ADAPTABLE PROCESSES: CONCEPTS, DESIGN AND IMPLEMENTATION IN CAMUNDA

Ana Roman

Supervised by:
D. Karastoyanova
V. Andrikopoulos

March 22, 2021
University of Groningen

Abstract

Due to the generic approach that they take while modelling scenarios, as well as their independence from application domains, business workflows have also grown increasingly popular among the scientific community and applications. In a business context, process automation can drastically improve an organization's performance and efficiency, and thus reduce costs in terms of time and money. In the scientific context, on the other hand, process automation can be used as a tool for conducting experiments, bringing with it the advantage that such an approach allows for automation as well as reproducibility of the results, which is a key element in the academic research field.

In the past years, Camunda has proven itself to be a powerful workflow and decision automation platform, that provides numerous tools to facilitate the modelling, execution and analysis of business processes. At the same time, Camunda is BPMN compliant, coming as an answer to the problems that scientists often face when relying on tools developed by other scientists, that are more often than not outdated or based on outdated programming languages.

Despite the extensive set of tools that the state of the art business workflow systems currently have, they still lack the possibility of trial-and-error modelling that is needed when working with processes that do not have a complete set of instructions, and thus cannot support incomplete, partially defined workflow models, such as in the case of scientific experiments or in a business process where processes can evolve over time.

The present study addresses these limitations that are also present in Camunda, with regards to the possibility of flexible development and execution of models and processes. The system will be extended with advanced flexibility features that will allow users from both worlds (business and scientific) to create and run workflows in an explorative manner in order to increase the robustness of their applications.

Contents

List of Figures

Acronyms

BP Business Process. 11

BPM Business Process Management. 11

BPMN Business Process Model and Notation. 18

DMN Decision Modelling and Notation. 23

UML Unified Modeling Language. 18

WfMC Workflow Management Coalition. 16, 18

WfMS Workflow Management System. 12, 18

Chapter 1 | Introduction

At the core of Business Process Management (BPM) lies the observation that every product offered by a company on the market is the result of a number of activities [1]. Business Processes (BP) coordinate these activities as well as the interrelationships between them, and as such, are crucial to a company's performance. A deep understanding of the processes that take place in an organization can lead to consistent outcomes and improvements, such as costs or execution times reduction, reducing error rates and also more opportunities to innovate which can bring about competitive advantage [2]. Instead of focusing on the execution of individual activities, BPM focuses on managing the entire chains of events, activities and decisions that are valuable to an organization, its customers and its stakeholders [3]. *Processes* can be formally defined as the sum of events, activities and decisions that lead to an outcome [1]. BPM can be seen as a mechanism for controlling the execution of these processes by providing a set of tools such as concepts, methods, documentation, monitoring and means of optimization [4].
According to the literature [2], quality of the processes and the way in which the processes are conducted may lead an organization to outperform another one offering similar services.

In the past years, BPM has started to receive more attention from other communities outside of the business world. In particular, the scientific community has started using some of the tools provided by BPM when coordinating their own research and experiments. Through workflows that are used for scientific means, also called *scientific workflows*, scientists can benefit from the extensive set of tools that conventional workflow management systems have to offer, and have access to a significant body of knowledge from the heavily researched BPM domain. Scientific workflows can be particularly useful to specify the control and data flow of experiments, or to orchestrate software modules and services that scientists develop themselves [5].

Offering a BPMN compliant engine and a number of applications that enable non-IT specialists to develop and deploy processes, Camunda has become increasingly popular as a Workflow Management System (WfMS) between both business and scientific communities. It is an open, ongoing project that is continuously improved, and has a large community of developers and users from a vast number of domains. Due to its recent increase in popularity and the set of tools that it already offered, Camunda was selected as the candidate for the topic of this thesis.

1.1 Motivation

Despite the already existing set of tools that they provide, and the extensive work that has been done on the research of BPM, existing WfMS still lack features and characteristics that are desirable in scientific contexts and applications. For example, most systems do not provide the deployment of flexible, adaptable processes, that would enable a trial-and-error way of developing processes. In an experimental context, not all the steps are known beforehand and a lot of activities are subject to change. Processes are usually long-running, with activities that could take days, even weeks [6]. In the light of these issues, one can already understand why the flexibility of a process instance would be preferred over the redeployment and repetition of the same steps.
However, not only the scientific community could benefit from such capabilities. Business processes usually handle extensive amounts of sensitive data and have to follow rules imposed by the surrounding environment, such as legislation, changing market conditions or rules of conduct. According to the literature, changing a workflow instance is easier and would allow a workflow management system to be more responsive to the changes in its environment [7].

The vision of the current work is to aid users from both worlds and contexts in improving their processes, by providing them with the concepts and a proof-of-concept implementation of adaptable processes on the Camunda OSS.

1.2 Problem Statement

The topic of adaptable processes has been thoroughly discussed in the past by the literature. In their work [6], Karastoyanova and Sonntag have laid the foundation of a new technique called *Model-as-You-Go*, which allows the development and deployment of adaptable processes on a BPEL engine. In his dissertation thesis [8], A Weiß extends the *Model-as-You-Go* technique such that it can be applied to choreographies.
However, all of the previous work and research has focused mainly on the BPEL language and engine. Up to the point of writing this thesis, the subject of adaptations on

the Camunda OSS has not yet been researched or implemented.

1.3 Research question and objectives

With regards to the aforementioned problem statement, the overall goal of the current thesis is to explore the concept and requirements of a WfMS that provides flexibility capabilities. The current work will investigate previous work done on this topic, will select relevant ideas and will put together a conceptual blueprint. At the same time, it will describe how the concepts derived from the literature can be translated into implementation, which will then be applied on the Camunda software.
As such, the research question that will be explored is the following:

How can the concepts extracted from existing literature be applied when extending Camunda with flexible execution of processes?

1.4 Summary of contributions

With the research objectives and question in mind, the contributions of the current thesis can be outlined. The main focus of this work is to contribute with a concept that will allow users of the Camunda WfMS to improve the robustness of their applications by enabling the ability to deploy adaptable processes.
This contribution can be further divided the following way:

(I) A concrete list of requirements that, according to the literature, are fundamental to any WfMS that should allow adaptable processes.

(II) A proof-of-concept implementation of an extension on the Camunda software, that enables the deployment of adaptable processes
This second sub-contribution has been split in two:

  (a) Providing the implementation of the `Instance Migration` approach

  (b) Providing the implementation of the `Data Transfer` approach

1.5 Structure of the thesis

The thesis is structured as follows:

Chapter 1: Introduction This chapter focuses on introducing the core ideas behind the current work and the motivation that supports the importance of it. It describes the problem statement, the research question and the objectives that are to be met by this work.

Chapter 2: Background and Related Work This chapter is split up in two parts. Firstly, it will begin by introducing a lot of the core concepts behind processes, business processes, BPM and the BPM Lifecycle and Management. Then, it will describe Workflows, their components and make the difference between business and scientific workflows. Afterwards, a high-level overview will be given as an introduction to Camunda.
Secondly, the relevant literature is presented, with focus on the *Model-As-You-Go* approach.

Chapter 3: Methodology Here, the requirements will be presented, extracted from two separate use cases. The concept and the flow of adaptable processes will be explained at a high level.

Chapter 4: Suggested Solution and Implementation This chapter will firstly take a look at the basic implementation of Camunda, and secondly it will present the architecture, together with the design decisions of the extensions brought to Camunda.

Chapter 5: Evaluation and Conclusion The last chapter focuses on the evaluation of the extensions brought to Camunda. The implementation will be evaluated based on its' compliance with the requirements and its usefulness towards the presented Use Cases. The chapter will conclude the work with the summary of the results, a discussion on the limitation as well as the ideas for future work.

Chapter 2 | Background. Related Work

This section introduces fundamental concepts, definitions and ideas that have served as the foundation for the current research.
The section is split into two parts: firstly, section 2.1 will give an overview of the theory that served as a foundation for the current research, collected from the literature, such as books, manuals, official documentation and articles.
Secondly, section 2.2 will analyse the previous work that has been done on the current research topic, and presents the conceptual ideas found in the literature such that they can later be applied in this research.

2.1 Background

This section introduces the fundamental concepts, formalities and ideas that this thesis is based on and that are necessary in order to understand business processes, from modelling to execution. Firstly, the chapter will focus on the background information and the formal definitions of workflows and business processes. Then, a high-level overview of Camunda will be described, focusing on the components that were most relevant for the topic of this thesis.

2.1.1 Business Process Management

Business Process Management (BPM) supervises the way work and tasks are being performed in an organization, focusing on both outcomes and improvement opportunities at the same time [2]. At the core of BPM lies the *process*: the sum of activities, events and decisions that are all put into motion whenever a company wants to achieve a certain goal, from the micro-level, where only one or a few number of parties are involved (eg. packaging a product for delivery) to the macro-level, where multiple parties, systems and people have to work together and coordinate their actions (such as sending a Rover to the surface of Mars[1]). The importance of business processes has been extensively described in the first chapter, and the same motivation can be used when talking about the importance of business process management.

2.1.1.1 The BPM Lifecycle

According to the literature [2], at a conceptual level, all BPM initiatives follow certain steps in their life cycles. A depiction of the steps can be seen in figure 2.1.
The life cycle starts whenever a business problem is presented. As a result of this, an analysis of the process architecture is made in order to provide an overview of all the processes in an organization, which can be used to select the best-fitting process (or set of processes) that can be used to solve the problem.
Next, the previously selected processes are documented and presented as so called "as-is process models", which in turn are analysed in order to identify issues or to measure their performance; the set of found issues are then ranked based on a certain aspect, such as impact, estimated costs or time to solve.
The next step describes the changes that should be brought when addressing the issues, and generates an improved "to-be process model", and the next step will make the transition from the "as-is process" to the "to-be process", in both organizational changes (eg. changes to the participants) and automation (changes to the IT systems).
The newly formed process is then monitored, its data analysed and the performance measured, to ensure that the performance objectives have been met. Whenever errors or new issues are found, the cycle is then restarted and repeated - which can happen on a continuous basis.

2.1.2 Workflows

The Workflow Management Coalition (WfMC) was established in 1993, and is an international organization whose duty is to "*concentrate purely on process*" [2]. They develop reference models, documents and standards for the interoperability of workflow management systems.
According to the WfMC, workflows can be formally defined as *"the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules"* [9]. Informally, a workflow can be seen as the part of a business process that can be automated with the help of a computer system [8]. In a typical workflow

[1] https://mars.nasa.gov/mars2020/
[2] Official Website: https://www.wfmc.org/

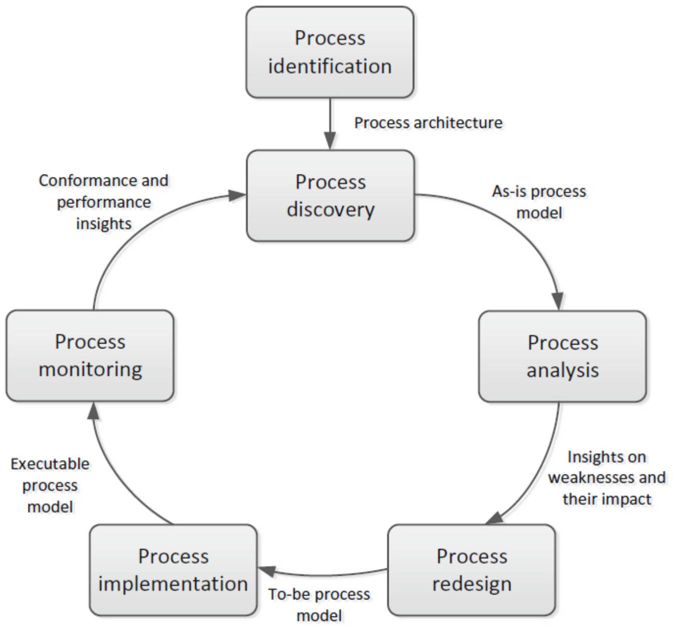


Figure 2.1: The BPM lifecycle, as found in [2]

scenario, both computer systems and people are involved [7]. The tasks are divided according to specific resource allocation rules, in order to achieve a certain objective (eg. highest efficiency, lowest error rate, etc).

2.1.2.1 Business vs. Scientific Workflows

The need for appropriate monitoring tools for business workflows has been discussed in the previous sections. The application of business workflows to other fields, such as science and simulations, is an issue that has been researched extensively by the scientific workflow and service composition communities [6]. By using conventional/business workflow technologies in the scientific field, natural scientists benefit from a multitude of tools and knowledge that is readily available.
However, the literature [5], [6] also identifies a few differences between the requirements of the two types of workflows, for example, the fact that scientific experiments often require a more explorative approach when designing their models, based on trial and error, or the fact that in a scientific context, the scientist is both the designer and the user of the workflow.

In their work [10], Leyman and Roller identify, define and exemplify four types of business workflows, as well as their direct counterparts in the fields of science and engineering. The four types of workflows are the following:

Collaborative - involve many individuals, and is usually focused on a single project. Such examples could be the release of a new product or the tracking of tasks and systems involved in an assembly line of a hardware component.

Ad-hoc - they are less formal in structure and response, and are usually targeted towards handling uncommon situations. For example, whenever a legal policy changes and a company has to change some of their activities, or when an exception is thrown in a software system and certain error-handling behaviors are required.

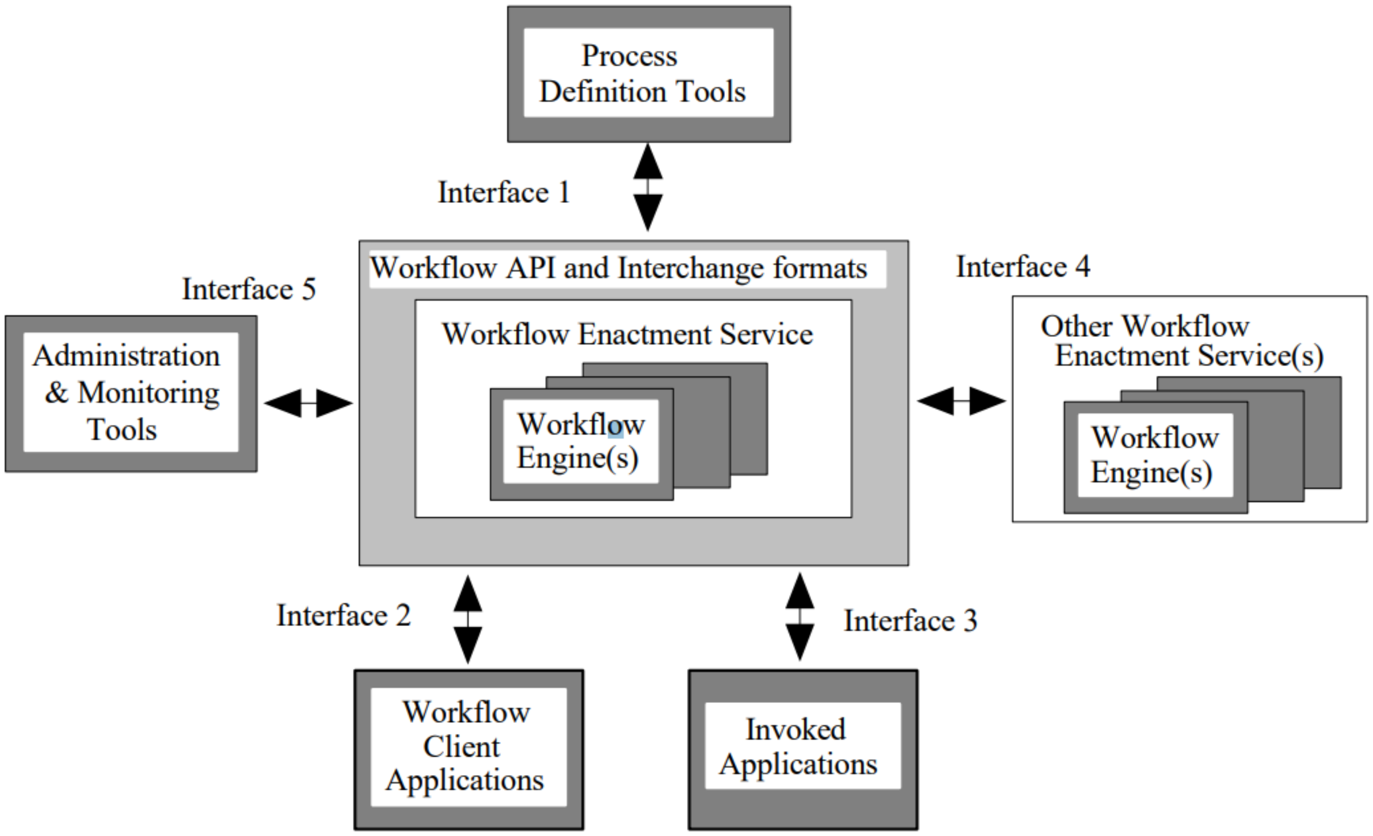
Administrative - they deal with tasks that should be done on a frequent basis, but are not directly tied to a product or an objective. Such activities include the management of systems, book keeping, scheduled maintenance or managing log/output files.

Production - the activities that lie at the core of a company, such as the steps involved in issuing an insurance policy or the weather systems that interpret sensor data to run simulations and try to predict extreme weather conditions.

2.1.2.2 Components

According to [9], all workflow systems contain a number of generic components which can interact according to a set of rules. In order to achieve interoperability between different workflow systems, standardised interfaces should be provided and common data formats should be established.
Figure 2.2 depicts the workflow reference model proposed by the WfMC, which can be seen as an overview of the architecture of a generic WfMS.

In their work, Leymann and Roller [10] identify two types of WfMS components, based on the type of functionalities that they have: *build time* and *run time* components.
The build time components allow users to create their workflow models, together with all the required information and data. In the Workflow Reference Model, this component is seen as a Process Definition Tool. There exists a multitude of languages for specifying process models, such as BPMN [11], UML Activity Diagrams [12], Case Management and Notation (CMN) [13] or Petri-Nets [14]. In the current thesis, BPMN will be used as the main language of reference.
The run time components allow the initialization of process instances, the navigation through the model graph and the execution of each activity's behavior. They allow the monitoring of instances as well as the modifications brought to the instances.



Process Definition Tools
Interface 1
Workflow API and Interchange formats
Interface 4
Interface 5
Workflow Enactment Service
Other Workflow Enactment Service(s)
Administration & Monitoring Tools
Workflow Engine(s)
Workflow Engine(s)
Interface 2
Interface 3
Workflow Client Applications
Invoked Applications

Figure 2.2: The Workflow Reference Model of the WfMC, taken from [9]

In his work [8], A. Weiß refers to workflow execution as "*the actual navigation through the workflow path from activity to activity, either sequentially or in parallel, using workflow data to determine the correct path and the manipulation of said data*". In the context of the current work, the same definition will be used and referenced.

Soundness

The workflow literature identifies one minimum requirement: *soundness* which should be met by every process. In their work [15], Aalst and Hee state that a process is *sound* if there exists no unnecessary tasks, and that every process that is started must be fully completed, without any remaining dangling active executions.

2.1.3 Flexibility

Depending on the domain of application or the nature of a research activity, the term *flexibility* can be defined in different ways [16].
Within the BPM domain, flexibility can be seen as the ability to take into consideration different activities from a multitude of disciplines, such as computer science, information science, economics, etc. [1].
However, a general consensus of the literature defines flexibility as the ability to react

to changes [2]. These changes can be from both internal sources, such as the replacing of a software system, or external, for example, whenever a change in the legislation does not allow certain parts of a process. Flexibility can also concern the ability of handling fluctuating amounts of workload, as well as the responsiveness to those changes.

Taking into consideration the two types of components that Leyman and Roller [10] have identified, described in section 2.1.2.2, flexibility can also be looked at from two dimensions. *Run time flexibility* focuses on the ability to change a business process during execution (after it has been started and before completion). *Build time flexibility* describes the ability to alter the structure of the business process.

The concept of workflow flexibility can also be found in the literature under the name of dynamic or adaptive workflows [7]. In this context, the adaptation of workflows can be seen as the extension of a static process whenever some change is required, by bringing modifications to the process rather than building a completely new model. This change can be ad-hoc, meaning that it only applies to a single process instance, or general, where a subset or all of the existing and future process instances should have these changes implemented [7].

Correctness Whenever a workflow is subject to changes, there must be a set of criteria, rules or specifications that can ensure the correctness of the workflow. When checking for correctness, a few questions can be asked, such as: *What changes are allowed to be made?* and *Is the resulting workflow process definition following the aforementioned criteria and specifications?*.
The literature [17] distinguishes between two types of correctness: *syntactic correctness* (eg. Is the resulting process definition still a sound model?) and *semantic correctness* (eg. Can the running instances that have been modified still be completed?).

The correctness of a workflow and its adaptations is an issue that should be guaranteed and checked. However important this issue is, it is outside the scope of this thesis, since it represents an entire research topic on its own. We leave it to the users to ensure the correctness of their workflows and adaptations.
More information on the correctness of workflows can be found in: [17], [18], [19].

2.1.4 Camunda

Camunda[1] is a Java-based framework for workflow and process automation. It makes use of BPMN models, and provides an extended set of tools that can be used to design, execute and automate business processes. These tools provide support for all the stages in the life cycle of a workflow, as mentioned previously.
In this section, the choice for choosing Camunda as a workflow framework will be argued for, and a brief overview of the architecture of the software will be given together with a small introductory use case.

2.1.4.1 Motivation

The reasoning behind choosing Camunda as the process automation framework is multifold. Keeping in mind the drawbacks of the existing scientific workflow management systems, such as the use of outdated languages and technologies, the ties to a particular domain or the lack of user-friendly interface, Camunda can be seen as a possible solution to the most significant of those aspects.

First of all, and as mentioned previously, Camunda is BPMN compliant. Since one of the important drawbacks of the previously used systems was the fact that they were, more often than not, using outdated languages and technologies developed in-house by scientists [6], BPMN does not have this drawback; it is a well-known modelling language that is easy to learn, and according to the literature, the most popular language used for describing processes [3]. Knowledge of BPMN is enough to get a user started with modelling and deploying their own models onto the system.
At the same time, Camunda is implemented in Java[2], and users can also write their own Java classes to add the desired behaviors to the services. As such, the problem that some of the existing WfMS had with regards to using outdated languages and technologies are not present with Camunda.

Secondly, the software has become increasingly popular as a commercial tool for process automation. Since Camunda works with BPMN models which are domain independent, it has a vast number of applications and can be easily integrated into existing solutions by a lot of organizations from a multitude of industries, such as finance, insurance and software[3].

Thirdly, Camunda is an ongoing project, being continuously developed and im-

[1]Official website: `https://camunda.com/`
[2]`https://www.java.com/en/`
[3]A list of organizations using Camunda: `https://camunda.com/about/customers/`

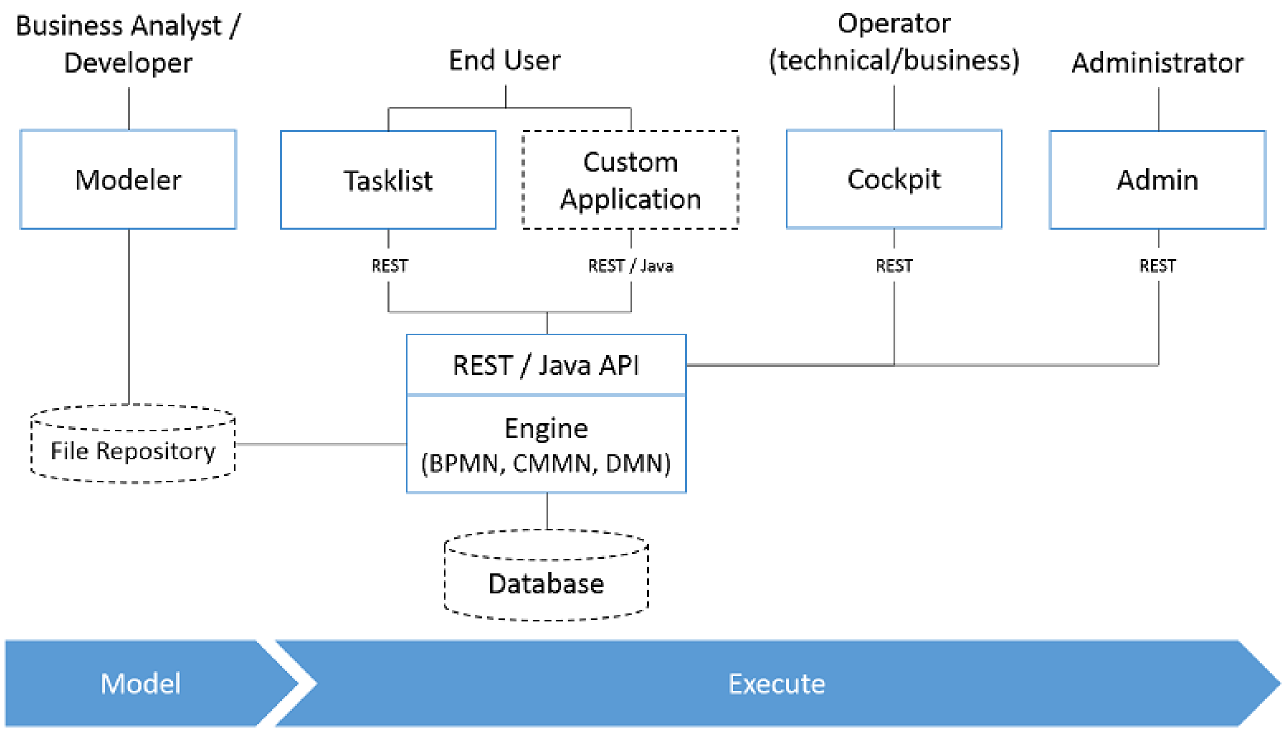


Figure 2.3: Overview of the architecture of Camunda

proved by both the 'parent' developers and the community. There are a lot of resources available to assist users with getting started with the application, such as extensive documentation, a multitude of examples and tutorials for beginners, a blog that is frequently updated and a forum where users can ask questions and discuss on various topics with other developers. New features are often suggested and rolled out by the community, and the development team is responsive to the ideas brought forward to them; in their official repository, individual users can make contributions by submitting their code via Merge Requests, which the team will then review, discuss and ultimately merge into the main project.

Lastly, Camunda is an open-source project, and the source code for the main, most important packages is openly available from the official repositories, and this comes with all the benefits (and disadvantages, of course) of an application that is open source. The source code of the Camunda Modeler and the Camunda Engine, together with the REST API are available for download from their official repositories.

2.1.4.2 Architecture Overview

As mentioned in section 2.1.2, most workflow management systems follow a similar pattern when it comes to their architecture. By analysing the different sub-applications that Camunda has, the same pattern can be observed.

Figure 2.3[1] gives a broad overview of the architecture of Camunda, as well as the typical user roles that will interact with the different components of the system [2].

The two different types of components that Leymann and Roller [10] attribute to a typical WfMS and that were mentioned in section 2.1.2.2, are also present in Camunda. Different terminology is used in this case; however, the concepts and the functionality behind each of these elements are in line with the ones mentioned in the literature.
On the one hand, the formally named *build time* components of a typical WfMS can be found in Camunda as part of the *Modelling* step, attributed to the `Modeler`.
On the other hand, the remaining sub-applications of Camunda fall under the *run time* components, or the *Execution* step. The `Engine` has the role of the *Workflow Enactment Service*, and it communicates with the other modules through interfaces provided by the `REST/Java API`.

Next, a brief introduction will be given for each of the modules. As previously mentioned, alongside the Engine, Camunda offers intuitive, user-friendly front-end applications that make it easy for users outside of the scientific world to use the software. Even though not all of these front-end applications were modified in the context of this work, they are worth mentioning because they were extensively used during the research phase and also are relevant for the Use Cases of this work.

The Engine The `Engine` is the element that lies at the core of Camunda. Whenever a BPMN model is deployed, this component will translate it into Java objects and store the its representation internally. It manages operations such as creation, activation, suspension or deletion of a process instance, a process deployment or a process definition. The Engine does the navigation through the model graph from one activity to another, executes all the activities' behaviors and communicates with the external process participants (such as web services or other engines). At the same time, it also handles the persistent data layer by managing with a database. According to the official documentation[3], The Engine provides an interface that can be used by other applications in order to communicate. At the core, the Engine is a lightweight application that can parse BPMN and DMN files, can be used standalone or together with all the other packages that Camunda has to offer. It can also interpret behavior representations (eg. Service Tasks that

[1]As found at: `https://docs.camunda.org/manual/7.14/introduction/`

[2]All of these tools are present in the Community Edition, which can be downloaded from the official website. An Enterprise Edition is also available and provides additional features. An overview of the differences between the two can be found here: `https://camunda.com/enterprise/`

[3]`https://docs.camunda.org/manual/7.14/introduction/architecture/`

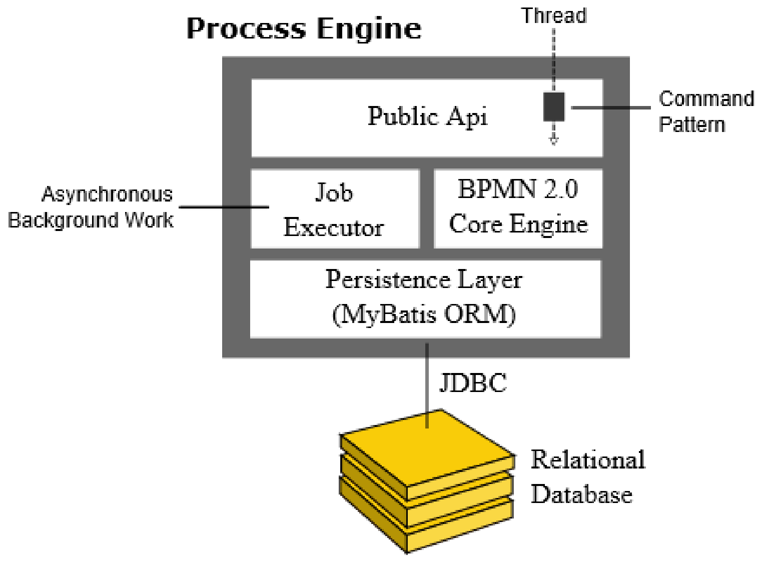


Figure 2.4: The architecture of the Camunda Engine, as presented on the official documentation website[1]

invoke a piece of code) and execute them during runtime.
An overview of the architecture of the Engine can be seen in Figure 2.4, as it is found in the original documentation.

Modeler The `Camunda Modeler` is a standalone application, a visual editor that can be used to produce BPMN models. It is a dedicated application that can be downloaded and installed on the user's local file system. The application offers a complete set of tools for generating models, such as modelling processes, assigning tasks to users, and it even has integrated functionality to allow models to be deployed to the Camunda Engine directly from the user interface.
Under the hood, the Modeler is a NodeJS[2] application. It is open source, and the source code can be downloaded from the official GitHub repository[3]. In order to communicate with the `Engine`, the `Modeler` sends requests to the `REST API`.
An overview of the UI of the Modeler can be seen in Figure 2.5.

One aspect to note about the `Modeler` is the fact that it does not insure against possible errors in the process model, such as deadlocks, livelocks, dead or useless transitions. This aspect should be taken into consideration by the user, who should make sure that the model adheres to the concept of *soundness* [15],

[2]https://nodejs.org/en/
[3]https://github.com/camunda/camunda-modeler

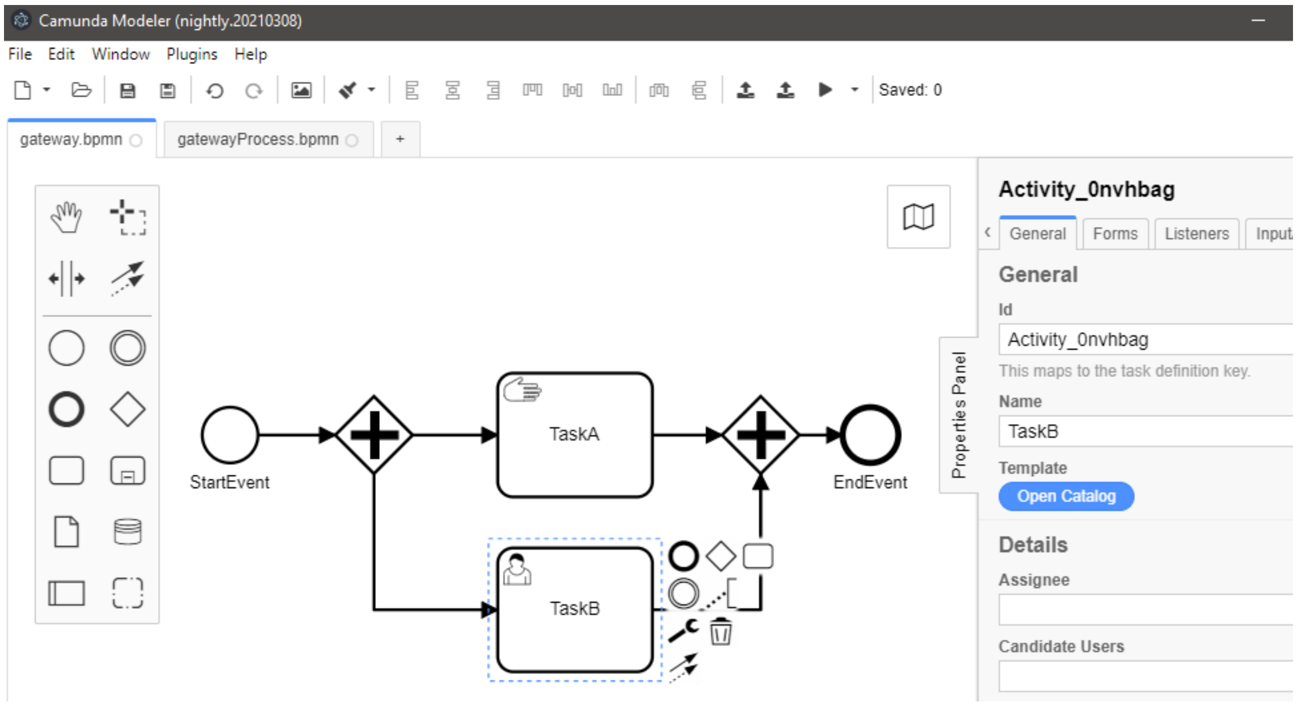


Figure 2.5: An overview of the Camunda Modeler

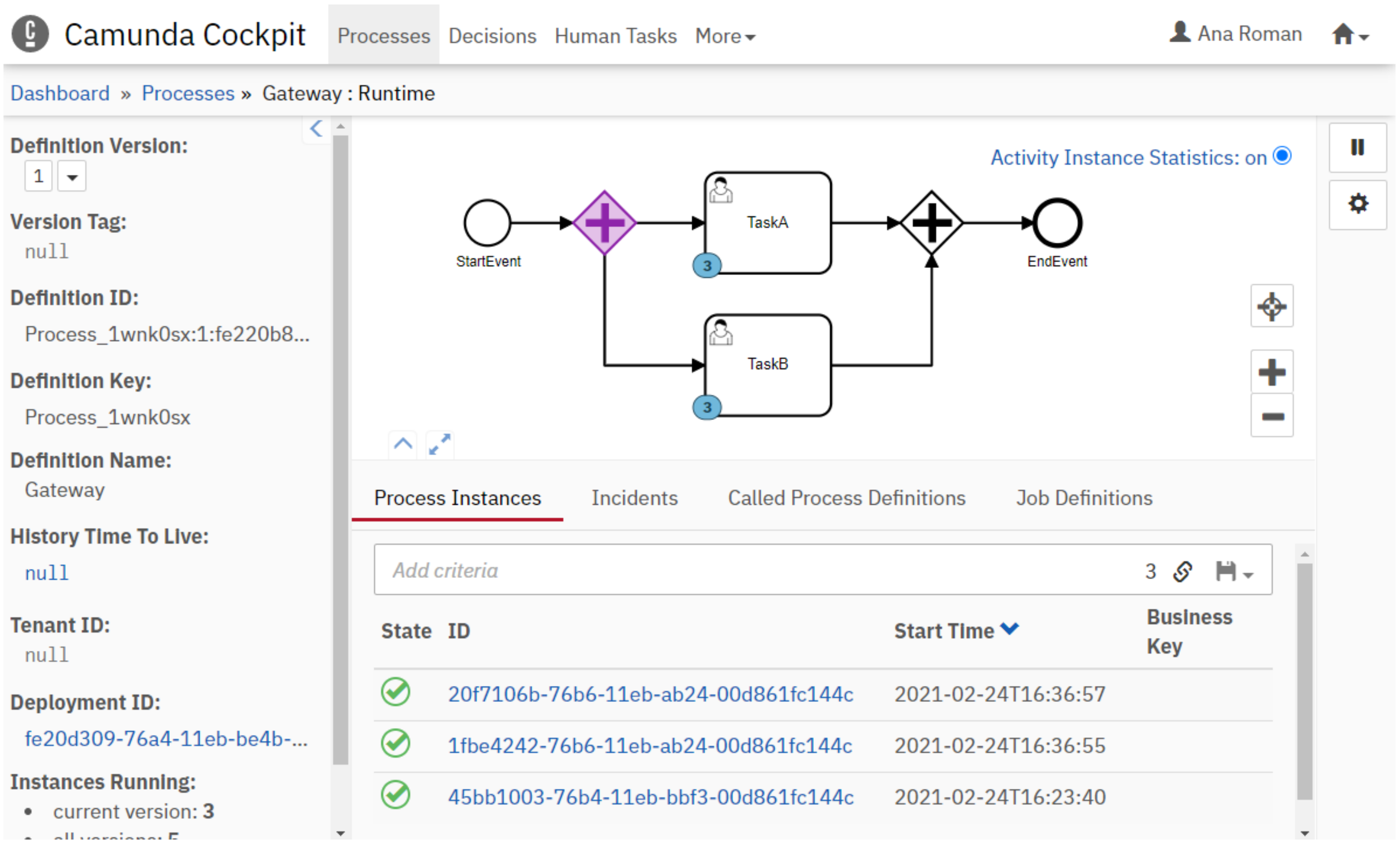
discussed in section 2.1.2.

Cockpit Out of the three front-end applications that Camunda has, the Cockpit is the most important from a development point of view, due to the multitude of functionalities that it has and the information that it provides. It can be used as an administration and monitoring tool for the process models and their respective deployments, process instances and the currently active activities. It allows the user to see specific information about a process, such as versioning, the number of running instances and which activities are being executed. It also allows the user to suspend and resume an entire process, or a single process instance.

The Cockpit also allows the inspection of a single (running) process instance, which allows the user to see the information specific to that instance, such as data (in the form of variables), as well as the User Tasks and the users that have been assigned to them.

A more extensive explanation of all the aspects and functionalities of this web application is given in the official documentation[1]. An overview of the Cockpit showing a deployed process definition that has three running process instances can be seen in figure 2.6.

[1] https://docs.camunda.org/manual/7.14/webapps/cockpit/



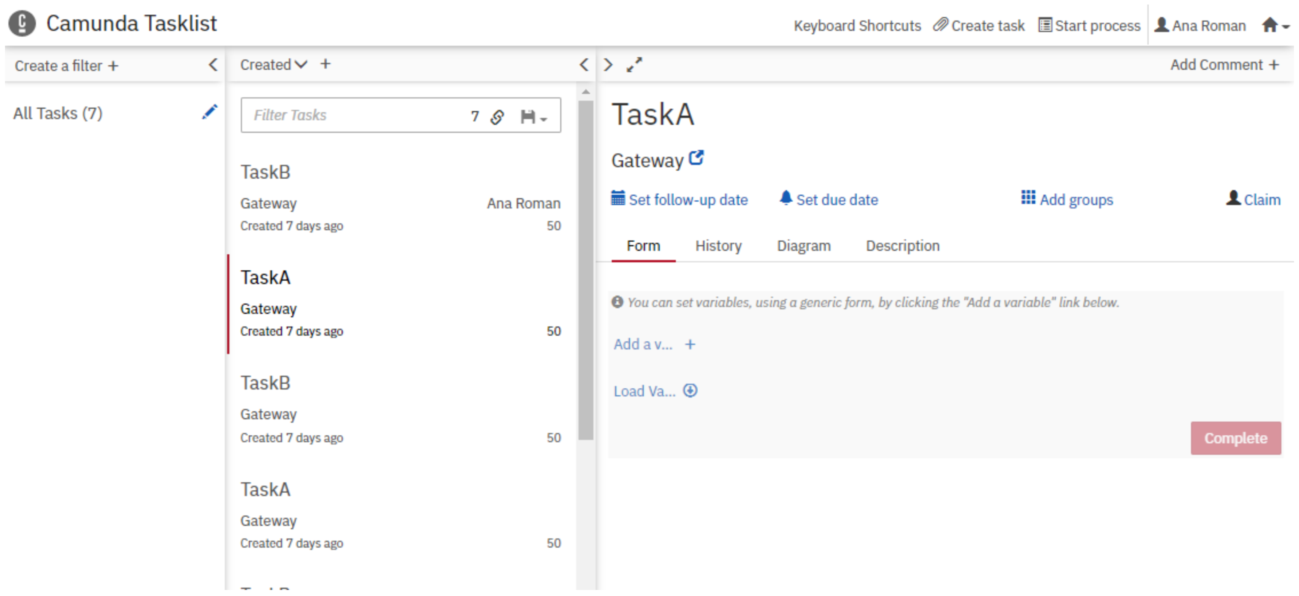
Definition Version: 1
Version Tag: null
Definition ID: Process_1wnk0sx:1:fe220b8...
Definition Key: Process_1wnk0sx
Definition Name: Gateway
History Time To Live: null
Tenant ID: null
Deployment ID: fe20d309-76a4-11eb-be4b-...
Instances Running:
- current version: **3**

State	ID	Start Time	Business Key
✓	20f7106b-76b6-11eb-ab24-00d861fc144c	2021-02-24T16:36:57	
✓	1fbe4242-76b6-11eb-ab24-00d861fc144c	2021-02-24T16:36:55	
✓	45bb1003-76b4-11eb-bbf3-00d861fc144c	2021-02-24T16:23:40	

Figure 2.6: An overview of the Camunda Cockpit

Tasklist This application allows for the management of User Tasks, such as searching, filtering, creating new tasks or completing active ones, as well as allowing users to claim tasks, to reassign them or to set due dates.
After a task has been assigned to a certain user in the Cockpit, the task will appear in that user's Tasklist view, and the user can access it, view all the information provided by the task or insert data whenever necessary. Afterwards, whenever a user completes their tasks, the Activity will be marked as completed in the process instance as well, and the execution will move on further in the graph.
Another important functionality that is provided by this webapp is the possibility to start a new Process, by pressing the `Start Process` button in the top right corner of the screen. Whenever that button is pressed, the user is shown a generic start form where variables can be specified and then a new process instance of the selected process will be started.
More information over the Tasklist can be found in the official documentation[1]. An overview of the user interface of this web application can be seen in figure 2.7.

[1] https://docs.camunda.org/manual/7.14/webapps/tasklist/



You can set variables, using a generic form, by clicking the "Add a variable" link below.

Figure 2.7: An overview of the Camunda Tasklist

Admin The Admin application is responsible for the User Rights management, handling user identity and authorization. The application can be used to have an overview of all the registered users of a given application, their roles, tasks, and Group membership. More explanations can be found in the official documentation[1].
For the purpose of the current work, this module has not been used or interfered with, and as such, will not be treated further.

Enterprise Edition The Camunda software also offers an Enterprise Edition. This edition provides a larger toolbox of functionalities for their users, and they mainly focus on providing User Interfaces for operations that the Community Edition allow only through code or `REST` requests.
Examples of such operations are the deletion of a `Process Definition` or providing access to the history of all `Process Instances`, including the completed ones (in the Community Edition, once a `Process Instance` has finished, it is no longer visible to the user in any of the Web applications). The Enterprise Edition also offers a user interface for the conducting of an `Instance Migration`[2]. The `Instance Migration` approach will be explained and discussed in more detail in section 4.2, together with all the limitations that it has. The same limitations that will be discussed in the following sections are also present in the Enterprise Edition. And at the same time, this edition is not open source and also sits behind a costly paywall, which scientists are not always guaranteed to have the means to pay.

[1] `https://docs.camunda.org/manual/7.14/webapps/admin/`

[2] More details can be found at: `https://docs.camunda.org/manual/7.14/webapps/cockpit/bpmn/process-instance-migration/`

2.1.5 Alternatives to Camunda

In this section, a few WfMS will be shortly discussed as possible alternatives to Camunda. One thing to note is that all the applications that will be mentioned are not open source, which was a factor of major importance in the selection process. However, they do exist on the market at the moment of writing this thesis, and as such are worth mentioning.

Kissflow Kissflow[1] provides its users with a worklow automation package, which also includes tools that cover the entire life cycle, from modelling to deployment, execution and monitoring. This platform is heavily focused on business processes and targeted towards businesses and customers, advertising advantages such as ”No code” and a visual interface that is easy to use - no BPMN, but a visual editor that is supposed to make modelling easier. These advantages can prove to be useful in a business context, where the employees of a company have little to no programming knowledge, but at the same time they can be detrimental to scientists and their experiments, because of the very limited amount of customization and the limited possibilities that are offered.

Others Other WfMS systems that provide means of modelling and executing processes are IBM WebSphere Process Server: [2] and the Oracle BPEL Process Manager [3]. However, those applications are heavily enterprise-oriented. While the set of tools that they provide are beneficial in the business context, they do not satisfy all the requirements that present themselves in the development of an application in the scientific domain. For example, ad-hoc adaptation features are not available.

[1]https://kissflow.com

[2]https://www.ibm.com/cloud/blog/websphere-trial-options-and-downloads

[3]https://www.oracle.com/middleware/technologies/bpel-process-manager.html

2.2 Related work

This section takes a look at the literature that covered topics similar to the research presented in this thesis. It discusses existing research done on adaptive processes and process flexibility in the context of e-Science and Business Process Management (BPM). These works serve as a foundation for the requirements analysis of developing flexible processes, which will be introduced in the following section.
This section is mainly based on the published paper “Model-as-you-go: An Approach for an Advanced Infrastructure for Scientific Workflows” [6] as well as the dissertation thesis of A. Weiß, “Flexible Modeling and Execution of Choreographies” [8].

2.2.1 The Model-as-you-go approach

As mentioned previously, the concept of *adaptable processes* has been studied before and various implementations have been proposed by the literature. One such study has been conducted by Sonntag and Karastoyanova in their work [6], which helped lay the foundation of a new approach that would extend an existing WfMS with advanced flexibility capabilities. The main goal of this study was to help the scientific community create applications and experiments, by providing them with an empirical design of a WfMS that would mirror the trial-and-error manner in which such work is conducted.

The new approach makes the development of processes more efficient and robust, because it allows the user to bring modifications to a running process instance without losing the progress that was already made. And this is a very important contribution, since scientific workflows can be long-running, spanning over the course of multiple days, can invoke multiple systems, and possibly uses resources that may not always be available. With these aspects in mind, it is evident that restarting such a process is not always an option.

Despite the fact that the application of their work is based on the BPEL modelling language, as well as a BPEL engine for executing the processes, most of the ideas that are discussed were presented on a conceptual level in order to allow them to be applied to other systems.

Wavefront One important detail to note is the fact that in their work, Karastoyanova and Sonntag [6] use the term *wavefront* to describe the currently active activities, which can also be seen as the present of an instance [20]. In Camunda, the concept of *wavefront* exists as well and is implemented by the means of a *token*. In the monitoring tool (eg. the Cockpit as seen in figure 2.6), the tokens can be seen as the number indicators on the bottom-left corner of an activity.

2.2.1.1 Conceptual requirements

When developing the empirical concepts for the WfMS, the following high-level requirements have been identified in their work. Below, the requirements will be listed and discussed. A number of these requirements are also applicable to the context of the present work, and will be treated in the following chapters.

MR-1 Providing an integrated tool that supports most of the workflow life cycle.

MR-2 Enabling the trial-and-error manner of development, whenever the workflows are not fully known prior to their execution.

MR-3 The merging of the execution and monitoring phases of the workflow life cycle.

MR-4 Enabling the management of executions directly from the modelling tool.

MR-5 The integration of the tools for instance monitoring and modeling, in order to view the results.

MR-6 Enabling the looping back from the execution phase to the modeling phase by allowing instance adaptations; at the same time, the users should be unaware of the technical details behind the adaptation process, and the migration to new versions should be seen as part of the modeling phase.

2.2.1.2 Requirements analysis & suggested solutions

In the current section, the requirements enumerated earlier in Section 2.2.1.1 will be discussed, together with the suggested solutions. The discussion will be kept at a conceptual level, since the implementation suggested by the authors concerns a different application.

MR-1 This requirement is rooted in the fact that scientists themselves handle every step of the workflow life cycle, and having a single tool that integrates most of the steps will be beneficial for their work, since it removes the need to switch between tools.
In order to achieve this requirement, the workflow engine was updated such that it would publish execution events about the process instances that were deployed (eg. information such as status - running or suspended, currently running activity, etc). The modeling tool would pick those execution events up

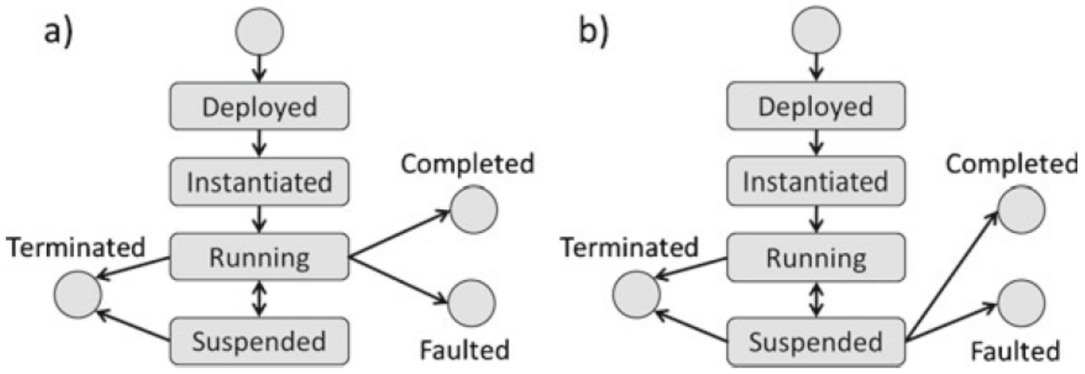


Figure 2.8: Process states in a conventional workflow system **a)** and in the Model-As-You-Go approach **b)**, as found in [6], where a process instance enters the *Suspended* state after execution is finished.

and display them to the user. At the same time, the workflow modeling tool was extended with functionality that allowed it to subscribe to the events published by the engine, such that it could display information about process instances.

MR-2 In order to enable the trial-and-error approach for development, there are a few steps that should be achieved: first of all, the user should be able to select, in the modeler, the workflow instance that they wish to change, and should have the possibility to suspend its execution. Second, the user will make the desired modifications to the model and then re-deploy it on the engine (as a new version), with the logic of the old workflow instance adapted to the new one.
Requirements MR-3, MR-4 and MR-6 are also related to the current one, and will give more concrete descriptions of the steps involved.

Another functionality that had to be implemented in order to enable this approach was the possibility to keep a workflow instance 'alive', even after the last activity in the process has been executed. For this, a modification to the process state life cycle has been introduced, namely, after completion, a process enters the "*Suspended*" state instead of going into the "*Completed*" state. From being suspended, the user can decide what to do with the instance: either to complete it, or adapt it and re-run it.
A depiction of the process state diagrams can be seen in figure 2.8.

MR-3 Since requirement MR-1 enforces the use of an integrated tool for the steps in the

life cycle of a workflow, this requirement MR-3 can be related to the previously mentioned one. Through the modeler, the user should be able to select a specific workflow instance, and see the state of its execution, for example, to be aware of which activities are being executed.
At the same time, the modeling tool should also be able to handle variable values for individual workflow instances. Therefore, this extension was also implemented.

MR-4 In order to enable the administration of executions directly from the modelling tool, alongside the monitoring properties mentioned in the previous requirements, the tool has been extended with further management functions, such as the possibility to suspend, resume, terminate and delete workflow instances. These operations were offered through a simple visual interface (eg. a toolbar), and were applied to the workflow instances that were currently selected and monitored by the user (as mentioned in MR-3). The workflow instance ID is used to correctly identify which processes should be altered.

MR-5 The integration between the monitoring and modeling tools has also been treated in MR-1 and MR-3. However, according to this particular requirement, the user should also be able to view the results of the workflow instance (such as the handled data, variables, etc). In order to achieve this, the workflow instance states are being saved to a persistent storage module. Whenever an instance is selected for monitoring, the information can be fetched from the storage and communicated to the modelling tool, which will then present it to the user.

MR-6 As a continuation of the previously mentioned requirement MR-2, this requirement targets the migration of a currently running (suspended) process instance to a new version of the workflow model.
On the one hand, the authors discuss the different issues that arise when handling different versions of a workflow, and introduce the term *Concurrent workflow evolution* to describe the situation when multiple versions of a workflow are concurrently active.
On the other hand, whenever an instance is migrated to a new version, the logic and data related to the instance should be ported and adapted from the old version. To ensure the compatibility between the two versions, a validation component has been implemented.

In order to implement the instance adaptations, the Model-as-You-Go approach also discusses the re-run of workflow parts. This operation can be done in two ways:

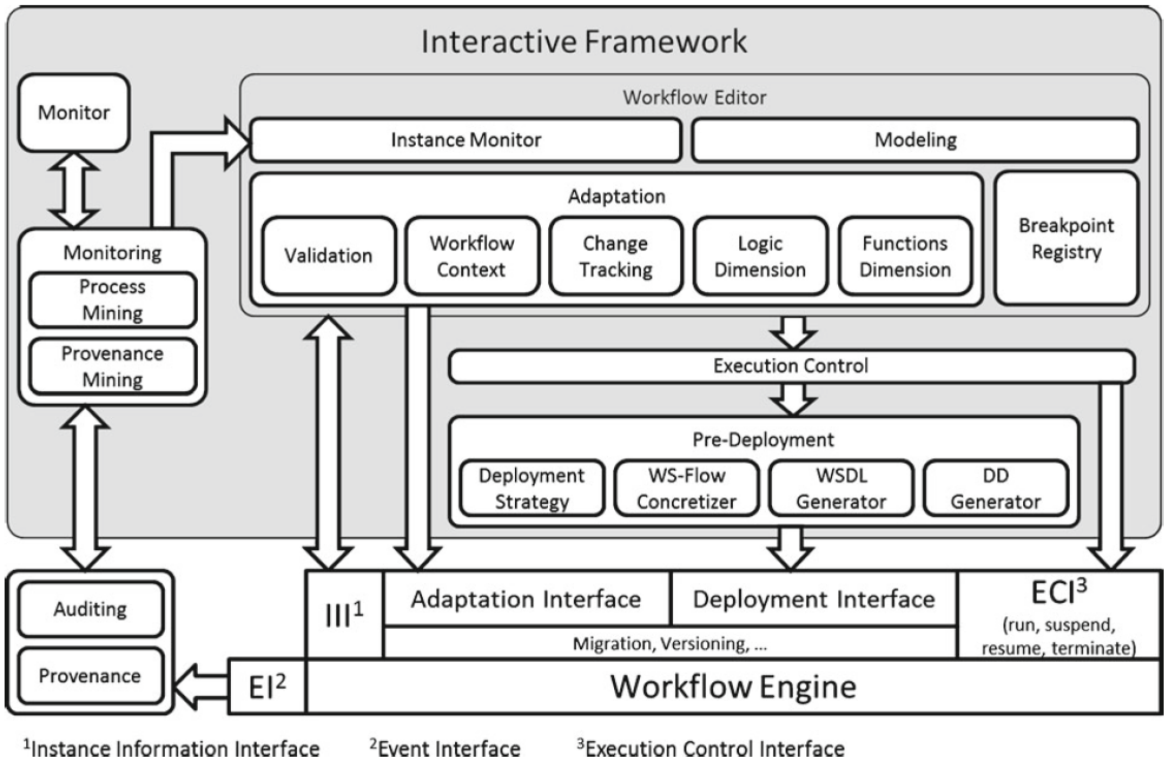


Figure 2.9: The architecture of Mayflower, as presented and described in [6]

(a) *iteration*: will run the activities again, starting from any given activity (possibly selected by the user).

(b) *re-execution*: will undo already completed tasks by executing their compensating behaviors, and then re-runs activities. This operation is especially important whenever idempotent services are executed or a workflow changes the state of a different system, for example.

The authors of the Model-as-You-Go approach have implemented all of the requirements mentioned above, and more, in a proof-of-concept application called *Mayflower*, based on the BPEL engine.
The requirements that the *Mayflower* system had to implement served as the foundation for the requirements of the current thesis. The fact that they were kept at a conceptual level, without mentioning any implementation-level details, made them translatable and portable to any system or engine that should be extended with the aforementioned capabilities.

The architecture of Mayflower can be seen in figure 2.9, and more explanations about the implementation details can be found in the original article [6].

2.2.2 "Flexible Modeling and Execution of Choreographies" - Andreas Weiß

The dissertation thesis of A. Weißputs together a number of prior research contributions, in order to achieve the flexible modeling and execution of choreographies. At the core of his work, lies the vision to *support users from diverse application domains [...] to solve problems in an easy, flexible manner* [5]. His vision is in line with goals of the Mode-As-You-Go approach authors, as well as with the vision of the current thesis, and some of the ideas that he presents were kept in mind when developing the requirements and solutions of this thesis.

The application of his work is also focused on workflows, workflow management systems and the functionalities of business and scientific workflows. His work brings forward a number of important research contributions, by presenting prior research done on the topics of flexible modeling and execution and extending them with an application to choreographies. An overview of his work's contribution can be seen in figure 2.10.

2.2.2.1 High-Level Requirements

Below, the high-level requirements of the dissertation are presented, together with a short explanation.

ChorR-1 User-friendly modeling
A user-friendly way of modelling processes, including the reuse of (parts of) existing models.

ChorR-2 Technical transparency
The technical and implementation details of the system should be hidden from the user as much as possible.

ChorR-3 User-driven control
A user-friendly way of controlling the execution of process instances, with actions such as starting, pausing, resuming or terminating.

ChorR-4 Integration of modeling and monitoring
The integration of the two reduce complexity and technicalities for the user and can be beneficial to non-IT experts.

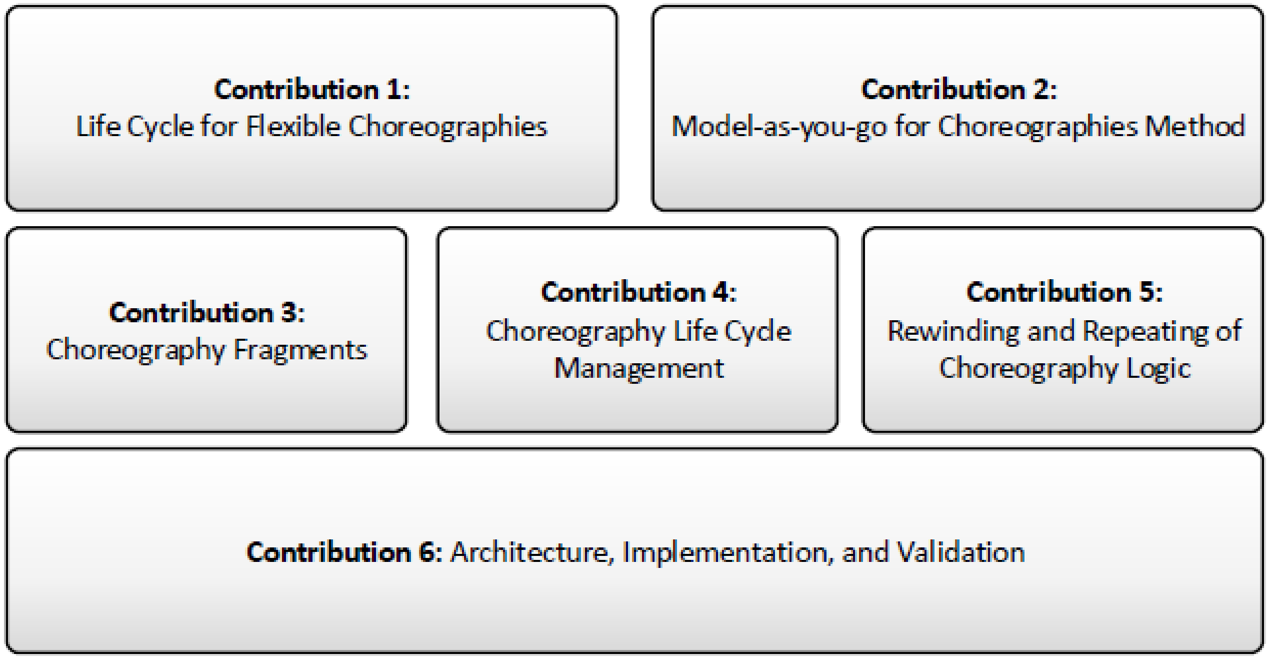

Contribution 1:
Life Cycle for Flexible Choreographies
Contribution 2:
Model-as-you-go for Choreographies Method
Contribution 3:
Choreography Fragments
Contribution 4:
Choreography Life Cycle Management
Contribution 5:
Rewinding and Repeating of Choreography Logic
Contribution 6: Architecture, Implementation, and Validation

Figure 2.10: The research contributions of A. Weiß, as presented in his dissertation thesis [5]

ChorR-5 Flexibility mechanisms
Enabling the flexibility mechanisms introduced by the *Model-as-You-Go* approach.

ChorR-6 Separation of concerns
This requirement describes the separation of concerns principle as known in the software engineering field [21].

ChorR-7 Standard compliance
The concepts and tools used for the development of his work should be compliant with the industry IT standards.

As one can already observe, some of these requirements are overlapping almost entirely or on a conceptual level with the requirements presented by the authors of the *Model-as-You-Go* approach. At the same time, these requirements were also kept in mind in the context of the current thesis.

One important aspect of the work of A. Weißis the size of the contribution that he has brought to the research field of workflows. As previously mentioned, he brings together a number of previous research topics and papers. From each of these studies, he adopts the ideas and concepts that are the most important and pieces them together, such that they form the foundation of his work. From there, he proposes a new proof-of-concept application, from the architecture to the implementation, that should encompass all the research contributions as well as all the requirements that he discovered.

His work is broad and it treats multiple subjects, and the concept of adaptive processes is only one of them. As can be seen in figure 2.10, he also introduces other concepts, such as *Choreography Fragments*, *Choreography Life Cycle Management* and *Rewinding and Repeating of Choreography Logic*. While these concepts are of outmost importance for the vision of his work and also for the current thesis, they are outside the scope of this thesis and will be left as ideas for future work.

Chapter 3 | Methodology

This chapter describes the requirements analysis for extending a workflow engine with the possibility of deploying adaptable processes. The related work and literature sections have already illustrated that this subject has been extensively researched, and the conceptual groundwork has already been laid by the authors of the respective studies. As mentioned before, the conceptual nature of their studies serves as a guide for the implementation of these requirements on specific engines. As such, the majority of ideas and requirements that are described in this section have been adopted from the literature and follow the same conceptual steps. The resulting requirements framework will be used as a reference when implementing the adaptable processes concept for business processes.

In the context of this work, the term *adaptable process* will refer to a process instance that has the ability to deal with changes and modifications. Process adaptability should be applied to the process instance during run time.

The chapter is structured as follows: Firstly, section 3.1 introduces two motivating scenarios for adaptable processes, each from a different domain of application. Then, it will describe the specific requirements that are to be developed to cover the issues that arose in the two scenarios. Secondly, section 3.2 will discuss the specific objectives of the adaptable processes.

3.1 Requirements

The current section will give an overview of the requirements which stand behind the implementation of the adaptable processes concepts for business process models, which will be presented in the next section.
The current section is structured as follows: At first, subsection 3.1.1 will introduce two use case scenarios, one from a business context and the second from a scientific experiment, where the users could benefit from the possibility of having an adaptive

process instance. These scenarios are either fictional or adopted from the literature. Then, with the insights of the literature presented in the earlier section together with the scenarios, a framework of requirements will be presented in subsection 3.1.2.

3.1.1 Use Case Scenarios

Table 3.1 presents two conceptual scenarios, each from a different application domain: business and scientific. For each scenario, its origin, a description as well as the problems that it comes across are presented.

Scenario	**Description**	**Problem Context**
Business domain		
UC-1 Loan approval system	Most of the steps are automated, except a few which require manual user input. In one case, a manager notices a mistake in the input, which ultimately changes the output of the process	- The process instance is already started. The user has already introduced his confidential data, which is not stored anywhere else. - There is no way of changing the course of the process through a user interface; this operation can only be done with the help of an IT expert - If the process is left to continue as-is, the output would be wrong
Scientific domain		
UC-2 Scientific simulation	When conducting an experiment or a simulation, a scientist would like to be able to check the output of each operation (activity) before moving on to the next steps, to be able to make the required changes to the data if needed.	- The scientist would like to be able to inspect the data after every step in the pipeline - The scientist would like to be able to make changes to the model after each step, to have a better overview of the output - The scientist wants to be able to go back to a previously executed step in order to introduce different parameters and to run it again, without losing the rest of the progress - The data can consist of high-resolution 3D images, so every step of the process is time consuming (eg. a few hours on a personal computer)

Table 3.1: Scenarios requiring adaptable processes and their requirements

The given scenarios in Table 3.1 illustrate that adaptive processes are needed in both domains. The situations deal with different types of data, but still do benefit

from the adaptation of a process instance that would transfer the data from one instance version to the new one, without the loss of data.
The benefit of having this type of adaptation are different in each case. For the loan system, it could correct an erroneously entered piece of information. For the scientist, it would enable an exploratory fashion of modelling a process, where the control and data flows are not entirely known from the beginning.

For both scenarios, restarting a process definition is not a feasible option, firstly because sensitive data is lost, and secondly because the individual activities can be too time consuming.

3.1.2 Requirements Analysis

The current subsection will give an overview of the requirements framework of the thesis at hand, deduced from the literature as well as the use cases presented in the previous section.

3.1.2.1 Requirements Framework

In order to produce the requirements framework of the current thesis, a few steps were taken.
First of all, the requirements of the literature study presented in section 2.2 were deduced and summarized. Then, the most relevant requirements were outlined and selected as requirements for the current work.

One aspect to note is the fact that Camunda already fulfilled some of the requirements that had to be implemented by the previous literature, and as such, those will be omitted. One such example is requirement MR-3, which involved the merging of execution and monitoring phases. This requirement can be omitted, since the Camunda Cockpit mentioned in section 2.1.4.2 already handles the two phases in the same user interface. Another example of such a requirement is ChorR-3, because Camunda already implements mechanisms for user-driven control.
The collected requirements were classified into four categories, which are presented below.

R-1 Concept and implementation
The current thesis should follow the relevant literature and present the theory that was applicable when extending the workflow engine, at a conceptual level.
At the same time, the thesis should give a high-level overview of the implementation details of the suggested solution, together with arguments to support the

decisions taken about the architecture and design of the application.
The implementation should follow the usual design principles, such as separation of concerns, as explained in ChorR-6, and it should be as non-intrusive as possible.

R-2 Runtime adaptation on the Control Flow
The flexibility mechanisms previously mentioned from the literature should be made available to the user, such as the migration of a running process instance to a new version of the process model. The adaptation should be done through a user-friendly graphical interface, such as to hide all the technicalities from the user.
The user should also be able to re-iterate parts of the model, starting from a specified activity.

R-3 Data flow preservation
Changing a process instance (through a migration or re-iteration) should preserve the data that had been obtained in the original instance, by transferring it to the new instance without any loss of information.

R-4 Organizational change
User Tasks or tasks that have assigned a certain system or endpoint should be able to be adapted as well. If it is desired that the new version of a process instance should re-assign an activity (or multiple), this change should be picked up by the process and it should be propagated to the new version.

3.1.2.2 Non-functional requirements

This subsection will outline the most important non-functional requirements at a conceptual level.

NF-1 **Usability**
In the vision presented in section 1.1, the focus was on assisting users of a WfMS with the added functionality of adaptations. As such, the users should be able to easily deploy an adaptive process on the engine, without any prior technical knowledge.

NF-2 **Usefulness**
Through the adaptable process concept, the dimensions of time, cost and quality should be improved in the context of both business and science processes, for example, by reducing the required execution time of a process or by allowing changes that correct an error in an activity.

NF-3 **Non-intrusive changes**
The implementation of the new functionality should be as non-intrusive as possible, and it should not interfere with the way that the engine functioned prior to the changes.

3.2 Adaptable Processes

Keeping in mind the aforementioned use cases in subsection 3.1.1 and the requirements of the previous subsections, this section will introduce the logic flow behind the adaptation of a process instance . The section will look at the two use cases and suggest possible solutions that cover all the issues mentioned in the problem context in each use case (eg. table 3.1, by using process instance adaptation).

3.2.1 Loan application procedure

Since the specific steps of a loan application procedure are outside the scope of this work, a simplified, high-level version of such a procedure can be seen in figure 3.1.

As explained in the use case, the situation where a problem could arise would be the case when the employee who was supposed to make the decision about the loan made a mistake, and marked the loan as rejected. As such, the process instance proceeds by executing the sub-process *Perform rejection procedure*, and the manager only notices the error when the task to set the loan application status to 'finished' shows up in his list of active tasks.
Before bringing any modifications to the engine, the process instance is rather static, meaning that any changes in the control flow of the process instance cannot be done easily through a user interface.

However, by implementing the adaptable concept, the manager who noticed the error can open the respective model in the `Modeler`, and decide how to handle the situation.

One possible solution would be to change the process definition model to include the possibility to loop back right before the decision was made and to change the decision made by the previous employee.
To do this, the manager would have to bring a modification to the already existing model, and indicate to the modeler that he would like to deploy the process in an adaptable manner, rather than a new deployment. Then, he can indicate in his assigned task that the process is not yet finished, and the outcome of the process instance will change. Since the adaptation flow will ensure that the data belonging to the old process is preserved and migrated to the new version of the model, the manager does not have to worry about losing the client data that was already present.
The new version of the model can be seen in figure 3.2.
Since this modification is ad-hoc, brought only to the specified process instance, it will

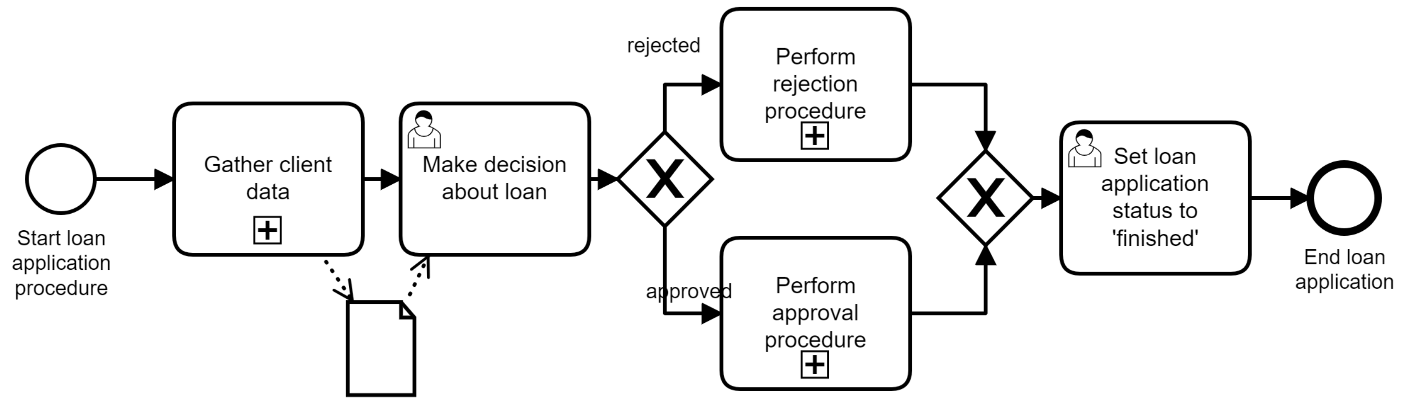

rejected
Perform rejection procedure
Gather client data
Make decision about loan
Set loan application status to 'finished'
Start loan application procedure
approved
Perform approval procedure
End loan application

Figure 3.1: The initial model of a loan application procedure

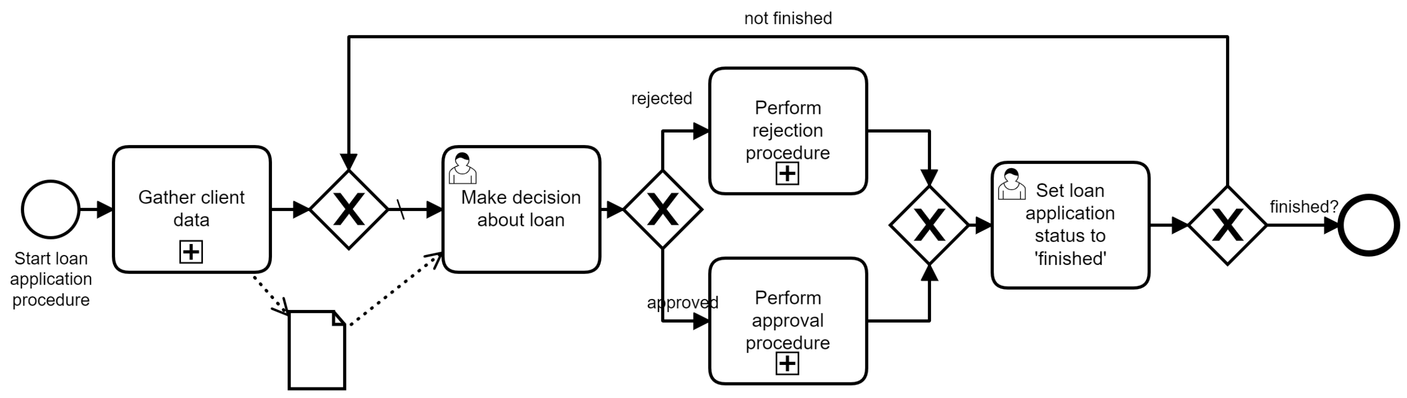

not finished
rejected
Perform rejection procedure
Gather client data
Make decision about loan
Set loan application status to 'finished'
finished?
Start loan application procedure
approved
Perform approval procedure

Figure 3.2: The modified model of a loan application procedure

not interfere with the other running process instances of the same process definition.

3.2.2 Scientific simulation

For this use case, we can take as an example the typical process involved in conducting an experiment. Supposing that a scientist first has a sub-process ready that does some pre-processing on some data. Then, the scientist has to provide some parameters and some files as input, execute some code and then inspect the results of the execution. If the scientist decides that the results of the two steps are satisfactory, the simulation can be started. The example process model can be seen in figure 3.3.
However, if after inspecting the results of both steps, the scientist notices that the output data of the second step still needs some modifications before it can be used as input into the simulation, one more activity should be added to the model; the activity should execute a piece of code that handles these modifications. Then, the scientist performs the adaptable logic on the running process instance by deploying the new model and migrating the running instance to the new version. The new version of the running instance will then execute the newly added activity whenever the scientist decides to complete the user task *Inspect results of second step*, and the

process will continue with the simulation. All the data of the previous process is also preserved and transferred to the new version. The new version of the model can be seen in figure 3.4.
There is no limit to the number of modifications that can be brought to a model. This represents an advantage, since scientists do not have to worry about missing steps in their models, or not being able to make small changes to their work. These cases can easily be handled through adaptations.

Another option that the scientist has is to keep only the data that has resulted from the pre-processing step, and restart all the other activities. In this case, the new activity *Execute modifications* can be added in the modeler and the new version of the model can be deployed as an adaptable process. If the ID of the activity *Provide directory of data files* is also passed to the *deploy adaptable* window, the new process instance will know to pick this activity as the starting point.
When this happens, the process will re-execute all the other activities except the pre-processing step, and all the data from the old version will be overwritten in the new process with the new values.

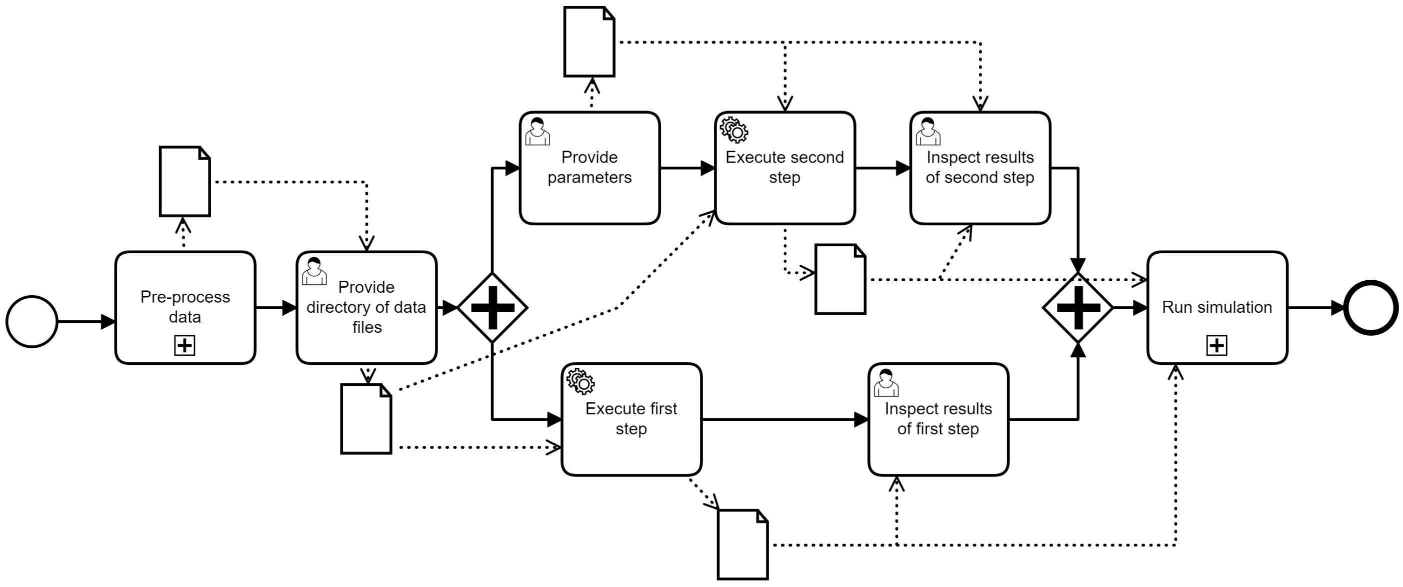

Pre-process data
Provide directory of data files
Provide parameters
Execute second step
Inspect results of second step
Execute first step
Inspect results of first step
Run simulation

Figure 3.3: The initial model of a scientific simulation

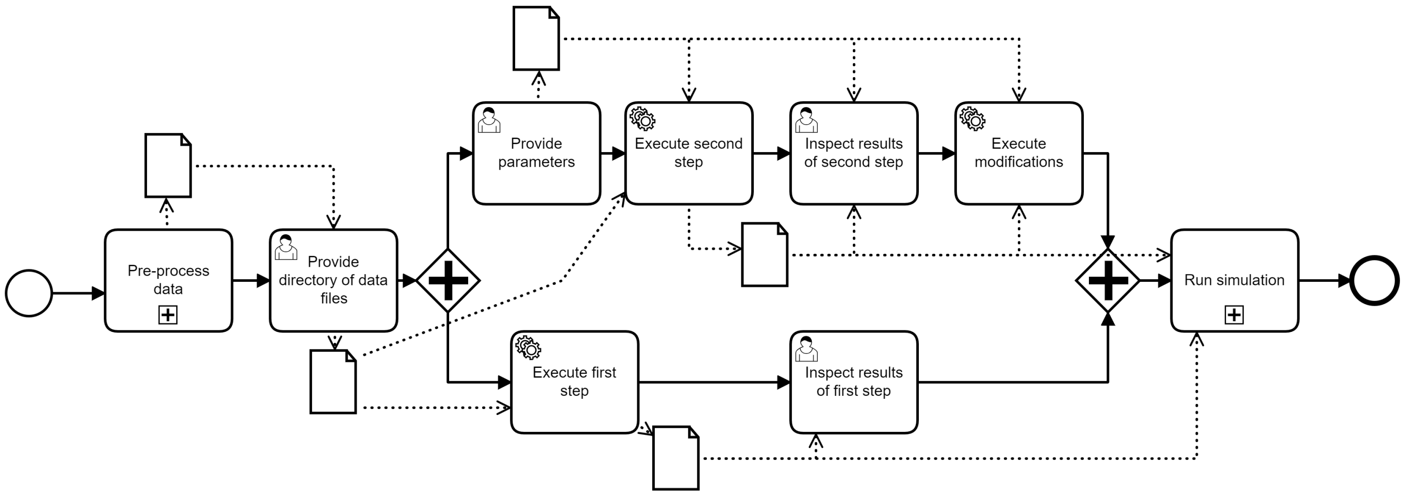


Figure 3.4: The modified model of a scientific simulation

Chapter 4 | Suggested Solution and Implementation

The previous chapter presented the conceptual requirements that should be followed when extending a workflow engine with the capability of deploying adaptable processes.

4.1 Camunda - Basic Concepts

The current section will present an overview of the architecture and design of Camunda, as it is implemented at the time of writing this thesis[1]. Keeping in mind that Camunda has a lot of functionalities to help users develop workflows, only the aspects that were changed and that were most relevant for the context of this thesis will be mentioned. More information on the implementation of Camunda can be found in the official documentation[2].

4.1.1 The Process Engine

As mentioned previously in section 2.1.4.2, the `Process Engine` is the module that lies at the core of Camunda. Figure 2.4 gave an overview of the architecture of the engine.

The implementation of the engine is very complex and it encompasses a lot of functionality. To better handle its high degree of complexity, the developers have split the main package into multiple services that have specific functionalities. An overview of the main, most important services can be seen in figure 4.1. Below, a brief introduction will be given to the services that have been used when writing the implementation of the current research.

[1]Since Camunda is an ongoing project, some of the details mentioned in this section might be subject to change.

[2]Camunda official documentation `https://docs.camunda.org/`

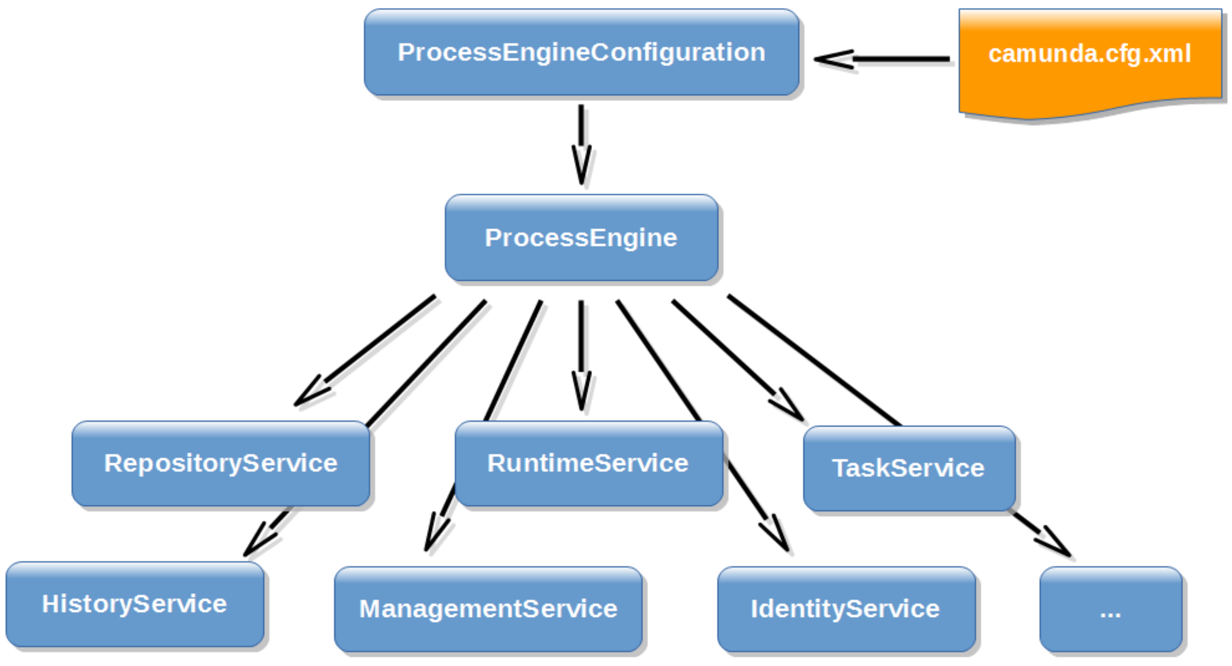


Figure 4.1: Overview of the Camunda Engine services, as found in the official documentation

The complete list of services together with more detailed explanations on the functionalities of each specific service can be found in the official documentation[1].

- `ProcessEngineConfiguration`

  This class will read a file that contains specifications of the Engine. Then, it will initialize and build a process engine based on those specifications whenever Camunda is started. This process engine will be used throughout the whole lifetime of the application.

- `ProcessEngine`

  This is the object that the `ProcessEngineConfiguration` will build on startup. It is a Java object representation of the engine. It gives access to a public API that can be queried in order to interact with the other services.

- `RepositoryService`

  This service is the most important for the context of the current work. It can be used to manage and manipulate static data, such as process deployments and definitions that are known to the engine. It uses the internal database

[1] https://docs.camunda.org/manual/7.14/user-guide/process-engine/process-engine-api/

to store and fetch this information.
For example, whenever a new model is deployed on the engine, the `RepositoryService` uploads the model file onto the engine, it inspects it and parses it, creating the internal representation of the model. Then, it creates a new `Deployment` object and assigns it a unique deployment ID; afterwards, it creates a new `Process Definition` with a unique ID. In this context, a process definition can be seen as the representation of the process model on the engine. The process definition can now be used to start process instances.

In short, the functionalities of `RepositoryService` are the following:

1. it can query for processes definitions and deployments;
2. it handles the versioning of the definitions; Whenever a new version of the same definition is deployed on the engine, the service will assign to it a version number higher than the previous one.
3. it can be used to suspend process definitions, which will not allow new instances of the respective definition to be started any more; at the same time, it also allows re-activation of a definition, which does the opposite;
4. it can retrieve resources related to process definitions, such as the model files, diagrams or their internal representation. For example, the `RepositoryService` can retrieve the list of Java objects that represent all the `Activities` in a model;

Listing 4.1 provides a code snippet that illustrates how the `RepositoryService` can be used to query for a `ProcessDefinition` by using its ID.

```
ProcessDefinition originProcessDefinition =
        engine.getRepositoryService()
                .getProcessDefinition(originProcessDefinitionId);
```

Listing 4.1: How the `RepositoryService` can be used to query for static data

- `RuntimeService`

Whereas the `RepositoryService` handles static data, the `RuntimeService` does the opposite. For each `Process Definition` that is deployed on the engine, the `RuntimeService` can be used to start new `Process Instances` of those definitions. In Camunda, process instances are seen as execution of a process definition. For each process definition known to the engine, and each version of a process definition, multiple `Process Instances` can be running at the same time.

The most important functionalities of this service are the following:

1. the service can be used to query for process instances and executions. The concept of an execution is indicated by the activity token, and has been mentioned before in the literature under the name of *wavefront*;
2. it can manage the life cycle of an instance, through operations such as suspending (pausing), resuming or cancelling a process instance

   Note: One important aspect of the life cycle management, is the fact that the `RuntimeService` can be used to start a `ProcessInstance` from any node or transition in the model, and not only at the start event. This aspect is important, and it will be used later in the implementation of the Data Transfer approach.

3. it handles the data of process instances; the service can be used to query for `process variables` (which are variables specific to an instance)
4. the `RuntimeService` can be used to perform `instance migrations`, where a process instance can be migrated to a new version without the loss of data.

Listing 4.2 is a code snippet from the implementation, that shows how the `RuntimeService` can be used to query for a process instance based on its ID, as well as how to suspend a process instance.

```
    // Find a ProcessInstance based on ID
    ProcessInstance originProcessInstance =
        engine.getRuntimeService()
              .createProcessInstanceQuery()
              .processInstanceId(originProcessInstanceId)
              .singleResult();

    // Suspend a ProcessInstance based on ID
    engine.getRuntimeService()
          .suspendProcessInstanceById(originProcessInstanceID);
```

Listing 4.2: How the `RuntimeService` can be used to manage process instances

The fact that the `Engine`, through the `RuntimeService` allows for instance migrations is positive, and will be beneficial for the context of this research. However, this func-

tionality is only available through a request to the `REST API` and it has to follow specific guidelines, which are not always clear.
According to requirement R-2, a user-friendly interface is the desired manner in which this change should be done. As such, this research will focus on implementing this interface and will take into consideration the implications which a process instance migration brings. These will be discussed in detail in section 4.2.

4.1.2 The `REST API`

The `REST API` package lies on top of the `Engine`, and it can be queried by other applications in order to interact with the Engine. The module has a lot of pre-defined endpoints that can be used to perform different actions on the engine, such as deploying a new model, starting a new process instance, fetching information about deployments, definitions or process instances, just to name a few. A complete list of the endpoints, together with their parameter specifications and expected outputs can be found in the official documentation[1].

In its implementation, the `REST API` makes use of the functionality provided by services mentioned in the subsection above in order to complete the requests that it receives, without interfering with the implementation of the engine.
For example, whenever a request is sent to the endpoint that should return information about a deployment based on the deployment ID, the `REST API` invokes the `RepositoryService` to search for the deployment in the engine. If a deployment was found with the given ID, information about it is returned as a response; otherwise, an error is thrown.

As mentioned in section 2.1, the `Modeler` uses the `REST API` to deploy `.bpmn` model files onto the engine, by sending a request to a specific URL, with the model as a parameter.
This has led to the idea of creating a similar behavior for the deployment of adaptable processes. By interacting only with the `REST API`, the extension brought to the code could make use of all the functionality that is made available through the `Engine`, without bringing any changes to its implementation. This would be in line with the specifications of requirement R-2, which mentions that the implementation of the extension should be as non-intrusive as possible.
At the same time, having access to the `Engine` object would allow the performing of `Instance Migrations`, as described previously.

[1] `https://docs.camunda.org/manual/latest/reference/rest/`

The following sections will present the implementation of the extensions brought to the `Modeler` and the `REST API`, in order to allow the user to deploy adaptive processes.

4.2 Instance Migration

The official documentation[1] of the `Instance Migration` procedure gives a detailed explanation of the steps that need to be performed in order to execute the migration. According to the documentation, there are two steps involved when creating a migration.
To note is the fact that for the rest of the thesis, the origin process instance (the process instance that needs the adaptation) will sometimes be referred to as simply *origin*, and the target process definition (the new version of the adaptable process) will be called *target*.

The two steps required for an `Instance Migration`:

(I) Creating a `MigrationPlan`
Through a set of *migration instructions*, the migration plan should describe how a process instance should be migrated from one definition to a different definition, or to a different version of the same definition. The migration plan identifies the origin and target process definitions by their IDs.
The set of migration instructions will be used to create a mapping between the activities of the initial process definition and the activities of the target definition. Whenever activities do not change from the original definition to the target one, the *migration plan* can also be instructed to map all the activities that are equal (they have the same ID, type and scope), without having to provide instructions for all of them.

One aspect that the documentation mentions is the fact that the mapped activities have to be semantically equivalent. This means that tasks that have different types (eg. a User Task and a Script Task) cannot be mapped through the instructions. The implication of this limitation is the fact that it does not allow the activity to change type from one version to the other. This limitation will be discussed more in the following sections.

(II) Creating a `MigrationPlanExecutionBuilder`, passing it the `MigrationPlan` and executing it

[1]URL: `https://docs.camunda.org/manual/latest/user-guide/process-engine/process-instance-migration/`

Second of all, when the `migrationPlan` has been created, it needs to be applied to a (set of) process instance(s). For this, a `MigrationPlanExecutionBuilder` needs to be created. The builder will look at the migration plan and apply all the instructions to the process instance that is specified. The target process instance is identified through its ID.

A code snippet, extracted from the implementation, that has been used to create a `migrationPlan` can be seen in listing 4.3.

```java
// Create the migrationPlan
MigrationPlan migrationPlan = engine.getRuntimeService()
    .createMigrationPlan(originProcessDefinitionId, targetProcessDefinitionId)
    .mapEqualActivities()
    .updateEventTriggers()
    .build();

// Create the MigrationBuilder and pass it the migrationPlan
MigrationPlanExecutionBuilder builder = engine.getRuntimeService()
    .newMigration(migrationPlan)
    .processInstanceIds(originProcessInstanceId);

// Execute the migration
builder.execute();
```

Listing 4.3: Example code for creating and executing a MigrationPlan

4.2.1 The Implementation of the Instance Migration

Whenever a user decides to migrate one of the running process instances, first the Modeler will be used to apply the desired changes to the respective model. Then, the option to *deploy adaptable* should be available; by selecting this option, the running instance of the process will be migrated to the new model. The changes should be visible whenever the user opens up the Cockpit.

In the background, multiple things need to happen. First of all, the process instance that should be migrated should be suspended (if it is not already). Second of all, the updated version of the model that the user has worked on needs to be deployed to the `Engine`. The previous section presented the two steps that need to be taken whenever an instance migration should be performed on a process instance. However, for the creation of the *migration plan*, the first step mentions that both process definitions, the origin and the target, should be present on the engine. By deploying the new version of the model on the engine, we ensure that both versions

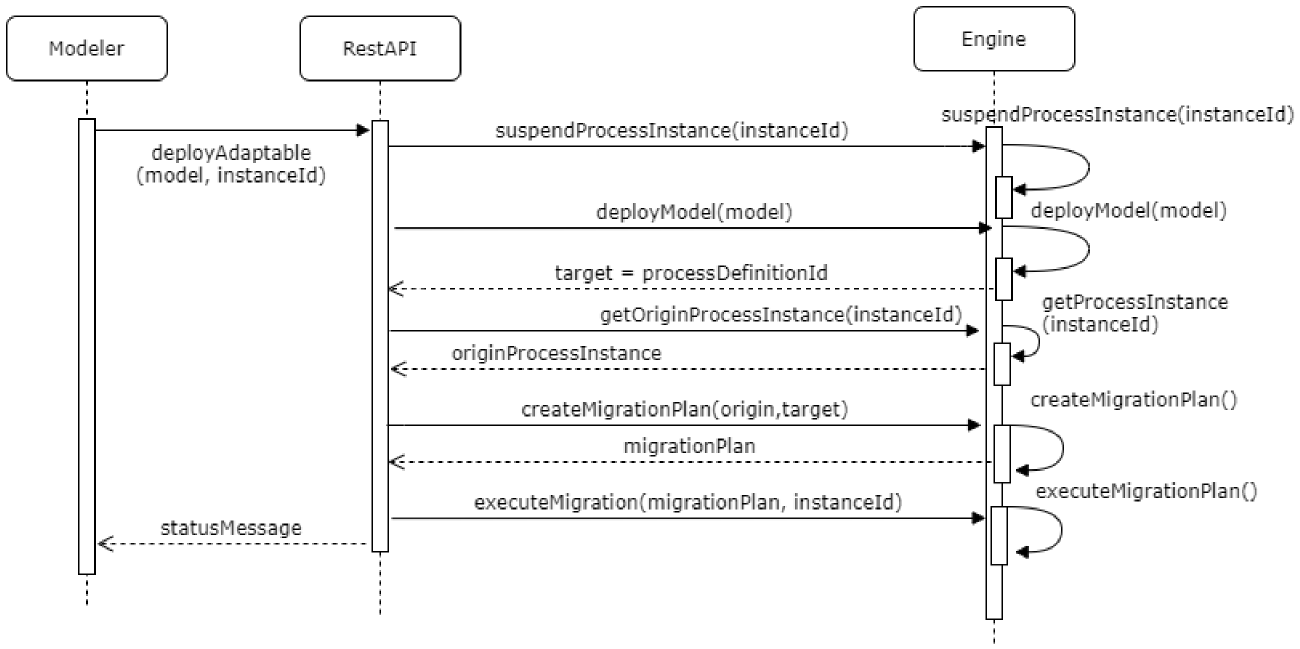


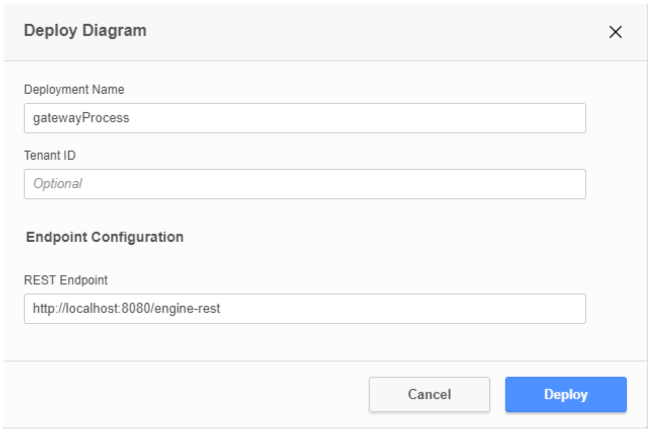
Figure 4.2: Conceptual illustration of the interactions between components when performing an Instance Migration

are present.
At the same time, the `REST API` should have the means to make a distinction between the two types of deployment - either a normal deployment, where a new process model is sent to the engine, or an adaptable deployment, such that the running process instance can be migrated to this new deployment. Whenever the *deploy adaptable* action is sent to the `REST API`, the service will proceed with the adaptable logic, by making the `migrationPlan` and executing it.

The UML diagram in figure 4.2 presents a conceptual illustration of the interactions between the `Modeler`, the `REST API` and the `Engine`.

4.2.1.1 The Modeler

In order to allow the user to deploy a new version of a model onto the engine and to mark it as an adaptation of a process instance instead of a new deployment, the `Modeler` had to be extended with a new functionality, namely, the ability to *deploy adaptable*. As such, a new button was added to the `Modeler`, that would open a new window where the user can enter the ID of the process instance that should be migrated. By opening this new window instead of the typical deployment one, the difference between the two types of deployment is also made and this information can


Deploy Diagram

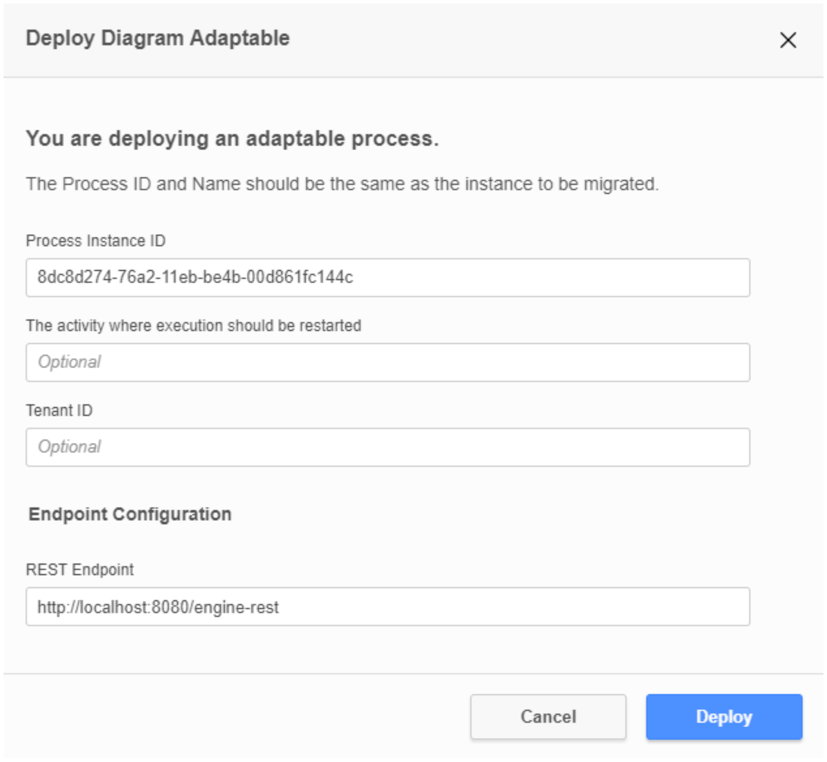
Deployment Name: gatewayProcess

Tenant ID: Optional

Endpoint Configuration

REST Endpoint: http://localhost:8080/engine-rest

(a) The basic deployment window in the Modeler


Deploy Diagram Adaptable

You are deploying an adaptable process.

The Process ID and Name should be the same as the instance to be migrated.

Process Instance ID: 8dc8d274-76a2-11eb-be4b-00d861fc144c

The activity where execution should be restarted: Optional

Tenant ID: Optional

Endpoint Configuration

REST Endpoint: http://localhost:8080/engine-rest

(b) The adaptable process deployment

Figure 4.3: The two types of deployment allowed through the Modeler

be sent to the engine, together with the new model. Figure 4.3 shows a side-to-side view of the two deployment windows.

4.2.1.2 The `REST API`

The `REST API` was also extended in order to implement the adaptable process concept. First of all, a new endpoint was added: `/deployment/deploy-adaptable`, that can accept a new model, together with data. By accessing this endpoint instead of the standard deployment one, the distinction between the two cases can be easily made. The endpoint that can be accessed in order to make a normal deployment is the following: `deployment/create`

Listing 4.4 provides a code snippet to present the code that was added to the `DeploymentRestService` interface and to the `DeploymentRestServiceImpl` class. The two already existing classes have minimal alterations brought to them, which is in line with requirement R-2.

Afterwards, a new class has been added, the `AdaptableDeploymentService`, that orchestrates all the logic behind the adaptation. It firstly fetches the ID of the process definition that the adapted process instance belongs to. Secondly, it handles the deployment of the new model and fetches the ID of the new `ProcessDefinition`. Then, it calls on the new class `MigrationService`, which compares the two process definitions in order to decide whether the migration can be safely performed. If the migration can be performed without issues, it will be executed and the user should be able to see the results in the Cockpit.
However, there are a few cases when the migration cannot be performed, and these cases will be discussed in the following subsection 4.2.1.3.

The UML diagram in figure 4.4 illustrates the interactions that take place within the `REST API`, whenever a request is sent by the `Modeler` to deploy an adaptable process which requires the creation and execution of a `migrationPlan`.

```
// DeploymentRestService.java
    @POST
    @Path("/deploy-adaptable")
    @Consumes(MediaType.MULTIPART_FORM_DATA)
    @Produces(MediaType.APPLICATION_JSON)
    DeploymentDto deployAdaptable(@Context UriInfo uriInfo,
                                  MultipartFormData multipartFormData);

// DeploymentRestServiceImpl.java
    public DeploymentWithDefinitionsDto deployAdaptable(
                UriInfo uriInfo,
                MultipartFormData multipartFormData) {

        AdaptableDeploymentService service =
            new AdaptableDeploymentService(getEngine(), multipartFormData);

        return service.deployAdaptable();
    }
```

Listing 4.4: The extensions brought to the REST API

4.2.1.3 Edge cases

Throughout the development of the implementation, a few limitations of the `Instance Migration` According to the documentation and our own tests, the migration **is** guaranteed to perform as expected in the following cases:

- Whenever the changes that are brought to the model are made either in the past (changes made to activities that have finished executing) or in the future

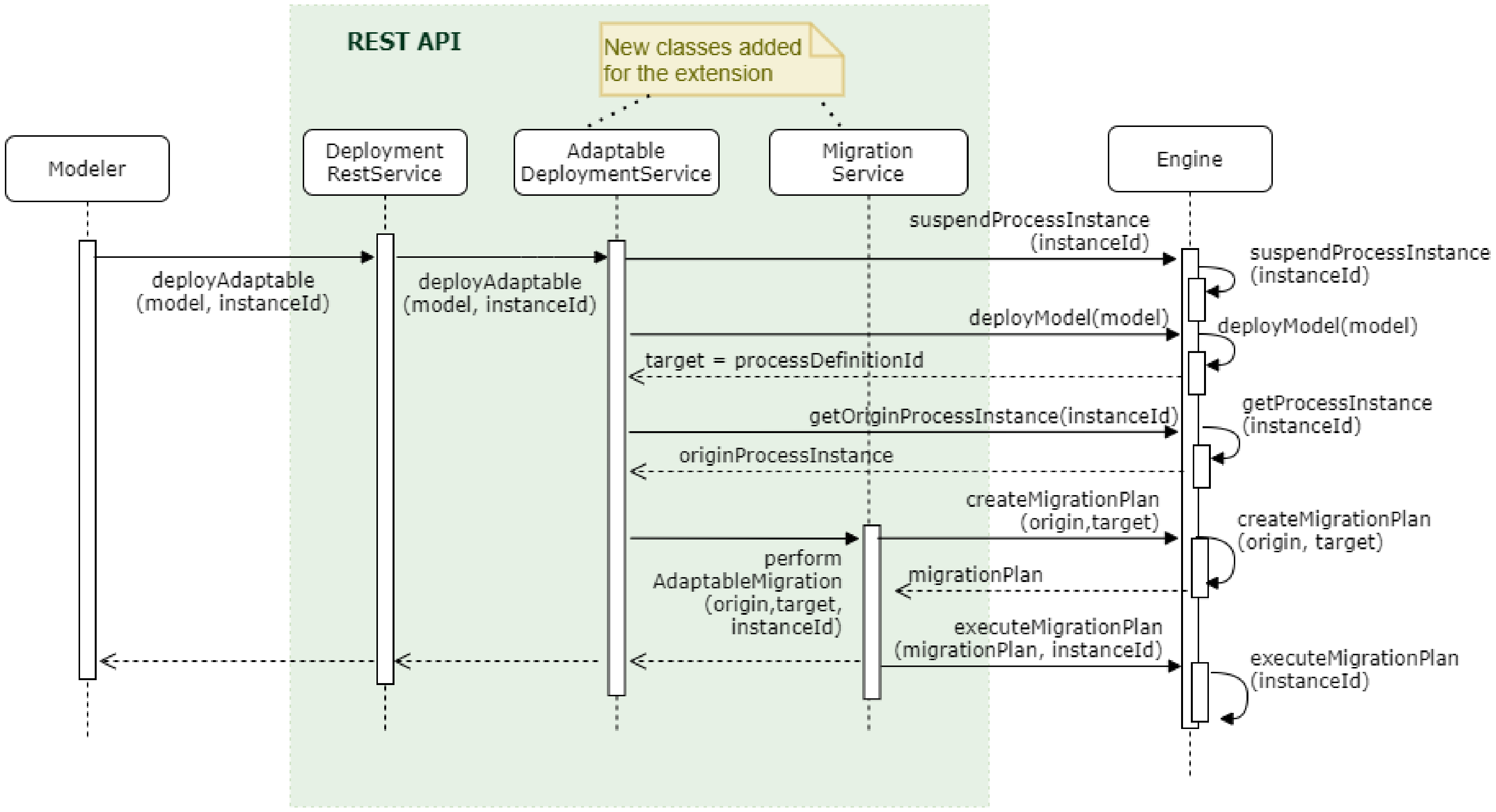

REST API
New classes added for the extension
Modeler
Deployment RestService
Adaptable DeploymentService
Migration Service
Engine
deployAdaptable (model, instanceId)
deployAdaptable (model, instanceId)
suspendProcessInstance (instanceId)
suspendProcessInstance (instanceId)
deployModel(model)
deployModel(model)
target = processDefinitionId
getOriginProcessInstance(instanceId)
getProcessInstance (instanceId)
originProcessInstance
createMigrationPlan (origin,target)
createMigrationPlan (origin, target)
perform AdaptableMigration (origin,target, instanceId)
migrationPlan
executeMigrationPlan (migrationPlan, instanceId)
executeMigrationPlan (instanceId)

Figure 4.4: Conceptual illustration of the interactions between the `REST API` components and the `Engine` when performing an Instance Migration

(activities that have not been reached yet).
This includes operations such as deleting, adding or changing activities in the future or in the past.

- If there are changes made to active activities, then the activities should not change type or ID. However, their names or internal behavior may change.

Those being said, during development we have encountered situations when a migration could not be performed. One such example is whenever the user is trying to bring modifications to an active activity (either by deleting it or changing its type). Whenever an `Instance Migration` would be performed on an instance where an active activity would be deleted, an error will be thrown because the migration will not know how to proceed with the execution of the active activity. The same will happen whenever an active activity changes type - from a User Task to a Manual Task, for example.

However, these situations are very likely to happen in practice, and should not be a limitation of the adaptable process concept. To address this limitation, changes should be brought to the internal implementation of the `Engine`. But, according to requirement R-2, the extensions brought to Camunda should be non-intrusive. Making changes to the `Engine` would be an intrusive operation, and as such, is not an option for the current research.

In order to cover the cases that cannot be handled by a migration, a different approach has been found. While it does not solve the problem of the limitation, the different technique will be able to handle the edge cases. And the user will not be aware of the differences between the two. This approach will be presented in the following section, 4.3.

4.3 New Instance Deployment with Data Transfer

From the documentation and from experimenting with the Instance Migration, the limitations in functionality that it provides have become clear. The migration cannot be performed in cases that are crucial to the development of adaptable processes, such as the deletion or replacement of an active task.

To allow the possibility of these use cases, a new approach is proposed. The steps will be similar as with the `Instance Migration`, up to a certain point:

1. The user makes changes to the model and selects the *deploy adaptable* method

2. The origin process instance is suspended and the new model is deployed onto the engine

3. The lists of activities from the two process definitions (origin and target) are compared. Then, the extension will establish what kind of changes were made, and will decide on the course of action:

`CASE I` If no active tasks were changed:
- proceed with the migration as previously explained

Note: Until this point, the logic is the same as with the `Instance Migration`. The following steps have been added for the implementation of the new approach.

`CASE II` If an active task was deleted or changed:
- fetch all the data that was computed by the origin process instance
- pick the node right before the deleted activity as the start point for the new process instance
- use the `RuntimeService` to start a new process instance of the newly deployed model
- assign the data of the origin to the target

The main idea behind this technique is the transfer of data between the two process instances. In accordance with requirement R-3, whenever a process instance is migrated, data should also be handled and moved from the origin instance to the target. This approach handles this requirement, and ensures that the data of the old process instance is assigned to the new instance before the instance is created. When the instance is started, all the data is readily available and the execution token will be right before the deleted activity. To the user, the new instance will look as a modification of the old one, and not as a completely new proces definition.

Because of the way this technique works, it will be referred to as the `Data Transfer` approach.

There are a few advantages to using an `Instance Migration` instead of the Data Transfer method. First of all, the `Instance Migration` will transfer all executions from a process instance to the other. This means that, whenever two branches are concurrently active in the origin process, the executions will be transferred and the branches will also be active in the target process.

However, the `Data Transfer` approach can only start one execution (one branch). This has the implication that the user should ensure the re-iteration of the process instance from a 'safe' point in the model - one that would ensure that both the execution branches that were previously active, will be active in the target process. This implication will be further discussed in section 4.3.1.
Second of all, if an instance is adapted through a migration, the new instance will maintain a lot of the information from the origin instance. The ID of the process instance will stay the same. To the user, the transition will seem as if the model of the process instance 'transforms' into the new version. However, in the background, a new process instance of the target process definition is started, and the information is transferred from one instance to the other (not just the variables or output), also in the database and in the internal representation of the model inside the engine.

Keeping those aspects in mind, for the context of this thesis we have decided to provide both methods for the adaptation of processes. The new class `ProcessInstanceStarted` has been added, such that we have a better separation of concerns between the `MigrationService` functionality and the starting of a new proces.

Figure 4.5 illustrates the interactions between the different components in order to realize this approach. To note is the fact that some of the initial steps that were presented in the previous diagrams (eg. in figure 4.4) have been merged into a more generic one, since they are the same, and instead, the focus was on the modified interactions.

4.3.1 Specifying a starting Node ID

In the `CASE II` of section 4.3, a decision was made to pick the node right before the changed activity as a starting node for the new process. This can be seen as a temporary solution, because it has a lot of downsides; for example, if the changed activity was in a parallel gateway branch, then the new process will start the execution on the branch of the changed activity only, and will lead to a deadlock, since the second gateway expects two tokens, however, it will only receive one.

This example has been illustrated in figure 4.6, where there are two active tasks, `Execute Task B` and `Execute Task C`. If the user decides to swap out or completely remove the activity `Execute Task B`, then, according to the logic explained above, the new process instance would start at the node before this task.
In the example, the starting activity would be `Execute Task A`. However, this logic does not take into consideration the context of the change.

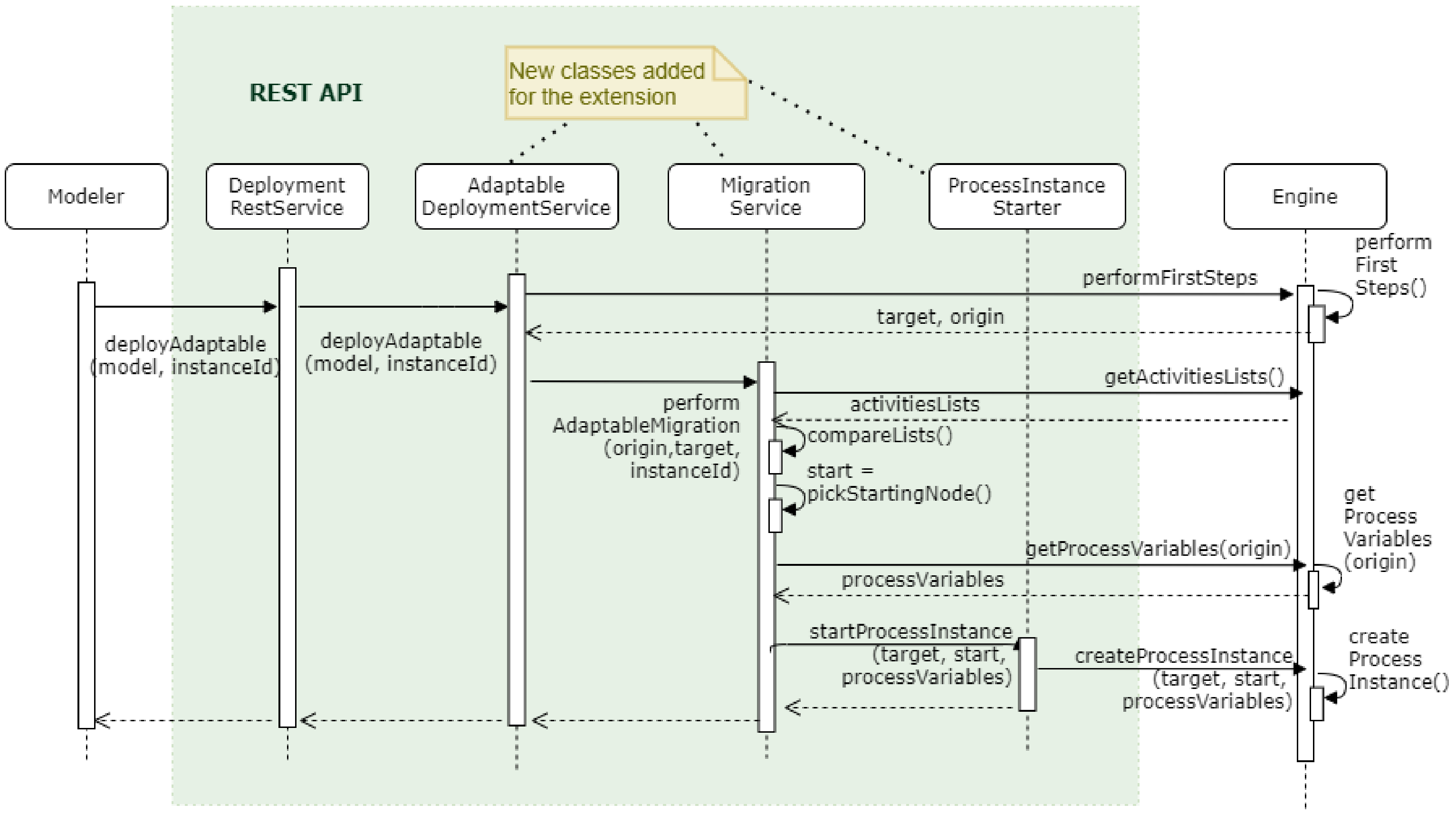

REST API
New classes added for the extension
Modeler
Deployment RestService
Adaptable DeploymentService
Migration Service
ProcessInstance Starter
Engine
deployAdaptable (model, instanceId)
deployAdaptable (model, instanceId)
performFirstSteps
perform First Steps()
target, origin
perform AdaptableMigration (origin,target, instanceId)
getActivitiesLists()
activitiesLists
compareLists()
start = pickStartingNode()
getProcessVariables(origin)
get Process Variables (origin)
processVariables
startProcessInstance (target, start, processVariables)
createProcessInstance (target, start, processVariables)
create Process Instance()

Figure 4.5: Conceptual illustration of the interactions between the `REST API` components and the `Engine` when performing the Data Transfer approach

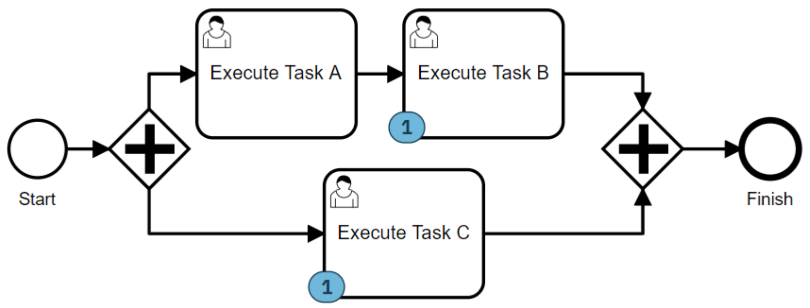


Figure 4.6: Possible case when, by starting a process instance at the node preceding the changed activity `Execute Task B`, a deadlock could be caused

When the process instance starts at `Execute Task A`, the respective execution branch will proceed, and the execution will reach the parallel gateway. However, the execution on the second branch will not be restarted. The gateway expects two tokens before it can proceed - and because the second branch is never restarted, the gateway will be waiting for a second token indefinitely.

One possible solution to this problem would be the implementation of an algorithm to dynamically determine the restart point that would not lead to a deadlock. In the example presented in figure 4.6, this point could be the Gateway after the Start activity.

The dynamic determination of re-iteration points is a research topic on itself. In his work, A. Weiß proposes a number of algorithms that can be used to dynamically find the best starting points of an instance, such that no errors, like the deadlock mentioned above, could occur. Due to the complexity of the topic, and to the limited time available for conducting the current research, we have decided to leave the topic open for future work.

One alternative solution for the problem described above is to allow the user to specify which node the new process instance should start the execution from. This alternative leaves it to the best-judgement of the user to select the appropriate start node (activity or gateway) that allows for a correct execution of the instance. It also brings along the advantage of the re-iteration of the process from any given node, if the user would like to make changes to their model and re-run parts of it.

To facilitate this, the `Modeler` was extended with the option to specify a Node ID as a parameter when deploying an adaptable process. This can be seen in figure

4.3(b), where the field for the Activity ID is present and optional.
Whenever an Activity ID is provided, the Data Transfer approach will be used to migrate from one process instance to the new one. The Activity ID will be passed to the `RepositoryService` as the starting point. The actual implementation, however, will start the execution at the transition right before the mentioned activity, to ensure that the activity itself will be executed.

Listing 4.5 is a code snippet from the implementation that starts a new `ProcessInstance` from a given transition and assigns to it a list of variables.

```java
ProcessInstance processInstance =
      engine.getRuntimeService()
            .createProcessInstanceById(targetProcessDefinitionId)
            .setVariables(variables)
            .startTransition(sequenceFlow.getId())
            .executeWithVariablesInReturn(false, false);
```

Listing 4.5: Example code for starting a ProcessInstance at a given transition ID

To separate the concerns of the two approaches of adaptation, a new class has been added along with the `MigrationService`, namely the `ProcessInstanceStarter`. This class will focus only on the starting of a process instance from a specific node.
This class offers two functions, that can be used depending on the type of node that the new instance should use as a starting point: `startProcessInstanceAtTransition()` and `startProcessInstanceAtGateway()`. The first function is used to start a new process at the transition right before the selected activity, so it will look at the incoming transition of that activity. The second function is needed when dealing with gateways. If the user would like to start the process on a gateway, there are two possible cases that we need to take into consideration.

- If the gateway brings together multiple branches (so the number of incoming transitions is higher than 1), we cannot restart the process before the gateway, because only one execution will be active and this will result in a deadlock again. As such, we need to restart the execution *after* the gateway.

- If the gateway only has one incoming transition, it is safe to start the process on that specific transition. This will also take care of the case in which the gateway is a conditional one. By starting the process *before* the gateway, we ensure that the conditions will also be taken into consideration, because the behavior of the gateway will be executed in order to select the correct path.

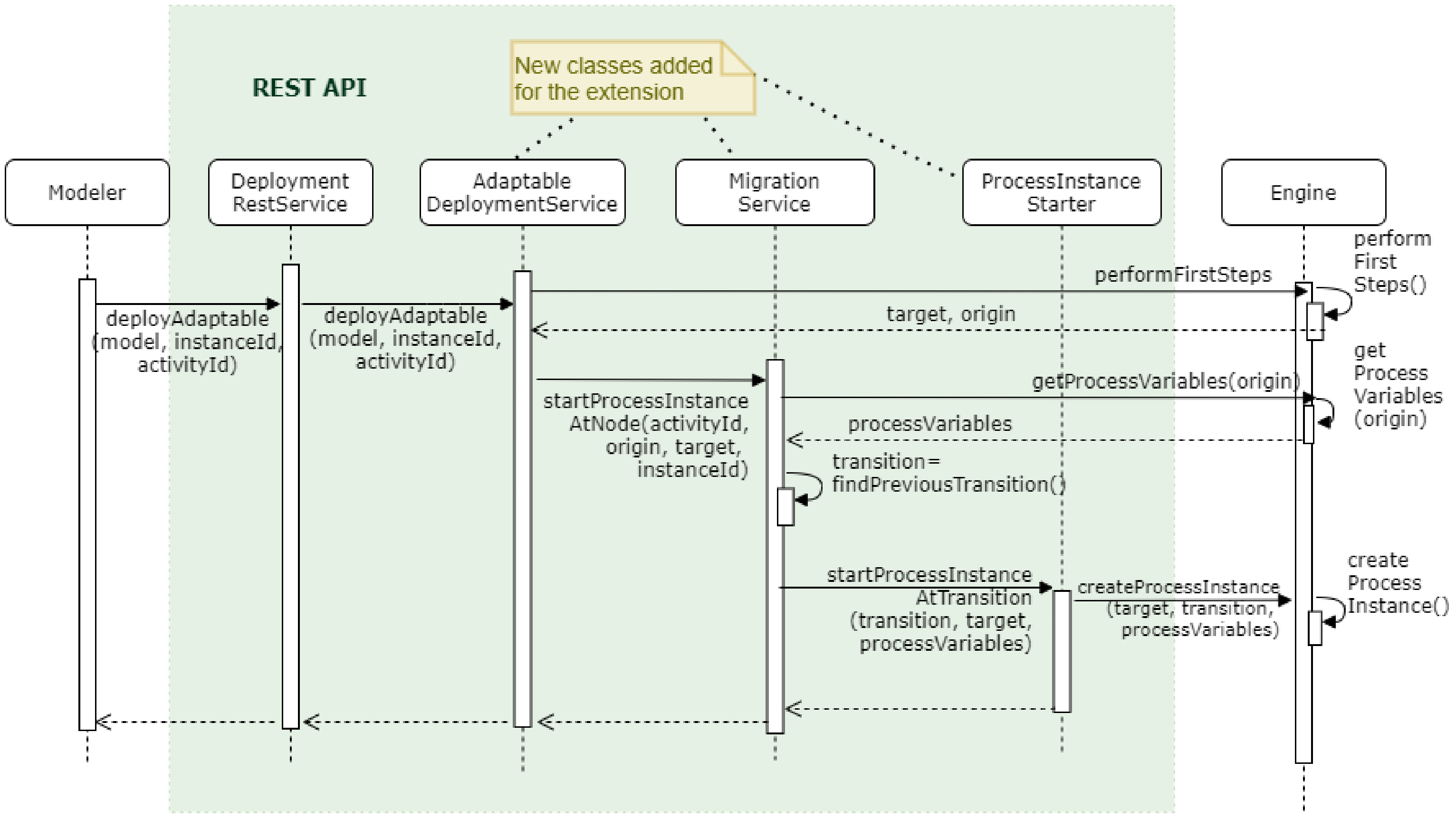

REST API
New classes added for the extension
Modeler
Deployment RestService
Adaptable DeploymentService
Migration Service
ProcessInstance Starter
Engine
deployAdaptable (model, instanceId, activityId)
deployAdaptable (model, instanceId, activityId)
performFirstSteps
perform First Steps()
target, origin
startProcessInstance AtNode(activityId, origin, target, instanceId)
getProcessVariables(origin)
get Process Variables (origin)
processVariables
transition= findPreviousTransition()
startProcessInstance AtTransition (transition, target, processVariables)
createProcessInstance (target, transition, processVariables)
create Process Instance()

Figure 4.7: Conceptual illustration of the interactions between the `REST API` components and the `Engine`, when the user provides an Activity ID as a starting point

4.3.2 Conclusion

The proof-of-concept implementation explored the ways in which Camunda could be extended with the functionality of deploying adaptable processes. The application already had a basic version of an instance migration, however it was not done through a user interface, but only accessible through the REST API. The implementation of the current research included an extension brought to the Modeler that makes the flow of deploying an adaptable process available through a friendly user interface.
The section described the instance migration also in terms of its limitations, and it proposed an alternative solution for the cases where the built-in instance migration would fail to execute. The code that implements all the functionality mentioned in this chapter, and some minor supporting extensions such as logging, error handling, etc., can be found in the GitHub repository[1].

[1] https://github.com/ana-roman/camunda-adaptable-processes

Chapter 5 | Evaluation and Conclusions

After having introduced the implementation of the Camunda extension that will allow the deployment of adaptable processes, the current chapter will present the validation of the implementation, by looking at the use cases presented in section 3.1.1 as well as the requirements presented in 3.1. The evaluation will be described in section 5.1. Afterwards, section 5.2 will draw the conclusion of the current thesis, by summarizing the work and discussing its limitations and topics for future research.

5.1 Evaluation

The current section will take a look at the suggested implementation that was described in chapter 4. Firstly, it will look at the problem context of the Use Cases presented in section 3.1.1 and will discuss whether the current implementation would be beneficial for the two. Secondly, it will look back at section 3.1, to discuss whether the implementation fulfills all the requirements that were proposed.

5.1.1 Application to the Use Cases

Section 3.1.1 presented two Use cases that, despite being fictional, were used to illustrate the need for adaptable processes in the development of business process.
A few video clips that demonstrate how to deploy adaptable processes can be found in the GitHub repository of the project.

5.1.1.1 Loan approval system

In the use case *Loan approval system*, the manager needed to find a way to correct the process instance that had faulty data. The presence of the faulty data would have as implication a wrong output of the process, which could lead to potentially upset clients.

In this case, two primary requirements had to be met. First of all, the process model of the process instance should be modified, such that the user could overwrite the faulty data and change the final outcome of the process. Second of all, the data of the client had to be preserved, since the data was sensitive and not directly available.

Through the state-of-the-art implementation of the Camunda extension, the manager should be able to make the desired changes through the user interface of the Modeler, without asking for additional help. The manager could use the Instance Migration approach to solve this problem.
Firstly, the `Modeler` could be used to make changes to the process instance model. One such change could be the indication a mistake and the ability to loop back to an earlier activity. This behavior has been illustrated in figure 3.2. Then, by choosing the *deploy adaptable* option in the modeler, the user only needs to specify the ID of the process instance that should be corrected, and then deploy the adaptable process. The process instance will be migrated, together with all the data. This change should be visible in the Cockpit, where a new process instance will be visible with all the data that was already present. Now, the manager should be able to correct the output of the process by modifying the fields that were incorrect.

5.1.1.2 Scientific simulation

The use case of the scientific simulation had an extra requirement on top of two already mentioned by *Loan approval system*. Alongside the ability to bring changes to a model and the preservation of data, the scientist wanted to be able to restart the execution of a process instance from a specific activity.
The implementation of the adaptable processes that was described in this thesis would also assist the scientist, by allowing the option of performing experiments in an exploratory fashion. As such, there can be an initial version of the process on the engine, and then modifications can be brought to it, without the loss of data. The same steps will be followed by the scientists, as in the case of the manager: the process model will be modified in the `Modeler` and the option to *deploy adaptable* will be chosen; this can be repeated until a version of the model that follows all of the requirements is reached. Whenever the scientist notices that some steps in the process should be ‘backtracked’, `Data Transfer` approach can be used to deploy the adaptable process and to start at a given activity, while keeping all the data that was previously generated. However, the variables that resulted as output from activities that are executed again will rewrite the ’old’ values of the variables (the ones transferred from the origin process instance).

5.1.2 Requirements Fulfillment

In this section, the requirements presented in section 3.1 will be named, and the discussion will focus on whether the implementation of the adaptable processes extension fulfills each requirement.

R1 Concept and implementation
For the fulfilment of this requirement, the thesis had to provide a conceptual overview of adaptable processes, and how this concept could be used to extend the Camunda OSS. Sections 3.1, 3.2 together with chapter 4 discuss in detail both the concepts and the implementation details of this extension.
As such, requirement **R1** has been fulfilled.

R2 Runtime adaptation on the Control Flow
Chapter 4 gives a detailed explanation on how the concept of adaptable processes has been implemented. By deploying an adaptable process, a user can achieve ad-hoc runtime adaptations on the control flow.
At the same time, the extension brought to Camunda has been minimally intrusive. The existing classes of the `REST API` package were extended with the addition of a few extra lines of code, such that none of the already existing features were interfered with. The rest of the implementation was done through dedicated classes and services.
Requirement **R2** has been fulfilled.

R3 Data flow preservation
Both the `Instance Migration` and `Data Transfer` approaches handle the transfer of data from a previous instance to the other, by reassigning the variables from the origin to the target process instances.
Requirement **R3** has been fulfilled.

R4 Organizational change
Whenever a task gets reassigned (eg. a User Task is reassigned to a different User/User Group or a Service Task changes the endpoint that it is talking to), these changes will be picked up by both adaptation methods.
Moreover, as a side note, Camunda already had the functionality of reassigning an active User Task implemented in the Cockpit's user interface.
Requirement **R3** has been therefore fulfilled.

5.2 Conclusion

This section provides a summary of the main results of the current research, and concludes the work. It will present the limitations that were found for the current implementation, as well as ideas for future work.

5.2.1 Summary of results

Section 2.2 described how the concept of adaptable processes was studied in the past by the literature. A lot of ground work had already been laid and a vast number of concepts were well established. The literature provided a very clear blueprint that any workflow management system should follow, whenever it should be extended with the capability of adaptable processes. However, with so much material available and all the different functionalities that all seemed to be of outmost importance, a selection had to be made in order to centralize the focus strictly on the requirements, behaviors and capabilities of adaptable processes.

The first main contribution of this work is related to the study of the literature. In this thesis, the literature has been studied and summarized, and the most relevant information has been extracted and presented. Following the same pattern as the literature, the theoretical findings of this thesis have been presented at a conceptual level. A list of high-level requirements was established, and they were also kept separate from the implementation, in order to allow them to be independent from the workflow management system.

Secondly, the thesis presented the implementation of the concepts on the Camunda Workflow Management System. And this brings us to the second main contribution. In the light of the literature and the established requirements, the current thesis described in detail the implementation of the extension that was brought to Camunda. The second contribution is made up of two parts: one for each of the two approaches that allow the deployment of adaptable processes, namely the `Instance Migration` and the `Data Transfer` methods. Depending on the type of adaptation that the user wants to perform, one of the two methods can be used and the Evaluation section has explained how this can be done.

5.2.2 Limitations and discussion

The first limitation that the current implementation has, was mentioned in the implementation of the `Data Transfer` method. Whenever this approach is used for the modification of an active task without specifying an node ID to start from, the method is set to start the new process instance from the node that precedes the modified task. This can lead to errors such as deadlock, as it was described in the section. Although it can be surpassed by providing a node ID as a starting point, this is only a work around and it does not solve the actual issue. At the same time, human error can still lead to problems whenever a user selects a ‘wrong’ activity to start from (in this context, a *wrong* activity can be seen as an activity that, whenever chosen as a starting point, would lead to potential errors).

The second limitation that has not yet been discussed, is the inability to compensate already executed activities whenever a process instance is restarted from an earlier point in time. In their work [6], Karastoyanova and Sonntag outline the need for compensating activities. For example, some activities might perform actions that are not idempotent. As such, the re-execution of such an activity without it being firstly compensated might lead to errors or unwanted results.

Lastly, the current work does not implement *keep-alive* mechanism as the authors of the *Model-as-You-Go* described, and as it was illustrated in figure 2.8. In order to set ’break points’ in the model, the user has to add Nodes that implement wait states (states that require further input to continue) such as User Tasks or Message Receiving Tasks. For the same reason, the current implementation does not support the deployment of a model that is not complete. In order for a model to be deployed, it needs to have a start and an end event. So the flow of the current implementation is different from the one described in the *Model-as-You-Go* approach.

Despite these limitations, the current implementation does touch upon and fulfills all the expected requirements. The topic of adaptable processes in Camunda has never been previously researched; as such, the current thesis lays the foundation for the basic concepts surrounding this topic, and only focused on the core, indispensable requirements.

5.2.3 Future research

One of the possible future research topics has already been mentioned in section 4.3. The dynamic decision of the best possible starting point is an interesting topic, and it definitely needs further investigation. The benefits of having this functionality would be substantial. For example, it could ensure that whenever new instances are started, they are started from a safe point, from where the successful completion of the instance could be guaranteed.

Other interesting future research topics stem from the limitations addressed previously. The lack of compensations when re-iterating (parts of) a model can lead to errors. As such, this topic is worth researching, because it could solve the problems that arise whenever activities do not have idempotent behaviors, as mentioned previously.
At the same time, something similar to the *keep-alive* mechanism mentioned in the literature could be researched and implemented. This mechanism would make the development of adaptable processes even easier from the user's point of view, because there would be no need to add wait-state activities in the model such that an instance does not finish executing.

Bibliography

[1] M. Weske, *Business Process Management: Concepts, Languages, Architectures.* 01 2007. (Cited on pages 11 and 19.)

[2] M. Dumas, M. L. Rosa, J. Mendling, and H. A. Reijers, *Fundamentals of Business Process Management.* Springer Publishing Company, Incorporated, 2018. (Cited on pages 11, 15, 16, 17, and 20.)

[3] P. Harmon, *The State of Business Process Management.* 03 2016. As found at: `http://www.bptrends.com/bpt/wp-content/uploads/2015-BPT-Survey-Report.pdf`. (Cited on pages 11 and 21.)

[4] W. Aalst, "Business process management: a comprehensive survey," *ISRN Software Engineering*, 01 2012. (Cited on page 11.)

[5] A. Weiß, V. Andrikopoulos, M. Hahn, and D. Karastoyanova, "Model-as-you-go for choreographies: Rewinding and repeating scientific choreographies," *IEEE Transactions on Services Computing*, vol. 13, no. 5, pp. 901–914, 2020. (Cited on pages 11, 17, 34, and 35.)

[6] M. Sonntag and D. Karastoyanova, "Model-as-you-go: An approach for an advanced infrastructure for scientific workflows," *Journal of Grid Computing*, vol. 11, 09 2013. (Cited on pages 12, 17, 21, 29, 31, 33, 34, and 71.)

[7] J. vom Brocke and M. Rosemann, *Handbook on Business Process Management 1: Introduction, Methods, and Information Systems.* Springer Publishing Company, Incorporated, 1st ed., 2010. (Cited on pages 12, 17, and 20.)

[8] A. Weiss, "Flexible modeling and execution of choreographies," 09 2018. Found at: `http://dx.doi.org/10.18419/opus-10224`. (Cited on pages 12, 16, 19, and 29.)

[9] J. Eder, *Workflow Management and Workflow Management System*, pp. 3545–3549. Boston, MA: Springer US, 2009. (Cited on pages 16, 18, and 19.)

[10] F. Leymann and D. Roller, *Production Workflow - Concepts and Techniques*. 01 2001. (Cited on pages 17, 18, 20, and 23.)

[11] OMG, "The complete business process handbook," 2015. Found at: https://www.omg.org/news/whitepapers/Business_Process_Model_and_Notation.pdf. (Cited on page 18.)

[12] OMG, "Unified modeling language (uml)," 2015. Found at: https://www.omg.org/spec/UML/2.5.1/PDF. (Cited on page 18.)

[13] OMG, "Case management model and notation (cmmn)," 2016. Found at: https://www.omg.org/spec/CMMN/1.1/PDF. (Cited on page 18.)

[14] W. Aalst, "The application of petri nets to workflow management," *Journal of Circuits, Systems, and Computers*, vol. 8, pp. 21–66, 02 1998. (Cited on page 18.)

[15] W. V. Aalst and K. V. Hee, "Workflow management: Models, methods, and systems," in *Cooperative information systems*, 2002. (Cited on pages 19 and 24.)

[16] M. J. Alter, *Science of Flexibility*. 2004. (Cited on page 19.)

[17] W. Aalst, van der, T. Basten, H. Verbeek, P. Verkoulen, and M. Voorhoeve, *Adaptive workflow: On the interplay between flexibility and support*, pp. 63–70. Netherlands: Kluwer Academic Publishers, 2000. (Cited on page 20.)

[18] M. Reichert and P. Dadam, "Adeptflex—supporting dynamic changes of workflows without losing control," *Journal of Intelligent Information Systems*, vol. 10, pp. 93–129, 2004. (Cited on page 20.)

[19] M. Reichert and B. Weber, *Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies*. 10 2012. (Cited on page 20.)

[20] M. Sonntag and D. Karastoyanova, "Concurrent workflow evolution," *Electronic Communications of the EASST*, vol. 37, 2011. (Cited on page 29.)

[21] R. Mitchell, ed., *Managing Complexity in Software Engineering*. Computing, Institution of Engineering and Technology, 1990. (Cited on page 35.)