

MASTER THESIS

FACULTY OF SCIENCE AND ENGINEERING

COMPUTING SCIENCE

Building a software system to autonomize business process adaptation

Author:
Evrikli XHELO

Supervisors:
Dimka KARASTONAYOVA
Fadi MOHSEN



university of
 groningen

faculty of science
and engineering

August 2021

Contents

Acknowledgements	4
Abstract	5
1 Introduction	6
2 Key concepts and literature review	8
2.1 Business processes	8
2.2 Business Process Management	9
2.3 BPM systems	9
2.4 Process adaptation	10
2.4.1 Adaptation triggers	10
2.5 Adaptation strategies	11
3 Problem analysis	14
3.1 Adaptation pipeline	14
3.2 The challenges of process adaptation	15
3.3 Research question	16
3.4 System requirements	17
3.4.1 Functional requirements	17
3.4.2 Non-functional requirements	18
4 Architecture	20
4.1 Proposed solution: constructing an Adaptation Dashboard	20
4.1.1 Designing an adaptable process	21
4.2 System design	22
4.2.1 Service communication	22
4.2.2 Sequence diagrams	24
5 Implementation	28
5.1 Microservices	28
5.1.1 Mapping server	28
5.1.2 External task service	29
5.1.3 KPI system	30
5.1.4 Violation-check system	31
5.1.5 Process resumption system	31
5.1.6 Camunda engine	32
5.2 System prerequisites for the end-user	32
5.2.1 Implementation of the checkpoint life-cycle	32
5.3 Technology stack	33
6 Discussion	35
6.1 Evaluating the Adaptation Dashboard using an adaptable process model	35
6.1.1 Setting up the stages of the checkpoint life-cycle	35
6.1.2 Initializing the services of the AD system	37
6.1.3 Initializing process execution in Camunda	37

6.1.4	Using the Adaptation Dashboard	38
6.2	AD and third-party services	43
6.3	Setting up AD	45
6.4	Evaluation of the non-functional requirements	45
6.5	Process adaptation pipeline	46
7	Conclusions	50
7.1	Thesis contributions	50
7.1.1	Construction of the Adaptation Dashboard	50
7.1.2	Implementation of the life-cycle for the checkpoint state	51
7.1.3	Implementation of frameworks for future types of performance metrics and adaptation triggers	52
7.1.4	Implementation of Stage I of the adaptation pipeline	52
7.2	Main takeaways of the thesis	52
7.3	Limitations of our work	54
7.4	Future work	54
	Bibliography	57
	Appendices	59
A	Adaptation Dashboard	59
B	Camunda modeller	66
C	Camunda's API requests	66

Acknowledgments

I would like to thank my parents, Vaso and Katerina, for being a constant source of love in my life. Their support in all my endeavours has been lighting my paths even in the most difficult situations. I would also like to express my gratitude to prof. dr. Dimka Karastonayova for guiding me throughout these past months, while not only being my supervisor, but also a friend and colleague. Lastly, I would like to express how grateful I am to all the people I have met throughout my studies at the University of Groningen in the past five years, that have turned into friends and colleagues.

Abstract

The discipline of business process management (BPM) is an integral aspect of businesses, systems and other disciplines that work with business processes. Business processes are used by any establishment whose end-goal is to provide a service to their customers. Such establishments use business processes to structurally define the steps that are followed in order to produce the goal of their outcome (the happy flow) or the other possible paths. The life-cycle of a BPM system consists of the following five steps that could potentially also overlap: 1. designing the business process, 2. modelling it, 3. execution of said process, 4. monitoring it and lastly, 5. optimization of the process model based on the results of phase 4. Recently, however, it has become clear that there is a need for a new (and optional) phase in the BPM life-cycle, called process adaptation. The rise of predictive process monitoring has made it possible to learn about the outcome of a running business process before its execution is finished (phase 3). When the outcome is not as expected, it should be possible to adapt the process accordingly and resume its execution without having to start from the beginning. Numerous research studies have already been conducted that put forward the why and the how of process adaptation. Today, workflow engines that facilitate the execution of business processes also provide developers with the means to make process adaptation possible, however, in most cases this feature has to be implemented in-house by the organisation's developers. In this thesis, we have created a system that makes process adaptation convenient to oversee. One of the primary products of this system is an extensible, user-friendly dashboard that depicts information about process instances that could potentially require adaptation. In this thesis, we delve into the development process of this software, how it is used (for the end-user) and how it can be extended (for the developers).

1 Introduction

One of the most important aspects of an organization are its processes; they get to provide detailed descriptions of said organization's operations including, but not limited to, the individuals that are in charge of maintenance, the administrators that are involved, as well as the resources that are allocated for the execution of the organization's operations. Processes are important because they can be used to not only visualize the target goals of an organization, but also track them, giving maintainers the opportunity to control whether a certain process is performing as expected. The term 'organizations' can refer to big business corporations and companies, but also to smaller rely such as schools, scientific experiments and every other subject whose goals relies on a set of structurally defined activities that produce a desired outcome.

Managing the life-cycle of processes Processes are designed to emulate the workflow of an operation, from start to finish, using a set of pre-defined notations whereby different types of activities are represented using different notations. One of the most widely-used representation languages for designing processes is called BPMN ¹. By designing an organization's operation workflows, processes can offer a new outlook on said organization's tasks from a daily, monthly and even yearly perspective. In this way, processes have also been established as an instrumental tool for helping highlight an organization's points of improvement.

The research field of BPM ² is concerned with the study of processes (also referred to as business processes), specifically their design, execution and maintenance. The BPM life-cycle of a process consists of five phases - *design, model, execute, monitor, optimize*. The first three phases of the life-cycle are focused on designing the process and its execution. In the last two phases we work with the results of the process' execution to yield information about its performance and what can and cannot be improved in its future iterations. The subject of such improvements are primarily focused on the target goals of the operations of the organisation in question. In recent years the ability to improve a process while it is still executing has become quite relevant, if it means that the process performance will improve to its expected levels. In BPM, this is referred to as process adaptation.

Adapting processes The focus of our study is placed on the adaptation phase of a business process. Our goal has been to construct an adaptation system that can generalize the adaptation flow of a process in spite of the various possible adaptation causes and proposals. This has lead to summarizing the procedure of process adaptation as a pipeline that is divided into two parts, one part concerned with determining which processes require adaptation, and the other part concerned with the execution of the adaptation. We refer to the pipeline in question as the "adaptation pipeline". This thesis presents our research study and efforts in designing and constructing a system that can confine the functionalities of both stages of the adaptation pipeline.

The key contribution of our work is a *dashboard* which provides information about *adaptable* processes that can help end-users determine whether said processes are in need of adaptation or not. Additionally, we introduce an implementation for the life-cycle of a *checkpoint*, a 'buffer state' which can be used to determine whether a process instance requires adaptation. This implementation is crucial to the definition of an adaptable process that is supported by our dashboard system. Lastly, our

¹Business Process Model and Notation

²Business process management

Key Performance Indicator and *violation-check* systems serve as frameworks for the development of future types of performance metric- and adaptation trigger- services respectively.

For the initial version of this adaptation dashboard we have resorted to providing an application of the first stage of the adaptation pipeline, that is, effectively determining which adaptable process require adaptation. The second stage of the pipeline, that is, executing a successful process adaptation, remains as part of the future work for our research and software development.

This report consists of seven chapters and an Appendix in §7.4; chapter 2 is dedicated to the research study that has been conducted for this thesis, primarily focused on BPM and process adaptation. Chapter 3 presents the results of our problem analysis, which we have used to define the system requirements. Chapter 4 introduces our system proposal and architecture, followed by chapter 5 which provides an extensive overview of the system's description and implementation. Chapters 6 and 7 coincide with the discussion and conclusion chapters respectively.

2 Key concepts and literature review

In this chapter of the thesis we present a detailed overview of the concepts that we have gathered and used throughout the research and conduction of this study. This information also surmounts to a literature review of the existing research in the domain of process adaptation. Working with an adaptation system requires reasonable understanding of business processes and how we can effectively work with them. The main focus of our study is well-represented in [Karastonayova and Ghahderijani, 2021]. However, we first take a step back and look into business processes and BPM systems, so that we can present a cohesive timeline of the process development from its inception to its possible adaptation. Every section of this chapter pertains to a relevant concept in the field of process adaptation and the core study of this thesis. We take a closer look at every concept individually.

2.1 Business processes

A business process is essentially a set of activities whose outcome provides some value. Such processes are extensively used by business organizations or scientists in order to depict the clear path they have designed to achieve the goals of the organisation or scientific experiment respectively. [Karastoyanova, 2010] provides a complete outlook on how processes can be used in support of scientific experiments, while [Kazhamiakin et al., 2010] is one of the multiple available papers that presents how business organisations can use processes to achieve their target goals and oversee how the executions are laid out and whether there is need for improvements.

There are two crucial concepts in the realm of business processes that are worth mentioning, a process model and a process instance. Modelling a process serves to provide a portrait of what was perceived as a fact at the time the process model was created. They are used to represent the internal elements of business processes, including the activities involved as well as their dependencies, the dataflow, the roles and actors assigned, and the goals [Lindsay et al., 2003]. A process instance, on the other hand, represents the real-life execution of the process model. The activities of the process model could be implemented using in-house services, external systems or could also be manual tasks that have to be completed by individuals (for instance, signing a printed document). Either way, the execution of a process implies that all these activities will occur when and how is indicated in the process model. For any existing process model, there can be multiple instances of processes being executed. When working with process models and instances, it is just as important to introduce two crucial concepts, the BPMN representation and workflow engines.

BPMN: Business Process Model and Notation BPMN is a representational language for defining business process models. Its main objective is to provide an easy-to-understand language for all the individuals that work with business processes regardless of their roles and levels of expertise [Weiss and Amyot, 2007]. BPMN models exhibit the definition of the business process by making use of four basic categories: flow objects, connecting objects, swim lanes, and artifacts. These basic elements provide support for modeling sequence flow, roles, activities, events, and process hierarchies. In Figure 1 we present a trivial process model in BPMN notation.

Workflow engines The execution of process instances is rendered by workflow engines. Such engines can usually provide a set of other features such as parallelism, remote building, resumability, and logging of process instances and their activities [Bedó et al., 2020]. For the purpose of our study, we use Camunda's workflow engine.

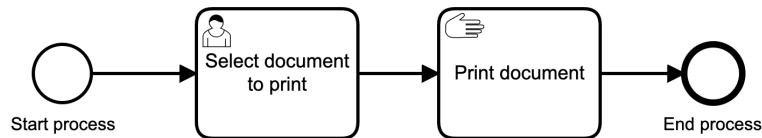


Figure 1: The BPMN representation of a fairly simple process model with two tasks/activities, the first one being “Select document to print” and the latter one being “Print document”. Thus, this process model defines the procedure of choosing a document to print and printing it. The left-most circle represents a start event, whereas the right-most circle represents the end event of a process.

2.2 Business Process Management

The process model and process instance are the outcomes of the modelling and execution phase, respectively, in the life-cycle of a business process. This life-cycle is studied by the Business Process Management (BPM) discipline. BPM is constituted by a set of well-established principles, methods and tools that combine knowledge from information technology, management sciences and industrial engineering with the purpose of improving business processes [Van Der Aalst et al., 2016]. The complete BPM life-cycle consists of five phases: *design*, *modelling*, *execution*, *monitoring* and *optimization*. In the design phase, analysts work on establishing the facts and details surrounding a specific business process which can then be used to model the process. In the monitoring phase, which can also happen during the execution phase, certain tools can be used to supervise and surveil the running process instance. The results of the monitoring phase are then used in the optimization phase of the life-cycle to establish if and how the process model can be improved for future iterations.

2.3 BPM systems

Organisations can opt to implement BPM by making use of BPM systems that are developed to increase the effectiveness and efficiency of the organization [vom Brocke et al., 2014] and its business processes. The phases of the life-cycle are as much contingent on one another as they are independent. They are contingent because the outcome of one phase serves as input for a different one; there would be no process instance without a process model and no monitoring results without the execution of a process instance. Structurally, however, each one of these phases represents an independent entity with a set of functionalities that produces the desired output. A BPM system is a composition of such entities that are able to manifest the life-cycle of a business process.

Designing the entities of a BPM system is just as important as designing the process models. These entities have to be compatible with the target goals of the organisation and must be easy-to-use for their intended end-users (for instance, business analysts or other individuals that are in charge of maintaining the business processes). Extensive research has been conducted with primary focus on each one of the entities. [Wetzstein et al., 2010] present a complete overview of the monitoring life-cycle in (cross-organizational) business processes. The framework they propose derives its requirements from a specific business process, however, it can also be applied to different use cases. What this and other monitoring frameworks have in common, is their requirement of establishing a *monitoring agreement*. This implies that business organisations which opt to monitor their process instances, must establish at least one *subject* for the monitoring entity. This subject will be scrutinised during the monitoring phase and any optimization strategies that will be developed will aim to improve process performance based on that particular subject.

[Karastonayova and Ghahderijani, 2021] introduce an extensive and complete diagram of possible subjects for monitoring frameworks. They also place their focus on predictive monitoring frameworks which are able to make predictions about the subjects' values before the termination of the process execution. This gives process maintainers the opportunity to improve the outcome of their process instances before their termination which, in turn, has great implications for the target goals of the business organisations and their values.

2.4 Process adaptation

The ability to improve a process instance's outcome before its termination is not a functionality of any of the five phases of the BPM life-cycle. Instead, it is referred to as *process adaptation* [Karastonayova and Ghahderijani, 2021]. The advancement of predictive monitoring technologies has increasingly put forward the need for tools which can apply the prediction results to process instances that have already began their execution. Moreover, in the majority of cases, discovering human-errors on process instances would lead to a definite termination of the execution. Assuming that certain process instances have a lengthy running time, terminating mid-execution could lead to not only down times but, potentially losses in money, customers or data amongst other examples. Of course this is not definite for all business processes, which is why the motivation for applying process adaptation thoroughly relies on the individuals responsible for the processes and their life-cycle.

Scientists have presented a great deal of interest on the field of process adaptation, with many studies introducing clear overviews of how process adaptation can fit into the BPM life-cycle. Two of these studies, [Kazhamiakin et al., 2010] and [Wetzstein et al., 2012], indicate how business organisations can apply process adaptation on the basis of the monitoring results of 'process quality factor analysis' and 'KPI violations' ³ respectively ⁴. The latter uses data from monitoring historical process instances to build decision tree algorithms that can learn the dependencies between the KPI-s. These decision trees are then used to generate predictions for the run-time process instances. We can, therefore, understand that the application of process adaptation results in the development of a complex system whereby a new set of requirements and sub-systems need to be defined.

[Karastonayova and Ghahderijani, 2021] propose the functional architecture of a BPM system which supports autonomous process adaptation based on the results of a predictive monitoring tool (Figure 2). Such a system would be able to autonomously decide whether there's a need for process adaptation and then selecting the most appropriate adaptation strategy for its application.

This architecture aims to create a general approach to process adaptation which can be applied regardless of the monitoring or adaptation subjects. A pivotal part of this proposal is the presence of an entity that can recognize an *adaptation trigger* and enact the process adaptation based on the answer. We discuss this in the following section.

2.4.1 Adaptation triggers

Adaptation triggers are the *cause* to the *effect* of process adaptation. This means that a process instance can be adapted run-time only if there has been an adaptation trigger. Different types of triggers are caused by a multitude of different reasons. Figure 3 provides an extensive diagram of the various

³In 4.1, we briefly introduce the concepts of KPI-s and violations in the context of our study. In §5.1.3 and §5.1.4 we present the service implementations for a KPI and a KPI violation respectively.

⁴These serve as the subjects of the monitoring agreement that we introduced in §2.1

reasons that could cause an adaptation of a process instance. We can note how every reason has the potential to create an adaptation trigger and ultimately, cause process adaptation.

The first two axes of the diagram represent the reasons which can make a process instance prone to process adaptation, for instance, changes in infrastructure of an organisation or a shift on the available resources (second axis). The different types of metrics on the first axis can be used as subjects to predictive monitoring tools that make predictions of the metrics' values. When the results of these predictions indicate that the organisation's target goals may not be met, the process instance will be prone to adaptation.

Adaptation triggers (third axis) indicate *why* the 'reasons' from the first two axes can prompt the adaptation of a process instance. For example, unsatisfactory results from predictive tools indicate a violation of the organisation's target goals; recent changes in a specific law indicate that a re-alignment of the organisation's policies may be due. One particular 'reason' can be mapped to one or more adaptation triggers and one or more adaptation triggers can then be mapped to (possibly more than) one appropriate adaptation strategy. The relevance of recognizing the most accurate adaptation triggers is crucial in assisting the system to define the most appropriate adaptation strategy (fourth axis).

2.5 Adaptation strategies

The identification of adaptation triggers is succeeded by the application of the process adaptation. Ideally, an organisation would be able to choose from a set of available *adaptation strategies*. These strategies could focus on different aspects where adaptation could be applied, such as the functional services and the control and/or data flows amongst others. More importantly, it is possible to apply adaptation to the whole process model (such that future process instances can abide by it) or strictly to one process instance. [Karastonayova and Ghahderijani, 2021] provides an elaborate diagram of the different dimensions wherein process adaptation is applied. An organisation's choice of adaptation strategy depends on the set of available strategies.

Adaptation strategies in Camunda [Roman, 2021], provides an implementation for two adaptation strategies, "Instance Migration" and "Data Transfer", using Camunda⁵. The idea of Instance Migration is as follows: let us assume that the most appropriate adaptation strategy for a process instance P , would be to update its process model's BPMN, α . The process maintainer, could define the newly updated BPMN of the process model as β . P would, thus, need to be 'migrated' from its initial α notation to the more recent β notation. In addition to the implementation of the "Instance Migration", [Roman, 2021] also extends the Camunda modeller such that the individuals could use it to update the BPMN notations and have them uploaded directly to the workflow engine, where the process instances were initially running.

Data Transfer is an adaptation strategy similar to Instance Migration. Whenever the β model includes the deletion, update or replacement of an active task in the α model, this adaptation strategy will ensure that no data is lost during the migration of the process instances to the new model.

⁵Camunda has established its own Migration API that can be used to implement the Instance Migration method according to the user's preferences: <https://docs.camunda.org/manual/7.15/reference/rest/migration/>.

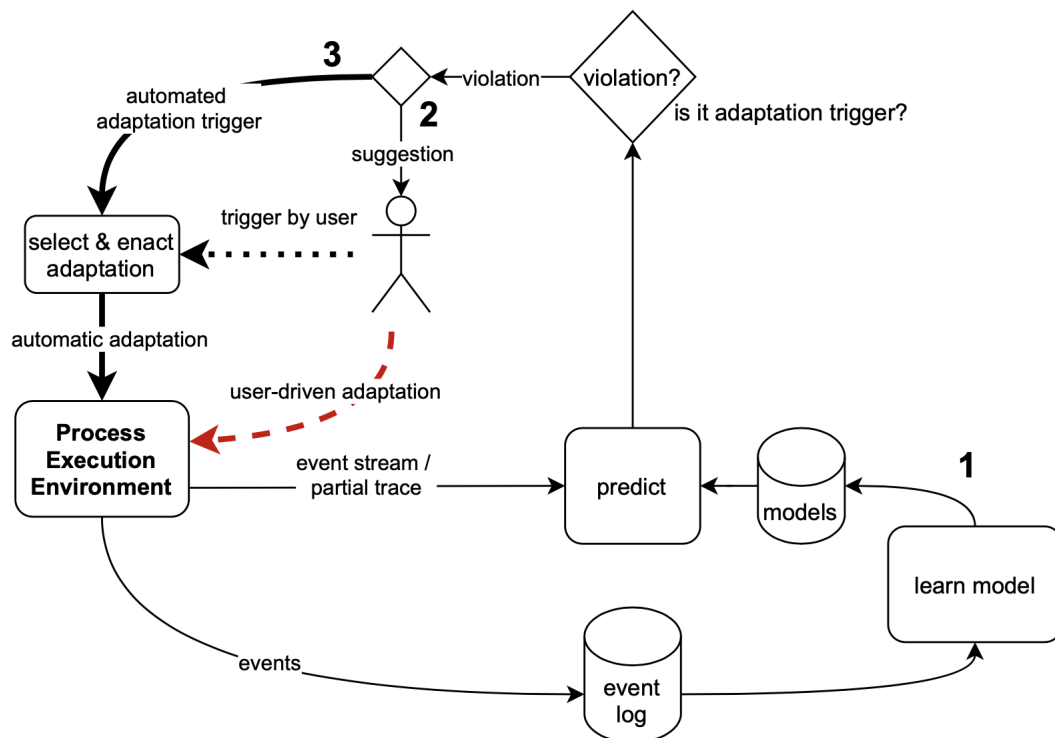


Figure 2: A workflow proposal from [Karastonayova and Ghahderijani, 2021] of a BPMS that supports process adaptation. Path 1 depicts how monitoring tools can use event streams produced by the process execution environment to create prediction models with the purpose of making predictions for running process instances. Paths 2 and 3 both indicate two different ways that process adaptation can be enacted. The former indicates how an adaptation strategy is enacted manually from an individual end-user of the system, while the latter represents how the adaptation system is responsible for selecting an appropriate adaptation strategy.

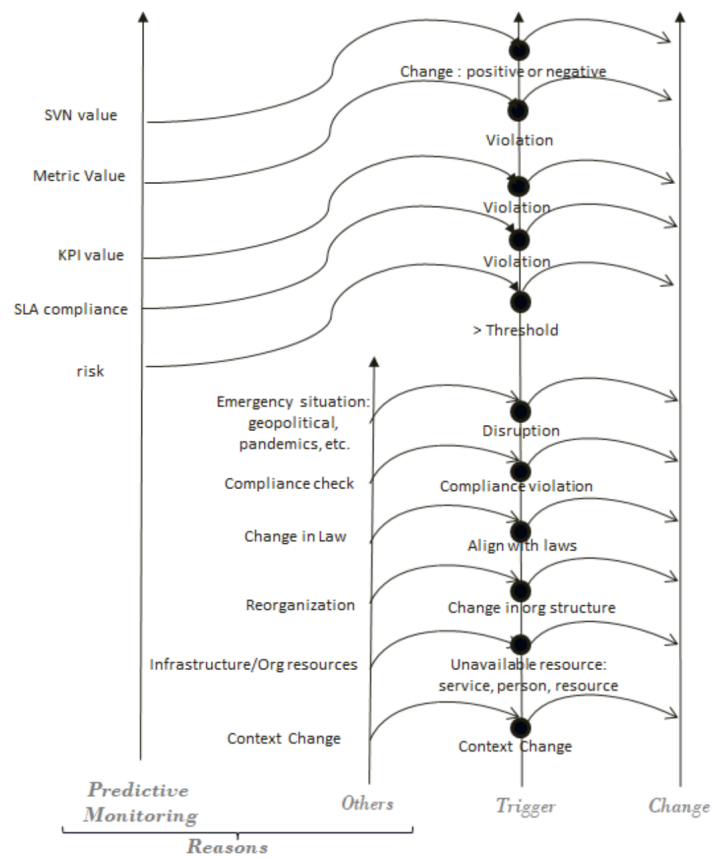


Figure 3: An extensive diagram providing an overview of possible reasons that may cause adaptation triggers, which in turn, cause process adaptation; retrieved from [Karastonyova and Ghahderijani, 2021].

3 Problem analysis

Given the vast interest in process adaptation and the research that has been put forward, it is understandable that a lot of effort be placed on creating methods for its application. However, this raises new concerns that need to be addressed, with new challenges that need to be taken into account. If we want to build a system that not only emulates process adaptation but also autonomizes it, we have to inspect these challenges so that we can derive a set of requirements on which to base the software system. Furthermore, we have to understand just how much autonomy can such a system have; where does the human involvement begin and where should it end. We use our research information from §2 to conduct an analysis of the problem that we are going to tackle in this thesis.

As stated previously, the foundation of our work relies on [Karastonayova and Ghahderijani, 2021]’s research and their workflow proposal of a BPMS⁶ that supports process adaptation (Figure 2). This proposal introduces three key **entities** that engineer the adaptation process:

1. a *predictive tool*, which we will assume predicts the values for an arbitrary metrics μ
2. a *mapping tool*, which is able to map an adaptation trigger with an adaptation strategy
3. the *adaptation application* entity, which is responsible for applying the chosen strategy to the process instances

Listing 1: List of entities that we derive from the proposed architecture (Figure 2).

The interaction of the three entities is as follows: the results of the predictive tool (entity 1) are used to calculate whether the process instance requires adaptation. Depending on these results, the mapping tool (entity 2) will match the cause that triggers process adaptation to one or more possible adaptation strategies. Finally, the mapped strategy is applied to the process instance using entity 3. This proposed architecture also offers the opportunity to purposely allow human involvement (path # 2 in Figure 2) by having individuals enact the adaptation strategy. We get to understand that an adaptation system will comprise of several sub-systems that can enact the functionalities of each entity in the architecture. Throughout the rest of this chapter we will discuss the concept of an adaptation pipeline, as a means to present a high-level overview of the adaptation procedure in §3.1, we introduce the challenges of process adaptation in §3.2, followed by the main research question of our study in §3.3. Finally we present the system requirements that we have designed in accordance with the results of this analysis in §3.4.

3.1 Adaptation pipeline

We use Figure 4 to provide a high-level representation of an adaptation workflow, which is essentially an enactment of the architecture we discussed in Figure 2. The pipeline constitutes of two stages, each of which is concerned with the execution of a specific service. Stage I comprises a service that can indicate whether a process instance requires adaptation, whereas Stage II consists of a service that can apply the appropriate adaptation strategy to the processes. If we were to think of these stages as two separate functions, then the input for Stage I would be a specific process that is a candidate for process adaptation. We define a candidate process as any process instance that could *potentially* require adaptation. The output of Stage I states whether a candidate process requires adaptation, that is, an adaptation trigger has been defined for the process instance in question.

⁶Business Process Management System

On the other hand, the input for Stage II of the adaptation pipeline is the process instance that requires adaptation, whereas the output would be the process instance once one (or possibly more) appropriate adaptation strategies have been applied to it.

Stage I coincides with entity 1 of Listing 1, while Stage II coincides with entity 3. As we mentioned in §2.5, [Roman, 2021] puts forward the possibility of working with adaptable processes in Camunda as one way of approaching the adaptation of process instances by implementing Instance Migration, therefore providing a concrete implementation for entity 3. As such, for the purpose of this study, we need to position our focus on constructing the systems that can implement the other two entities. If we were to rephrase the preceding statement in accordance with the workflow in Figure 4, we aim to build a system that can recognize a process instance’s need for adaptation and why.

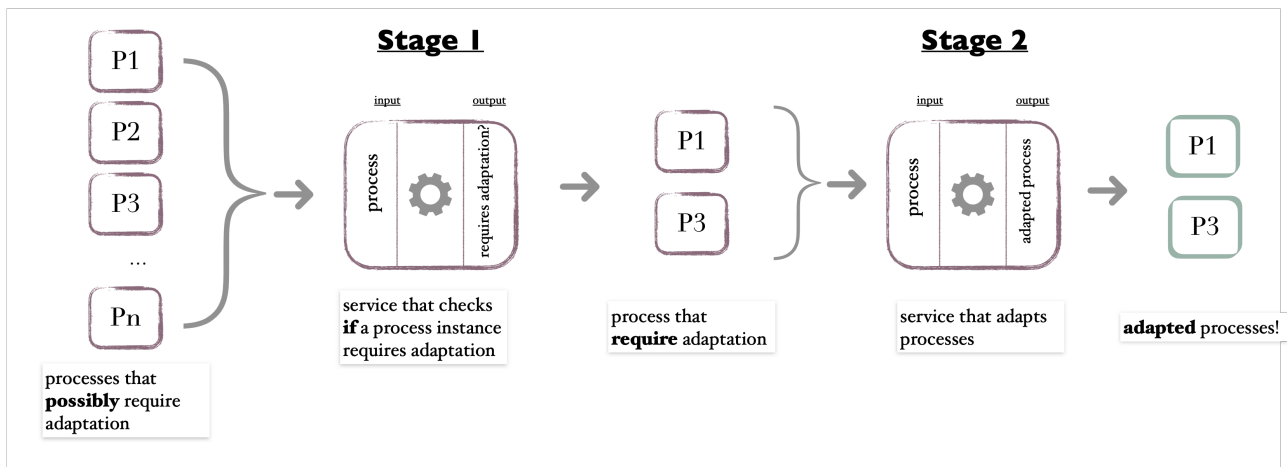


Figure 4: A high-level representation of a process adaptation pipeline; let us assume that process instances **P1** and **P3** require adaptation. At the start of the pipeline, each one of the process instances is a candidate for adaptation. Further on in the pipeline, we see the service in Stage 1 has determined that two of the instances need to be adapted (for some arbitrary reason). Stage 2 adapts both process instances. At the end of the pipeline we see how both processes have been adapted; the outlines of their blocks are different so as to demonstrate the differences between the instances at the beginning of the workflow and the instances at the end.

3.2 The challenges of process adaptation

Before we can commence building a system that can apply process adaptation to process instances, we need to study how the theory on process adaptation can be merged with the practice. This is important in order to understand what is feasible for implementation with the current existing technology. We wish to make use of the available studies from §2 while building a system which creates new value(s) without having to reinvent the wheel. For this reason, we first need to understand some of the known challenges that a process adaptation system must accommodate.

Process adaptation is not one-dimensional The reasons why a process instance would require adaptation can be multi-fold. While it would make sense for a wrong input variable to cause a process instance to acquire an instance migration ⁷, what happens when another process instance runs out of

⁷Process instance migration in Camunda is one of its core functionalities wherein the new deployment of a process model does not affect the execution of process instances that were based on the previous process model.

resources that it can make use of? The necessary adaptation strategy in such a case would have to make the process be eligible to use a new set of resources. Thus, perhaps an adaptation of the process model might be more suitable for this situation (wherein a new set of resources is made eligible in exchange for the unavailable ones). Either way, it is evident that a process adaptation system has to be able to understand the reason why a process instance may require adaptation before it can choose an appropriate solution to the adaptation trigger.

Similarly, it could sometimes be the case that a process instance may run into more than one issue during its execution, each of which requires a special adaptation strategy for its attendance. Therefore, we have taken note how process instances could require adaptation not only for potentially different reasons in every execution, but sometimes also for more than one reason at a time.

Process adaptation can be risky BPM systems that support process adaptation must also ascertain that a thorough analysis of the process' execution is conducted. The results of such an analysis are used to determine whether a process instance is in need of adaptation. As intricate as such an analysis could be in determining which aspects of a process instance require adaptation, it must also be able to take into account whether the outcome of applying an adaptation strategy is better than the outcome of allowing the process instance to finish execution in its original, planned form. In a worst case scenario, an incomplete analysis could suggest an adaptation strategy that exacerbates the expected outcome of the process execution.

Processes are prone to disruption The BPM lifecycle demonstrates that processes are in a constant state of modification. The results from the monitoring phase are used to determine possible optimization strategies for the new iteration of process instances. It is safe to assume that each aspect of a business process is prone to change and as such, the potential issues that are revealed during the execution of its process instances are, similarly, prone to change as well. This could present a challenge for adaptation systems to recognize issues because they have already been established based on inspections of past process instances. Furthermore, new issues could possibly require introduction of new adaptation strategies.

3.3 Research question

Having mentioned several challenges that surround the topic of process adaptation, we now present the key research question of our study: *How can we construct an autonomous adaptation system that is able to tackle the challenges of adapting a process instance?* This research question places focus on two important aspects; firstly, constructing a *complete* adaptation system and, secondly, constructing an autonomous system. Let us discuss both aspects separately, starting with the first one.

For the purpose of this thesis, we define a *complete* adaptation system one that takes into consideration the challenges that were introduced in §3.2. Such a system has to be able to discern the (possibly multiple) issues that a certain process instance is facing and identify the adaptation strategies that can *improve* the outcome of the process instance. Furthermore, such a system has to be versatile such that whenever the process model is updated, the adaptation system would be just as viable in recognizing (new) issues and strategizing (new) adaptation methods.

The second aspect that our research question puts forward is the need for constructing an autonomous system. We have derived this requisite from [Karastonyova and Ghahderijani, 2021]'s research,

which places a great deal of focus into the collaboration of predictive process monitoring and process adaptation for achieving process performance improvement. An autonomous software system is self-sufficient when is able to execute services with minimal to no human involvement [Walch, 2020]. An autonomous process adaptation system would have to be able to identify possible issues with process instances, possibly using predictive tools, and offer the most appropriate adaptation strategy, independent of the system's maintainers.

3.4 System requirements

We finalize this chapter by presenting the set of requirements that we have derived from our problem analysis. We will use these requirements to construct a complete and autonomous adaptation system that is able to recognize a process instance's need for adaptation and, in doing so, will prepare the process instance for the application of whichever adaptation strategies are available *and* appropriate. In order to attain system completeness as defined in §3.3, we have to construct an extensible system. The system's extensibility will make it possible to introduce not only new metrics and adaptation triggers, but new calculation methods for them as well. Furthermore, seeing how this system is only the first stage in the adaptation pipeline in Figure 4, it must ensure inter-operability with the services of the other stage(s) as well. To provide a more complete outlook on the system's functionalities and its performance, we present the list of functional and non-functional requirements.

3.4.1 Functional requirements

Functional requirements are used to describe the functionalities of a working software system. They are based not only on the goals of the product (system's) owner, but also on the analysis conducted by the developers to determine the feasibility of the system's properties. Let us first introduce two important concepts that are relevant to this system's development and its requirements.

Violations The concept of violations was first introduced in §2.4.1 as an adaptation trigger of the unsatisfactory results of the predictive systems that indicate noncompliance with a particular's organisation's target goals (Figure 3). We wish to emphasise on this concept because it is a pivotal aspect in building an initial version of our system and it will be used extensively throughout this study.

Metrics criteria We use metrics criteria to allow our adaptation system to recognize any violation of the available metrics. This system can use the criteria to understand the (predicted) values of a process instance's metrics and whether they are within the appropriate norms. The criteria are defined based on the type of the metric (KPI value, SLA compliance etc), which in turn helps dictate the functions that will conduct the evaluation. As an example, for KPI values, the criteria is defined based on the mapping function.

We now introduce the list of functionalities that we have defined for the adaptation system:

- F1** The system must be able to recognize adaptation triggers for a process instance based on all the available reasons introduced in the first two axes in Figure 3.
- F2** The system must be able to understand the criteria of the process' metrics.
- F3** The system must be able to recognize whether there has been/is expected to be a violation of the process metrics' values, using the available metrics criteria.
- F4** The system must be able to update the definition of the metrics criteria, using end-user input.

- F5** The system must be able to introduce new metrics criteria.
- F6** The system must be able to introduce new adaptation triggers.
- F7** The system must be able to work with external predictive tools that are used for predicting the values of the available metrics.
- F8** The system must be able to allow application of available (external) adaptation strategies.

Listing 2: List of functional system requirements that we derive from our problem analysis.

3.4.2 Non-functional requirements

Based on the list of properties that we defined in Listing 2, we have identified a set of non-functional requirements, that can demonstrate *how* the system is expected to perform. These are pivotal in helping us determine the architecture and practical structure of our system. The list is as follows:

- N-F1** The system should be extensible.
- N-F2** The system should be inter-operable.
- N-F3** The system should be scalable.
- N-F4** The system should be maintainable.
- N-F5** The system should be user-friendly.

Listing 3: List of non-functional system requirements that we derive from our problem analysis.

Let us discuss each one of the requirements separately and understand our decision for their inclusion and how they are relevant to our system's goals.

Extensibility The principle of extensibility in software engineering describes a software system's prospects for future growth. It is measured on two aspects: the ability to extend the system *and* the level of effort that is required to implement the extensions. In order to create a system that is able to recognize adaptation triggers despite the changes in a process model causing possible disruptions in the metrics criteria, we must ensure its extensibility. The application of this principle gives our system the opportunity to implement functional requirements **F4**, **F5** and **F7**, which results in a fulfilment of functional requirement **F1**; the implementation of these properties ensure the completeness of the system (§3.3).

Inter-operability The principle of inter-operability in software engineering depicts how two or more software systems can cooperate with one another despite differences in language, interface, and execution platform [Wegner, 1996]. This is relevant for our software system because it needs to be able to communicate with external systems that provide different services such as application of process adaptation or predictive monitoring. At the same time, inter-operability helps fulfil functional requirements **F7** and **F8**.

Scalability The principle of scalability supports a software system's growth in a certain dimension without having to waive the user experience or other functionality. It is important for our system to be scalable as it grows into supporting more adaptation triggers and, therefore, supporting greater amounts of data. We can opt to build a scalable system by designing a modular architecture (as introduced in Figure 2).

Maintainability The principle of maintainability allows for the design of software systems wherein the addition of new requirements does not risk the introduction of new errors [Sommerville, 2007]. This principle is quite relevant to our system not only because it will be prone to extensibility, but also because of the possible attraction of other developers working with process adaptation pipelines. Similar to the principle of scalability, a modular architecture proposes more advantages to maintainability compared to other designs.

Usability The principle of usability is concerned with the user-experience of a software system. It is important for the Adaptation Dashboard to provide simple and straight-forward approaches for the end-users to conclude their tasks. Furthermore, the target users of the system are, potentially, people of different academic backgrounds, which is why it is important for the dashboard's functionalities to appear as effortless as possible.

4 Architecture

In this chapter of the thesis we make use of our analysis' results to tackle the research question - "How can we construct an autonomous adaptation system that is able to tackle the challenges of adapting a process instance?" - by constructing a software system that follows the requirements that were introduced in Listing 2 and Listing 3. This chapter is divided into two key sections; in §4.1 we present our solution proposal to the research question and in §4.2 we introduce the system design we have developed based on the solution proposal.

4.1 Proposed solution: constructing an Adaptation Dashboard

In order to accommodate the requirements that we have established for the adaptation system, we propose to construct an *adaptation dashboard*. Creating such a dashboard will allow end-users to visualize the adaptation phase of the business processes (the two stages in Figure 4). At the same time, the existence of a user-interface allows end-users to enable the adaptation strategies of their choice, thus enacting path # 2 in Figure 2. In the Appendix (§7.4), we have included images of the end-result of our system, a user-friendly adaptation dashboard that (among others) can calculate whether a process instance that is running on a Camunda engine, requires adaptation. In order to provide a clear depiction of how this system can be used, we will introduce a trivial process model that will be executed by Camunda's engine in Figure 5. By providing a simple walk-through of how one can make use of the adaptation dashboard, we aim to elaborate on our proposed solution and provide some initial insight into the system before we get to introduce more specific details.

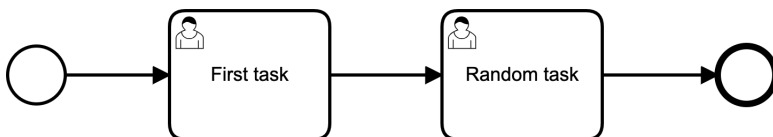


Figure 5: BPMN representation of a trivial process with two user tasks.

The process model in Figure 5 consists of only two tasks; upon the completion of both tasks, a process instance of this model will terminate definitely. If we were to assume that this process would require some sort of adaptation before its termination, then we can state with certainty that it would not be possible to apply it because the model clearly dictates how the flow of the process shifts from one task to the other without any possibilities of diverging. As a result, in order to have the option of adaptability with running process instances, we propose working with *adaptable processes*.

Adaptable processes Adaptable processes are process instances which have the option of being adapted during their execution, regardless of the type of adaptation. Designing adaptable processes can be dependent on numerous factors, including the process execution environment, target goals of the end-users and available tools, among others. In this study, we define adaptable processes as processes whose models include at least one *checkpoint*.

Checkpoints The concept of checkpoints has been previously introduced in various research papers on the field of process adaptation, including [Wetzstein et al., 2012] and [Metzger et al., 2015]. Checkpoints are included in processes models to present a 'check-up' moment, where an external

service or an end-user can inspect the parameters of the process and note whether the process is performing as expected. In the case when something appears to be wrong, it can be established that the process can possibly be adapted. Checkpoints can also serve as a means to correct mistakes that were only noted after the start of the process instance execution. In §4.1.1 we introduce our implementation of a checkpoint's life-cycle and how we can put it into practice in a process model.

KPI-s In §3.4.1 we introduced the concepts of *violations* and *metrics criteria*. The metrics criteria that are used to check the performance of a process are defined by the process maintainers. The default metrics that we have used in designing in our adaptation system are the Key Performance Indicators (*KPI-s*). The most common adaptation trigger that is associated with KPI-s is the *KPI violation*.

KPI-s need to be defined appropriately such that they can help determine the quality of the process' performance. Our definition of a KPI is based on the definition from [Wetzstein et al., 2012], according to which, a KPI consists of three elements:

1. KPI metric: a type of metric that is derived from a designated function whose input usually depends on specific metrics that are extracted from the process' execution environment.
2. KPI classes: a set of at least two nominal values to interpret the process performance (e.g., "good", "medium", "bad")
3. mapping function: a target value function which maps values of the KPI metric to KPI classes

Listing 4: Definition of a KPI, based on [Wetzstein et al., 2012]

The mapping function of a KPI is quintessential in determining whether a KPI violation has occurred. In such situations, process maintainers would usually opt to adapt the process such that the violation can be modulated. An example of a KPI would be the running time of a process instance. If there are only two KPI classes ("green" to dictate that the process is running as expected and "red" to dictate that the process requires adaptation) and a mapping function that maps process runtime-s that are smaller than the average process runtime-s to *green*, while every other value is mapped to *red*, then a *red* class would indicate that a violation of the KPI has occurred.

4.1.1 Designing an adaptable process

We can convert any process into an adaptable one if we introduce at least one checkpoint in its model before the terminating event (the bold-lined circle in Figure 5). Given the research that has been conducted for this study and the system development, we propose an implementation for the *life-cycle of a checkpoint* as follows:

1. checkpoint send task
function: pass along a message with the state of the process instance
2. message receive task
function: receive a message that can allow the resumption of the process instance in question
3. (optional) message processing service task
function: execute other functionality that are/can be passed along with the message from the "message receive task"

Listing 5: Our definition of implementing a checkpoint's life-cycle, in line with our system's requirements (§3.4) and architecture (§4).

Given the definition of the checkpoint state, we can update the process model from Figure 5 into an adaptable process by introducing a checkpoint state between the two user tasks; the updated model can be seen on Figure 6.



Figure 6: BPMN representation of the adaptable process model from Figure 5. We can note the “checkpoint send task” labelled as Checkpoint, the “message receive task” labelled as Check is over and the “message processing service task” labelled as Process message, all three of which make up the “checkpoint state”.

4.2 System design

In this section of the chapter we place focus on the design of the adaptation system by presenting an overview of the system architecture and the sequence diagrams of the main use-cases. The system architecture overview is presented in Figure 7. The Adaptation Dashboard (AD) system consists of *six* entities in total, three of which share a data layer. This architecture is an application of the microservices architecture style, where the system consists of smaller independent services that communicate together via lightweight mechanisms [Aderaldo et al., 2017]. Each one of the entities (microservices) comes with an API⁸ that renders it accessible to the rest of the services. It is important to note that “Camunda engine” is an external service, however, its API makes it possible for it to send messages back and forth with the other services. Following, is a list of all the services that compose AD.

1. *External task service* - connecting running adaptable process instances with “Mapping server”
2. *Mapping server* - key service, main linking point between the end-user and process instances
3. *Process resumption system* - connecting “Mapping server” with adaptable process instances
4. *Violation-check system* - calculating violation checks
5. *KPI system* - storing and retrieving information about the KPI-s
6. *Camunda engine* - rendering functionalities from Camunda accessible to AD

Listing 6: List of all the services that compose the Adaptation Dashboard system.

4.2.1 Service communication

Presenting the relationships between the services that compose a software system is an important aspect of said system’s architecture. In Table 1 we introduce all the one-to-one relationships between the entities as demonstrated in Figure 7. The existence and definition of these relationships is especially important in a microservices architecture, because they are crucial to the implementation of the system’s functionalities.

⁸Application Programming Interface

Initiator	Receiver	Relationship functionality
External task service	Mapping server	Pass along information about a process instance in the checkpoint state.
Camunda engine	External task service	ETS receives a message from a process instance that has reached the checkpoint state.
Mapping server	Camunda engine	MS acquires and receives specific information about the history of a process instance.
Mapping server	Process resumption system	MS passes along information about a process instance that will resume execution.
Violation-check system	Mapping server	VCS acquires and receives information about process instances.
Violation-check system	KPI system	VCS acquires and receives information about KPI-s.
Process resumption system	Camunda engine	PRS correlates messages (usually from the Mapping server) with Camunda engine.

Table 1: Overview of all the service couples that communicate with one another. We present a short introduction of main functionality for every relationship; further details will be introduced in the succeeding section(s).

Microservices abbreviation:

ETS: External task service

MS: Mapping server

VCS: Violation-check system

PRS: Process resumption system

KS: KPI system

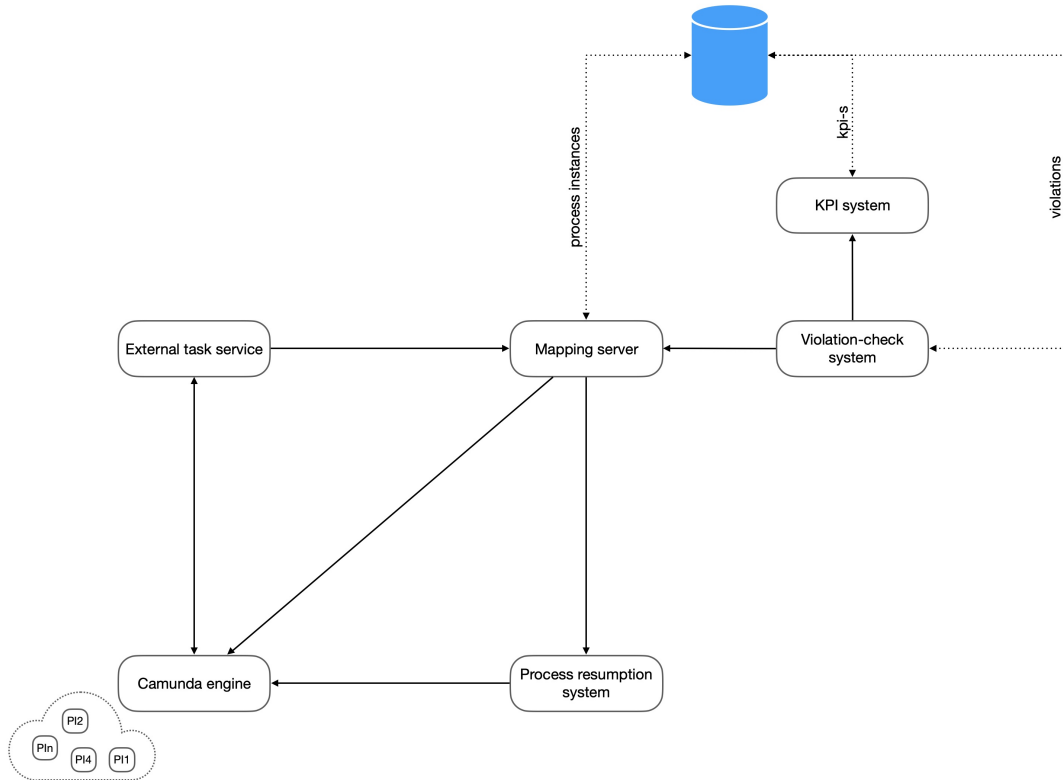


Figure 7: System architecture overview: the AD architecture applies the microservices architecture style; it consists of five smaller systems (or services) - Mapping server being the key service due to its functionalities - and Camunda engine as an external service; three systems also share the data layer. The relationship of two services connected by an arrow is explained as follows: the service on the bottom of the arrow is the one initiating communication with the service at the head of the arrow. For example, in the relationship between the “Violation-check system” and the “KPI system”, the former initiates communication with the latter.

4.2.2 Sequence diagrams

We use sequence diagrams to depict the sequence of activities that constitute the three main use cases of the adaptation dashboard, which are the following: (1) listing all the process instances that have reached a checkpoint state in Figure 8, (2) resuming the execution of a process instance in Figure 10 and (3) calculating whether a certain process instance has had a KPI violation in Figure 9. We elaborate on all three use-cases respectively.

Listing process instances in a checkpoint state When an adaptable process instance is being executed, it will eventually reach at least one checkpoint state. This state is designed such that upon its trigger, a message is sent to the ETS (via Camunda platform), containing relevant information about the process instance that is being executed. The ETS microservice will then send a message to the MS microservice about the new process instance resource that has been established. MS also serves as the backend service for the Adaptation Dashboard, therefore rendering the list of all the available process instances that are in a checkpoint state, for the end-user to see. An example of the dashboard listing two process instances can be noted in Figure 22 of the Appendix.

Perform violation check for a KPI Before opting to resume the execution of a process instance or deciding to adapt it, the end-user has the option to see whether there have been any violations of the KPI-s that are available in the system. The front-end sends a message to the VCS microservice with the identifiers of the process instance in question and the KPI for which the violation check should be calculated. Therefore, VCS sends a message to the KPI microservice with the KPI identifier and, in return, receives the KPI entity; a similar transaction occurs between VCS and the MS microservice as well. Upon obtaining both entities, the VCS microservice performs the violation-checks accordingly. The result of the calculations are then broadcasted in the front-end for the user to see. Figure 25 and Figure 26 depict how the user can trigger the violation checks and then see the results respectively.

Resume execution of a process instance When the end-user of the Adaptation Dashboard has already ascertained that a process instance does not require adaptation, they can opt to allow said process to resume its execution. This flow is triggered from the front-end, i.e. the end-user themselves. Upon its triggering, the MS microservice sends a message to the PRS microservice. The former is able to pass along an entity which encapsulates information about the execution state of the process instance in question. Upon receiving a message from MS, PRS will then correlate a message⁹ with Camunda’s engine, such that the latter can trigger the resumption of the appropriate process instance. Figure 23 depicts the button on the dashboard that can trigger this flow.

- * end user aka the UI frontend
- ** including the engine and the modeler

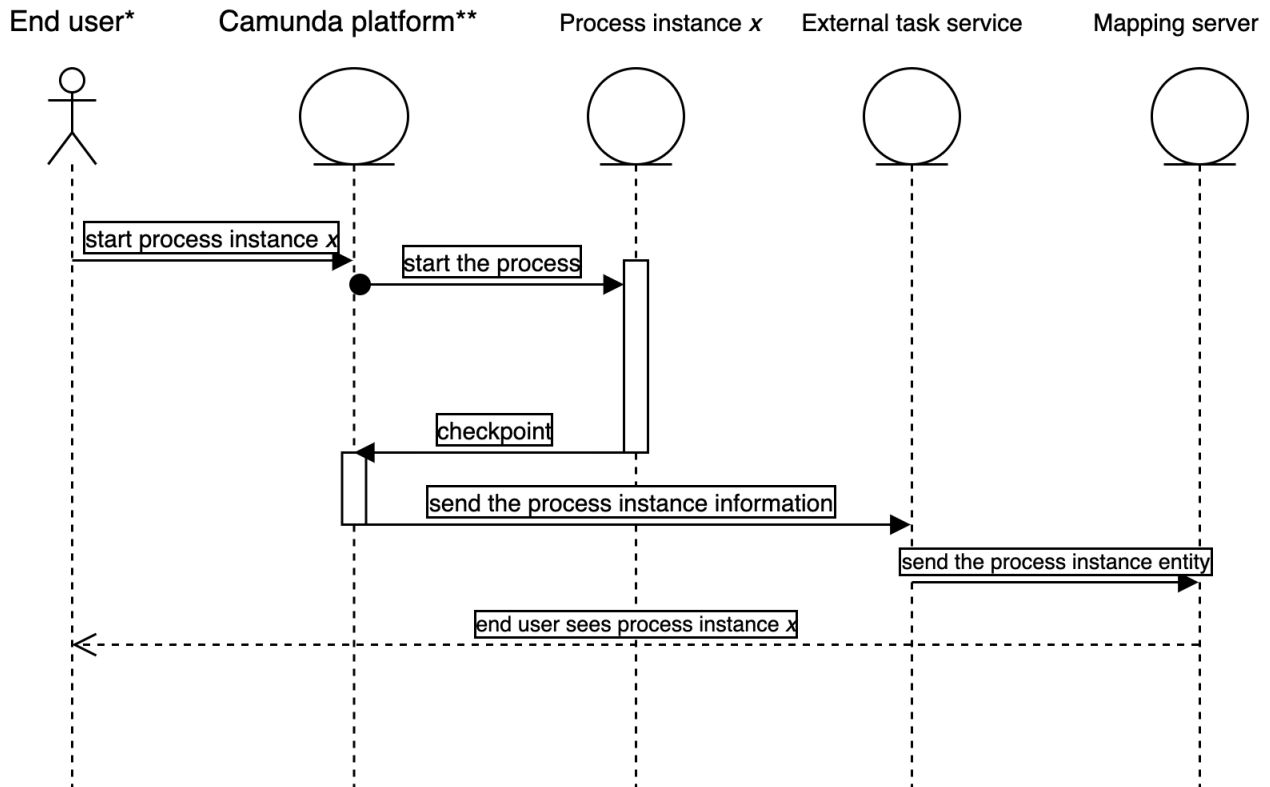


Figure 8: Sequence diagram: when a process instance ‘x’ reaches a checkpoint state.

⁹Process instances that have reached a *receive task* during their execution are awaiting for a message in order to move forward. In Camunda, the step of mapping such a receive task with any type of published message is called *message correlation*.

* end user aka the UI frontend

** including the engine and the modeler

FUNCTION to represent the situation when the violation calculation requires use of all the process instances

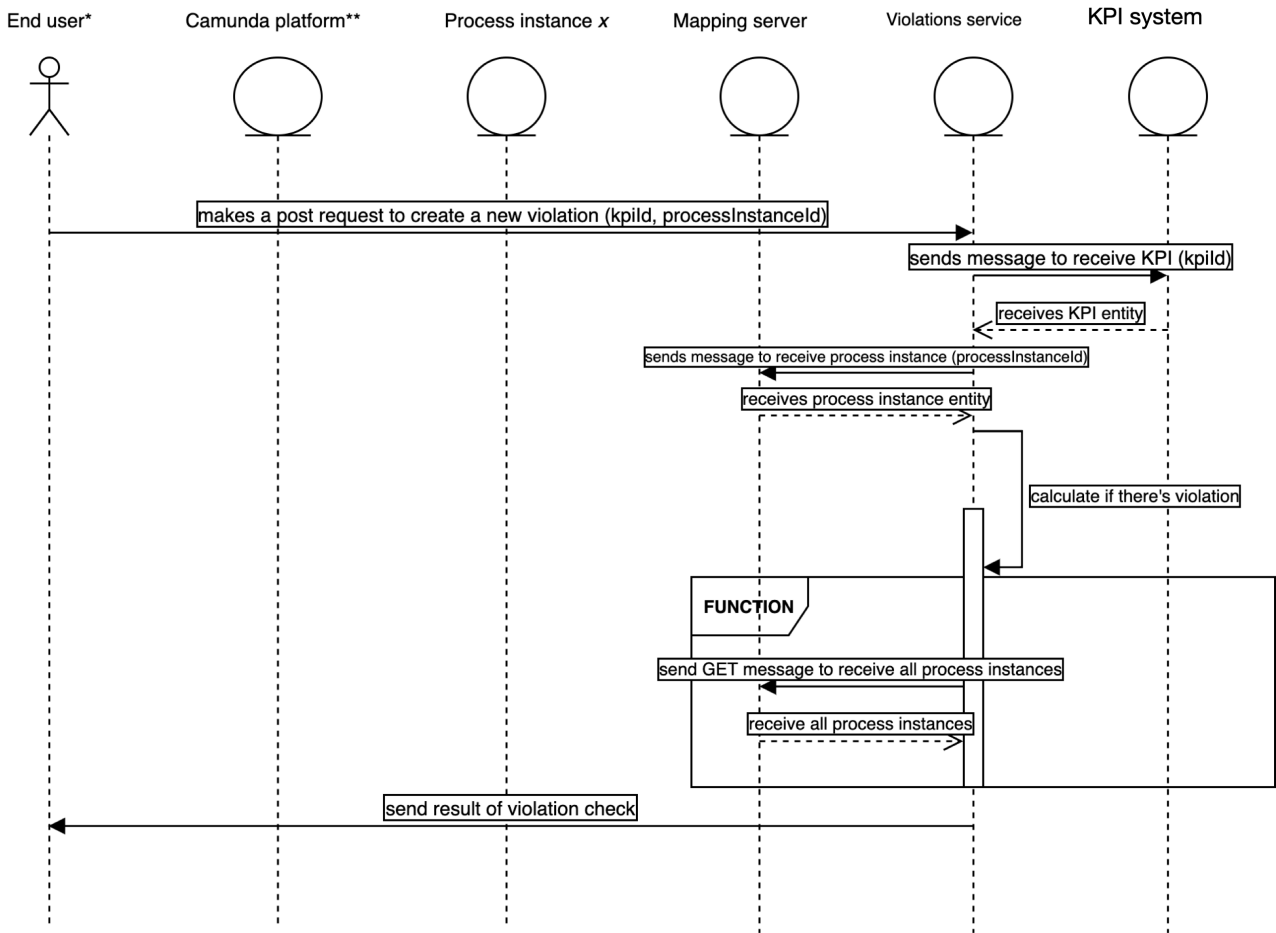


Figure 9: Sequence diagram: when the user opts to check for a KPI violation using the AD dashboard. The FUNCTION frame is initiated whenever the calculation requires information that is to be retracted from all the previously ran process instances.

* end user aka the UI frontend
** including the engine and the modeler

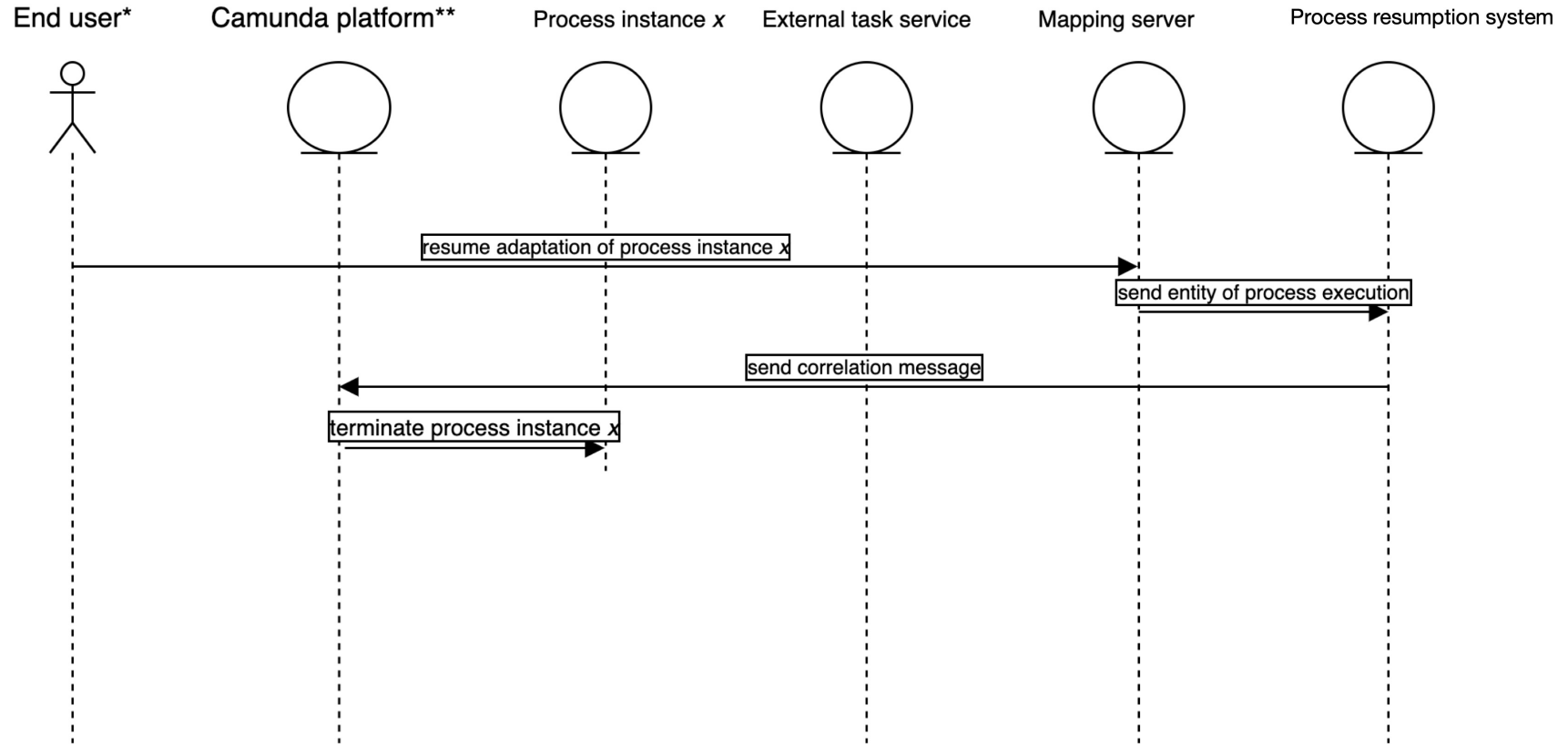


Figure 10: Sequence diagram: when the end-user opts to resume a process instance 'x' using the AD interface.

5 Implementation

This chapter will introduce the implementation details of the system that we have built, the *Adaptation Dashboard* system. We wish to present the inner workings of the system as well as its functionalities, all of which lie within its own components. The system has been built especially with non-functional requirement **N-F1** in mind (system extensibility), such that current components can be easily extended and new components can be just as easily integrated with the existing ones. Throughout the span of this chapter, we refer to the Adaptation Dashboard system as AD.

5.1 Microservices

We use this section of the thesis to elaborate on the Adaptation Dashboard (AD) system and all its software components, which we refer to as microservices, including their functionalities and how they communicate with one another. We reiterate that the diagram of the system's architecture can be noted in Figure 7, whereas Listing 6 presents all the microservices and their key functionality.

The AD system consists of six microservices, all of which communicate with at least another service. The Camunda engine is an external service which represents the execution environment of Camunda's platform. The service is equipped with its own API that is used for sending and receiving requests from the other microservices. All the other microservices run independently from one-another, and three of them are also connected to a common database. We use a MySQL database to persist relevant information about the entities that are shared amongst the microservices during their interactions. All the interactions are established by means of API-s, rendering AD a RESTful system. In the following sections we will provide more detailed information on each one of the services.

5.1.1 Mapping server

The MS microservice is the backbone of the AD system because it contains information about the adaptable process instances. Every microservice, existing or upcoming, that needs to make use of the information about process instances, has to be able to communicate with this service. MS and its resources are accessible through its API implementation.

Adaptation Dashboard history The user-interface of the Adaptation Dashboard consists of several tabs, one of which is dedicated to the history of adaptable process instances whose executions have terminated. Figure 30 of the Appendix presents an example overview of the history tab. MS is the service responsible for correctly storing information about the history.

Data layer The key entity of this service is the *adaptable process instance* entity. All the entities are persisted in the database table, where we currently store the following information: *checkpoint time* - as the time when the process instance reached the checkpoint state, *process instance id* - as the process identifier that is created from Camunda, *start time* - as the starting time of the process instance execution, *complete process running time* - as the process runtime after it has finished execution, *xml process definition* - as the xml representation of the process model, *new xml process definition* - as the xml representation of the new process model created via AD's editor, *is in checkpoint* - as a variable to denote whether the process in question is in a checkpoint state or not, *is terminated* - as a variable to denote whether the process in question has terminated execution or not, *end time* - as the ending time of the process, *process definition key* - Camunda-defined variable that is used to provide

human-readable names to process models, *process definition id* - Camunda-defined variable to denote the definition of a process model ¹⁰.

Functionalities The following listing provides an overview of MS’s functionalities.

1. Receive information for adaptable process instances that are in a checkpoint state from ETS.
2. Store adaptable process instances as entities in the database.
3. Retrieve information for all process instances, active and inactive.
4. Calculate the running time of process instances that are still active.
5. Retrieve the history of process instances from Camunda’s platform.
6. Delete the history from the database.
7. Resume execution of adaptable process instances.
8. Store new `xml` definition for a process instance.

Listing 7: Functionalities: Mapping server

5.1.2 External task service

The main goal of the External task service (ETS) is to serve as the connecting link between the process instance running in Camunda and our adaptation system. An adaptable process instance that reaches a checkpoint state is prepared to send a message containing relevant information about itself. ETS is designed to be on the receiving end of such messages. The implementation of ETS exploits two crucial aspects introduced by Camunda, *message correlation* and the *External Task pattern*. We have devised a process model - which we call “Checkpoint message receiver” - whose purpose is to receive the messages that are sent by any adaptable processes whenever they reach a checkpoint state ¹¹. The process model for the ETS service can be noted in Figure 11; its instances run in Camunda.

The model consists of two tasks, a ‘*receive task*’ called “Checkpoint” and a ‘*service task*’ called “Process message”. The former is used to correlate the incoming messages from the adaptable processes, provided that the processes are using the specific identifier that is expected by the task. The latter, being a service task, is responsible for implementing ETS’ main functionality: passing all the relevant information that is received from the messages along to the MS service. The External Task pattern is used to dictate Camunda that the implementation of a certain task is to be established using an external system, as opposed to other means of implementations (such as `JavaDelegate-s`, `scripts`, `expressions` and more). In the case of “Process message”, its external system is the ETS service itself.

In Camunda, we create a correlation between a ‘*send task*’ and a ‘*receive task*’ by defining the value of the *correlation* that is shared between the two. Therefore, the ‘*receive task*’ of process model “Checkpoint message receiver” will only be correlated to the ‘*send task*’-s that initiate a message correlation with correlation value `checkpoint-message`.

When the “Process message” task of “Checkpoint message receiver” is executed successfully, ETS initiates a new execution of the process in question; therefore, the “Checkpoint” receive task will be activated again, rendering the process ready to receive new messages from other adaptable process instances in a checkpoint state.

¹⁰Note that *process instance id* serves as an identifier of a running process instance, whereas *process definition id* serves as an identifier for the process model.

¹¹Please refer to Listing 5 to reiterate the definition of the checkpoint state

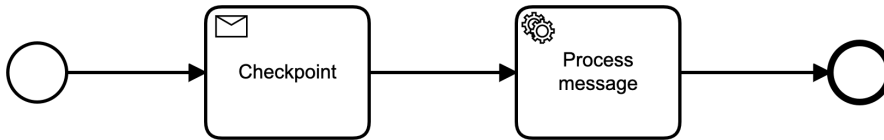


Figure 11: BPMN representation of process model “Checkpoint message receiver” that is used to correlate the messages that are sent from adaptable process instances in their checkpoint states and process them using ETS.

Functionalities The following listing provides an overview of ETS’s functionalities.

1. Application of the External Task pattern for the *service task* of process model “Checkpoint message receive”.
2. Receive information about the adaptable process instances that have reached a checkpoint state.
3. Send request to Camunda’s engine to retrieve the `xml` representation of the process’ BPMN representation.
4. Create an entity to encapsulate the information of the adaptable process instance.
5. Send created entity to MS.
6. Restart a new instance of the process “Checkpoint message receiver”.

Listing 8: Functionalities: External task service

5.1.3 KPI system

This service (KS) is designed to represent the different types of metrics that are used to control the process performance. In this study we focus specifically on KPI-s that are concerned with the running time of the process instances, but the implementation of this system can serve as a framework for other metrics as well (such as SLA compliances, SVN values and more). One of the tabs of the Adaptation Dashboard presents a list of all the KPI-s that have been defined in the system. Figures 27 and 28 of the Appendix demonstrate the KPI tab and the form that is used to create a new KPI respectively.

Data layer The main entity of this service is the *KPI* entity. This resource is created from user input in the Adaptation Dashboard. Similar to the other two key entities, KPI-s are saved in one of the tables on the database, where we store the following information: *KPI name*, *description* - a short text to describe the type of the KPI and how it can be used, *green class indicator* - a value from set $\{-1, 0, 1\}$ and *threshold* - to indicate a number (in milliseconds) that together with the ‘green class indicator’ are used to define the KPI’s mapping function ¹².

The KPI-s that we are working with in this adaptation system are expected to have three classes, a *green* one to indicate that the process instance should resume execution without adaptation, an *orange* one to dictate that the process instance could safely resume execution is on the limit of the threshold and a *red* one to indicate that a process requires adaptation because one specific KPI has been violated. We use the *green class indicator* property to inform the system about the way that it should perform the violation checks; a value of -1 indicates that “*the violation check maps to a green class for all the process run-times that are smaller than the threshold*”.

¹²Please refer to Listing 4 for a reiteration of the definition of KPI.

Functionality The following listing provides an overview of KS’s functionalities.

1. Store information on new KPI-s in the database.
2. Retrieve information on all the available KPI-s.

Listing 9: Functionalities: KPI system

5.1.4 Violation-check system

We have designed this service (VCS) as a means to externalize the violation checks that have to be performed for different KPI types. It can serve as a framework for future types of metrics and process performance indicators that will have to be checked for violations (or other types of adaptation triggers). One of the tabs of the Adaptation Dashboard is dedicated to the history of all the violation checks that have been performed using our system (an example overview of this tab can be noted in Figure 29 of the Appendix).

Data layer The main entity of the VCS system is the violation entity. This type of resource is created whenever the user triggers a violation check from the user-interface of the Adaptation Dashboard. The result is automatically stored in the database and can be retrieved only through VCS’s API. We store the following information: *kpi id* - as the KPI identifier of the KPI for which the violation-check was calculated, *kpi result class* - as the class of the violation-check result and *process instance id* - as the identifier of the process instance for which the violation-check was conducted. Consequently, we can infer that the violation entity is dependent on the process instance and KPI entities.

Functionality The following listing provides an overview of VCS’s functionalities.

1. Perform violation checks.
2. Store information on violation checks in the database.
3. Retrieve information on all the previously calculated violations.
4. Acquire and retrieve information on a specific KPI from the KS microservice.
5. Acquire and retrieve information on a specific process instance from MS.
6. Acquire and retrieve information on all the process instances from MS.

Listing 10: Functionalities: Violation-check system

5.1.5 Process resumption system

The purpose of this system (PRS) is to serve as a linking point between the Mapping server and a specific process instance. PRS’s most common use case has to do with the resumption of process instances that are in the second stage of the checkpoint life-cycle (the ‘message receive task’ stage, according to the definition from Listing 5). Upon receiving a request, which includes Camunda’s identifier of the process instance, from the MS microservice, PRS performs a message correlation with the process instance in question.

Functionality The following listing provides an overview of PRS’s functionalities.

1. Receive information on the state of process instances from MS.

2. Use the information to correlate with the ‘receive task’ of the adaptable processes that are in the second stage of the checkpoint life-cycle.

Listing 11: Functionalities: Process resumption system

5.1.6 Camunda engine

This external service is an important connecting point between the process instances and the rest of the AD system. The execution environment that is established from Camunda, also known as the Camunda engine, holds a multitude of information pertinent to the adaptable process instances that the users get to design, a few examples being the complete running time of a process instance or its activity status. This type of information can be stored in variables and passed along to other process instances or services; the variables in question are called *process variables*. We deem it important that one of the components of the adaptation system is able to provide the functionalities that make it possible to work with the process instances directly as well as have accessibility to their process variables.

In order to exploit the services that Camunda has to offer, we have focused on two methods, its API-s and the `JavaDelegate` classes, also referred to as delegation classes. API requests are the primary form of communication between Camunda and the rest of the system’s microservices. The delegation classes are used to implement the functionalities of the ‘service tasks’ in the process models. We use the delegation class to implement the first stage of the checkpoint state (“checkpoint send task” according to Listing 5) of an adaptable process. By using delegation classes, we are able to use Camunda’s runtime environment to pass along all the relevant information of an adaptable process instance stored as process variables to the “Checkpoint message receiver” process (Figure 11).

5.2 System prerequisites for the end-user

The system description of the Adaptation Dashboard enables us to derive a list of prerequisites that the end-users of the dashboard have to bear in mind before opting to use it. We present the prerequisites in Listing 12.

1. Users have to use Camunda as the execution environment for their process instances.
2. Users have to design their processes as adaptable process models, meaning that every process model must contain at least one checkpoint state, as defined in Listing 5.
3. The first stage of the checkpoint state must be implemented using a `JavaDelegate` class.

Listing 12: Prerequisites for the end-users of the Adaptation Dashboard.

It is important that the end-user of the Adaptation Dashboard (AD) uses Camunda’s engine to run process instances because our system is currently designed to communicate with Camunda’s execution environment. The starting point of the AD system is the External task service (§5.1.2) which is also an implementation of Camunda’s External task pattern, a feature that is specific only to Camunda’s platform. All the other microservices, however, are independent of Camunda. This entails that the AD system could be updated to operate with other process execution environments, provided that a new service that connects adaptable process instances with the AD system be implemented.

5.2.1 Implementation of the checkpoint life-cycle

In §4.1 we elaborate on adaptable processes and why they are crucial to the quest of establishing an adaptation dashboard where end-users can visualize the adaptation process. We present an imple-

mentation for the checkpoint life-cycle (Listing 5) to denote how a process model can be converted into an adaptable one. This, in turn, creates the opportunity for potential adaptation to the process in question, if proven necessary. Our proposed implementation consists of two stages and, a third, optional one. The first two stages represent a send-and-receive communication pattern between the adaptable process and the AD system.

First stage The first stage of the checkpoint state dictates that an adaptable process instance sends a message, possibly with one or more process variables. The receiving end of such a message is the process instance “Checkpoint message receiver” (Figure 11). We contrive this one-way transaction by using a `JavaDelegate` class to implement the send task of the adaptable process. We use this type of class due to its simplicity and efficiency.

Second stage The second stage of the life-cycle is concerned with enabling the running process instance to receive information back from AD. The main use-case for this stage is when AD is used to notify the process instance that it can resume its execution in the normal flow. Nonetheless, it is possible to implement other types of messages that need to be communicated from AD to a running process instance. In our system, we have designed a specific service, Process resumption system, that is used to enact this part of the checkpoint life-cycle.

Optional third stage The optional third stage of the checkpoint life-cycle is designed to give users of the Adaptation Dashboard the opportunity to pass along additional information from AD to the running process instances. In the definition of the life-cycle’s implementation in Listing 5, this stage is defined as a service task. Therefore, users have the freedom to design this task using the framework that would suit their requirements the most, including a `Java Delegate` class or an external service.

5.3 Technology stack

In this section we provide an overview of the technology stack that has been used throughout the development of this software project. We present the stack in Listing F.

- A. Spring Boot framework
Java framework is used for implementing the microservices ¹³.
- B. `JavaDelegate`
Java interface that is used to implement the functionalities of the service tasks in the process models ¹⁴.
- C. Camunda
Software platform that consists of several components, two of which being a process modelling tool and a process execution environment ¹⁵.
- D. MySQL
The software choice for the database system ¹⁶.

¹³<https://spring.io/projects/spring-boot>

¹⁴<https://docs.camunda.org/javadoc/camunda-bpm-platform/7.3/org/camunda/bpm/engine/delegate/JavaDelegate.html>

¹⁵<https://camunda.com>

¹⁶<https://www.mysql.com>

E. RESTful API-s

The architectural style of choice for the implementation of the interfaces of the microservices.

F. React

Front-end library for the implementation of the Adaptation Dashboard ¹⁷.

¹⁷<https://reactjs.org>

6 Discussion

In this chapter, we aim to provide an evaluation of the Adaptation Dashboard (AD), by taking into consideration the following parameters: *(a)* the correctness and efficiency of AD *(b)* the time it takes to set up and prepare AD to get up and running *(c)* the evaluation of the non-functional requirements and *(d)* how can AD work with third-party services. We will discuss each one of the parameters separately through sections §6.1 - §6.4. Finally, in §6.5 we discuss how the Adaptation Dashboard fits into the adaptation pipeline that was introduced in chapter 3, Figure 4.

6.1 Evaluating the Adaptation Dashboard using an adaptable process model

In order to determine the correctness and efficiency of our system, we will make use of a simple process that will be executed using Camunda’s execution environment. We model the process using Camunda’s modeller (Figure 12a). Let us refer to this process as the `Print` process. We note that the process model is quite trivial, in that it consists of three tasks, a gateway and the start- and end-events. In Figure 12b we present how the process model is converted into an adaptable process, by introducing three new tasks that pertain to the three stages of the checkpoint life-cycle (Listing 5).

Having built the adaptable process model, the next step is to set up the delegate classes for the checkpoint stages of the `PRINT` process in §6.1.1. After the process has been fully established, we can commence the initialization of the AD system, including the front-end service of the user interface; we talk about this in §6.1.2. At this point, we can initialize the execution of the process in Camunda’s environment and test out the functionalities of the Adaptation Dashboard; these steps are discussed in §6.1.3 and §6.1.4 respectively.

6.1.1 Setting up the stages of the checkpoint life-cycle

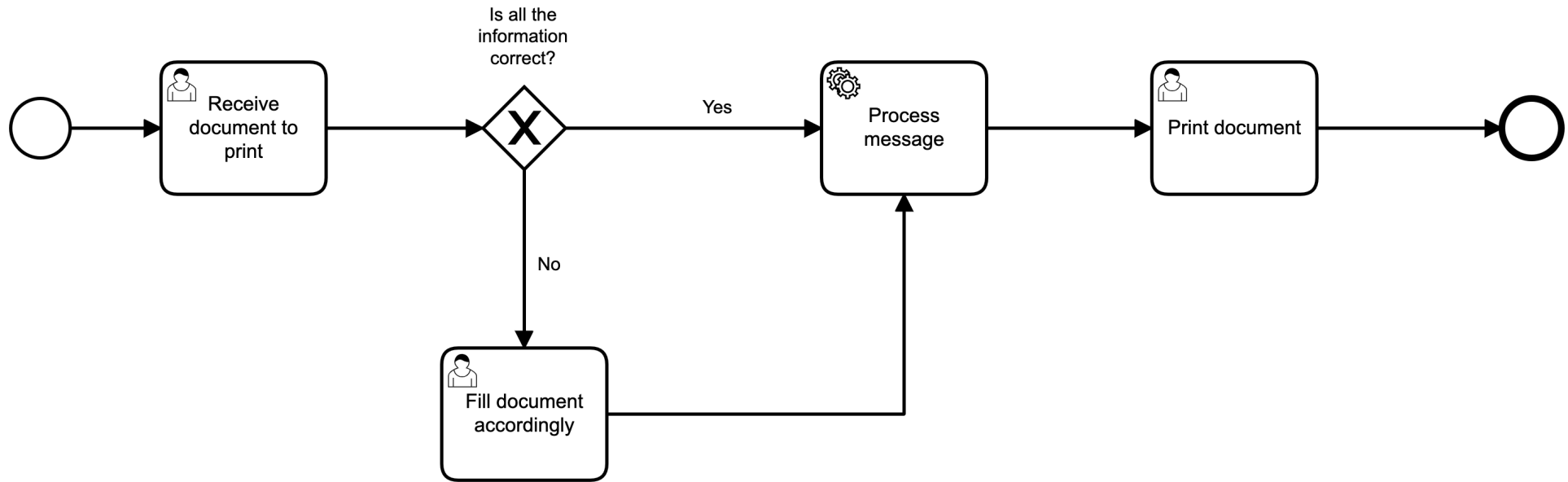
First task/stage: “Checkpoint” An adaptable process such as the `PRINT` process needs to contain at least one checkpoint implementation in its model, meaning that it should contain *at least one sequence of a send task* called ‘Checkpoint’, a *receive task* called ‘Check is over’ and a third, optional, service task called ‘Process message’. The first task, ‘Checkpoint’, is responsible for setting up communication between the `PRINT` process and the “Checkpoint message receiver” process (Figure 11); establishing this connection is important because the later is responsible for passing relevant information about the `PRINT` process to our AD system. The message that is sent from the ‘send task’ in `PRINT` is received in the ‘receive task’ of the “Checkpoint message receiver”.

In order to complete the establishment of the first task, we design a `JAVA` delegate class and copy the path reference of this class to the Camunda modeller, in the *Properties Panel*. More specifically we copy `delegates.CheckpointDelegate` to field name ¹⁸:

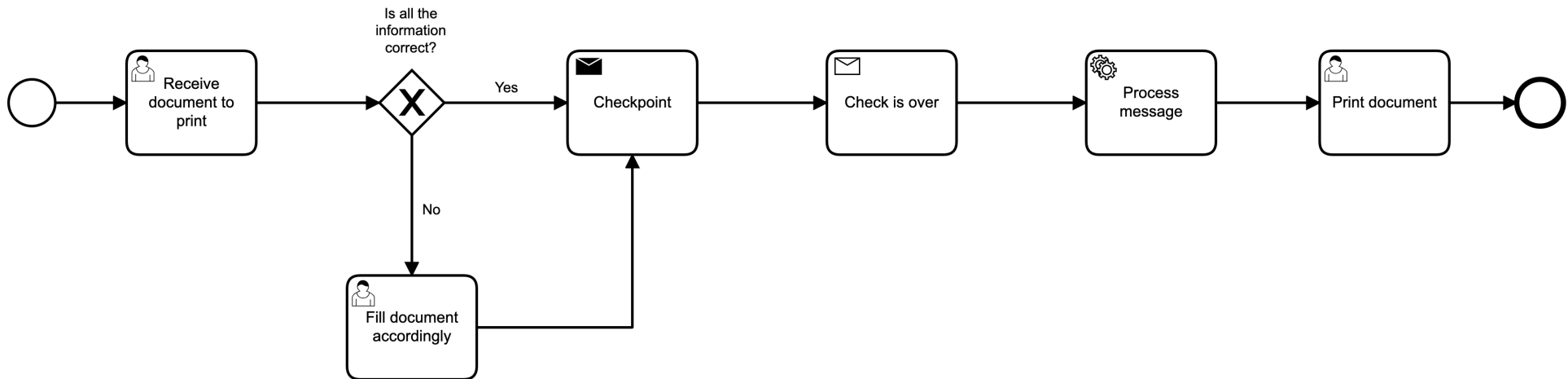
```
Camunda modeller/PRINT process/'Checkpoint' task/Properties Panel/
Details/Java Class
```

The information that we pass along with this message is the *process instance identifier*, the *process definition identifier*, the *process start time* and the *time* when the process reached the first stage of the *checkpoint*. We will need all this information later on.

¹⁸Please note a screenshot of this location in Figure 31 of the Appendix.



(a) BPMN representation of the PRINT process. It consists of three states, one gateway event and the start- and end-events.



(b) BPMN representation of the PRINT process as an adaptable process. We have introduced three new states which coincide with the three stages of the implementation of the checkpoint life-cycle. The states are strategically introduced right before the "Print document" task so as to mirror the fact that end-users may need to double-check the document before printing.

Figure 12: BPMN representations of the PRINT process. On the top we note the representation of the model as a simple process, whereas on the bottom we note the representation of the model as an adaptable process.

Second task/stage: “Check is over” The second task, “Check is over”, is a ‘receive task’. This task is specialized in receiving messages and it, therefore, expects one. In order to complete the definition of this class, we set the name of the message identifier as `state-of-execution` in the field name:

```
Camunda modeller/PRINT process/'Check is over'/Properties Panel
Details/Message Name
```

This task will receive a message under the identifier `state-of-execution` from the Process Resumption system (PRS) (§5.1.5). Once the outgoing message from PRS is matched to this task, the `PRINT` process will resume its execution as normal and it will proceed to the next task, “Print document”.

Third task/stage: “Process message” This task is optional, meaning that end-users of the AD system need not implement it or define it. It is a ‘service task’ that we can use to receive more information from PRS, such as *variables*. The way this task works is by processing the message that is received from the previous stage. For our evaluation, we will provide an implementation for this task such that the `PRINT` process can receive some information from the AD system. We define a new `JAVA` delegate class and copy its path reference to the Camunda modeller in the following field:

```
Camunda modeller/PRINT process/'Process message' task/Properties Panel/
Details/Java class
```

By implementing the three stages of the checkpoint life-cycle, we have set up the `PRINT` process as an adaptable process that can be used with the Adaptation Dashboard. The next step is to boot the services of the AD system.

6.1.2 Initializing the services of the AD system

The system that we have built consists of *five* microservices, each of which runs separately and independently from the other ones. The front-end of the user-interface is a separate entity, which we trigger first. The `MS` service (§5.1.1) is the core of the AD system because it is used for storing the information of process instances, which is displayed on the front page of the Adaptation Dashboard. As such, `MS` is the first microservice that we initialize.

The `ETS` system (§5.1.2), on the other hand, serves as the link between Camunda’s environment and the AD system. `ETS` consists of two entities: the “Checkpoint message receiver” process model (Figure 11) and the `JAVA` application that implements the External Task pattern for the receive-task in the process model. We initialize the application first and then the execution of the process model.

Lastly, we can initialize the rest of the microservices. There is no hierarchy amongst them as their functionalities are triggered by the user’s input through the user-interface. As such, the order in which they are initialized is not important. At this point, we are ready to initialize the `PRINT` process.

6.1.3 Initializing process execution in Camunda

In order to use Camunda, we make sure that we are already running its server in our work-space environment. There are several ways in which a process instance can be executed in Camunda. Before a process can be executed, it must first be deployed into Camunda’s servers. In our application it is possible to do so either by using Camunda’s modeller and triggering the model’s deployment or by

using Camunda’s API. The latter is done by performing a GET request to the following line (assuming that Camunda’s servers are running on localhost:8080):¹⁹

```
http://localhost:8080/engine-rest/deployment/create
```

After a successful deployment, process models are presented on Camunda’s Cockpit interface. From there, we can trigger the execution of the process. We opt to do this using Camunda’s API, by performing the following GET request (where key stands for the process definition keyword that can be set using Camunda’s modeller):

```
http://localhost:8080/engine-rest/process-definition/key/process-two/start
```

Having initialized the execution of the PRINT process instance, we can now begin to use the Adaptation Dashboard and evaluate its functionalities.

6.1.4 Using the Adaptation Dashboard

Once the PRINT process is initialized, we let it execute as it is supposed to, without having to interfere. The first task to be executed is the manual task “*Receive document to print*”. If the evaluation of the gateway “*Is all the information correct?*” evaluates to false, the next task to be executed is the manual task “*Fill document accordingly*”, otherwise the send task “*Checkpoint*” is the one to be executed. For the purpose of this discussion, we opt to evaluate the gateway to *true*. The execution of the “*Checkpoint*” task sends a trigger to our system; the information with the process instance is now persisted in the system and we can note it on the home page of the Adaptation Dashboard (Figure 13). Concurrently to working with the dashboard, we also navigate to Camunda’s cockpit where we can observe the execution of the process in real-time (Figure 14)²⁰. We note that the PRINT process is currently in the “*Check is over*” task and the “*Checkpoint message receiver*” is back to its initial task, waiting for new incoming messages from other process instances.

[Load process instances](#)

List of process instances that are in a checkpoint.

Process instance 243: PRINT

[2e9de99f-2da4-11ec-8ead-fa0b86accac1](#)

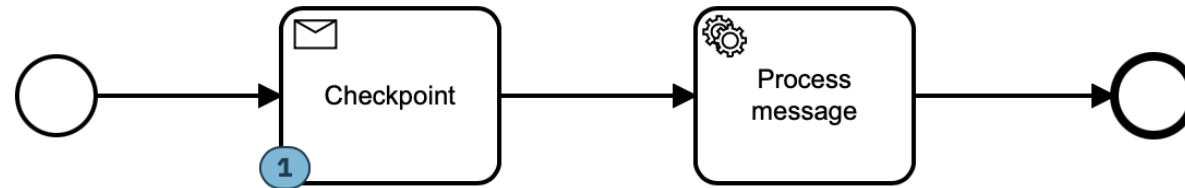
- start time: Fri Oct 15 12:39:32 CEST 2021

[Resume process instance](#)

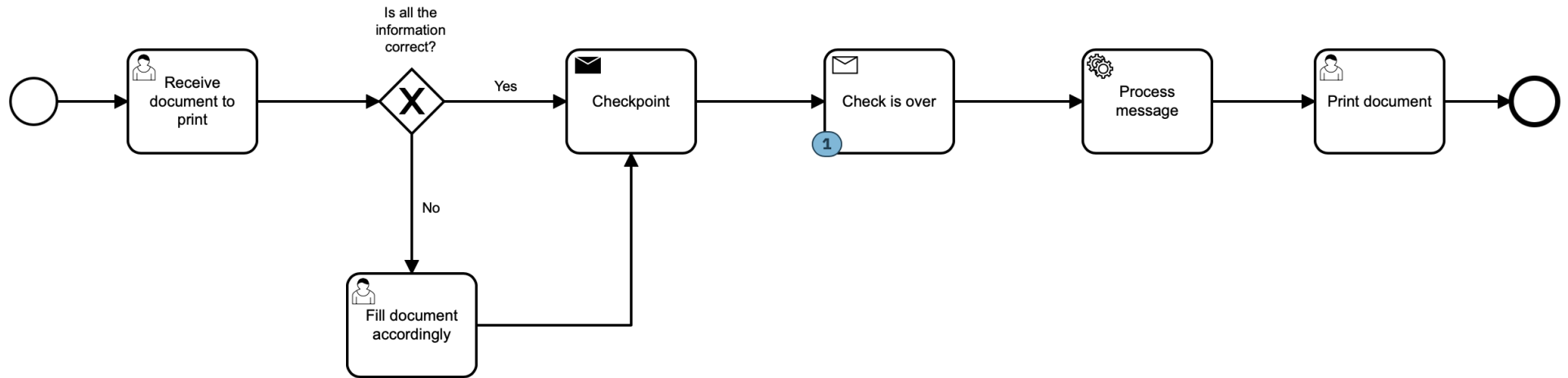
Figure 13: The PRINT process on the home-page of the Adaptation Dashboard; we take note of the start time and the link with Camunda’s identifier of the process instance. By clicking on the identifier’s link, we can navigate to the detail page of the process.

¹⁹More information on the API requests that are used in this project can be found on §C of the Appendix.

²⁰Camunda’s cockpit can be located on <http://localhost:8080/camunda/app/cockpit/default/#/dashboard?searchQuery=%5B%5D>, assuming that Camunda’s server is running on the localhost:8080 of the workspace.



(a) The “Checkpoint message receiver” (CMR) process as seen in Camunda’s cockpit. Once the PRINT process is in the first phase of the checkpoint life-cycle, CMR loops back to its first task, “Checkpoint” so as to receive new messages from other process adaptable instances. All adaptable process instances can correlate this CMR by setting the identifier of the message to “checkpoint-message”.



(b) The PRINT process in Camunda’s cockpit, after the execution of the first stage of the checkpoint life-cycle. During this time, the user gets to work with the process instance in the Adaptation Dashboard.

Figure 14: Real-time view of our two working processes in Camunda’s cockpit after the execution of the first stage of the checkpoint life-cycle.

At this point, we can opt to take note of the existing KPI definitions in the system, namely we navigate to the second tab of the dashboard, *KPI definition*. In Figure 27 of the Appendix we note that there exist four KPI definitions for the running time of the process. We need to be aware of this information for when we trigger violation checks on the process instances.

Having looked into the KPI-s, we now go back to the dashboard, such that we can move further into the detail page of the process instance in question (Figure 16). In the detail page, we firstly note the input form “*Select type of KPI:*”²¹ where we can select whichever one of the existing KPI-s for which to run a violation-check. We opt for the first option, *AVG running time*, and click on *Check for violations*; a modal pop-up is triggered, where we are presented with the results of the check. In Figure 15 we note that the results for the `PRINT` process indicate that no violation of the KPI in question can be traced in the `PRINT` process. This means that the running time of `PRINT` is lower than the running time of processes that have run previously in the Adaptation Dashboard.

Next up, on the detail page, we note the BPMN notation editor where the current model of the process instance is displayed. We opt to edit the notation and save it in the system, by clicking on *Save new diagram*. In Figure 17 we present a side-by-side comparison of the model notation that we created in the editor and the persisted `xml` file when displayed in Camunda modeller. With this comparison we would like to demonstrate the accuracy of the editor of the system and its persistence functionality.

Finally, we can trigger the resumption of the `PRINT` process by going back to the Adaptation Dashboard and clicking on “*Resume process instance*” (Figure 13). We navigate back to Camunda’s cockpit where we note that `PRINT` has resumed its execution and is currently executing manual task “*Print document*” (Figure 18). After said task has finished execution, the process instance terminates successfully. In the *History* tab of the Adaptation Dashboard we present an overview of all the adaptable process instances that have terminated their execution in Camunda²². Once we navigate back to the dashboard (and click on “*Get most recent history*” to make sure that the system is most up-to-date), we take note of the `PRINT` process and its properties (Figure 19).

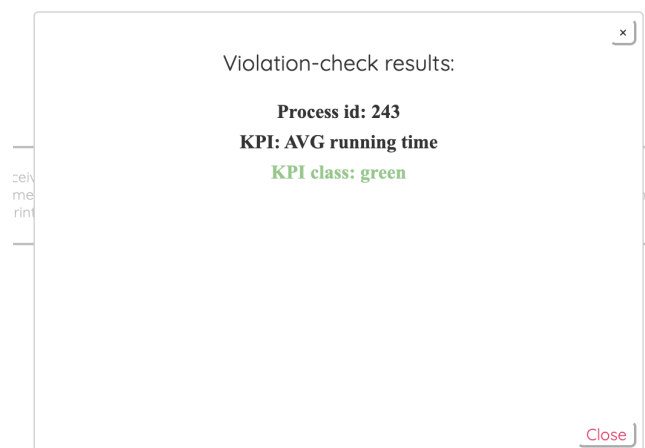


Figure 15: The results of calculating KPI violation on the `PRINT` process,

²¹Figure 25 of the Appendix.

²²More screenshots of the History tab can be found in the Appendix.

Process instance PRINT: 2e9de99f-2da4-11ec-8ead-fa0b86accae1

Select type of KPI: [Check for violations](#)

[Save new diagram](#)

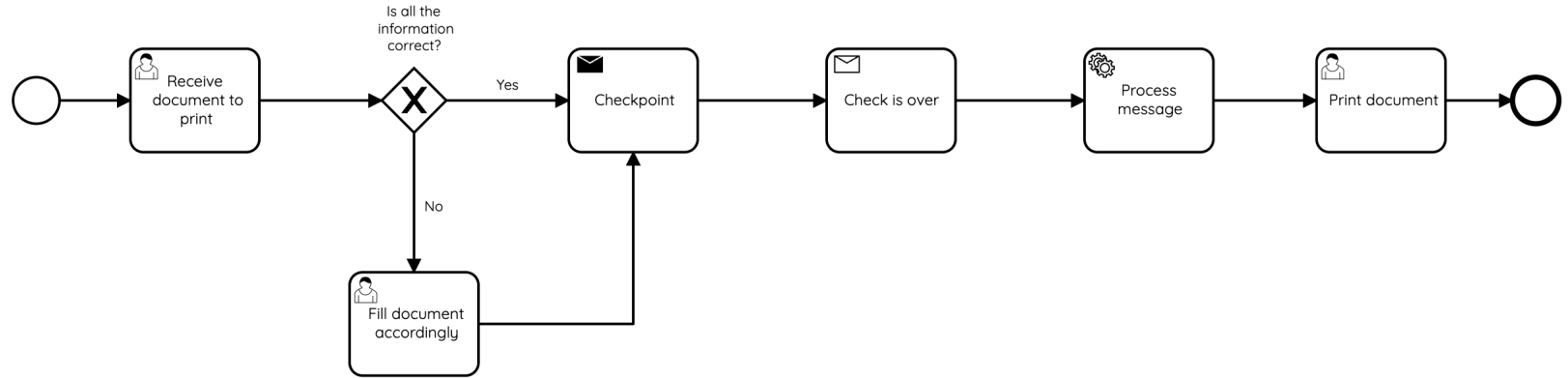
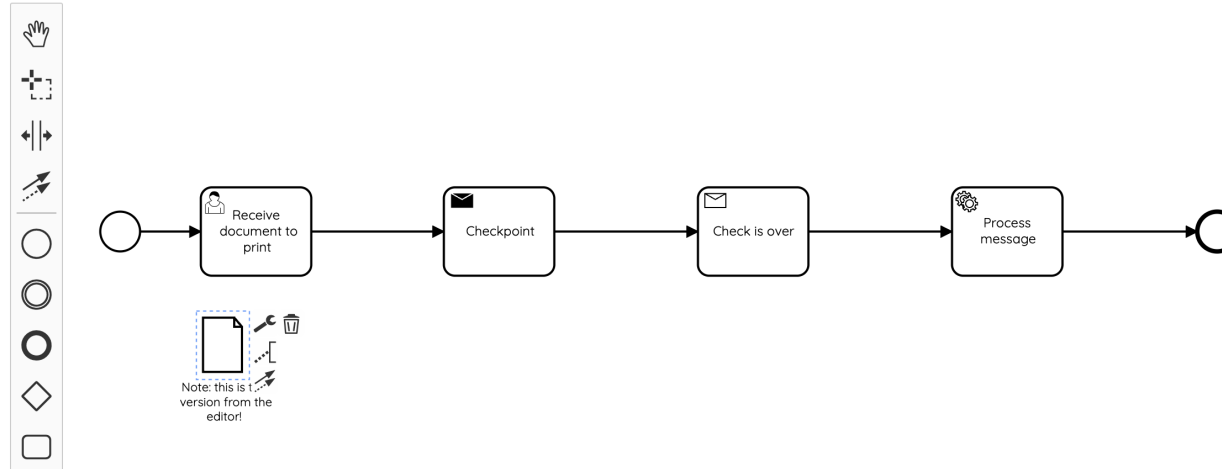


Figure 16: Detail page of the PRINT process in the Adaptation Dashboard.

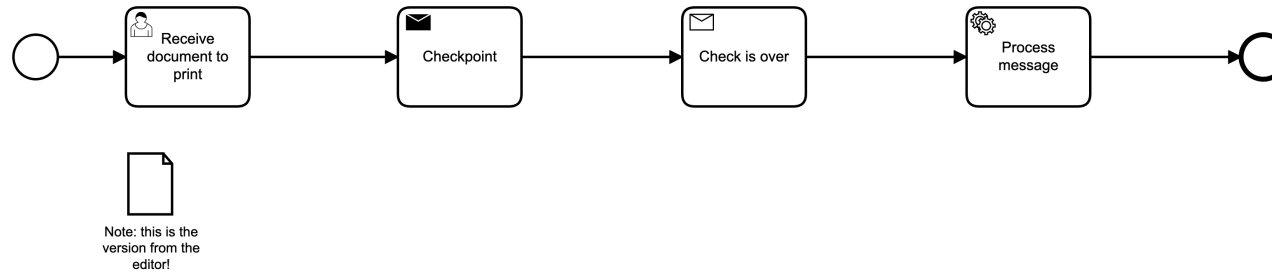
Process instance PRINT: 2e9de99f-2da4-11ec-8ead-fa0b86accae1

Select type of KPI: [Check for violations](#)

[Save new diagram](#)



(a) BPMN notation of the process model that we created in the editor of the Adaptation Dashboard for PRINT. Once we clicked on “Save new diagram”, the xml-definition of the model is persisted in the system.



(b) Depicting the BPMN notation of the new xml-definition for PRINT that is persisted on our system in Camunda’s modeller.

Figure 17: Side-by-side representation of the two BPMN notations for the process model that we create in the Adaptation Dashboard. The top one is the notation that we see when we get to use the editor of the dashboard. The bottom one is the notation we get to see when we paste the xml-definition of the model from the database of our system.

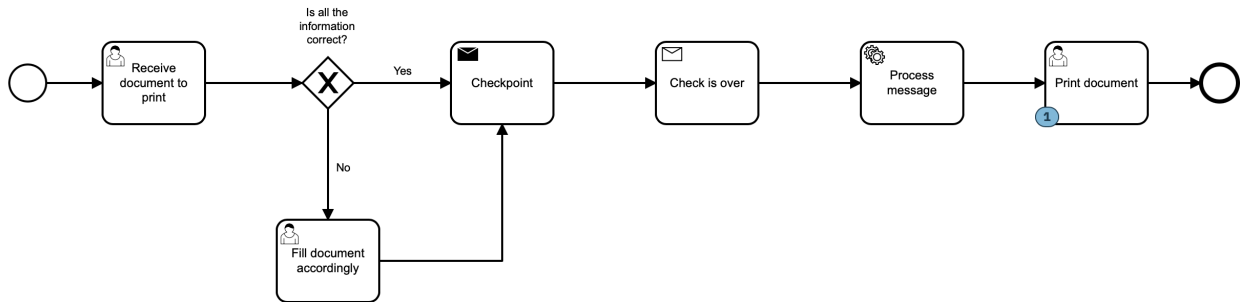


Figure 18: Real-time presentation of *PRINT* in Camunda's cockpit after we trigger 'Resume process instance'. At this point, the third stage of the checkpoint's life-cycle has been completed successfully.

Process instance: 243

Definition key: PRINT

End time: 2021-10-15T15:42:42.866+0200

Running time: 3 hours, 3 minutes, 10 seconds

Process instance id: 243

Figure 19: After *PRINT* has completed its execution successfully, the system is completed with information about its end- and running- times.

In this section of the chapter we wanted to present a how-to manual of using the Adaptation Dashboard with processes that are modelled and executed in Camunda. In doing so, we were able to depict the main functionalities of the dashboard, how it can be used as a tool to work with adaptable processes and discuss the efficacy of the system.

6.2 AD and third-party services

The Adaptation Dashboard has been created on the basis of a microservices architecture, wherein several different services, in establishing communication with one another, are able to generate a specific end-goal based on their defined functionalities. A detailed overview of the architecture is presented on §4. Each one the services of the system is equipped with a REST API. As a consequence, setting up connection with either one of the services or the system as a whole is fairly straightforward. In this section, we provide a list of the existing API methods of the Adaptation Dashboard's services.

Mapping server

1. Get a list of all the available adaptable process instances.
Method GET
`/all_process_instances`
2. Get a list of all the *active* adaptable process instances.
Method GET
`/process_instances`

3. Get a list of all the inactive process instances (i.e. process instances whose execution has terminated).
Method GET:
`/process_instances_history`
4. Delete the current history from the system.
Method GET:
`/clear_process_instances`
5. Save a new process instance in the system.
Method POST:
`/process_instances`
6. Retrieve a specific process instance from the system.
Method GET:
`/process_instances/{processInstanceId}`
7. Resume the execution of an active process instance.
Method PUT:
`/process_instances/{processInstanceId}`
8. Retrieve the xml representation of the definition of a specific process.
Method GET:
`/process_instances/{processId}/xml`
9. Retrieve the running time of a process instance in milliseconds.
Method GET:
`/process_instances/{processId}/running_time`
10. Save a new process definition (in xml) for a running process instance.
Method PUT:
`/process_instances/{process_id}/new_xml_definition`

Listing 13: List of the existing API methods for Mapping server.

KPI system

1. Create a new KPI.
Method POST:
`/kpis`
2. Retrieve all available KPI-s from the system.
Method GET:
`/kpis`
3. Retrieve specific KPI from the system.
Method GET:
`/kpis/{kpiId}`

Listing 14: List of existing API methods for KPI system.

Violation-check system

1. Retrieve information about all available violation-checks.
Method GET:
`/violations`

2. Retrieve information about specific violation-check.

Method GET:

```
/violations/{violationId}
```

3. Create a new violation.

Method POST:

```
violations
```

Listing 15: List of existing API methods for Violation-check system.

Process resumption system

1. Receive a message about the state of a running process instance.

Method POST:

```
/send_to_process_instance
```

Listing 16: List of existing API methods for Process resumption system.

6.3 Setting up AD

Setting up the Adaptation Dashboard consists of starting up the entirety of its microservices and the SQL servers. A `dockerfile`²³ is used to establish the initialization of said services. We could then use said file to initialize all the services simultaneously; using a `dockerfile` is beneficial because it ensures that the system and its functionalities will behave as expected, despite the type of operating system, virtual machine, cloud work-space.

6.4 Evaluation of the non-functional requirements

In this section of the chapter, we look into how the design and architecture of Adaptation Dashboard (AD) fit the requirements that were defined in Listing 3 in our analysis chapter (§3). Let us take a look at each one of non-functional requirements separately and evaluate their fulfillment.

i. Extensibility

By designing a microservices-based architecture for AD, we are able to easily introduce and integrate new services to plug into the complete framework of the system's architecture. The only requirement for a new microservice would be the implementation of a RESTful API such that it can easily interact with the rest of the existing services.

ii. Inter-operability

All the microservices of the system are RESTful and are equipped with a specific API which can be used to retrieve relevant information. As a result, establishing communication with AD is fairly straightforward, provided that the user is knowledgeable of the available requests.

iii. Scalability

The concept of scalability for our system is primarily focused on extending AD such that it

²³The official website for Docker can be found here: <https://www.docker.com>.

can support more adaptation triggers and types of performance metrics. Having a modular architecture (due to the microservices-based design) implies that new services can be easily introduced without hurdling the load of the system and downgrading AD's performance.

iv. Maintainability

Given that AD has a fairly modular architecture, the difficulty levels for its maintainability are quite low, thus rendering AD as a fairly maintainable system.

v. Usability

The design of the Adaptation Dashboard is built to be simple yet effective. It is a single-page application with several endpoints, which represent the set of resources that are available to the system. Our goal has been to retain a minimalist overview of the adaptable processes while also providing the user with the most relevant information and functionalities in a compact way. In §A of the Appendix, we present images of the different pages of the Adaptation Dashboard.

Listing 17: Fulfillment of non-functional requirements

While we provide reasoning about the fulfilment of the system's non-functional requirements, we wish to emphasize that it does not constitute of a complete proof. There exist various methods of evaluation analysis which can be used to showcase the fulfilment of a software architecture's requirements, with ATAM²⁴ and SAAM²⁵ being two of them.

6.5 Process adaptation pipeline

In §3.1, Figure 4 we presented a high-level representation of the process adaptation pipeline. We divide the pipeline in two stages; **Stage I** is dedicated to processes which are 'candidates' for adaptation, that is, adaptable processes which are in the first stage of the checkpoint life-cycle (Listing 5). The goal of this stage is to determine whether these candidate processes require adaptation. At the end of the first stage of the pipeline, we can note which process instances should possibly be adapted and which ones can resume their execution safely. **Stage II** is dedicated to executing the appropriate adaptation strategy(ies) to all the process instances from Stage I that do require adaptation. In this section, we discuss how our system, the Adaptation Dashboard, is an implementation of Stage I.

We interpret Stage I of the pipeline by dividing it into two *procedures*. In the first procedure, the input (which consists of adaptable process instances) is prepared such that it can be depicted in the user-interface of the Adaptation Dashboard. The second procedure is, therefore, focused on determining whether the process instances that are present in the dashboard require adaptation. Figures 20 and 21 present the BPMN representations of the first and second *procedures* respectively. We discuss each one of the procedures individually.

Adaptation pipeline, Stage I, procedure 1: Present information about candidate adaptable processes

The BPMN in Figure 20 contains *five* different pools: P1, Checkpoint message receiver, External task service, Mapping server and Adaptation Dashboard. Let P1 represent PRINT, the running instance of

²⁴Architecture trade-off analysis method

²⁵Software architecture analysis method

the adaptable process from Figure 12b. The “Checkpoint message receiver” is the running instance of the process model that was introduced in Figure 11. The External task service and the Mapping server are two of the microservices of our system (more information on §5.1.2 and §5.1.1 respectively). The Adaptation Dashboard represents the user-interface that the end-user interacts with (an example model of the UI can be found in Figure 22). The procedure modelled in Figure 20 is a high-level representation of the steps that were discussed in §6.1.

Given this high-level outlook, we can begin to understand the integration of the microservices and how they execute their functionalities into the establishment of the Adaptation Dashboard’s requirements. The starting point of this procedure is the running instance of an adaptable process model, such as `PRINT`. By the end of said procedure, the information about the state of the process instance is made available to the Adaptation Dashboard system and can be used to calculate whether the process requires adaptation or not.

Adaptation pipeline, Stage I, procedure 2: Inspect whether a process instance requires adaptation The second procedure of Stage I in the adaptation pipeline is to render a conclusion about the adaptable process instance (current example being `PRINT`) on whether the process requires adaptation or not. In our system, the starting point of this procedure is when the end-user triggers the calculation of a violation-check for a particular available KPI and a particular process instance through the user-interface (depiction of this trigger can be seen on Figure 25). As of that moment, the system begins calculations based on the logic that is defined on the VCS and KPI systems (more information on §5.1.4 and §5.1.3 respectively). In Figure 21 we present the BPMN notation of this procedure. By the end of the procedure we note that the end-user is informed about the status of process `PRINT` and whether it needs to be adapted accordingly.

By depicting the two procedures in Figures 20 and 21 we have showcased how the Adaptation Dashboard serves as an implementation of the adaptation pipeline’s Stage I, wherein given an adaptable process instance, we have the opportunity to see whether said process needs to be adapted. The parameters of the algorithm that inspects if a process requires adaptation are dependent on various factors, including the performance metrics of choice and their definitions in the system. The current version of the Adaptation Dashboard uses KPI-s as performance metrics.

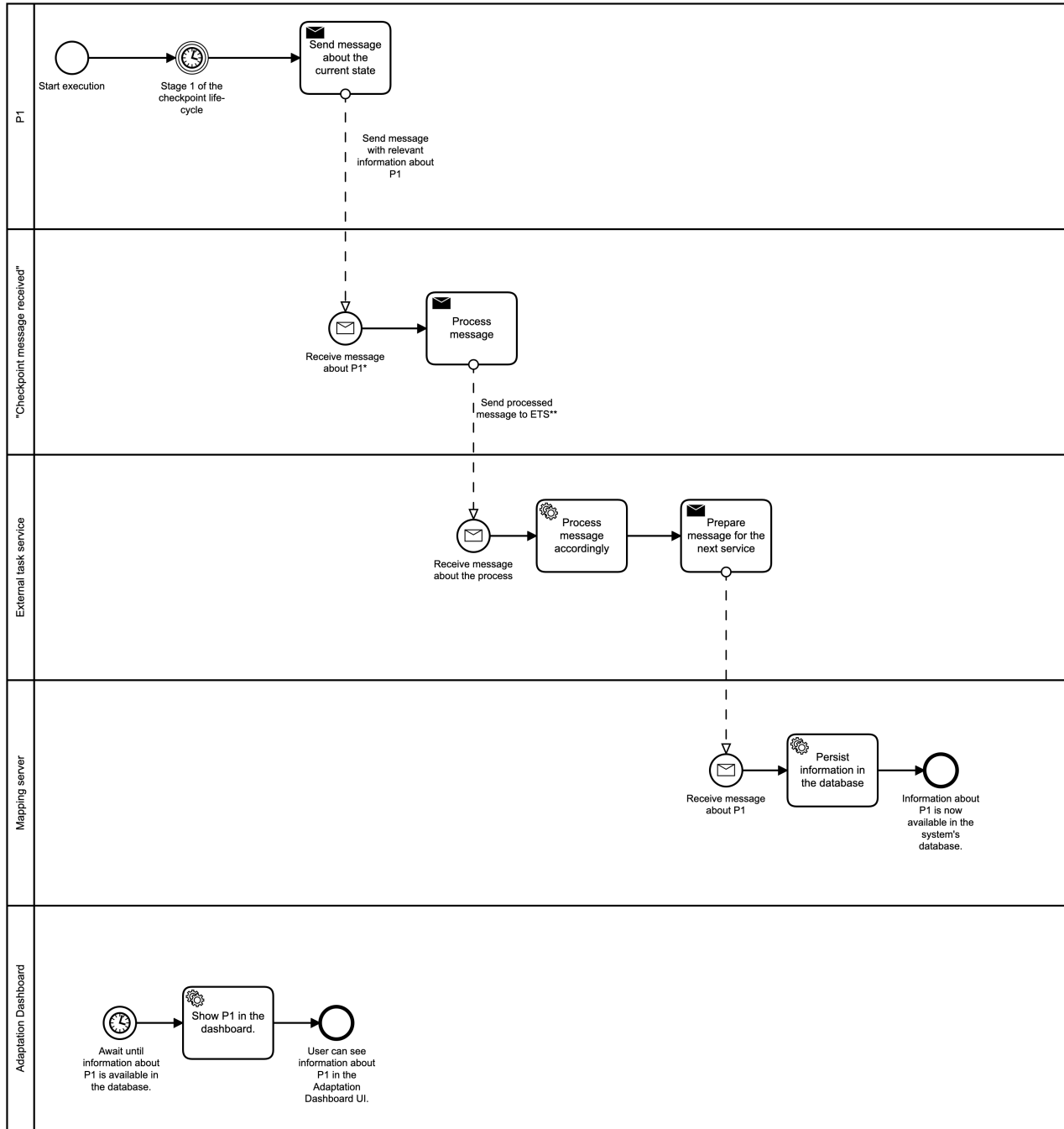


Figure 20: This is a BPMN representation of the procedure wherein an adaptable process instance is depicted in the Adaptation Dashboard.

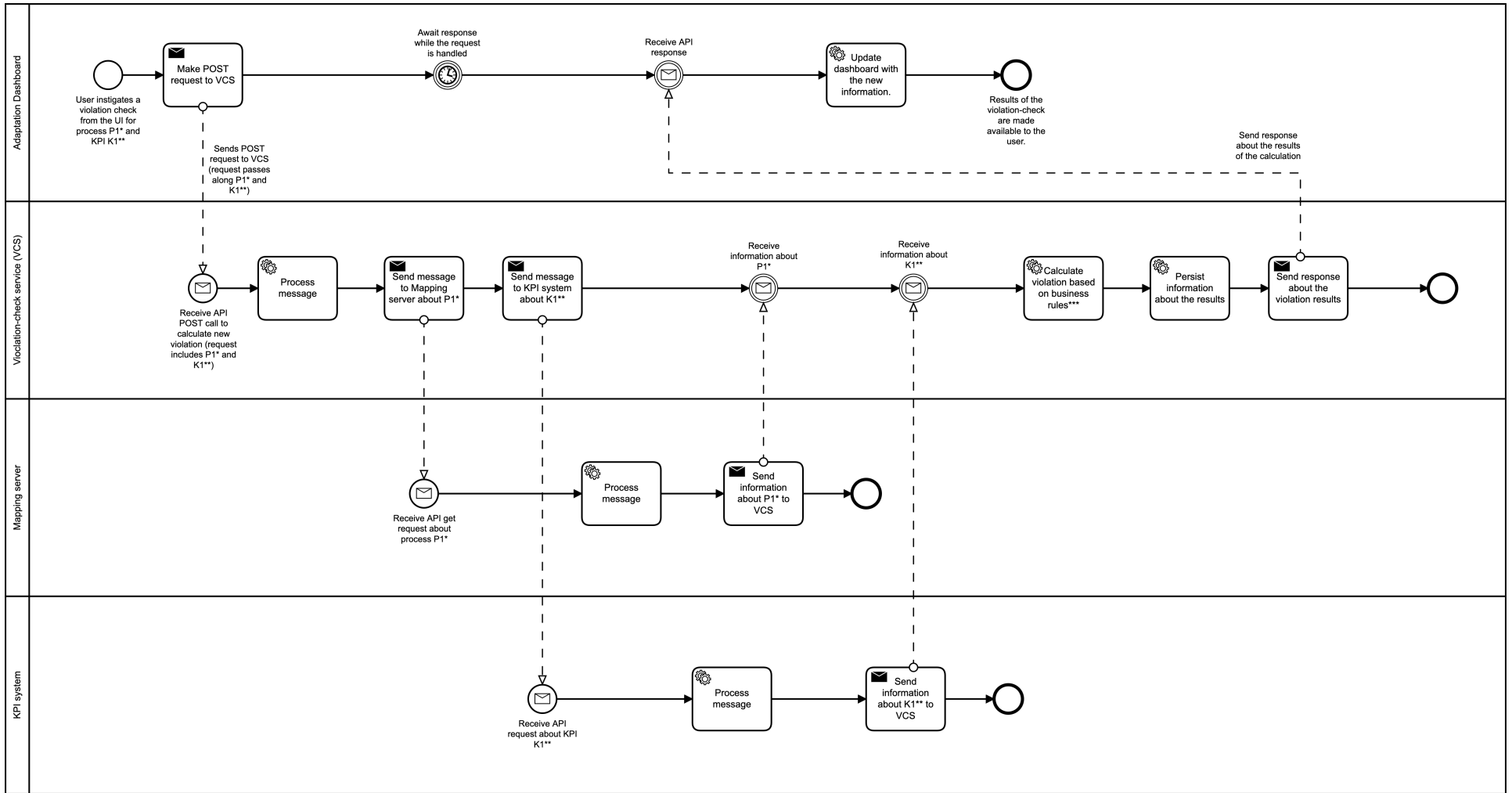


Figure 21: This is a BPMN representation of the procedure wherein the system investigates whether an adaptable process instance requires adaptation.

7 Conclusions

In the final chapter of this thesis, we aim to provide a summary of the work that has been conducted throughout this research study. The main theme of the thesis has been business processes and their execution, while its objective was to autonomize the adaptation of a process instance. More specifically, we have been seeking to answer our research question: “How can we construct an autonomous adaptation system that is able to tackle the challenges of adapting a process instance?”. In our quest of answering this question, we designed a high-level overview of an adaptation pipeline, presented in Figure 4. Based on this overview, we have constructed the first version of the *Adaptation Dashboard*, a microservices-based software tool that allows end-users to visualize the procedure of adapting a process instance running in Camunda. In section 7.1 we provide an extensive outlook on the key contributions, where we discuss thoroughly the existing functionalities of the Adaptation Dashboard.

In chapter 2 of this thesis we have presented the literature review based on the key research papers that have been most relevant for this study; chapter 3 presents the results of our problem analysis which were used in establishing the (functional and non-functional) requirements of the Adaptation Dashboard. In chapter 4 we present an overview of the system’s microservices architecture. In chapter 5 we delve into detailed introductions of each one of the services in the system and its core functionalities. In chapter 6 we discuss how the implementation of the Adaptation Dashboard fits into the adaptation pipeline and provide an extensive evaluation of the system.

In this chapter, we will firstly present and discuss the contributions of our thesis, followed by a compact summary of the main takeaways from our study and system implementation. We then talk about the limitations of the initial version of the adaptation dashboard - matters such as what it can and cannot do. We conclude the chapter by discussing the future possible steps for the development and extension of the Adaptation Dashboard.

7.1 Thesis contributions

In this section, we present an overview of the contributions that our research and study has put forward and made possible. The list of contributions can be noted in Listing 18. Throughout the rest of this section we will dive further into each one of the contributions.

1. Construct version 1 of the Adaptation Dashboard.
2. Propose an implementation of the life-cycle for the checkpoint state.
3. Introduce a blueprint that developers can use for new types of violations that can be used with this system, and a separate blueprint for new types of performance metrics.
4. Provide an implementation for Stage I of the proposed adaptation pipeline.

Listing 18: List of contributions

7.1.1 Construction of the Adaptation Dashboard

The key contribution of our work is the development of a dashboard that can be used to visualize the steps that *adaptable* processes take from the moment that they become candidates for process adaptation until their eventual resumption in Camunda’s execution environment. More specifically, the dashboard will depict all the adaptable processes that are in the first stage of the checkpoint’s

life-cycle, the “*checkpoint send task*” (Listing 5). The interface of the Adaptation Dashboard gives the end-user the opportunity to do the following: ²⁶

- F1*: See a list of all the adaptable processes that have reached the first stage of the checkpoint’s life-cycle.
- F2*: Manually resume adaptable processes (i.e. can trigger a process instance instance to resume its normal course in the Camunda execution environment).
- F3*: The dashboard is equipped with a BPMN editor for each one of the active process instances where the user can view the current notational representation of the process model.
- F4*: The user can update the BPMN representation of the process model in the dashboard’s editor and opt to store it in the system.
- F5*: There are two default KPI-s in the system, one that compares the runtime of an active process to the minimum existing runtime in the system, and one that compares it to the average runtime of previous processes in the system.
- F6*: The user can define new KPI-s that are specific to the process running time. They can set a threshold (in milliseconds) of their choice and define the KPI classes on the basis of said threshold.
- F7*: The user can trigger calculations of violation checks for each one of the available KPI-s.
- F8*: The user can see a history of the process instances that have ran in the past (in the adaptation dashboard) and are no longer active in the Camunda execution environment.
- F9*: When the user opts for the option “*Get most recent history*” in the dashboard, it will trigger an examination of all the previous active process instances; should they have terminated their execution in the Camunda environment, the history will be updated accordingly.
- F10*: The user can see a history of the violations that have been calculated in the past and their individual outcome.

Listing 19: List of functionalities for of the Adaptation Dashboard V1.

The code-base of the Adaptation Dashboard can be retrieved from the following GitHub repository: <https://github.com/evixhelo/master-thesis-projects>.

7.1.2 Implementation of the life-cycle for the checkpoint state

In constructing the Adaptation Dashboard we have designed and developed an implementation for the checkpoints of adaptable processes. The idea of introducing checkpoints in process models has been put forward in previous research studies like [Wetzstein et al., 2012]. Our implementation was influenced by two key aspects: using Camunda as the execution environment for the process instances and enabling communication between said execution environment and our software system. As a result, our implementation consists of two stages (and an optional one), which we presented in Listing 5.

The first stage, “checkpoint send task”, is concerned with establishing the communication between Camunda and our microservices architecture. In order to achieve this, we designed the “External task system” (ETS). In §5.1.2 we provide specific information about the ETS system and how it enables said communication. This first stage is the starting point of the process adaptation pipeline where an adaptable process is rendered a candidate for process adaptation.

²⁶Please note that a complete overview of the dashboard can be noted in the Appendix (§7.4).

The second stage, “message receive task”, is concerned with successfully establishing the connection between our system and Camunda’s execution environment. This stage is crucial in allowing end-users to manually resume a process instance in the dashboard. In §5.1.5 and paragraph 4.2.2 we talk about the “Process Resumption system (PRS)” and how the communication is established respectively.

The third and final stage of the checkpoint’s life-cycle is optional; we call it the “message processing service task”. The idea of this stage is to allow developers to pass along information from our system to the execution environment in Camunda. Such information can possibly be relevant for the rest of the process instance’s execution. Possible scenarios could vary from trivial ones such as comparing the running time of the process to a specific threshold or updating the values of existing process variables to more complicated ones such as skipping certain tasks based on their identifier and more.

7.1.3 Implementation of frameworks for future types of performance metrics and adaptation triggers

Before there can be process adaptation, there needs to be a cause that triggers it. In our literature review we presented the concept of adaptation triggers and a diagram with an extensive overview of possible mappings between such triggers and their possible causes (Figure 3). In this first version of the Adaptation Dashboard, we take into consideration one type of cause and one type of adaptation trigger. More specifically, we work with KPI-s that function on the basis of process run time. As such, the adaptation trigger would be the violation of these KPI-s. The KPI system and the “Violation-check system (VCS)” are used to mirror the behaviours of performance metrics and adaptation triggers respectively; specific information about their functionalities can be found on §5.1.4 and §5.1.3.

It is possible to introduce new performance metrics as well as new types of KPI-s that can be used in the Adaptation Dashboard. The introduction of new performance metric gives rise to the possibility of introducing new types of adaptation triggers as well. The existing services can serve as a framework for developing new services that can be easily ‘plugged’ into the current microservices architecture.

7.1.4 Implementation of Stage I of the adaptation pipeline

When conducting our own analysis of constructing a BPM system that supports process adaptation, we presented a high-level overview of the pipeline for process adaptation, in Figure 4, which we have split into two stages. In this thesis, we have provided the architecture and design of a system that implements the first stage of the pipeline in question. The Adaptation Dashboard can be effectively used as a tool to help determine if adaptable process instances require adaptation, on the basis of pre-defined sets of rules (which are implemented as part of the system itself). In §6.5 we presented a detailed overview of how the Adaptation Dashboard is ‘mapped’ into Stage I of the pipeline.

7.2 Main takeaways of the thesis

Having introduced the contributions that this thesis has put forward, in this section, we would like to present a brief summary of the conclusions that we have reached from our research study and implementation work. We first present a list of the conclusions (Listing 20) and then provide some commentary on them.

1. The Adaptation Dashboard system can serve as the starting point of the adaptation phase in any BPM system.
2. This system fulfills the non-functional requirements that were introduced in Listing 3²⁷.
3. The Adaptation Dashboard is suitable only for adaptable processes.
4. An adaptable process is any process model that contains at least one *checkpoint*.
5. Through creating the Adaptation Dashboard, we have proposed a life-cycle for the checkpoint state and a possible implementation for the life-cycle in question.
6. In building this system, we have created the “Checkpoint message receiver” (Figure 11), a process model that establishes the connection between the Adaptation Dashboard and the process instances that run in Camunda.
7. Some of the microservices of the system can serve as a blueprint for new types of metrics and adaptation triggers.

Listing 20: Main takeaways of the thesis

Conclusion 1

The end-result of this thesis is a microservices-based system that we refer to as the ‘Adaptation Dashboard’ (AD) system. This system can serve as the starting point of any BPMS where the process adaptation is incorporated in the life-cycle of said BPMS. This is because the system is designed to retrieve information about adaptable processes and calculate whether they would require adaptation based on the set of services, functionalities and algorithms that have been defined in the system itself. The current version of AD has incorporated one type of performance metric (KPI-s that are specifically focused on the running time of the process instances) and one type of adaptation trigger (violations of the KPI). Additionally, the dashboard provides users with the possibility to manually resume the execution of a process instance once it has arrived in the checkpoint.

Conclusion 3

AD is designed to work with adaptable processes because in this way, we provide a BPMS with a strategic gap where the adaptation can be applied. From the technical point of view, setting a prerequisite for the types of processes that can use the AD system offers a better end-user and software developer experience.

Conclusion 4

We define adaptable processes as process models that contain at least one *checkpoint*. We propose an implementation of the life-cycle of a checkpoint state in Listing 5. The state is designed to represent a send-and-receive type of communication, which is analogous to the two-way communication between the adaptable process instances and AD. The checkpoint state also includes a third, optional, stage called “message processing service task”. We have designed this state so as to give developers the chance to be able to pass along extra information from AD to the process instances.

Conclusion 6

In order to devise a connection between the running process instances and AD, we have established a process model called “Checkpoint message receiver” (Figure 11), which will be constantly in an active state, ready to receive messages from adaptable process instances that are in the first stage of their checkpoint state. The information that is received is then passed along to AD’s other microservices (Mapping server, more specifically). It is important to note that the current version of this process

²⁷In chapter 6 we evaluate how the Adaptation Dashboard has fulfilled the requirements that we defined in chapter 3.

model can only await for one message at a time.

Conclusion 7

Two of AD's microservices - the KPI and Violation-check systems - serve as a blueprint for other types of performance metrics (SVN values, SLA compliances and more) and adaptation triggers respectively. We can implement any new services accordingly and ascertain that they also have a RESTful API such that they can be easily integrated into AD.

7.3 Limitations of our work

Having presented the list of contributions and the main takeaways of this thesis, we now discuss the limitations of the current research, which could be used as starting point(s) in future research work for process adaptation and upcoming versions of the Adaptation Dashboard. In this initial version of the dashboard, it is not possible to adapt any process instances. While the system can persist the new changes in the BPMN representation that the end-user generates via the dashboard's editor, no service has been implemented to trigger the execution of the BPMN in question.

Another limitation of the dashboard concerns the mapping possibilities between the performance metrics and the adaptation triggers, which, as we have discussed in §2, are crucial in deducing a process' need for adaptation. In the first version of our system, there exists only one possible mapping of the type; the performance metric is a KPI metric designated to check the running time of a process instance and the adaptation trigger is the violations of the KPI, as defined in the mapping function of the KPI in question.

Lastly, it is important to mention that in the initial release of the Adaptation Dashboard, Camunda's environment is the only third-party service that has been established in the structure of the system. Nonetheless, integrating with any of the compound services is rather straight-forward as discussed in §6.2, wherein we introduced the list of existing API methods of the Adaptation Dashboard.

7.4 Future work

This system is the initial version of an adaptation dashboard that is able to depict information about adaptable process instances and perform violation checks for the available KPI-s. In this section, we have compiled a list of possible updates that can be developed as part of the future development of the Adaptation Dashboard; we discuss every point separately.

Implement stage II of the adaptation pipeline An important step for the future of this system would be the implementation of stage II of the adaptation pipeline (Figure 4). Given that the second stage is concerned with the application of an appropriate adaptation strategy for a given process instance, there are two key concerns that need to be thought of; firstly, how can we design a system where we can define adaptation strategies and effectively select the most appropriate (set of) strategy(ies)? And secondly, how can we make sure that the strategy is applied? These questions need to be addressed by more scientific research, however an appropriate minimum viable product (MVP) of stage II would introduce [Roman, 2021]'s tool as an adaptation strategy.

Introduce new types of KPI-s The current setup of the system works with KPI-s whose designated functions are focused on the running time of a process instance. This means that whenever a new

KPI is introduced in this system, it will also operate on the basis of process runtime. One possible next step would be the introduction of new KPI-s, whose designated functions are focused on other parameters. The type of parameters can be determined based on the context of the business process. For instance, KPI-s based on the context of marketing can focus on the website traffic or the number of articles that get published. The introduction of a new type of KPI will most likely result in the development of a new service which is specific to the KPI in question. In order to implement this next step, there are two important things that need to be done: defining the KPI (as we have done for the process run-time-based KPI in Listing 4) and developing the requirements accordingly (as discussed in §7.1.3).

Introduce other types of adaptation ‘reasons’ and triggers Another possible next step would be to introduce a new type of reason which can be mapped to an adaptation trigger. This ‘reason’ could be either a performance metric or some other type of measurement (axes 1 and 2 in Figure 3). This would imply the implementation of a new service for the reason in question as well as a service to represent the possible adaptation triggers that can be exerted from said reason.

In §2.4.1 we talk about how the monitoring of different existing types of metrics can lead to possible violations, which in turn can be mapped to adaptation triggers. Some examples of such metrics are service-level agreements (SLA-s), risk metrics or SVN (subversion) values. Given the proper definition and implementation, it would be possible to also implement other adaption reasons such as changes in laws, (process) context or even resources. Albeit similar to the previous point, in order to develop this one, it is important that new, appropriate definitions for the metrics or reasons in question be developed. This could potentially also lead to the definition of an appropriate type of adaptation trigger. Once this step is done, the next thing to do is to implement a new microservice as discussed in §7.1.3.

Extend the functionalities of the dashboard Another possible step in the future would be to extend on the existing set of functionalities that the dashboard provides to the end-user (as presented in Listing 18). For example, since it is currently possible to use an editor to update the existing BPMN notation of a process instance and store it in the system, a next step would be to store it as a local file on the user’s workspace. The type of functionalities should focus on enhancing the experience of the end-user.

Working with third-party services A crucial step towards the development of our system is its integration with third-part applications. We use third-party (or external) applications to refer to any external systems, services or frameworks that are not affiliated with the Adaptation Dashboard but whose functionalities can be used by it. The importance of integrating our system with external services lies in the fact that our system is part of a bigger *pipeline* (Figure 4) whose end-goal is the (possible) adaptation of an adaptable process instance. By integrating with external services we are able to our-source the steps of the adaptation pipeline to appropriate services that can conduct them (what we have referred to as Stage II in the adaptation pipeline). In this way, we would be able to complete the development of the process adaptation pipeline.

As a result, we would firstly have to determine the appropriate services for the steps of the adaptation pipeline, namely metric prediction *and* execution of the adaptation. Afterwards, we can establish the integration between the services in question and the Adaptation Dashboard. Each one of the mi-

crosservices in the system is equipped with a REST API; as such, external systems that need to gather information from the Adaptation Dashboard can make use of the API interface.

Using other execution environments for process instances This initial version of the Adaptation Dashboard supports adaptable processes that are executed in Camunda's execution environment. An important future step for the development of the system would be the implementation of the system's compatibility with other execution environments such as Activiti²⁸ for instance.

²⁸Official website can be found on <https://www.activiti.org/>.

Bibliography

- [Aderaldo et al., 2017] Aderaldo, C. M., Mendonça, N. C., Pahl, C., and Jamshidi, P. (2017). Benchmark requirements for microservices architecture research. In *2017 IEEE/ACM 1st International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering (ECASE)*, pages 8–13.
- [Bedó et al., 2020] Bedó, J., Di Stefano, L., and Papenfuss, A. T. (2020). Unifying package managers, workflow engines, and containers: Computational reproducibility with BioNix. *Giga-Science*, 9(11).
- [Karastonayova and Ghahderijani, 2021] Karastonayova, D. and Ghahderijani, A. (2021). Autonomous process performance improvement.
- [Karastoyanova, 2010] Karastoyanova, D. (2010). On scientific experiments and flexible service compositions. pages 175–194.
- [Kazhamiakin et al., 2010] Kazhamiakin, R., Wetzstein, B., Karastoyanova, D., Pistore, M., and Leymann, F. (2010). Adaptation of service-based applications based on process quality factor analysis. In Dan, A., Gittler, F., and Toumani, F., editors, *Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops*, pages 395–404, Berlin, Heidelberg. Springer Berlin Heidelberg.
- [Lindsay et al., 2003] Lindsay, A., Downs, D., and Lunn, K. (2003). Business processes - attempts to find a definition. *Information and Software Technology*, 45:1015–1019.
- [Metzger et al., 2015] Metzger, A., Leitner, P., Ivanović, D., Schmieders, E., Franklin, R., Carro, M., Dustdar, S., and Pohl, K. (2015). Comparing and combining predictive business process monitoring techniques. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45(2):276–290.
- [Roman, 2021] Roman, A. (2021). Adaptable processes: concepts, design and implementation in Camunda. Master’s thesis, University of Groningen.
- [Sommerville, 2007] Sommerville, I. (2007). *Software Engineering* 8. Addison-Wesley Professional.
- [Van Der Aalst et al., 2016] Van Der Aalst, W., La Rosa, M., and Santoro, F. (2016). Business process management: don’t forget to improve the process! *Business & Information Systems Engineering*, 58(1):1–6.
- [vom Brocke et al., 2014] vom Brocke, J., Mathiassen, L., and Rosemann, M. (2014). Business process management. *Business & Information Systems Engineering*.
- [Walch, 2020] Walch, K. (2020). The autonomous systems pattern of AI. Forbes. Available at <https://www.forbes.com/sites/cognitiveworld/2020/05/30/the-autonomous-systems-pattern-of-ai/?sh=4880e1b46a6b>.
- [Wegner, 1996] Wegner, P. (1996). Interoperability. *ACM Comput. Surv.*, 28:285–287.
- [Weiss and Amyot, 2007] Weiss, M. and Amyot, D. (2007). *Business Process Modeling with the User Requirements Notation*.

- [Wetzstein et al., 2010] Wetzstein, B. et al. (2010). Cross-organizational process monitoring based on service choreographies. In *2010 ACM Symposium on Applied Computing (SAC '10)*, page 2485–2490. ACM.
- [Wetzstein et al., 2012] Wetzstein, B., Zengin, A., Kazhamiakin, R., Marconi, A., Pistore, M., Karastoyanova, D., and Leymann, F. (2012). Preventing KPI violations in business processes based on decision tree learning and proactive runtime adaptation. *Journal of Systems Integration*, 3:3–18.

Appendices

A Adaptation Dashboard

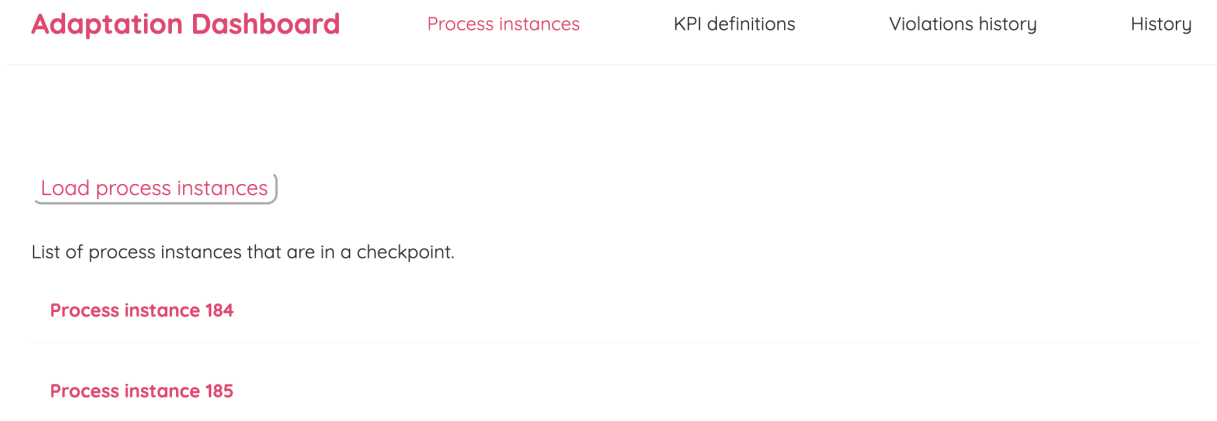


Figure 22: Overview: the main page in the adaptation dashboard. This page lists all the process instances that have reached a state which gives them the opportunity to be checked whether they need to be adapted or not. At the time of this image, we note that there were two process instances which have reached such a state.

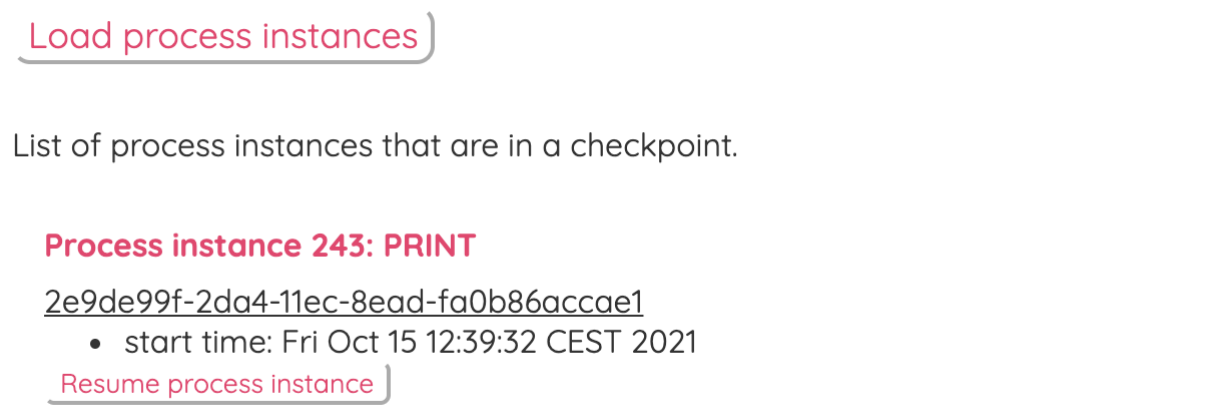


Figure 23: Overview: each one of the listed processes in the dashboard has a detail page, which can be reached using a referral link. The end-user can dictate a process instance to resume its execution in the Camunda engine, by using the [“Resume process instance”](#) button.

Adaptation Dashboard

Process instances

KPI definitions

Violations history

History

Process instance 184: d314c517-0330-11ec-b4f8-acde48001122

Select type of KPI: [Check for violations](#)

Figure 24: Overview: detail page of a process instance, where the user can note the BPMN notation of the process model.

Process instance 184: d314c517-0330-11ec-b4f8-acde48001122

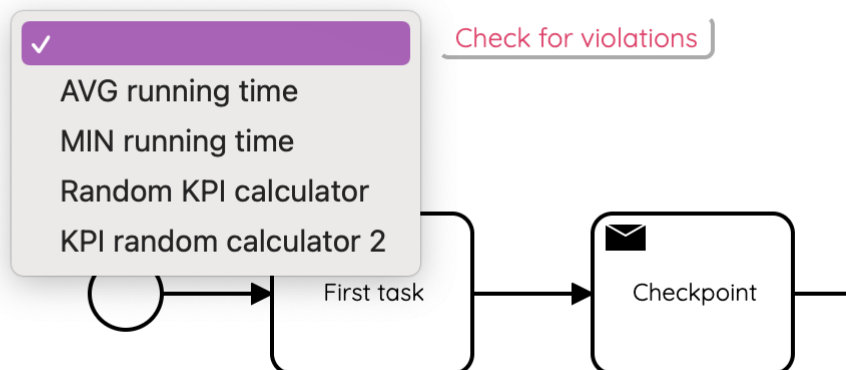
Select type of KPI: [Check for violations](#)

Figure 25: Overview: for every process instance, the user can opt to check whether there have been any violation of the KPI-s that have been already established in the system. First, the user would have to select the KPI for which they would want to perform the check; then, they would have to click on “Check for violations”.

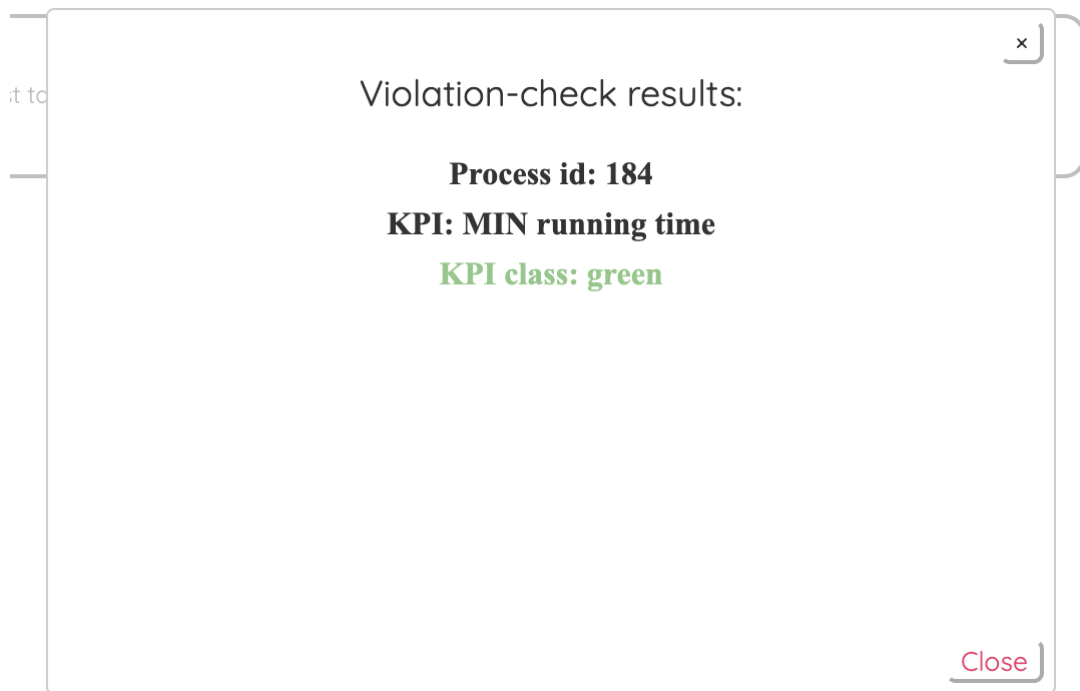


Figure 26: Overview: triggering the button from Figure 25 will generate a violation check whose results will be shown on a pop-up modal for the user to see.

Adaptation Dashboard

Process instances

KPI definitions

Violations history

History

[Define new KPI](#)**KPI name: AVG running time**

KPI description: This KPI focuses on the average running time of the process instances.

KPI name: MIN running time

KPI description: We use this KPI to compare the running time with the minimum running time of other process instances.

KPI name: Random KPI calculator

KPI description: We use this KPI to compare the running time of processes with the threshold that we have defined here.

KPI threshold: 35

Green class mapper: For values smaller than 35


KPI name: KPI random calculator 2

KPI description: We use this KPI to see if the running time of process instances is greater than the set threshold.

KPI threshold: 40

Green class mapper: For values greater than 40

Figure 27: Overview: a list of all the KPI-s that have been defined in our system. For every KPI except for the first two, the user gets to see their name, description, threshold and ‘green class mapper’. The latter two are used to define the mapping function of a KPI such that it can be used for violation check-ups.



The image shows a pop-up window titled "Define new KPI:". The window has a close button (an 'x' in a square) in the top right corner. Inside the window, there are four input fields arranged vertically:

- "KPI name:" followed by a text input box.
- "Description:" followed by a larger text input box with a small diagonal icon in the bottom right corner.
- "Threshold:" followed by a text input box.
- "Green class:" followed by a dropdown menu with a downward arrow.

In the bottom right corner of the window, there is a "Save" button.

Figure 28: Overview: pop-up form that allows the user to create a new KPI that can be stored in our system.

Adaptation Dashboard

Process instances

KPI definitions

Violations history

History

List of violations that have been calculated.

Process id: 155
KPI name: Random KPI calculator
KPI class: red

Process id: 164
KPI name: Random KPI calculator
KPI class: red

Process id: 164
KPI name: KPI random calculator 2
KPI class: green

Process id: 174
KPI name: KPI random calculator 2
KPI class: green

Process id: 164
KPI name: Random KPI calculator
KPI class: red

Process id: 174
KPI name: MIN running time
KPI class: green

Figure 29: Overview: a list of all the violations that have been calculated and stored on our system as a result of the end-user triggering the button on Figure 25.

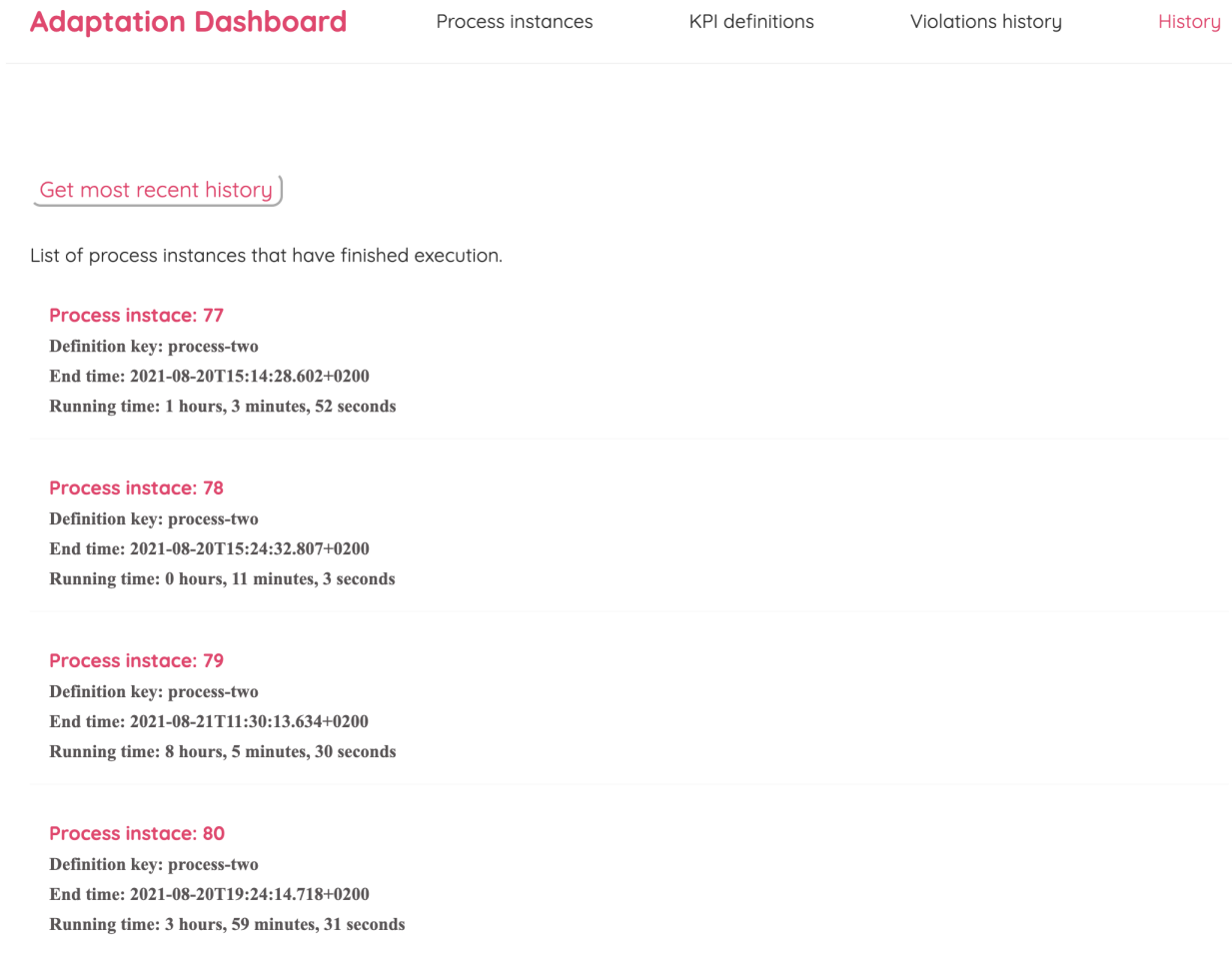


Figure 30: Overview: a list of all the process instances that have terminated, i.e. finished execution from the Camunda engine. The system can opt to makes use of these processes details if/whenever it is needed.

B Camunda modeller

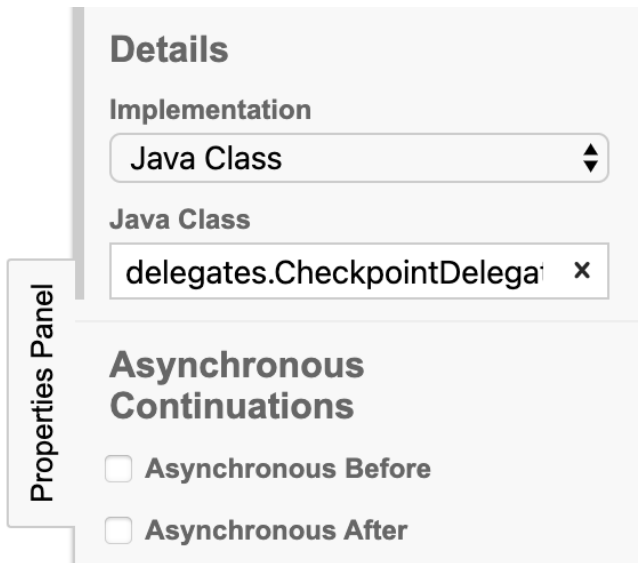


Figure 31: Screenshot of the 'Properties Panel' in Camunda's modeller.

C Camunda's API requests

In this part of the Appendix we present a list of useful API requests that can be used to work with Camunda's execution environment as well as to retrieve information about its process instances. These requests prove very useful in establishing the communication between Camunda and the Adaptation Dashboard (AD) system as well as evaluating/testing the functionalities of AD. We assume that Camunda's servers are running on `localhost:8080`. More requests can be retrieved from <https://docs.camunda.org/manual/7.15/reference/rest/>

1. Deploy a process instance in Camunda's environment.
Method POST:
`http://localhost:8080/engine-rest/deployment/create`
2. Start the execution of a process instance using its definition key.
Method POST:
`http://localhost:8080/engine-rest/process-definition/key/process_instance_key/start`
3. Get all the active process instances in your Camunda environment.
Method GET:
`http://localhost:8080/engine-rest/process-instance`
4. Delete a process instance from Camunda's environment using its identifier.
Method DELETE:
`http://localhost:8080/engine-rest/process-instance/process_definition_id`
5. Delete a process definition (i.e. its deployment) from Camunda's environment.
Method DELETE:
`http://localhost:8080/engine-rest/process-definition/process_definition_id`
6. Correlate a message with a process instance in Camunda.
Method POST:
`http://localhost:8080/engine-rest/message`

7. Get the history instance for a specific process in Camunda.

Method GET:

`http://localhost:8080/engine-rest/history/process-instance/process_instance_id`

Listing 21: Camunda: useful API requests