 10: Loop with additional activities before returning](image)
### step 1: import the bpmn's xml file
import_BPMN(filepath = 'C:/Users/Yoren/Downloads/Loop_option2.bpmn')

### step 2: Add Additional information

probabilities to XOR-split
70% chance of continuing without entering the loop
set_probabilities_to_XOR_split(Split1, first_elements = c('A4', 'END'), probabilities = c(0.3, 0.7))

*Code snippet 12: set probabilities for Split1 of figure 10*
5. Importing BPMN models with subprocesses

When exporting a BPMN model with subprocesses as a BPMN 2.0 XML file in Signavio, Signavio offers the option to include linked the linked subprocesses. If the user of the BPMNsimulator- package wants to do his analysis on the activity level of the subprocess he should mark this option before exporting (as is shown in Figure 11). If the user wants to treat the subprocesses in the same way as activities in the head process he should leave the option blank, this means he can set a 1 resource responsible and 1 duration for each subprocess as a whole.

![Export BPMN 2.0 XML](image)

*Figure 11: GUI of Signavio offering the option to include the linked subprocesses of the BPMN model*

![BPMN model of head process with a subprocess](image)

*Figure 12: BPMN model of head process with a subprocess*

![BPMN model of Subprocesses1 of the BPMN model represented in figure 10](image)

*Figure 13: BPMN model of Subprocesses1 of the BPMN model represented in figure 10*

An illustration is given for the BPMN model represented in figure 12 (the subprocess is represented in figure 13). As the user has marked the option to include the linked subprocesses as illustrated in figure 11 and want to import this XML-file in R, he should set the ‘subprocesses_included’ parameter
of the *import_BPMN()*-function to TRUE (default value is FALSE). When executing this command as in code snippet 13, the user will see that an r-list is created for A1 and A2 of the head process as well as A3, split1, A4, ... of the subprocess. Now he can specify activity durations, possibilities to follow alternative paths and resources responsible on the level of the subprocess.

```r
### step 1: import the bpmn's xml file
# subprocess included in the BPMN file
import_BPMN(filepath = 'C:/Users/Yoren/Downloads/process_subprocess.bpmn',
             subprocesses_included = TRUE)
```

*Code snippet 13: Import BPMN model of figure 12 with subprocesses included*

When the user wants to set a general activity duration and resource responsible for subprocess1 (basically treat it in the same ways as A1 or A2), the exporting in Signavio should be done with the option of figure 11 left blank and imported into R afterwards with the parameter ‘subprocesses_included’ of the *Import_BPMN()*-function set to FALSE.