Institute of Architecture of Application Systems
University of Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Master Thesis No. 0838-005

BPMN 2.0 Relevant Process Fragments (RPF) Repository

Abdul Wahab

Course of Study: INFOTECH

Examiner: Prof. Dr.Dr.h.c. Frank Leymann
Supervisor: M.Sc.,M.A. Marigianna Skouradaki
Commenced: 08.06.2015
Completed: 04.12.2015

CR-Classification: H.4.1, H.2.4, H.2.7, H.3.3, H.3.4

Abstract

The re-use of process models is an objective frequently discussed in the literature. The state-of-art expresses numerous efforts towards this direction, by providing a great variety of methods, models, algorithms and tools. More particularly, many approaches utilize similarity metrics, patterns, repositories and fragments in order to facilitate process model re-usability. This thesis is motivated from the need to re-use recurring parts of process models expressed in Business Process Model and Notation 2.0 (BPMN 2.0) language, and use them to synthesize representative, executable, synthetic BPMN 2.0 process models, that are utilized for benchmarking purposes. More particularly, the focus is on an already available algorithm that detects re-occurring structures in a collection of BPMN 2.0 process models. These structures are named "Relevant Process Fragments" (RPFs) and they are stored in a separate collection.

In the scope of this thesis, an RPF Repository is developed, which enables the composition of the artificial, executable BPMN 2.0 process models, with respect to user-defined, benchmark-related criteria. To this edge, the contributions of this work are to design and implement a prototype with the following functionalities: a) automatically characterize the RPFs with respect to benchmark related characteristics, and store them in an RPF repository; b) retrieve the appropriate RPFs from the repository with respect to benchmark related criteria; c) compose synthetic BPMN 2.0 process models that are compliant with the BPMN 2.0 standard; d) validate the representativeness of the process models through empirical rules; and e) automatically export the executable and/or deployable forms of the synthetic process models for two different BPMN 2.0 workflow engines. Our methodology is validated through simple and complex use case scenarios.

Acknowledgement

I would like to express my gratitude to Prof. Dr. Frank Leymann from the Institute of Architecture of Applications Systems (IAAS) for giving me the opportunity to do research in this field. I would also like to thank my supervisor Marigianna Skouradaki for her continuous guidance and support during the whole thesis. I would further like to thank all my Professors of University of Stuttgart who have taught me.

I would like to take this opportunity to thank my parents as well for all their prayers, sacrifices and never fading love. Special thanks to my friends and younger brothers, Abdul Basit and Sohaib Iqbal, your encouragement has always been a true motivation for me.

Contents

List of Figures

List of Listings

1 Introduction

In the last few years, technology has improved exponentially. The number of tools available to perform a particular task are also growing. Moreover, not all the tools are similar. Some have different architecture and some provide different set of functionalities in their domain. Users want to know that which tools among the set of tools provide the required functionality. So, tools need to be validated. Further, if more than one tool provides the same functionality, then users are interested to find out which tool would be the best fit in a particular domain. This leads to a scenario in which there is a need to benchmark the tools to find out the best one. For benchmarking, there is a need of an application that allows to develop simple as well as complex scenarios. These scenarios can then be tested on different tools available and eventually, the performance of each tool is compared.

1.1 Research and Problem Statement

Business Process Modeling Notation (BPMN 2.0) is a standard way of describing the internal procedures of an organization as well as collaboration with other organizations[JEay]. It is also a widely used modeling language [SRL+15]. [MPR12] describes workflows as flow of information through the Business Process or flow of tasks following a set of procedural rules. Additionally, there is a piece of software on which these workflows are executed called Workflow Management System [Tavay]. In the field of BPMN 2.0, there are also many Workflow Management Systems (WfMS). Benchflow project [Benay] addresses the problem on how to benchmark BPMN 2.0 WfMS. Most companies do not share their Business Processes because "Process equals product"[Ley01]. [Benay] proposes synthesizing Process Models using real world BPMN 2.0 Process Models. Those synthesized Process Models can then be used for benchmarking BPMN 2.0 WfMS. Performing the benchmark is not a part of this thesis.

[Benay] splits the benchmarking of BPMN 2.0 WfMS into multiple steps. First, some real world scenarios are collected then, relevant structure from the Process Models are extracted. Afterwards, Process Models are synthesized using the relevant structures. Lastly, the synthesized Process Models are used for benchmarking the WfMS. [SGHL15] uses some real world BPMN 2.0 Process Models and extracts frequently occuring structures from them. These structures are Process Fragments which can be re-used later [SKK+11]. This leads to the requirement of an application which can synthesize a Process Model based on user-defined,bechmark-related criteria.

In this thesis, a prototype application is developed which fulfills the requirement of synthesizing a valid BPMN 2.0 executable Process Model. The approach is to use the Process Fragments from [SGHL15] work. Then by using the metadata information of those Process

Fragments, Process Models are synthesized. The prototype application developed provides the following main functionalities:

- A repository of Process Fragments (from [SGHL15] work) and associated metadata.
- Synthesizing of Process Models based on the benchmark-related criteria.
- Automatic creation of the Process Engine executable version of synthesized Process Model for two Process Engines (Camunda and jBPM).
- Automatic creation of '.war' package which can then be deployed on Process Engine (Camunda).

1.2 Outline

The rest of the document is structured in following way:

Chapter 2: Discusses the fundamental concepts and ideas which builds the elementary knowledge base of the work in this thesis.

Chapter 3: Discusses some related work and how that work matched with the work in this thesis.

Chapter 4: Discusses the functional, non functional requirements and the proposed architecture of the application.

Chapter 5: Explains the implementation and configuration of the prototype application.

Chapter 6: Validates and evaluates the prototype application against the requirements.

Chapter 7: Summarizes the work done in the thesis and describes the potential scope for enhancements and extensions.

1.3 List of Abbreviations

The following list describes the abbreviations used in this document.

API	Application Program Interface
AST	Abstract Syntax Tree
B2B	Business to Business
Blob	Binary Large Object
BPMN	Business Process Modeling Notation
CFC	Control Flow Complexity
DOM	Document Object Model
FTP	File Transfer Protocol
jar	Java Archive
JMS	Java Message Service
REST	Representational State Transfer Protocol

RPFs	Relevant Process Fragments
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
URL	Uniform Resource Locator
war	Web Application Archive
WfMS	Workflow Management System
WSDL	Web Service Definition Language
XML	Extensible Markup Language
XSD	XML Schema Document

2 Fundamentals

This chapter describes the fundamental concepts and ideas which are related to the work in this thesis. Section 2.1 explains the definition and basic concepts of Business Process Modeling Notation. Section 2.2 defines repository and discusses the fundamentals of metadata and the usage of metadata in BPMN 2.0 Models. Section 2.3 discusses the general idea of benchmarking. Finally, section 2.4 discusses the idea of Workflow Management Systems (WfMS) and two of many available WfMS.

2.1 Business Process Modeling Notation

Business Process Modeling (BPMN 2.0) is a standard in which internal Business procedures of an organization are defined in graphical manner so that those procedures can be communicated within an organization in a standard way [JEay]. Further, BPMN 2.0 advances the capabilities and can also handle the B2B Business concepts [JEay]. BPMN 2.0 is based on the older version of BPMN 1.1 and additionally, it contains an XML based format to exchange BPMN Models [Ley14]. [Sil11] describes three levels of BPMN namely: a) Descriptive Modeling b) Analytical Modeling and c) Executable Modeling. Descriptive Modeling and Analytical Modeling are used to describe the processes of business. The former has relaxed completeness criteria in a way that some parts can be skipped in the Model, while the latter should have precise semantics. The third is Executable Modeling which are the executable versions of the Process Models. In this thesis, the focus is also to create the executable version of the synthesized Process Models. The mentioned levels are not part of BPMN 2.0 specification [Ley14].

BPMN 2.0 mainly consists of three parts [Ley14] [JEay]:

- **Orchestrations:** Flow of tasks, decisions, events and data. The Processes could be internal (executable) Process or public process.
- **Choreographies:** While Orchestration focuses on the flow of the Process Model, Choreography diagrams focus on the exchange messages between organizations.
- **Collaborations:** An interaction between two or more participants. For example, a customer calling help centers.

In this thesis, the focus is on Orchestration and not on Choreographies and Collaborations. There are some standard elements in BPMN 2.0 and every element has some characteristics. The elements have been categorized in five different categories, namely [JEay]:

- Flow Objects
- Data

- Connecting Objects
- Swimlanes
- Artifacts

The focus in this thesis is on Flow Objects and Connecting Objects. A Process is mainly a flow of tasks, gateways and events [JEay]. A valid Process must have a starting point (start event) and it must reach an end event [JEay]. An end event can be reached from multiple paths. The exact path is determined by the gateways. Gateways are used to converge or diverge the paths within a Process [JEay], which means that they must have either multiple incoming sequence flows or multiple outgoing sequence flows or connections. A sequence flow shows the direction of flow of Process from one element to another [JEay]. There are different types of gateways and every type has its own significance. For example, parallel gateway with multiple outgoing connections is used when there is a requirement to create parallel flows [JEay]. Similarly, there are many types of tasks as well. The purpose of a task is to perform a particular kind of work [Ley14]. Again every type of tasks has its own significance. For example, script tasks are used to execute the script, call activities are used to call other Processes etc. The elements of BPMN 2.0 which are the focus of this thesis are shown in figure 2.1. [JEay] explains the details of elements.

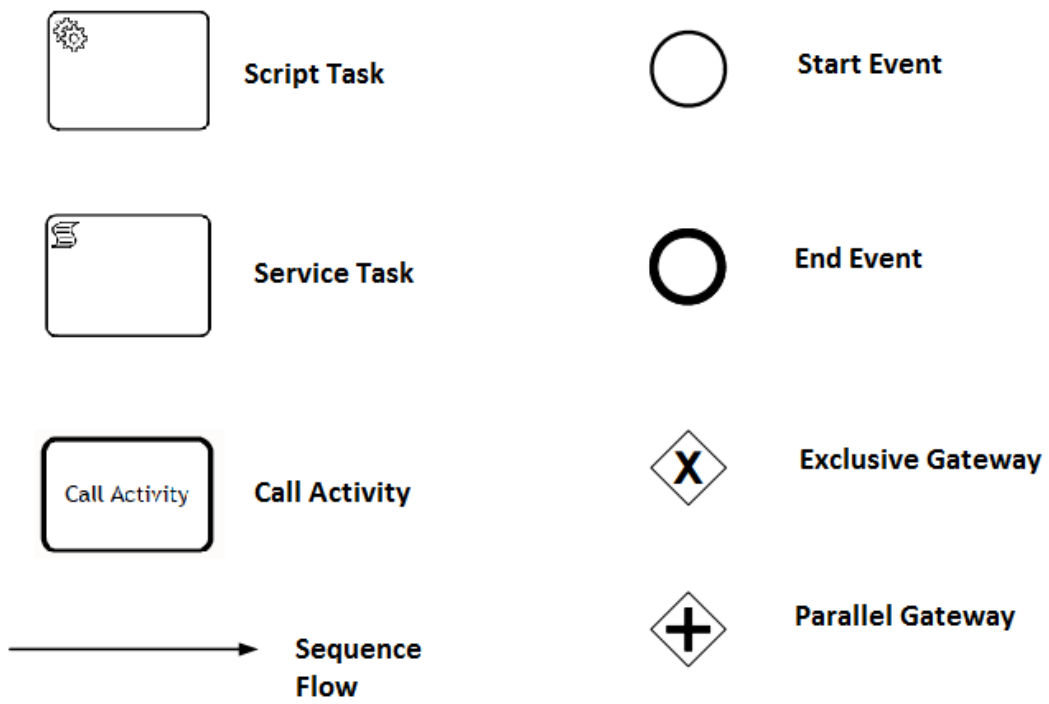


Figure 2.1: Few BPMN 2.0 Elements [JEay]

2.1.1 Process Fragments

There is ever-increasing desire to reach goals quickly and efficiently. Every one is looking to enhance the speed of the work in their domain. People prefer that if some work is already done, there should be no need to do it again. The idea of having Process Fragments is because of the same aforementioned reasons.

[SKK+11] explains Process Fragments as Fragments which can be re-used later and having some relaxed completeness criteria. Process Fragments may or may not have a start or end event [SKK+11]. An example is shown in figure 2.2.

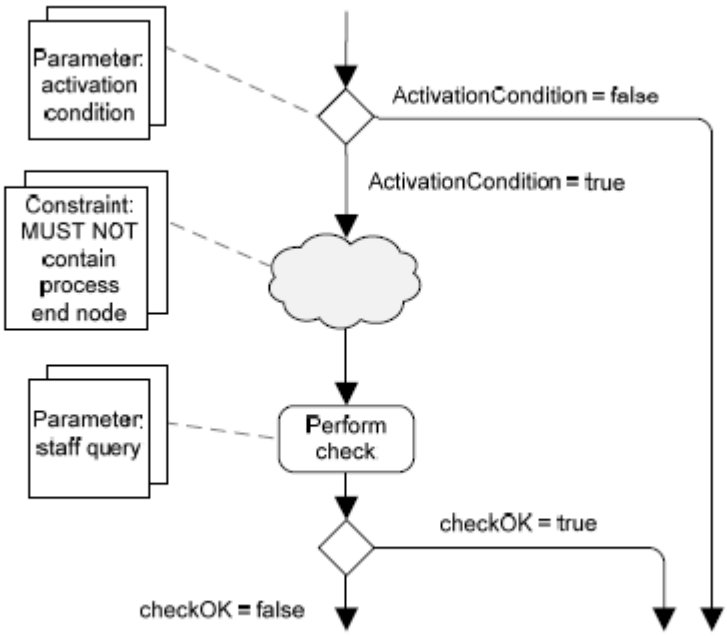


Figure 2.2: Process Fragment Example for Approval with Constraint. [SKLS10]

The Process Fragment in figure 2.2 shows a typical approval Process, which itself is not a complete BPMN 2.0 Model but it has some parameters and according to [SKLS10] these parameters needs to be set when a user wants to use and concretize this Process Fragment. Concretize means integration of Process Fragment into a Process Model. The parameter in this Process Fragment which needs to be set is for example, providing the staff query which performs the check. The Process Fragment in figure 2.2 shows a region known as placeholder. [SKLS10] defines that regions may or may not have constraints. For example, the constraint in this example says that there should not be an end event at that place if one wants to use this Process Fragment. More details about the usage and importance of Process Fragments are discussed in section 3.1.

2.2 Metadata and Repository

Metadata is data about data or data that is used to describe an information resource [NISay]. The usage of metadata is not just limiting to describe a resource. It also helps to locate, retrieve or manage the resource [NISay]. [NISay] explains three types of metadata:

- Descriptive Metadata: The metadata which is used to find a resource. For example, the name of a writer who has written a novel.
- Structural Metadata: provides the structural information related to a resource. For example, order of books to form a library.
- Administrative Metadata: provides information about when and how a resource is created.

BPMN 2.0 Models can also be assigned with metadata which could later be used to find out the Process Model. The terms 'annotation' and 'metadata' are used interchangeably in BPMN 2.0 but the semantics of both terms are the same and mostly used for descriptive metadata associated with Process Models. [Lin08] in his Phd thesis proposes a semantic annotation approach to annotate the Process Models. The purpose of annotating is to find Process Models and reuse them. [Eli15] also proposes an application to store the Process Models with annotations. The abstract flowchart of the application proposed by [Eli15] is explained in 3 steps:

1. Populating the repository:
    - Create Process Model.
    - Save the Process Model (On User Interface a form is shown to user so that it can add annotations).
    - After manually adding and then submitting the form, the annotations are saved.
2. Accessing the repository:
    - Provide search criteria.
    - Analyse the search results.
    - Select candidate Process Models.
    - Analyse the annotation of candidate Process Models.
    - View Process Model.
    - Modify Process Model (if needed).
    - Save Process Model (if modified).
3. Accessing the external repository:
    - Provide search criteria.

- Translate the query.
- Analyse query results.
- Select candidate process.
- Import Process Model.
- View Process Model (at this point the Process Model is saved in the local repository as well).
- Modify Process Model (if needed).
- Save Process Model (if modified).

The important thing to note is that annotations are basically describing the descriptive metadata and descriptive metadata can not be used to synthesize Process Models. Whereas the focus in this thesis is on synthesizing Process Models which can be done if the structural metadata is available. In either case, metadata needs to be stored somewhere. Usually it is stored in the repository. A repository is a place where data is kept in an organized manner. A repository could be a file storage system or it could be a database. Usually, database is preferred over file system because database is expected to improve performance over file system.

2.3 Benchmarking

[HHM11] describes benchmark as a continuous activity in which performance is measured and comparisons are made between the competitors in a relevant context. It is not a one time activity, usually organizations perform the benchmark regularly to improve the stature of their product and to achieve the world class status [HHM11]. [BR98] explained different classifications of benchmarking. Some of the classifications which are based on the required objectives are explained below [BR98]:

- Competitive: when the performance of an organization is needed to be compared against a competitor.
- Process: when a particular Process is needed to be compared with the Process of the competitor company.
- Industry: when the performance is needed to be compared compared with not only competitors but with all the organizations in the same industry.
- Performance: when the performance of a particular product of an organization is needed to be compared with the performance of the corresponding product of other organizations.

Among the set of classifications, performance classification is of prime importance as the focus in this thesis is to synthesize a Process Model which will then be used to benchmark the performance of the Process Engines based on key performance indicators (KPIs) mentioned in [Benay].

2.4 Workflow Managment Systems

Business Process in computing environment is called workflow [Ley14]. [MPR12] describes Workflow Management Systems (WfMS) as a piece of software which runs workflows. [SMS98] explains that WfMS allows to reach business goals efficiently by managing work activities and by invoking appropriate information resources linked with those activities.

[SMS98] describes that usually one workflow engine is used by the WfMS but in a distributed environment multiple engines can also run depending on the requirement of an organization which runs the WfMS. WfMS can execute multiple instances of a workflow and almost all the WfMS uses databases to store the instances and data associated with those instances [SMS98]. In this thesis, the focus is on two open source BPMN 2.0 Process Engines: Camunda and jBPM which are explained in the following subsections.

2.4.1 Camunda

Camunda [GmBay] is an open source platform to deploy Business Processes. Camunda is written in Java to give the Java developers leverage in implementing and running the Process Models on Java Virtual Machine (JVM) [GmBay]. Camunda has three main components on an abstract level; BPMN 2.0 Core Engine, a Job Executor and a relational database [GmBay].

Figure 2.3 shows the architecture of the Camunda Engine in which the Core Engine is responsible for translating the BPMN files to Java Objects because Camunda has been written in Java [GmBay]. Further, the Core Engine also presents the implementation of different constructs like service tasks or events. Job Executor is responsible for all the asynchronous work. For example, handling the execution of a particular Process [GmBay]. As mentioned earlier, the state of the instances are stored in a database therefore a relational database in this case saves all the information related to the instances of the Process Models. Whereas, Persistence layer ensures that the instances persist in the database [GmBay]. Tomcat server is usually used to deploy the Process Models on Camunda [GmBay]. This is usually done by deploying a Web Application Archive (.war) package on Tomcat. Further, Camunda also supports RESTful API to directly deploy the Process Model on Camunda [GmBay].

2.4.2 jBPM

jBPM Engine is a light-weight open source workflow Engine and it is also purely written in Java [Hatay]. jBPM allows to not only execute and manage the Process Models on the

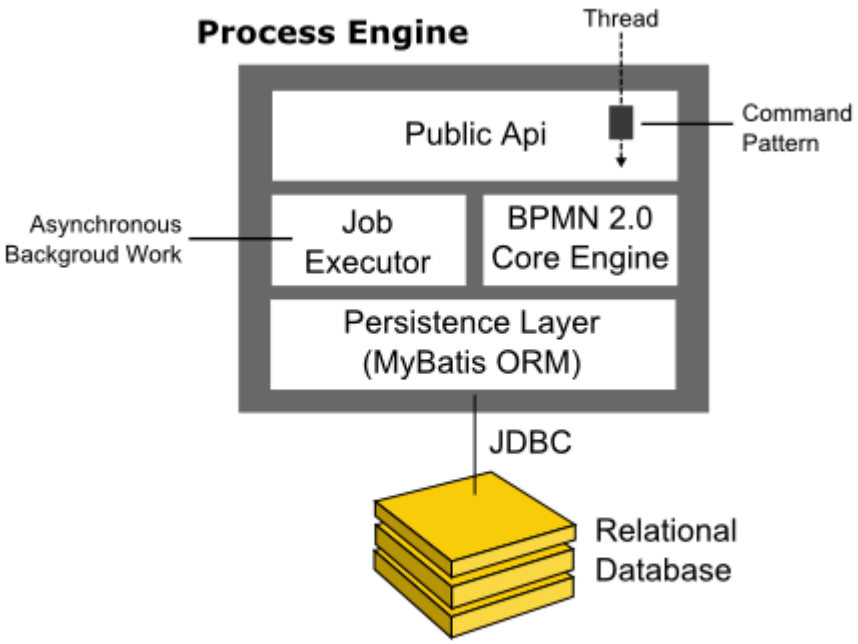


Figure 2.3: Architecture of Camunda Engine. [GmBay]

workflow Engine but also to model them using Eclipse-based and web-based editor [Hatay]. jBPM Engine supports the complete BPM life-cycle [Hatay]:

- Modeling and Deployment
- Execution
- Runtime Management
- Reporting

Figure 2.4 shows the different components of jBPM Engine in which the core Engine is at the heart of every other component. The application can also remotely connect to the Engine using RESTful API and JMS API [Hatay]. As in the case of others, jBPM also uses the database to persist the state of running instances [Hatay].

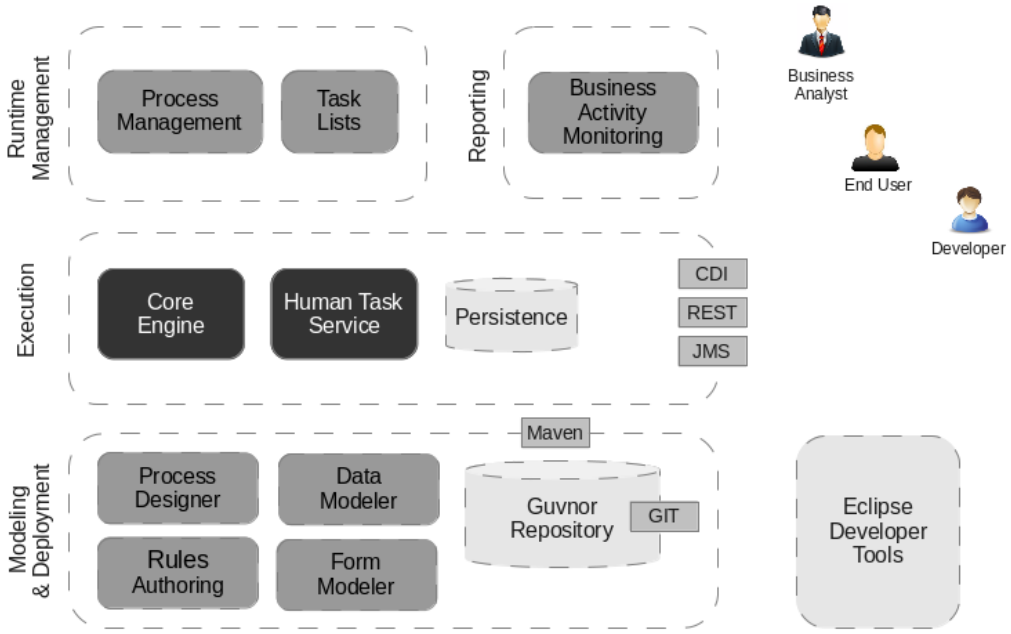


Figure 2.4: Components of jBPM Engine. [Hatay]

3 Related Work

In this chapter, similar work related to Process Fragments Repositories and benchmarking is discussed. In section 3.1 the emerging definitions of Process Fragments and how one can obtain and utilize them is discussed. In the section 3.2, the related work regarding Process Model repositories which have been developed and how our repository will be different from them is discussed. This thesis not only focuses on how to create a repository of the Process Fragments but also on how to retrieve and store the metadata or annotations related to them. Hence, similar work related to the metadata of Process Models and Process Fragments is also discussed in section 3.2. Finally, since the last contribution of this thesis is to synthesize executable Process Models for benchmarking purposes, Section 3.3 discusses related work regarding synthesizing Process Models, while Section 3.4 overviews the state-of-the-art in Workflow Management Systems benchmarking.

3.1 Process Fragments

[SKK+11] claims that the concept of Process Fragments is quite similar to the concept of libraries in typical programming languages where one can re-use the functionality without starting it from the scratch. [SKK+11] describes Process Fragments as a reusable structure that has some relaxed completeness criteria. They argue that Process Fragments help to develop a new Process Model easily and efficiently. Re-usability is not just important in the field of Business Process Management but also in the other fields such as biochemistry [Mayay] where they also used Fragments library. [RP93] introduces new term "Process Chunks" in the requirement engineering Process. It has a knowledge store that is useful in making decisions when it comes to requirement engineering. In requirement engineering, people focuses on the requirements of the user to develop any required system. But in this thesis the focus is not on requirement engineering.

There is also a concept of sub process [RMD10]. According to [UEKL10] sub process definition is similar to Process Fragments in terms of re-usability but represent different conceptual context in a way that sub process gives all the decision making power into the hands of a parent Process. [UEKL10] also claims that Process Fragments are part of the complete Process Model which means that they define their autonomy unlike sub processes who give up all of their autonomy. [ELS+10] also points out the difference between Process Fragments and sub processes. According to the author, Process Fragments posses incomplete Process knowledge and they need to be integrated with some other Process Fragments or Process Model to have a complete Process knowledge. [ELS+10] also presents a Process Fragment modeling language based on BPMN 2.0 standard through which different Process Fragments can be composed so that they can be re-used at later point in time.

[SKLS10] also emphasizes on the importance of having Process Fragments. Re-use is an integral part of enhancing the quality and minimizing the time spent on a particular task and that is why there is a need of Process Fragments. Process Fragments can also be realized as a subset of Process Models. [SKLS10] defines further characteristics of Process Fragments. According to the author, they may have but not required to have a start or end node, they may contain multiple incoming/outgoing edges and they must contain at least one activity. According to [SKLS10] the characteristics of Process Fragments are not limited to above, once could also impose further requirements on Process Fragments depending upon the application domain. For example, a Process Fragment has to obey some kind of compliance. Compliance means following some rules defined by a company or following some regulations or law. The compliant specific Process Fragments are not allowed to be edited freely unless where specified. [SKLS10] also describes some characteristics for using Process Fragments when it comes to compliance. For instance, the Process Fragments could be allowed to parametrized in order to define the areas in the Process Fragments that can be varied. One could define the constraints or regions in the Process Fragments. Means how those spaces or placeholders must be filled with activities or other Process Fragments. Placeholders are just an empty area which needs to be filled with something.

The important parameter to note is that these kind of Process Fragments have some context or some functional semantics (Process area/domain, Process type etc) associated with them. The regions and constraints both are usually related to functional semantics. Whereas in this thesis we exploit the concept of Process Fragments to synthesize new Process Models out of them. As we are mostly interested in the structural semantics of the Process Fragments, the aforementioned attributes of regions and constraints are not considered. The challenge in the work presented by [SLM+10] is also the same that it heavily based on the functional semantics of the Process Fragments, while in our work we exploit the structural semantics of the Fragments. [SLM+10] proposed two methodologies for the creation of the Process Fragments. One is Top-down approach in which a part of the Process Model has been extracted from a bigger Process Model. One could also realize it as a sub-graph of a complete graph. The other is Bottom-up approach in which a Process Fragment is build from the scratch. It is designed to fulfill requirements in some particular context. If this work presented by [SLM+10] not limits to functional semantics, then these Process Fragments could also be used in the prototype repository.

In [SGHL15] the authors propose an extended version of Process Fragments. In this work the authors observe that the original definition of Process Fragments given by [SKK+11] was based on the knowledge of the experts in the corresponding field. In order to automate the Process Fragment discovery the authors propose an approach that discovers and extracts frequently occuring structures in a collection of Process Models. The extracted Process Fragments can then be re-used for the creation of new Process Models. In order to increase consistency and re-usability the authors also provide an extended version of the approach which tags every recurring Process Fragment the number frequency appearance in the collection [SGHL15].

The extracted Process Fragments of this approach will be of prime interest for our thesis because we are also interested in the structural semantics of the synthesized Process Model

from Process Fragments.

3.2 Process Model Repositories

After getting the idea of what Process Fragments are, the main task in our thesis is to create a repository of them. The repository should store metadata information along with the Process Fragments. These will be later used to synthesize Process Models. The metadata information which we want is different than the already available metadata information of the Process Fragments [ML08] because we are basically focusing on the structural semantics of Process Fragments as in the end we want to synthesize Process Models using those Process Fragments. Also, we want to extract the metadata information automatically from Process Fragments. [SKLS10] proposes a Process Fragments library called "Fragmento". It is basically a repository for storing and managing the Process Fragments. One can search the required Process Fragment from the repository by doing a search (text search) on Process Fragment metadata such as it's name, the overall context in which it is used and other input parameters. Using Fragmento, one could even go one step ahead and perform advance search on the repository which is beyond the metadata like the structure of the Process Fragment. Further, there is also a feature available to validate the XML Schema through Fragmento. An example could be, validating a Process Model whether it contains a cycle or not. Fragmento also allows to perform changes upfront based upon user requirements. For example, deleting all the activities related to debugging. Finally, apart from a web interaction with Fragmento one can also use Fragmento features as Web Services using SOAP/HTTP message. Fragmento is designed and implemented for BPEL whereas in this thesis the focus is on BPMN 2.0. Also the code was not available so it can not be used in this thesis. Further, the metadata information focuses on the functional semantics of Process Fragments whereas in this thesis the focus is on the structural semantics of the Process Fragments.

[Eli15] in the Ph.D. thesis also developed a repository for Process Models. Before designing and creating the repository, [Eli15] did a survey regarding the already available Process Model repositories. [Eli15] found out through survey that inefficient search of the Process Model from the repository and publicly unavailability of the repository are the main loopholes when it comes to the Process Model repositories. Further, while developing the Fragmento library, only the requirements of the literature were kept in mind whereas in the repository made by [Eli15], not only the literature but also the experts experience were also taken into account to create the repository.

The Repository developed by [Eli15] can be used to; store the Process Model, search and navigate the repository, viewing the Process Model using web interface, adapt the Process Model and versioning the Process Models. The prototype developed in this work also stores the Model(Process Fragments). It also searches and navigates the Process Fragments of the repository. The difference arises when it comes to utilizing the Fragments, the prototype developed in the thesis uses Process Fragments and automatically tie them together to synthesize Process Model whereas using the repository proposed by [Eli15], one has to

perform manual steps for example; linking with other Process Fragments, tailoring the Process Fragments etc to synthesize Process Model.

The architecture of the repository developed by [Eli15] is different from the traditional three layers model in which there is Presentation layer, Backend layer (Business Logic layer) and Data layer. It has five layers, one of which, is the interoperability layer, which assists the user to easily exchange the model between different repositories by using wrapper which help to translate the queries depending upon the repository on the other side. The second additional layer is a service layer that exposes the functionality provided by the repository though Web Services and API's. The prototype presented in this thesis is also planed to be extended in order to expose the functionality through API's.

[Eli15] used database as a repository to save the Process Models and the prototype developed in this thesis also uses database. Further, it also uses metadata information, although the metadata information used with Process Models by [Eli15] are more focusing on the functional semantics of the Process Models (Process/Application Area, Process Type, Resources received, Resources provided etc). Whereas in the scope of this work, we are mostly interested in the structural semantics regarding the Process Fragment. By utilizing this information, we are then able to synthesize a Process Model out of selected Process Fragments. Additionally, we want all this metadata information to be stored in the Repository automatically by scanning/examining the Process Fragments whereas in [Eli15] the users have to actually provide the metadata manually through a web interface.

Valeria, Massimo and Salvatore in[SCG06] also present a repository for Process Fragments. Their contribution is similar to this work, but differs in context. More particularly, [SCG06] emphasizes on designing the new Process Model for multi-agent systems. [SCG06] has categorized the Process Fragments according to four basic criteria and all of these criteria focus on the functional semantics. For example, in which phase of the overall design Process, a particular Process Fragment can be reused. This is again a contradiction of our work that focuses on the structural semantics.

A repository for Process Models is also proposed by [VKL06]. One could think of their repository as an XML repository because Process Models are realized in XML and then stored in the repository. According to the author, at times the metadata of Process Models are also presented in XML files. The work [VKL06] based on the XML serialization of Process Models, while it allows the retrieval of XML data as Java objects or models of the Eclipse Modeling Framework(EMF)[1]. It is an important contribution by [VKL06] because searching through XML files can be very resource consuming and inefficient. The other important characteristics of the repository by [VKL06] are:

- Data handler part which takes care of serializing EMF objects to XML documents and de-serializing the XML files back to EMF objects. In short, the back-end repository just has EMF objects.

[1] `https://eclipse.org/modeling/emf/`

- User can query the Java objects (data) using Object Constraint Language (OCL). As the name suggests, OCL is a language that allows one to query Objects and in this case Java objects.

The architecture of the work [VKL06] focuses on retrieving,editing and storing the Process Fragments. It does not focus on synthesizing Process Model using those Process Fragments. Further, [VKL06] uses file system as a repository whereas in this thesis, database has been used to store the metadata along with the Process Fragments and this is expected to improve the query performance over a file system storage. Also [VKL06] is focusing and made for BPEL Process Models and we are more interested in the BPMN 2.0 Process Fragments.

3.3 Process Models Synthesizer

The next step in our thesis is to synthesize Process Models based on user defined criteria provided to the application. The user provides structural information about the desired Process Fragments and based on the criteria the application queries the Process Fragments from the repository and then join them together in order to synthesize an executable Process Model. With respect to this contribution, we analyze related works done in the scope of Process Model synthesizing.

Generating Process Model Collections is a research work proposed by [YDG14]. In the work [YDG14], new synthetic Process Models are generated which are similar to the real world Process Models, having approximately same properties. The first property which is considered by [YDG14] in order to generate the model is the size of the available Process Models. Then the second property is the number of the Process Patterns that appear in the Process Models, for example, loop, splits, sequences, joins. This has been done by analyzing the Process Models. Afterwards, the words that appear in the labels of the Process nodes and the co-occurrence of those words in the Process Models have also taken into account and lastly the type of nodes (end event, script task etc) that have been appeared in the Process Models.

[YDG14] presented the Process Models which have been used to generate new synthetic Process Models as graphs which have nodes (Tasks, Gateways, Events etc) and edges (Sequence Flows). The reason of this design decision is to enable the application of graph theory approaches. Then all the properties which has been mentioned above are applied for the Process Model generation. The work [YDG14] is different than what this thesis created in a way that [YDG14] used labels and their probabilities for the generation of Process Models and the outcome is not executable. Further, the prototype presented in this thesis mainly focuses on synthesizing the executable Process Models for benchmarking.

[EUL10] highlights one other aspect when composing a new Process Model using Process Fragments and that is fault handling. According to [EUL10], usually fault handling is done and modeled at design time when designing the Process Model but when Process Fragments are designed, the bigger picture is not available, which means that it is not known how the Process Fragments will be used and build into a Process Model. For that purpose,

[EUL10] proposes two fault handling strategies which allows to handle the fault in the Process Fragments. Since we are not focusing on fault handling part in our thesis therefore it can be ignored for now but it can be considered for the future work.

The creation of new Process Models out of Process Fragments is discussed in [ML08]. They have basically extended the life-cycle model of Process modeling. Additional steps have been added by them in the life-cycle model for different purposes. For instance, after extracting relevant Process Fragment from some repository there might be a need for adding more annotations/metadata against a Process Fragment so that it can be queried and searched more efficiently. Moreover according to the author, some Process Fragments might need some tailoring or modifications as well in order to align them according to the user requirement. But in this thesis the focus is on synthesizing Process Models automatically from Process Fragments based on the user defined, benchmark-related criteria and also the synthesized Process Models should be executable, whereas in the work [ML08] all of the steps need to be done manually.

There is one more aspect which should be considered to synthesize Process Models and that is structural characteristics of Process Models. The master thesis [Iva14] focuses on a statistical analysis of the structural characteristics of the Process Models. In the scope of this work, we have considered the contributions of [Iva14] in order to preserve the real-life attributes of Process Models.

To this edge, we also consider the work of Jorge [Car08]. In his work the author analyze the complexity of real world Process Models, and introduce the metric of Control Flow Complexity (CFC). [Car08] defined some ranges on CFC values which helps to identify the complexity of Process Models. We have used this metric in this thesis as well.

3.4 Workflow Management System Benchmarking

Although Process Model synthesis can be applied to many different use cases, our approach focuses on synthesizing Process Models for benchmarking purposes. The middleware that deploys and executes the Process Models is called Workflow Management System (WfMS). According to [Tavay], WfMS is a system which allows co-ordination of different individual components in a workflow, a process which is also called as Orchestration.

The benchmarking is important not just in the domain of WfMS but also in other domains [DES08], [Gra92] in order to find out which tool/engine would be a best fit [Hup09] in that particular domain.

According to the author of [SRL+15], there are a lot of Process modeling languages available but the most used one is BPMN 2.0 as it is a widely adopted standard. Furthermore, there are a lot of WfMS also available so there is a need to benchmark those WfMS in order to analyze the performance of those WfMS. The Benchflow project [Benay] and [SRL+15] addresses this problem on how to benchmark BPMN 2.0 WfMS. There are also many other approaches but Benchflow aims to have a standard benchmark. On a very abstract level the project has two main parts:

- **Defining the Workload**: According to [SRL+15] there are range of things which should be taken care in this point. First one is to collect the real world Process Models, secondly finding out the recurring structures in those Process Models and finally synthesizing Process Models with a specific workload out of those real world Process Models. [SRL+15] claims that the idea behind doing all these steps is to perform the benchmarking with the real world scenarios instead of any hypothetical Process Models. For that purpose, [SRL+15] have gathered some real world Process Models from different companies including IBM and then by using the RPF algorithm [SGHL15] all the recurring Process Fragments are derived. The next part according to them is synthesizing Process Models from those Fragments (which is part of this thesis) and at the end the synthesized Process Model will be used for the benchmarking.
- **Executing the Benchmark**: In order to execute the BPMN 2.0 Process Model on Workflow Engines, there is a need to do some Engine specific changes in Process Model in order to make them Process Engine executable. Until now, these changes were performed manually but by using the application developed in this thesis, user would be able to get the Process Engine specific Process Model in which all the required changes are performed automatically. This is necessary in order to speed up the benchmarking process. Further, Benchflow and [SRL+15] will provide a wide range of performance related metrics and aggregate them into valuable and some meaningful KPI's.

4 Concept and Design

In the first section 4.1 the functional and non-functional requirements of the application are discussed. In the second section 4.2 high-level architecture of the application is explained. Further, how different components interact with each other in the application are discussed in section 4.3. The detailed architecture of the application and the repository are explained in section 4.4. Finally, the pseudo code of the algorithms are discussed in section 4.5.

4.1 Functional and Non-Functional Requirements

In this section, the emphasis is to distinguish between functional and non-functional requirements related to the application. These functional and non-functional requirements must be taken into consideration while designing the architecture of the application.

4.1.1 Functional Requirements

Following are the functional requirements which should be fulfilled by the application:

REQ 1: (Store Process Fragments) The application should scan a particular pre-defined directory and retrieve all the Process Fragments. It should then store all of these Process Fragments in the repository, so that they can be queried later when required. Process Fragments must be assigned with a unique ID, so that they can be indexed and selected by the user at later point in time.

REQ 2: (Process Fragments Metadata) The application should automatically scan all the Process Fragments and retrieve all the metadata related to each Process Fragments. Since the aim is to synthesize Process Models, therefore Structural metadata should be stored in the repository for every Process Fragment.

REQ 3: (User-defined Criteria)The user of the application should be able to provide criteria to the application.

REQ 4: (Process Fragments Selection) The selection of the Process Fragments should be done based on the criteria provided to the application.

REQ 5: (Synthesize Process) The application should be able to synthesize Process Model using the already selected Process Fragments. The synthesized Process Model should be according to the BPMN 2.0 specification [JEay].

REQ 6: (Process Fragments Compatibility) Before synthesizing the Process Model, Process Fragments must be checked whether they are compatible with each other or not. Compatibility means, whether they can be joined with other Process Fragments or not, based on the metadata stored in the repository.

REQ 7: (Process Engine Executable Synthesized Process Model) The synthesized Process Model must be made Process Engine executable. This must be done according to the Process Engine requirements. This should be done for two different Process Engines preferably Camunda and jBPM.

REQ 8: (Automatic Deployment on Process Engine) The synthesized Process Model must be automatically deployed on one of the two Process Engines (Camunda).

4.1.2 Non-Functional Requirements

While designing the architecture of the application in this thesis, aspects like maintainability, reliability, flexibility and usability must be taken into the consideration. The details of the non-functional requirements (NFR) are below:

NFR 1: (Maintainability) The maintainability aspect should be given special attention while designing the architecture of the application in this thesis. Classes should be categorized in different packages and every package should have a specific purpose. Maximum attention should be given to the class definition so that functionality of every class is isolated from the other class with minimal dependency.

NFR 2: (Flexibility) Flexibility is an important part for every application. The application developed in this thesis should also be flexible in a way that for example, if at a later point in time there is a requirement to make synthesized Process Model executable for other Process Engine, that should be done without changing any part of the overall design of the application. Further, within the classes, functions with minimal length should be written so that if there is any requirement of modification or deletion at a later point in time, that can be done easily.

NFR 3: (Reliability) Maximum number of conditions should be tackled in the application developed in this thesis, to make sure that the application does not stop abruptly. Further, the application should always provide messages or feedback to the user, for example in the case of invalid user-defined criteria, the user should know that there is a need to change the criteria.

NFR 4: (Usability) Usability of the application should also be very straight-forward for the users. Users should be able to provide even the complex criteria with minimal efforts which implies that the associated XML schema should be easy for the user to understand.

4.2 Architectural Overview

This section proposes the high-level architecture of the application. The application has three core features on an abstract level. First one is to upload all the Process Fragments and associated metadata in the repository. In this thesis, database is used as repository because it is expected to improve query performance over file system storage. Second core feature is to synthesize Process Models based on user-defined, benchmark-related criteria and the last one is to make the synthesized Process Model executable with respect to Process Engine.

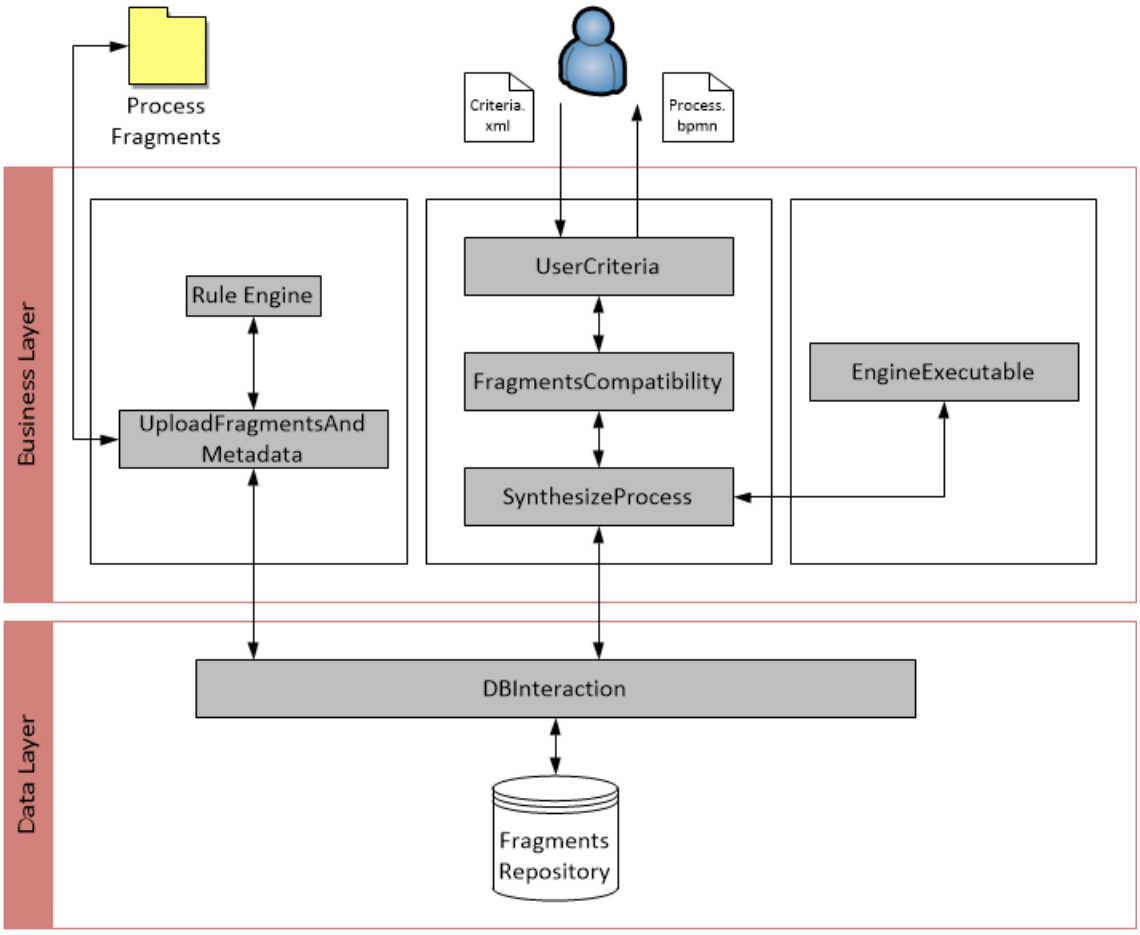


Figure 4.1: High-Level Architecture.

As show in figure 4.1, the proposed architecture has mainly two layers: Business layer and Data layer. Business layer has all the necessary components to fulfill the functional requirements discussed in section 4.1. Data layer allows to access the data stored in the database which are Process Fragments and their metadata. Initially the user of the application provides benchmark-related criteria through an XML file. The criteria is interpreted by the UserCriteria component of the Business layer in the application. The component then uses the criteria and retrieves the corresponding Process Fragments from the repository. After retrieving the Process Fragments, the application checks whether the Process Fragments can be joined with each other or not, based on the stored metadata in the repository. This is a compatibility check

and is done by the FragmentsCompatibility component. The SynthesizedProcess component then uses the data from the FragmentsCompatibility component to synthesize Process Model. The synthesized Process Model is then made Process Engine executable using EngineExecutable component. The third layer which is a Data layer have a DBInteraction component that allows to access all the Process Fragments and the associated metadata in the database. There is one more requirement in section 4.1 and that is to upload the Process Fragments and the metadata in the repository. UploadFragmentsAndMetadata component in the Business layer is responsible to fulfill that requirement. It is an isolated component from the other mentioned components in the Business layer.

4.3 Component Interaction

In this section, how different components of the prototype application interact with each other are discussed. Figure 4.2 shows the sequence diagram in which the overall flow of the application has been shown. First step is to provide the user-defined, benchmark-related criteria to the application. The criteria should be according to the already defined XML schema which is discussed in section 4.4.3. In this criteria, user can define different characteristics of the Process Fragments and different properties related to the synthesized Process Model.

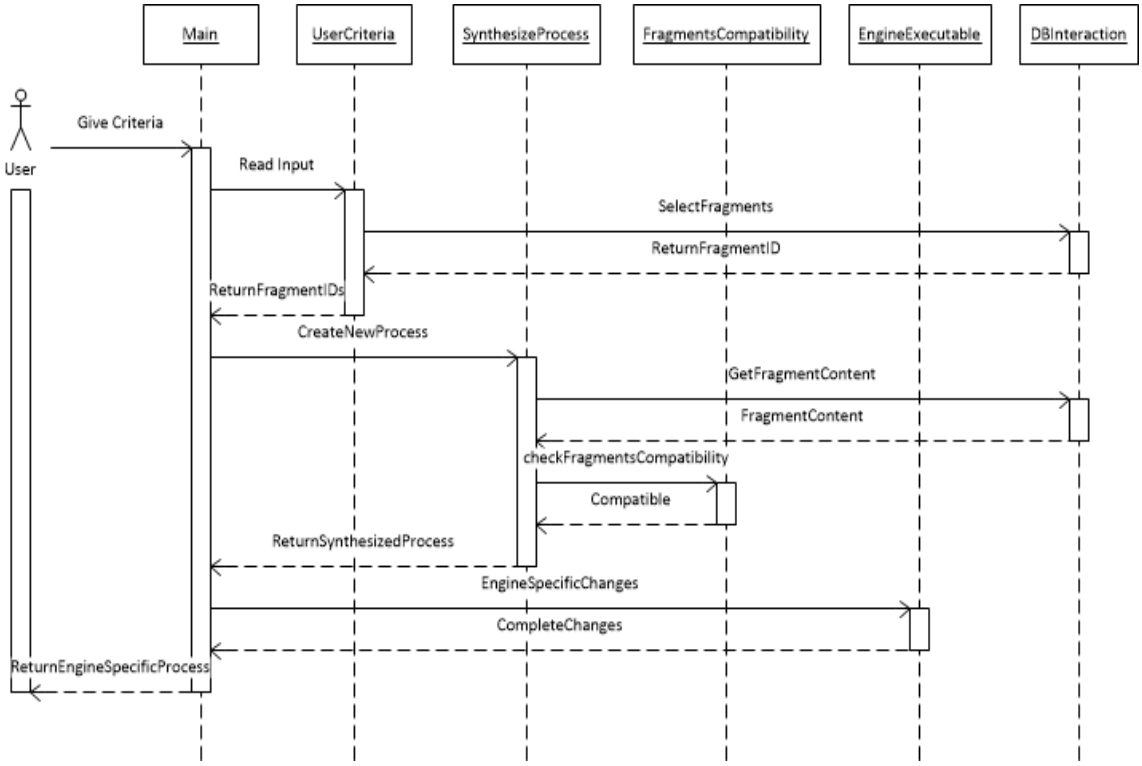


Figure 4.2: Sequence Diagram of the Application.

As shown in the figure 4.2, the user-defined, benchmark-related criteria is read and consequently all the inputs that have been defined by the user are determined by UserCriteria

component. Afterwards, the same component passes the inputs to the DBInteraction component which selects the corresponding Process Fragments from the database. For every input in the criteria, DBInteraction component returns the ID of the Process Fragment. Hence the IDs of all the Process Fragments constitute a list. This ID list is then passed to SynthesizeProcess component. Before synthesizing, the SynthesizeProcess component calls the FragmentsCompatibility component to verify whether the Process Fragments can be linked/joined together or not. The details of the compatibility check are explained in section 4.4. If the FragmentsCompatibility component returns true, then the Process Model is synthesized. If it does not return true, the user is prompted to change their defined criteria.

Once the Process Model is synthesized, it is then passed to the EngineExecutable component to make changes in the synthesized Process Model in order to make it Process Engine executable. In this thesis, the focus is on two different Process Engines (Camunda and jBPM), therefore the changes are according to either one of these two Process Engines as asked by the user. Finally, Process Engine specific synthesized Process Model is returned that is ready to be deployed on the Process Engine.

4.4 Fine-Grained Architecture

This section focuses on discussing the details of the proposed architecture which has been discussed in section 4.2. Initially, the class diagram which allows to store the Process Fragments along with the metadata and synthesizing an executable Process Model is discussed. Then, the Entity Relationship (ER) diagram of the database is discussed in which the Process Fragments and corresponding metadata are stored. Lastly, the XML schema associated with the user-defined, benchmark-related criteria is also discussed.

4.4.1 UML Diagram

Figure 4.3 shows the class diagram of the prototype application. The application evolves around main class.

Upload Fragments and Metadata Component

First part of the application is to upload the Process Fragments. This is UploadFragmentsAndMetadata component in section 4.2. The component comprises of a FileLoad class which allows to upload the Process Fragment. Two already implemented classes (BPMNCollectionSingleton and ModelInstance) have been extended and used in this thesis from the [SGHL15] work. The implementation of the FileLoad class has been extended from the class BPMNCollectionSingleton which allows to hold all the Models (in this case Process Fragments) as Java objects. FileLoad class scans a directory and retrieve all the Process Fragments in the specified directory. Before uploading the Process Fragments in the repository, there is a requirement to find out and store the metadata of the Process Fragments. The metadata focused in this thesis are:

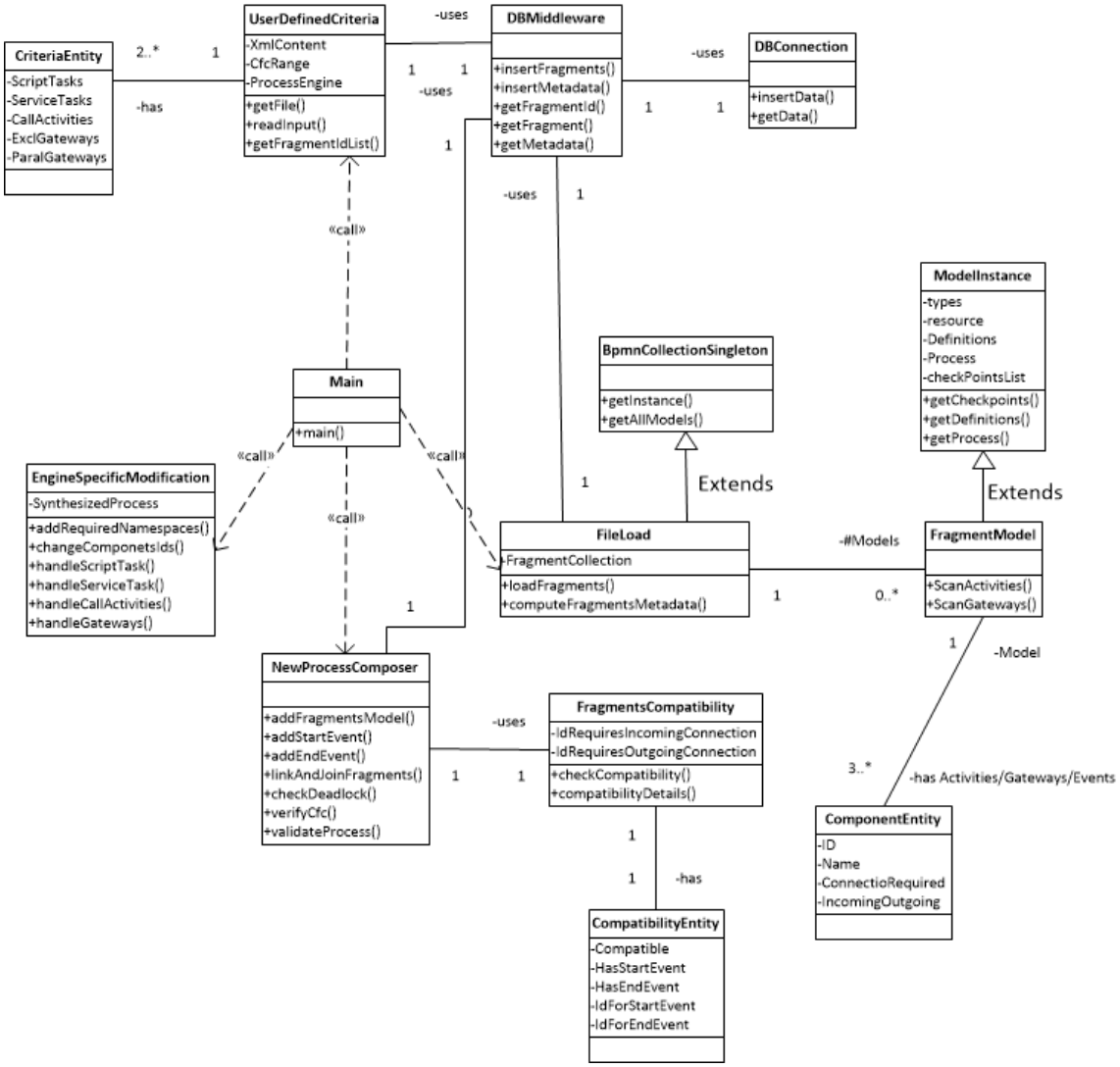


Figure 4.3: Class Diagram of Implementation.

- Is there any start event in the Process Fragment?
- Is there any end event in the Process Fragment?
- Does any task require additional connection in order to synthesize a valid Process Model?
- Is any gateway require additional connection in order to synthesize a valid Process Model?

This metadata enables the synthesizing of the Process Model. The connections required mean whether the task incoming and outgoing connections are according to the BPMN 2.0 standard [JEay]. For example, a task should have atleast one incoming and one outgoing connection in order to be considered as valid [JEay]. In this thesis, task means either script task, service task or call activity. Similarly gateway corresponds to parallel gateway and exclusive gateway. To get the metadata out of the Process Fragment, FragmentModel class

is used which is extending the implementation of the ModelInstance class. This is also part of UploadFragmentsAndMetadata component in section 4.2. ModelInstance class represent Model as Java objects. FragmentModel class uses ModelInstance class to represent the Process Fragments as Java objects and then those Java objects are scanned to find out all the metadata. ScanActivities and ScanGateways methods in the FragmentModel scan all the tasks and gateways to find out if those components in Process Fragments require any additional connections. Every component in the Process Fragment is translated as Java object of the ComponentEntity class as shown in the class diagram 4.3. For scanning, there is a need to write down conditions, so a rule engine is used. This rule engine allows to write the conditions independent of the business logic of the code. The details of how a rule engine works is explained in section 5.1.1. Example of the rule is; if a task has one incoming connection and no outgoing connection, then it means that the task require one outgoing connection to the other component in order to be considered as valid according to the BPMN 2.0 standard [JEay]. Similarly, if there is a gateway which has two outgoing connections and no incoming connection, it means that it is a split gateway and it must have one incoming connection with another component in order to be considered as valid [JEay]. All the metadata is then stored in the repository. The schema of the repository has been discussed in section 4.4.2.

Synthesize Process Model and Fragments Compatibility component

After storing all the Process Fragments and associated metadata, next part is to synthesize Process Model based on the user-defined, benchmark-related criteria. The user of the application provides the criteria through an XML file which should be according to the defined XML schema as discussed in section 4.4.3. According to the XML schema discussed in section 4.4.3, user-defined criteria should have atleast two fragment tags, which means that the criteria should be provided for atleast two Process Fragments. The maximum number could be unlimited. For every criteria under the fragment tag, a Java object of class CriteriaEntity is created. These classes are part of UserCriteria component in section 4.2. These Java objects are then passed to DBMiddleware class which then translates them into the corresponding SQL query. This query is executed on the database to search the Process Fragment. The DBMiddleware class is part of DBInteraction component in section 4.2.

After selecting the Process Fragments from the repository, next task is to synthesize Process Model. Before synthesizing, checkCompatibility method of the FragmentsCompatibility class is called which checks whether the selected Process Fragments can be joined together or not. The detail list of all the checks and the pseudo code for compatibility check is discussed in section4.5.

If the compatibility check returns false, then the user is prompted to change its criteria. If it is true, then the Process Fragments list is passed to the NewProcessComposer class. NewProcessComposer class then use the details from the FragmentsCompatibility class to synthesize Process Model. As mentioned earlier, the Process Fragments are treated as Java objects therefore initially, all the Java objects are added. Then, the connections are made for all those components which require additional connections (metadata). Afterwards, if there is a need to add start event or end event, that event is added to synthesize a valid Process Model.

The requirement to add start/end event is also received from the FragmentsCompatibility class. A condition for deadlock is also checked through method checkDeadlock in this class. This condition has been explained in detail in section 5.1.4. Two additional checks are also made, first one is to check whether the Control Flow Complexity (CFC) of the synthesized Process Model lies within the range as defined by the user (verifyCFC method). Second one is to check whether the synthesized Process Model is realistic in it's characteristics (number of tasks, maximum fanout value of gateway etc). These two conditions are also explained in more details in section 5.1.4.

Process Engine Executable component

After synthesizing the Process Model, last task is to make the synthesized Process Model, Process Engine executable. This is done using EngineSpecificModification class. The name of the Process Engine is provided by the user through XML file as discussed in section 4.4.3. Based on this input, an instance of the EngineSpecificModification is called (CamundaSpecificModification, JbpmSpecificModification). Process Engine specific changes are made which include adding required namespaces in the definitions element of the synthesized Process Model, handle script tasks, handle service tasks etc. These changes are executed using the corresponding methods in the class; for example (handleScriptTask, handleGateways etc). The pseudo code is discussed in section 4.5. Further details of how these methods work are explained in section 5.2.4 and section 5.3.1.

4.4.2 Entity-Relationship Diagram of the Repository

Figure 4.4 shows the Entity Relationship diagram of the backend repository. The repository store the Process Fragments and the corresponding metadata. The Process Fragments are stored in the Fragments table column Content. Every Process Fragment in this table gets a unique ID. This unique ID is an integer and also a primary key for the Fragments table.

The Fragments table has a one to one relationship with the Metadata table. Every Process Fragment has a corresponding metadata stored in this table. HasStartEvent, HasEndEvent are the metadata associated with the Process Fragments and these are saved in this table. Every metadata record has it's own ID (integer) which is also a primary key. A foreign key from the Fragments table is also in the Metadata table to identify which metadata belongs to which Process Fragment.

The metadata is not just limited to this. All the tasks and gateways in the Process Fragments also have metadata. The key point here is that the focus is on the metadata which can help to synthesize Process Model. Other metadata, for example the application area where the Process Fragments can be used, the type of the Process Fragments etc can be ignored because they do not contribute in synthesizing Process Model. The metadata related to the tasks is stored in the TaskMetadata table in which the component name (script task, service task or call activity) is saved. Connection Required column holds the value of how many connections are required by the component. There is another column IncomingOrOutgoing which holds either of the

two strings (incoming/outgoing). This column alongside the column Connections Required defines how many connections are required by the corresponding component and in which direction. Since a Process Fragment must have at least one task to be considered as valid Process Fragment, therefore it has a relationship of one to mandatory many with the metadata of a Process Fragment. Every row in this table also has a unique ID which is a primary key. The metadata related to the gateways is stored in the GatewayMetadata table. Since there is no restriction about the number of the gateways in a Process Fragment, therefore it has a one to optional many relationship with metadata of the Process Fragment. Further, the schema of the GatewayMetadata table is also slightly different than that of TaskMetadata. An additional column 'Type' is there in the GatewayMetadata table. Since the CFC value is dependent on gateways and type of the gateway, therefore this 'Type' column helps to compute the CFC value of the synthesized Process Model. The Gateway metadata also has columns like Connection Required and IncomingOrOutoging which has the same semantics as in the TaskMetadata. Every row in this table also gets a unique ID as primary key.

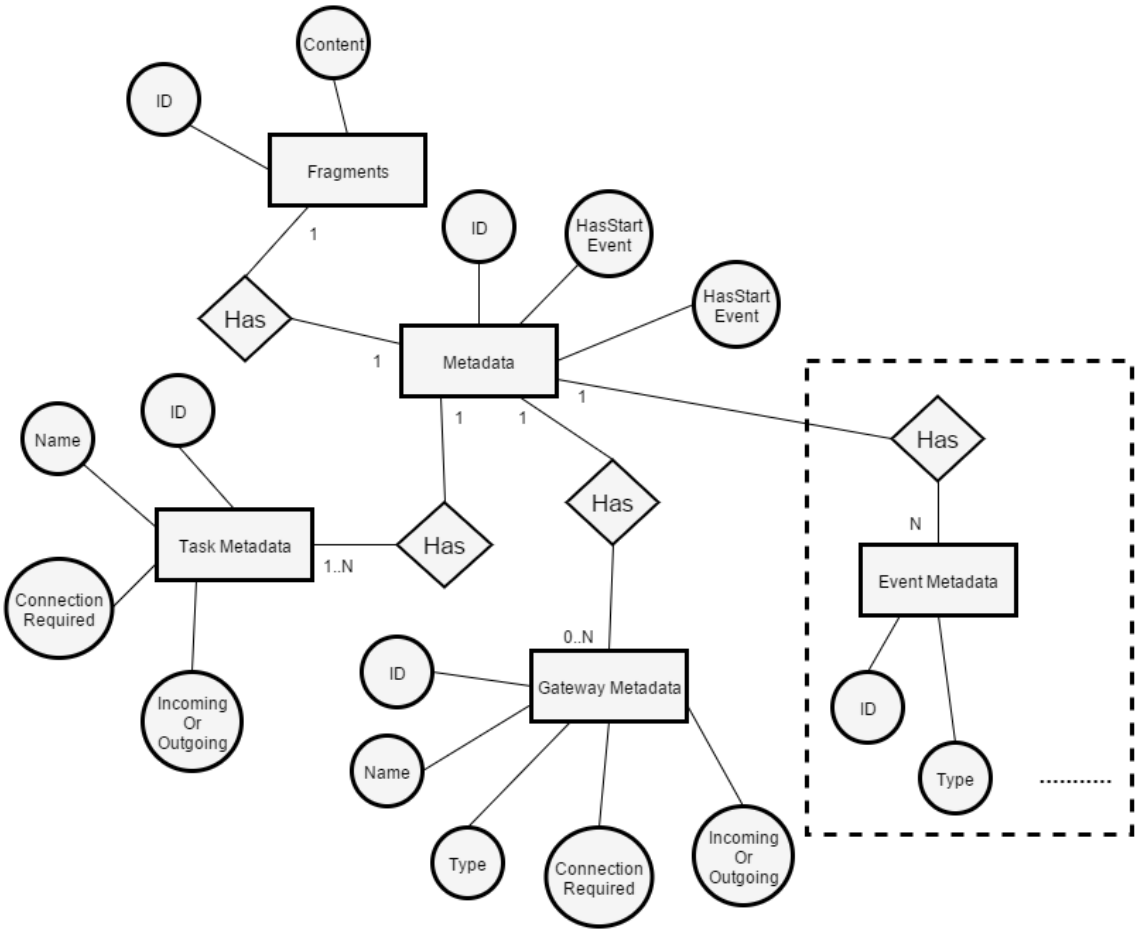


Figure 4.4: ER Diagram of the Implemented Repository.

As mentioned earlier, the focus in this thesis is with tasks (script task, service task and call activity) and gateways (parallel and exclusive). If at a later point in time, there is a requirement to add more components in this application like intermediate events, compensation activities etc which might have different semantics as compared to tasks and gateways, then there will

be a need to add more tables. That can be done without changing the whole schema as shown in dotted line in figure 4.4.

4.4.3 XML Schema for User-defined Criteria

To synthesize Process Model a user-defined,benchmark-related criteria is provided to the application by the user. This criteria is provided through an XML file. It is appropriate to describe the allowable XML document content/data by defining an XML schema document (XSD) because the XML schema validates the correctness of the data which in this case is user-defined criteria.

Figure 4.5 shows the XML schema which is used for the user-defined, benchmark-related criteria XML document. The XML schema has two main parts. First part is focusing on defining the Process Fragments characteristics and the second is focusing on the synthesized Process Model.

As seen in the figure 4.5, fragments tag has fragment and fragment is of FragmentType. The characteristics of every Process Fragments that requires to be selected from the repository are defined in this fragment tag. For example, number of script tasks, service tasks, call activities, exclusive gateways, parallel gateways. In order to further simplify for the user, there is also a tag 'activities' in which a user can define the total number of tasks (any combination of the summation of script tasks, service tasks and call activities) instead of providing details.

In the second part which focuses on the synthesized Process Model, user can define that the synthesized Process Model CFC must be within the range that has been asked by the user. That range is defined as attributes (Min and Max) of the CFC tag as shown in figure 4.5. There is one more significant property that a user can set and that is the Process Engine name. The synthesized Process Model is then made Process Engine executable for the defined Process Engine. Since the focus in this thesis is on Camunda and jBPM, therefore user can only write any of these two Process Engine names.

4.5 Algorithms

In this section pseudo code of the algorithms which are used in this application are discussed. First algorithm is related to check the compatibility of Process Fragments which are used to synthesize the Process Model. As discussed in section 4.4, there are some rules/conditions which are checked to verify the compatibility. The detail list of rules are below:

- **Rule 1**: If the number of outgoing connections required by the first Process Fragment are equal to the number of incoming connections required by the second one and similarly, the number of outgoing connections required by the second one are equal to the number of incoming connections required by the third one and so on then the Process Fragments are compatible. Consider there is a set having;

  N1,N2,N3... N Fragments

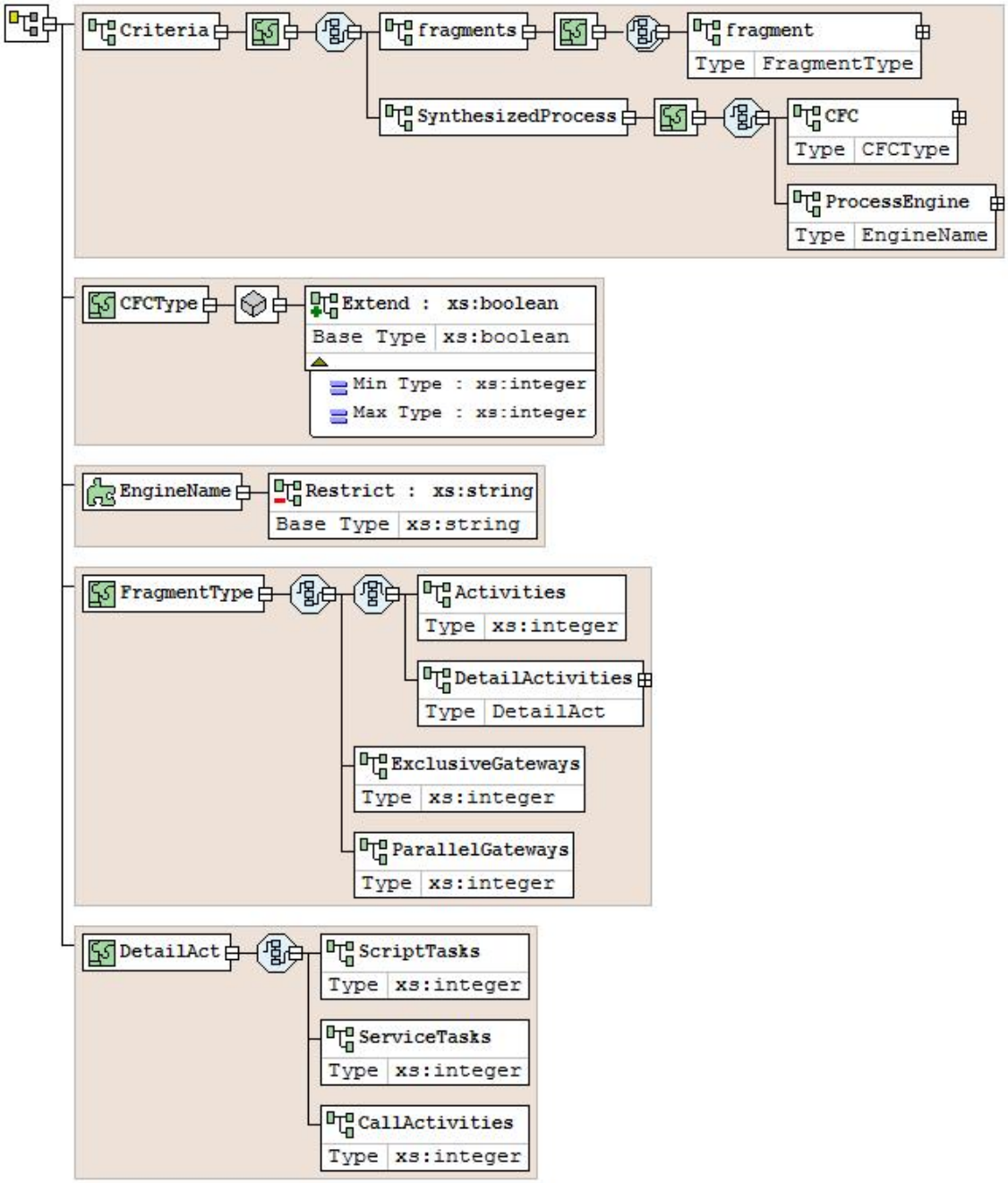


Figure 4.5: XML Schema for user-defined, benchmark-related criteria.

and which are selected from the repository based on the user defined criteria. Now consider the following subsets of this set;

$\{N1, N2\}, \{N2, N3\}, \{N3, N4\}\ldots$

We pick every subset and verify condition such that if N1 requires 'x' number of outgoing connections and N2 requires 'y' number of incoming connections and so on for $\{N2, N3\}, \{N3, N4\}\ldots$ then the first condition of Fragments compatibility is true

if and only if 'x' is equal to 'y' for every subset. Since the focus is only on the flow of the Process and not on the functional semantics of tasks (service task/script task/call activity) therefore "number of connections required" information is enough for this step.

- **Rule 2**: Two additional checks have to be made here: (1) Whether N1 Process Fragment from the set of N Process Fragments has a start event or not, (2) Whether Nth Process Fragment from the set of N Process Fragments has an end event or not. This information is passed forward so that if there is a need of having start or/and end event, those events could be added.

Algorithm 1 Fragments Compatibility algorithm

```
  procedure CHECKCOMPATIBILITY
Input: FragmentsList [N1, N2, N3...N]
Output: CompatibilityEntity Object
    if (Fragment N1 has startEvent) then
        CompatibleObject.setHasTrue = true
    else
        CompatibleObject.setHasTrue = false
    if (Fragment N has endEvent) then
        CompatibleObject.setHasEnd = true
    else
        CompatibleObject.setHasEnd = false
    for each Fragment N2..N − 1 do
        if (Fragment has startEvent || has endEvent) then
            CompatibleObject.compatible = false
     ......
    return CompatibleObject
```

- **Rule 3**: If N1 Process Fragment has a start event and it also requires one or more incoming connection, then this is also not compatible. Vice versa for the end event, i.e, if Nth Process Fragment has an end event and Nth Process Fragment also requires one or more outgoing connection then the compatibility is false. Further, if N1 requires one incoming connection and there is no start event, then a flag is set as true which tells the synthesizer that a start event should be added in order to synthesize a valid Process Model and vice versa for the end event.

- **Rule 4**: All the Process Fragments which are in the middle, from N2 to N-1th Process Fragment in the set of N Process Fragments should not have any start or end event. Since, the focus in this thesis is on synthesizing Process Models having one start and end event, therefore it is a violation for that.

- **Rule 5**: If any of the previous requirements are not fulfilled, then the Process Fragments are not compatible and no Process Model is going to be synthesized. Instead, a message is shown to the user that the Process Fragments are not compatible and the user must change it's defined criteria.

The pseudo code of executing the defined rules are shown in algorithm1.

Next algorithm is related to how Process Engine specific changes are imposed on the synthesized Process Model. For every Process Engine, the requirements are same (add required namespaces, handle script tasks, handle service tasks, handle call activities, handle gateways conditions) as discussed in section 4.4 but the syntax is different.

Algorithm 2 Process Engine Specific Changes algorithm

```
  procedure ENGINESPECIFICCHANGES
Input: Synthesized Process Model
Output: Engine Executable Process Model
    ADDNAMESPACES(ProcessModel)
    for each Element E in ProcessModel do
        CHANGEID(E)
    for each Task T in ProcessModel do
        if (T is scriptTask) then
            HANDLESCRIPTTASK(T)
        if (T is serviceTask) then
            HANDLESERVICETASK(T)
        if (T is callActivity) then
            HANDLECALLACTIVITY(T)
    for each Gateway G in ProcessModel do
        if (G is splitExclusiveGateway) then
            ADDGATEWAYSPLITCONDITION(G)
    return EngineExecutableProcessModel
```

The algorithm 2 shows how those changes are executed for a synthesized Process Model.

5 Implementation

This chapter discusses the implementation of the prototype application. The application has two main functionalities from implementation perspective, the first is to synthesize Process Model and second is to do changes in the synthesized Process Model with respect to Process Engines. The chapter has been divided in four parts.The first part explains the synthesizing Process Models in section5.1, second part explains the automatic deployment of synthesized Process Model on the Process Engine (Camunda) in section5.2, third part explains Process Engine specific changes of the synthesized Process Model for jBPM Workflow Engine in section 5.3 and finally, the last part in section 5.4 explains the limitations of the prototype application.

5.1 Synthesize New Process

The main goal is to synthesize Process Model from the Process Fragments which are stored in the repository. The overall steps to achieve this are shown in figure 5.1

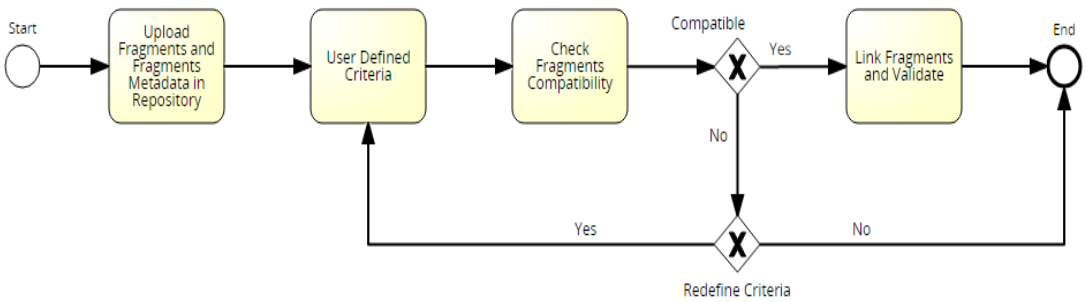


Figure 5.1: Steps to synthesize Process.

First, the Process Fragments along with their metadata needs to be uploaded on the database in a structured manner as discussed in the chapter 4. After uploading all the Process Fragments and their metadata, the user provides a criteria to the application. Then, the application determines whether the Process Fragments are compatible with each other or not based on their metadata which has been stored in the database. If the Process Fragments are compatible, a Process Model is synthesized by joining the Process Fragments together.

In the following sections the details of, how based on the user-defined, benchmark-related criteria a Process Engine specific Process Model is created and returned and in the case of Camunda a package is created and deployed on Camunda Process Engine, is discussed.

5.1.1 Upload Fragments and Metadata

First task is to upload all the Process Fragments on the database. MySQL database (v5.6.24) is used. The Process Fragments are serialized in XML files. To upload the Process Fragments, first the application scans the particular directory and gets all the Process Fragments in the directory. Then one by one, every Process Fragment is passed to DatabaseMiddleware class which creates an SQL query with corresponding values and passes it to the Database connection class which eventually executes the query. Every Process Fragment in the database has a unique ID which is automatically generated by the database.

Process Fragments need to be accompanied with their metadata in the database. The metadata is of utmost importance because that helps us to determine whether the Process Fragments can be tied together or not. The design of the database in which the metadata is stored has been discussed in the section 4.4.2. The metadata is automatically extracted from the Process Fragments. The metadata which is required in this thesis is discussed in the section 4.4.

First the Process Fragment is parsed to find out if there is any start event. If found, the metadata in the database is set to true. Same steps are replicated for the end event. Both of these information are stored in the metadata table of the database.

Next step is to find out all the tasks in the Process Fragments. As mentioned in the chapter 4, task means any one of the script task, service task or call activity. For every task, the name of the task (service task/script task/call activity), the component ID of the task and whether the task (service task/script task/call activity) requires any connection with other component are saved in the database as discussed in the chapter 4. Each component is also examined to figure out if it requires any incoming or outgoing connection.

Since these kind of scanning requires writing down the conditions, therefore, as discussed in section 4.4 a rule engine has been used. Drool rule engine(v5.2.0)[1] allows one to write all the rules in the rule file independent of the business logic so that rules can easily be added, removed or even modified. Using a rule engine is a much easier approach in terms of complexity and flexibility, as compared to writing down all the if/else conditions. For our thesis, two rule files have been written, one for tasks(service task/script task/ call activity) and the other one for the gateways (parallel/exclusive). All the conditions regarding the requirement of additional connections for the component have been written down in these rule files. When scanning of each task takes place, all the rules in the task rule file are fired. Based on the matching of the fired rule, number of required input/output connections are returned. These values are very critical to achieve the goal of synthesizing the Process Model. In the database, the values are saved according to the schema defined in the section 4.4.2. In the figure 5.2 one of the task rule is shown according to which the number of connections required is zero which means that if this rule is fired, i.e, task has one incoming and one outgoing connection then the number of connections are exactly according to the BPMN 2.0 rulebook and is completely valid.

After scanning the task, next step is to scan the gateways. As mentioned earlier, in this thesis the focus is on two gateway elements; exclusive gateway and parallel gateway. The schema

[1] `https://docs.jboss.org/drools/release/5.2.0.Final/drools-expert-docs/html`

```
rule "Activities no conn"
    when
        $node: FlowNode()
        $obj: ComponentEntity()
        eval($node.getIncoming().size()>=1)
        eval($node.getOutgoing().size()>=1)

    then
        $obj.setconnrequired(0);
    end
```

Figure 5.2: Example of a Task Rule in Rule File.

of the TaskMetadata table and GatewayMetadata is not similar and also the behavior and nature of these two components are totally different, therefore their metadata has been saved in totally different tables as discussed in the section 4.4.2. The overall steps to scan every gateway (exclusive/parallel) are also similar to the steps mentioned for scanning the tasks. The main difference is in the content of the rule file of gateway, which says that if there is some gateway component, it must be either a split one or a join one which means that the gateway must have multiple incoming or outgoing connections in order to be considered as valid in the Process Model. For that purpose, a different rulebook file has been added. All the rules regarding the gateways (exclusive/parallel) have been defined in this rulebook.

```
rule "Gateway 1 outgoing required"
    when
        $node: FlowNode()
        $obj: ComponentEntity()
        eval($node.getIncoming().size()>=2)
        eval($node.getOutgoing().size()<1)

    then
        $obj.setconnrequired(1);
        $obj.setincomingoutgoing("outgoing");
    end
```

Figure 5.3: Example of a Gateway Rule in Rule File.

For example in the figure 5.3, one of the gateway rule is shown which returns 1 and 'outgoing' if it is triggered. The rule says that if there is a gateway which has 2 incoming connections and no outgoing connections, it means that it is a join gateway and it requires atleast 1 outgoing connection in order to be considered as valid.

Similar to the tasks scanning scenario, these rules are fired for every gateway(exclusive/parallel) component and then, the corresponding metadata is stored in the database.

5.1.2 User Defined Criteria

After uploading the Process Fragments and storing corresponding metadata in the repository, user of the application can provide a benchmark-related criteria through an XML file to synthesize Process. XML file has to obey the XML schema which has been explained in the section 4.4.3. User through XML file can select as many Process Fragments (N number of Fragments) out of the repository as it wants but the user must provide the criteria for atleast two Process Fragments. This condition is enforced through XML schema as shown in figure 5.4. minOccurs has been set to 2 which means that there must be atleast two fragment tag. maxOccurs has been set to unbounded which means that the user can define as many Process Fragments criteria as it wants.

```
<xs:element name="fragments">
<xs:complexType>
<xs:sequence minOccurs="2" maxOccurs="unbounded">
<xs:element name="fragment" type="FragmentType"/>
</xs:sequence>
</xs:complexType>
</xs:element>
```

Figure 5.4: Fragment Tag in XML Schema.

For every Process Fragment, user provides how many script tasks, service tasks, call activities, exclusive gateways and parallel gateways it should have. To make it simple for the users, there is also a possibility for users in which they can define total number of tasks (sum of script, service task and call activities) instead of providing details as shown in figure 5.6.

In figure 5.6 user has provided the criteria for two Process Fragments. For the first Process Fragment, user has provided details like number of service tasks should be 0 and number of call activities should be 10. For the second Process Fragment, user did not provide any details and instead asked for a Fragment that should have 3 activities (means sum of any combination of script, service task and call activities should be 3).

After defining the Process Fragments, user can also define which Process Engine specific Process Model it wants. Since the focus is on two Process Engines (Camunda and jBPM), therefore user can only write either 'Camunda' or 'Jbpm'. This condition has been imposed in the XML schema using enumeration type as shown in the figure 5.5.

```
<xs:simpleType name="EngineName">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Camunda"/>
      <xs:enumeration value="Jbpm"/>
    </xs:restriction>
</xs:simpleType>
```

Figure 5.5: Process Engine Name in XML Schema.

As an example, again consider the criteria in figure 5.6 in which user has asked for jBPM Workflow Engine specific Process Model. And lastly, user can also define the constraint on the synthesized Process Model by providing the range on Control Flow Complexity (CFC) values. For example, the synthesized Process Model CFC value must be in the range 0-50 as shown in figure 5.6.

```
<Criteria xmlns="http://www.w3schools.com" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <fragments>
    <fragment>
    <DetailActivities>
        <ScriptTasks>0</ScriptTasks>
        <ServiceTasks>0</ServiceTasks>
        <CallActivities>10</CallActivities>
      </DetailActivities>
      <ExclusiveGateways>2</ExclusiveGateways>
      <ParallelGateways>0</ParallelGateways>
    </fragment>
    <fragment>
    <Activities>3</Activities>
      <ExclusiveGateways>2</ExclusiveGateways>
      <ParallelGateways>0</ParallelGateways>
    </fragment>
  </fragments>
  <SynthesizedProcess>
    <CFC Min="0" Max="50">true</CFC>
    <ProcessEngine>Jbpm</ProcessEngine>
  </SynthesizedProcess>
</Criteria>
```

Figure 5.6: Example of User Defined Criteria.

After defining all of the above, the application reads the user-defined, benchmark-related criteria and accordingly selects the Process Fragments out of the repository using SQL query as shown in listing 5.1.

```
1 select fragmentsrepository.metadata.fid as ID
2 from fragmentsrepository.metadata LEFT JOIN fragmentsrepository.
    taskmetadata
3 on fragmentsrepository.metadata.mid = fragmentsrepository.
    taskmetadata.mid
4 AND
5 fragmentsrepository.taskmetadata.taskname IN ("scriptTaskImpl","
    ServiceTaskImpl","CallActivityImpl")
6 group by fragmentsrepository.metadata.mid
7 having count(fragmentsrepository.taskmetadata.taskname)='"+
    criteria.getTotalTasks()+"'
```

Listing 5.1: SQL Query to Select Process Fragment.

The SQL query shown in the listing 5.1 selects the ID of those Process Fragments which have the total sum of script,service task and call activity equals to the value which has been defined by the user in the criteria. If more than one Process Fragment fulfills the criteria, then the application selects a random ID from the list.

After getting the list of Process Fragments, next task is to join them together (if possible) as explained in the below subsections.

5.1.3 Check Fragments Compatibility

After the specification of the user-defined, benchmark-related criteria, a set of Process Fragments are selected from the repository and returned. Now the next step is to find out whether those Process Fragments are compatible with each other or not. This is the core step to synthesize Process Model from the Process Fragments. The main aspect which is used in this step is the number of connections required by a Process Fragment. This is the total number of connections required by all the tasks (service task/script task/call activity) and the gateways(exclusive/parallel) of the Process Fragments as well as the direction in which those connections are required (incoming/outgoing).

```
select sum(taskmetadata.connrequired) as num
from taskmetadata join metadata
on taskmetadata.mid=metadata.mid
and metadata.fid='"+fID+"'
and taskmetadata.IncomingOutgoing='"+IncOrOut+"'
```

Listing 5.2: SQL Query to get the Connections Required.

For example, the SQL query in the listing 5.2 returns the connections required by a particular Process Fragment (fID) in a specified direction (IncOrOut). The total number of connections required, are computed based upon the BPMN 2.0 specification and already stored in the database as discussed in section 5.1.1. For compatibility check, there are set of conditions which needs to be checked in order to find out the whether the Process Fragments are compatible to be linked together or not. The list of conditions are explained in section 4.5. These conditions are checked here.

5.1.4 Join Fragments and Validate

If the Process Fragments in the previous step are found to be compatible, then all the Process Fragments should be joined or linked together as discussed in chapter 4 in order to synthesize Process Model. The overall flow to achieve this is shown in figure 5.7:

First the Process Fragments are combined in order to make one new Process Model, then there is a parameter "isExecutable" which is going to be set as true. After that, if there is some requirement to add start/end event then that event is added and after that the missing

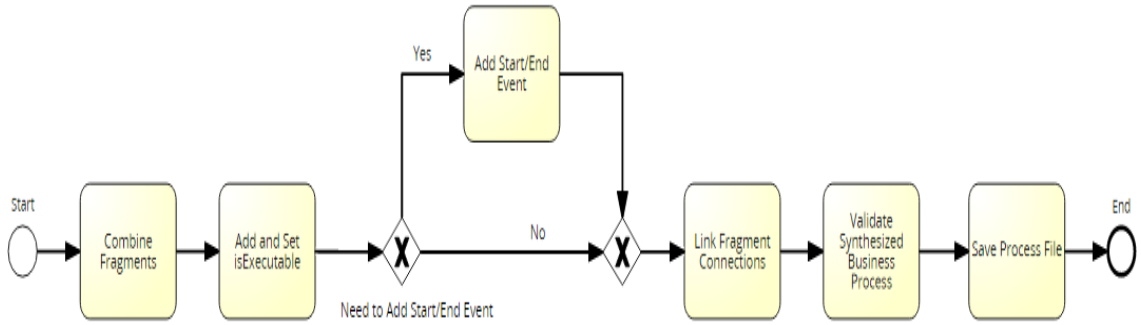


Figure 5.7: Join Fragments and Validate Synthesized Process.

connections between the Process Fragments are made. Lastly, the file is saved in a particular directory as a new Process (BPMN 2.0) Model.

Combine Process Fragments

As discussed in the section 4.4 of the design chapter 4, Process Fragments are realized as Java objects. Initially all the objects are added to generate one object which is going to be the synthesized Process Model as shown in figure 5.8. This is basically in mathematical terms a

```
private FragmentModel AddModels(List<FragmentModel> models)
{
    FragmentModel modA=models.get(0);
    FragmentModel modB=models.get(1);
    FragmentModel finalmod = addtwoModels(modA, modB);

    for (int i=2;i<models.size();i++)
    {
        finalmod=addtwoModels(finalmod, models.get(i));
    }

    return finalmod;
}
```

Figure 5.8: Add Process Fragments Objects.

union of all the Process Fragments, which means that there is one single root element of the synthesized Process Model, one single definition element and all the components from each Process Fragment along with all of their sequence flows.

Add and Set isExecutable

Since the aim is not just to synthesize Process Model but also to make them Process Engine executable, therefore after combining the Process Fragments, there is one parameter which needs to be set as true according to BPMN 2.0 standard definition in order to make the synthesized Process Model executable. The name of this parameter is "isExecutable" and it should be equal to true.

Add Start/End Event

As discussed in 5.1.3, there might be a scenario in which there is a need to add start/end event automatically in order to synthesize a valid Process Model. This addition (of start/end event) is made at this point (if required).

Link Fragments Connections

The next step is to link all the Process Fragments together. This linking is done by adding appropiate incoming/outgoing connections against a component and then, by adding a new sequence flow. For example, again consider that there are N1,N2,N3... upto N Process Fragments as discussed in section 4.5 and they are found to be compatible, now if N1 requires 'x' outgoing connections and N2 requires 'y' incoming connections, then those connections need to be established. Similarly, for other Process Fragments subset as well (N2,N3) and so on. This establishing connections is done here.

Validate Synthesized Process

Next step is to validate the synthesized Process Model. Following validation conditions are imposed:

- As discussed in 5.1.2 that user can impose the CFC value range and the synthesized Process Model CFC value must lie within the defined range. This check is going to be ensured here at this point. The CFC value of the synthesized Process is going to be calculated and compared with the user defined criteria.
- There could be a deadlock scenario in the synthesized Process Model and that would make the Process Model invalid. A classical deadlock scenario is shown in figure 5.9

  In the figure 5.9 parallel gateway always waits for the second token at its input but since there was an exclusive gateway in the start therefore that second token never arrives at the input of the parallel gateway and the Process remains idle there forever. To ensure that this deadlock does not exist in the synthesized Process Model, the application count all the split exclusive and parallel gateways and similarly, count all the join exclusive and parallel Gateways. If the count of split exclusive gateway is equal to the count of join exclusive gateway and same for parallel gateway then the synthesized Process

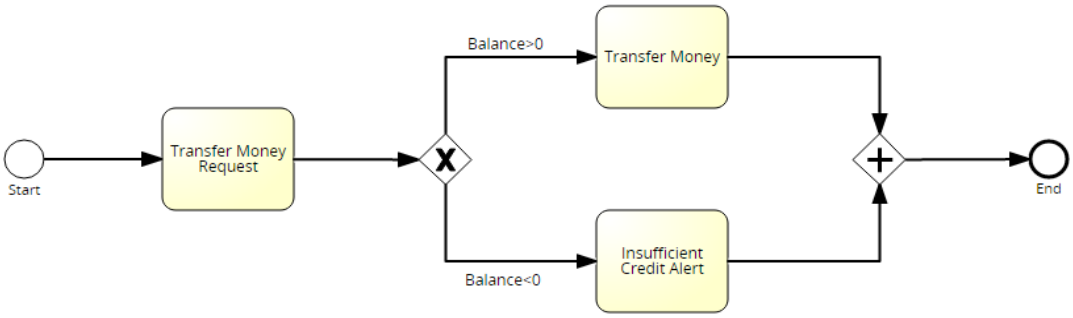


Figure 5.9: A Process with Deadlock.

Model is considered as valid and checkDeadlock method returns false otherwise invalid and the method return true as shown in figure 5.10.

```
return(!(SplitParallel==MergeParallel && SplitExclusive == MergeExclusive));
```

Figure 5.10: Condition to Check Process Deadlock.

- [Iva14] analyzes real world IBM Business Processes and it states that more than 90% of the Business Processes do not have more than 18 tasks. It means that sum of all the tasks (script task, user task, call activity and others) are not more 18. Similarly, [Iva14] also analyzed that the maximum number of the sum of all gateways(exclusive, parallel and others) are not usually more than 13 in a single Business Process. Moreover, the maximum Fanin value (number of incoming connection) and Fanout value (number of outgoing connections) of gateways (exclusive/parallel or other) are also observed to be 5. There are some more metrics which have been analyzed by the [Iva14] but in this thesis only above mentioned are handled. These parameters are called validation parameters in this thesis for the synthesized Process Model. It is important to consider all those parameters because as said in the chapter 3 that the aim is to do benchmarking and it is necessary that the benchmarking is done with the Process Models that look realistic. For that purpose, an additional class has been written, in which all these parameters threshold are defined and after synthesizing Process Model the application checks, whether those validation parameters conditions have been violated or not. For example, the total number of tasks (script,service task and call activity) in the synthesized Process Model should not be more than 18 otherwise, it violates the condition. If there is some violation in the synthesized Process Model, the Process Model is still going to be synthesized but in addition a message is shown to the user that the threshold values have been surpassed.

Save Process File

Linking the Fragments result in a new valid Process Model. As mentioned, the Process Model is an XML file and it has to be saved in some directory in order to reuse it again at a later

point in time. The path where the file should be saved is provided upfront and the file is saved there.

5.2 Camunda Engine Deployment

Tomcat Application server (v7.0.62) is used in our thesis to deploy synthesized Process Model on Camunda (v7.3). Camunda is discussed in detail in section 2.4.1.

After synthesizing the Process Model, the next step is to make the synthesized Process Model, Process Engine executable. But for Camunda, not only Process Engine executable Process Model is created but also packages are created automatically that allows to deploy the Process Model on Camunda more quickly. Tomcat server for Camunda needs '.war' packages of Process Model as discussed in section 2.4.1 that needs to be automatically generated in order to automate the whole deployment. '.war' package has some specific files that needs to be generated at runtime. Those files need to be stored in a particular directory. Synthesized Process Model also requires changes based upon the Process Engine and lastly the '.war' package needs to be copied to the Camunda Tomcat server.

The overall steps to achieve the output are explained with the figure 5.11.

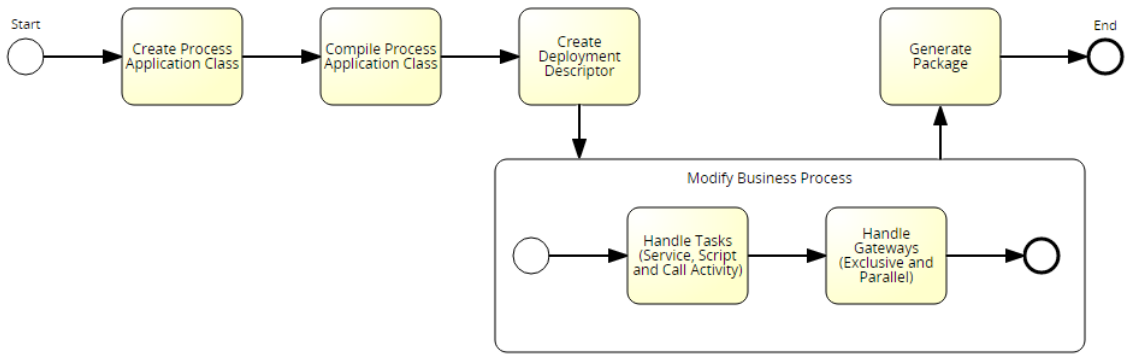


Figure 5.11: Process to generate .war Package.

The first step in the figure 5.11 is to create a Java class which is a Process Application class then the class file is compiled. Next step is to create the deployment descriptor file and modify the synthesized Process Model according to the Process Engine requirements. Lastly, all the files are grouped together in a proper structure which is explained below to create a package that is going to be deployed on the Process Engine application (Tomcat) server.

5.2.1 Create Process Application Class

After Synthesizing the Process Model, the goal now is to deploy the synthesized Process Model automatically on Camunda. For that purpose a '.war' package is required. Every '.war' package should have a Java class which is a Process Application class and according to

Camunda "The Process Application class constitutes the interface between your application and the Process Engine." [GmBay]

Since Tomcat Server is used, therefore ServletProcessApplication class is used as base class to develop Process applications. Further, since the whole process of creating the '.war' package is automated, therefore the Process Application class file needs to be created through code and for that purpose, Abstract Syntax Tree framework(v3.3.0)[2] has been used. This framework allows to specify everything which is required in our case. For example, defining annotations for a class, defining import packages, defining class name and defining the base class name. The name of the class has been randomly generated at runtime. The number of Java class files are dependent on the number of synthesized Process Model in a particular directory because as mentioned earlier, every '.war' package should have it's own Java file.

5.2.2 Compiling the Process Application Class

After creating the Process Application Class, next step is to compile the Class because Tomcat server does not understand the (.java) class file. The compilation of (.java) class files could be done through the runtime compiler command in Java but due to non availability of the extensive help another approach has been used. In this approach, all the (.java) class files which have been created in the previous step are saved in the directory of a different Maven project[3] and then that different Maven project is compiled using command prompt command (mvn). After the compilation, all the corresponding compiled (.class) files of the Process Application class are saved in a particular directory and then those compiled files are added to the '.war' package.

5.2.3 Create Deployment Descriptor

A deployment descriptor file which is an XML file also needs to be added in the '.war' package to describe the configurations associated with the Process Model and the backend of Camunda Process Engine. Figure 5.12 shows properties related to the deployment of

```
<properties>
  <property name="isDeleteUponUndeploy">false</property>
  <property name="isScanForProcessDefinitions">true</property>
</properties>
```

Figure 5.12: Process Model Configurations in Deployment Descriptor File.

Process Model on Camunda. As shown, 'isDeleteUponUndeploy' is set to false, which means that if the Process Model is undeployed from the Camunda then the Process Model will not be deleted from the Process Engine and vice versa if set to true. It is possible to add more properties in the deployment descriptor. The list of other properties are mentioned in

[2] http://www.eclipse.org/articles/Article-JavaCodeManipulation_AST/
[3] https://maven.apache.org/

[GmBay]. In the scope of this thesis, a pre-defined template of deployment descriptor file has been used. If required, more properties can be added in the template file. Further, if for every Process Model, there is a requirement of a different deployment descriptor file then this could be done using Document Object Model (DOM) parser[4] in Java.

5.2.4 Modify Process Model - Camunda

Synthesized Process Model is a newly created BPMN 2.0 Process. To make it executable on a particular Process Engine, the BPMN 2.0 file requires some changes. We do not claim that these are the complete list of changes but these are recognized for the scope of this thesis. First step is to change the ID of all the components namely call activity, service task, gateways (exclusive gateway and parallel gateway), start event and end event in the synthesized Process Model. The reason is that already alloted ID to the Process Fragment components are not valid because most of them start from number and for Camunda the ID must start from a character.

After committing the ID changes, next step is to find out and handle all the call activities, script tasks and service tasks within the synthesized Process Model. Call activities are handled by adding a reference (name) to an already deployed Process in the Process Engine. For this thesis, the reference for the call activity is an already deployed dummy Process. Script tasks are handled by adding a script against script task element. Since the focus is not on the functional semantics of the nodes therefore an empty script is executed.

Service tasks can be handled through multiple ways:

- it can invoke a RESTful Web Service or an API
- can invoke a SOAP based Web Service
- can invoke Java code.

To invoke a RESTful Web Service, the URL of the service is provided and also the method (GET,POST,PUT,DELETE). Further, the input and output parameters are also mentioned. To define all these configurations, 'camunda' namespace is added in the synthesized Process Model.

```
<bpmn2:serviceTask id="node2" name="SE1">
<bpmn2:extensionElements>
<camunda:connector>
<camunda:connectorId>http-connector</camunda:connectorId>
<camunda:inputOutput>
<camunda:inputParameter name="url">http://localhost:8080/engine-
    rest/deployment</camunda:inputParameter>
<camunda:inputParameter name="method">GET
</camunda:inputParameter>
```

[4] https://docs.oracle.com/javase/tutorial/jaxp/dom/

```
<camunda:outputParameter name="statusCode">${statusCode}
</camunda:outputParameter>
</camunda:inputOutput>
</camunda:connector>
</bpmn2:extensionElements>
<bpmn2:incoming>sid-785230FB-C3A2-4B8F-943D-DB5FA56FD0B3
</bpmn2:incoming>
<bpmn2:outgoing>sid-1F031708-167D-4406-A257-AC16481C9DFB
</bpmn2:outgoing>
</bpmn2:serviceTask>
```

Listing 5.3: Service Task in Camunda.

Listing 5.3 shows how a RESTful Web Service is invoked for a service task in the Process Model.

SOAP based service can be invoked by using 'camunda:connector' element which means that for a particular service task, this connector should be invoked. The SOAP connector should have a SOAP connector ID which basically points to the SOAP Http Connector and the input/output parameters. The number of parameters could be any and within those parameters, SOAP message can be defined.

Lastly, service task could invoke some Java code which could be:

- a method in the class
- a class that implements Java Delegate[5]
- a value expression evaluator.

For that purpose, the class should be defined and a reference should be made in the BPMN 2.0 file where required using 'camunda:expression' or 'camunda:class' element.

Last step is to handle the exclusive gateways within the synthesized Process Model. Exclusive gateway requires condition on every outgoing edge (if it is a split one). A condition expression should be added under every sequence flow. The evaluation of this condition expression determines which path of the Process Model is followed. The condition expression is a random value in a variable which should be provided through Camunda user interface by the Camunda Process Engine administrator while starting the Process. The name and specification of the variable is written under the start event element of the synthesized Process Model and is randomly generated.

[5] https://docs.camunda.org/manual/7.3/guides/user-guide/

5.2.5 Generate Package For Deployment

After creating the deployment descriptor file, modified synthesized Process Model and compile version of Process Application class, next step is to create the '.war' package. The overall structure of the '.war' package is shown in the figure 5.13

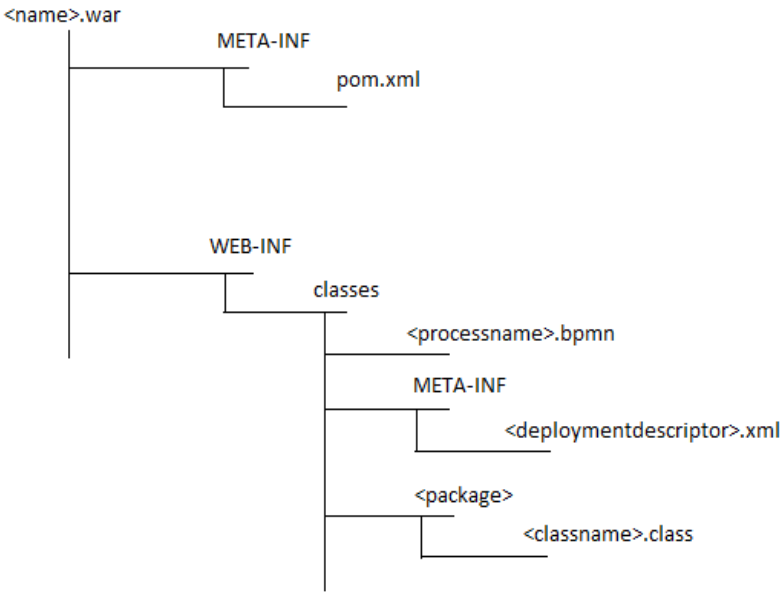


Figure 5.13: War Package Structure.

There are two main folders inside the package. The META-INF folder has the pom.xml file which has the record of all the dependencies. The other folder WEB-INF has Process Model, the compiled version of Process Application class and the deployment descriptor XML file. The creation of '.war' package has also been automated through code using the framework Arquillian shrinkwrap(v1.1.8)[6].The framework has extensive support for the developers to create the packages based upon their own requirement. '.jar', '.war' packages are the two mostly used by developers. Furthermore, within the package, developers can define the directory structure based on their own requirements.

5.2.6 Deployment on Server

Once the '.war' package has been created, next step is to deploy it on the Tomcat Application Server so that it can be run on Camunda. The deployment is assured by copying the already developed '.war' package to the 'Webapps' folder inside Tomcat application Server. The deployment can be automated with the help of FTP protocol which ensures the moving of

[6] `http://arquillian.org/guides/shrinkwrap_introduction/`

file from one machine to the other machine (Camunda Process Engine server). To verify whether the deployment has been successfully completed or not, log in to the web console of Camunda, synthesized Process Model name should be there, listed in the main section. The details of validating the deployment is discussed in section 6.3.

5.2.7 Deploying Mockup Services (REST)

In addition to the Process mentioned above, Camunda also provides RESTful API to interface all the important aspects of the Process Engine itself. For example, get all the deployments, delete a deployment or even post a deployment. Further, more information can also be extracted from the Camunda, for example get the total count of the tasks, get local task variable and others.[7]

To understand and verify the RESTful API feature, a dummy Process Model has also been deployed on Camunda. This is the second way to deploy Process Model other than the one in which '.war' package has been made and deployed. The downside of this method is that Camunda does not verify the correctness of the Process Model. For example, whether the conditions on the split exclusive gateway have been defined properly or not and so on. To deploy the Process Model via RESTful API and by using Java code, some parameters need to be passed in addition to the Process Model XML content. The additional parameters are: the name of the deployment, a boolean flag which basically tells whether the deployment should go ahead or not if the same Process has already been deployed. If set to true, rather then creating a new deployment, it will return the existing deployment.

5.3 jBPM Engine Specific Process Model

The focus is mainly on two different Process Engines in this thesis. First one is Camunda and the other one is jBPM (v6.2). After synthesizing Process Model, the Process Model itself requires some tailoring (similar to Camunda) in order to make it deployable on jBPM Process Engine. Tailoring means handling of service tasks, call activities, script tasks and gateways.

5.3.1 Modify Process Model - jBPM

After synthesizing Process, user can ask for a jBPM specific Process Model. For that purpose, there is a need to make some changes. We do not claim that these are the complete list of changes but these are recognized for the scope of this thesis. First step is similar to Camunda, change the ID of all the elements namely call activity, service task, gateways (exclusive and parallel gateway), start event and end event in the synthesized Process Model. The reason is that already alloted ID to the Process Fragment components are not valid because most of them have '-' and other special characters which should not be there when one wants to execute the Process Model on jBPM Engine. After committing the ID changes, next step

[7] `http://docs.camunda.org/latest/api-references/rest/`

is to find out and handle all the call activities, service tasks and script tasks within the synthesized Process Model. Call activities are handled by adding a reference (name) to an already deployed Process in the Process Engine. For that purpose, a dummy Process has been created and the application always refer to that dummy Process when there is a call activity. Next one is to handle script task; apart from setting ID and name which are typical parameters the other compulsory parameter which needs to be set is action. It is this parameter in which the associated script with that node is defined. Since the focus is not on the functional semantics of the nodes, therefore an empty script is executed. Next step is how to handle service tasks. They can be handled mainly through two ways:

- invoke Web Service
- invoke Java code (a method in the class).

Invoking Web Service is multiple steps process. First there is a need to tell jBPM Engine where the WSDL is so that it can be read. This can be done by using <import> element in the synthesized Process Model and by defining the location of the WSDL file in this <import> element. Next step is to define the operations and interface. This can be done using <itemDefinition> element in the synthesized Process Model. One can even define the message as well in this <itemDefinition> element. Finally, the exact operation which needs to be invoked is mentioned in the attribute of service task element in the synthesized Process Model.

The other way of handling service task is to invoke some method or operation of any Java class. For that purpose, first the Interface then the operations which are available through the Interface must be defined in the synthesized Process Model. Further, if operations require any message, that message should also be defined in the synthesized Process Model. And then in the service task element the operation name could be written to invoke that method as shown in listing 5.4.

```
<bpmn2:itemDefinition id="InMessageType" structureRef="java.lang.
    String"/>
<bpmn2:message id="InMessage" itemRef="InMessageType"/>
<bpmn2:interface id="ServiceInterface" implementationRef="com.
    sample.ClassService" name="com.sample.ClassService">
<bpmn2:operation id="ServiceOperation" name="serviceMethod">
<bpmn2:inMessageRef>InMessage</bpmn2:inMessageRef>
</bpmn2:operation>
</bpmn2:interface>

<bpmn2:serviceTask id="node7" implementation="Other" name="SE1"
    operationRef="ServiceOperation">
<bpmn2:incoming>sid-785230FB-C3A2-4B8F-943D-DB5FA56FD0B3
</bpmn2:incoming>
<bpmn2:outgoing>sid-1F031708-167D-4406-A257-AC16481C9DFB
</bpmn2:outgoing>
```

```
14 </bpmn2:serviceTask>
```

Listing 5.4: Service Task in jBPM.

In listing 5.4, a message 'InMessage' has been defined and it is of type InMessageType which is a string. Then an interface has been referenced and also the implementation of that interface. Then a method or operation has been referenced which has a name 'serviceMethod' and which takes message 'InMessage' as it's input. Finally, in the service task the ID of the operation has been referenced in order to invoke that method.

Last step is to handle the exclusive gateways in the synthesized Process Model similar to what is done for Camunda but the syntax is different for jBPM. Exclusive gateway requires condition on every outgoing edge (if it is a split one). A condition expression should be added under every sequence flow. The evaluation of this condition expression determines which path of the synthesized Process Model is followed. The condition expression is a random value in a variable. The type and the name of the variable has been randomly generated and written in the synthesized Process Model.

5.4 Limitations

Although the goals and the requirements which are mentioned in section 4.1 have been met yet there are some limitations in the implementation of the prototype application. Since the domain of BPMN 2.0 is very wide and the thesis is required to complete in six months, therefore following limitations exist in the application. As mentioned earlier, this thesis focuses on three types of tasks namely: script task, service task and call activity. Other types of tasks: user task, send task, receive task, manual task and transaction are not handled. Similarly, two gateway types are handled: exclusive gateway and parallel gateway. But there are other types of gateways that are not handled namely complex gateway, event based gateway and inclusive gateway. The next limitation is with the handling of exclusive and parallel gateway. As discussed in section 2.1 the gateways must have either multiple incoming connections or multiple outgoing connections.

But, if there is a scenario in which a Process Fragment has exclusive and parallel gateway as show in figure 5.14 (having one incoming and one outgoing connection) then in this prototype application these gateways are considered as split gateways. It means that the metadata in the repository tells that the gateway requires one outgoing connection. Whereas in reality, this type of gateway can be any of both (split or join). The next limitation is that there could be a scenario in which there are multiple start or end events but in this thesis the focus is to synthesize Process Models having one start event and one end event. These limitations could be addressed in the future work.

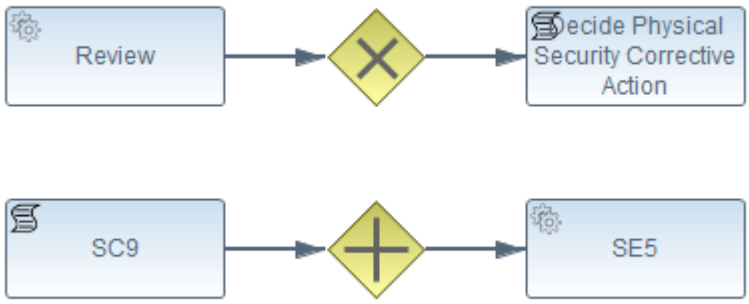


Figure 5.14: Limitation in Exclusive and Parallel Gateway.

6 Validation and Evaluation

In this chapter, the prototype application is validated. The validation is done to ensure that all the requirements which are listed in chapter 4 are fulfilled. All three core features of the application are validated. In section 6.1 the Process Fragments repository validation is done. Section 6.2 validates the synthesizing of Process Model and lastly, section 6.3 describes the validation of whether the Synthesized Process Model is Process Engine executable or not.

6.1 Process Fragments Repository - Validation

The application stores the Process Fragments and associated metadata in the repository. For validation, Process Fragments have been saved in a particular directory and the application started so that the application can read and save them in the repository (database). To validate whether the Process Fragments and associated metadata have been stored in the repository, SQL queries are written.

```
SELECT * FROM fragmentsrepository.fragments;
```

fid	file
15736	BLOB
15737	BLOB
15738	BLOB
15739	BLOB
15740	BLOB
15741	BLOB

Figure 6.1: SQL Query Request and Response.

Figure 6.1 shows the SQL to query the Process Fragments and the query returns all the Process Fragments. Usually for storing large objects (figures,text files etc), blob data type (binary large object) is used. Since Process Fragment is XML content, therefore blob data type is used. The blob content of first Process Fragment is also shown in figure 6.2

The metadata of the Process Fragment in figure 6.2 is also queried and the result is shown in figure 6.3.

Binary | Text

```
1   <?xml version="1.0" encoding="UTF-8"?><bpmn2:definitions
    xmlns:bpmn2="http://www.omg.org/spec/BPMN/20100524/MODEL"
    id="sid-bea53f11-2444-4f30-9a6d-bc8088e790d5">
2     <bpmn2:process id="sid-f1a4c52d-139d-4ba1-a295-dc982b33cd98">
3       <bpmn2:startEvent id="sid-B9CA1A51-96EA-41C8-8F70-F32B855869C6"
    name="">
4
    <bpmn2:outgoing>sid-EB24CA37-38B4-4E99-BC6A-4E4C086BC478</bpmn2:outgoin
    g>
5       </bpmn2:startEvent>
6       <bpmn2:scriptTask id="sid-CB2F92EA-AB25-4E75-9F48-B38826ABFE0C"
    name="">
7
    <bpmn2:incoming>sid-EB24CA37-38B4-4E99-BC6A-4E4C086BC478</bpmn2:incomi
    ng>
8
    <bpmn2:outgoing>sid-047F9A79-B08E-4748-A0A2-02BE22C17CFB</bpmn2:outgoing
    >
9       </bpmn2:scriptTask>
10      <bpmn2:scriptTask id="sid-B83188C1-AD3D-4B2F-8CA0-2BA0DF387667"
    name="">
11
    <bpmn2:incoming>sid-047F9A79-B08E-4748-A0A2-02BE22C17CFB</bpmn2:incomin
    g>
12      </bpmn2:scriptTask>
13      <bpmn2:sequenceFlow id="sid-EB24CA37-38B4-4E99-BC6A-4E4C086BC478"
```

Data Length: 1275 bytes

Figure 6.2: Process Fragment in Blob.

```
select fragmentsrepository.taskmetadata.*
from fragmentsrepository.metadata join fragmentsrepository.taskmetadata
on fragmentsrepository.metadata.mid = fragmentsrepository.taskmetadata.mid
and fragmentsrepository.metadata.fid=15736;
```

tid	mid	taskname	componentid	connrequired	IncomingOutgoing
21126	2896	ScriptTaskImpl	sid-CB2F92EA-AB25-4E75-9F48-B38826ABFE0C	0	NULL
21127	2896	ScriptTaskImpl	sid-B83188C1-AD3D-4B2F-8CA0-2BA0DF387667	1	outgoing

Figure 6.3: Process Fragment Metadata.

6.2 Synthesize Process Model - Validation

In this section, different validation cases related to synthesizing the Process Model are discussed. The output with changes in parameters in the user-defined, benchmark-related criteria are also observed in this section.

6.2.1 Trivial Case

In the first validation, a criteria for two Process Fragments is provided to the application through XML file as shown in figure 6.4. According to the criteria in the figure 6.4, the first Process Fragment should have 2 script tasks and no service tasks, call activities, exclusive gateways and parallel gateways. The second Process Fragment should have 2 call activities and no script tasks, service tasks, exclusive gateways and parallel gateways.

```
<fragments>
  <fragment>
    <DetailActivities>
      <ScriptTasks>2</ScriptTasks>
      <ServiceTasks>0</ServiceTasks>
      <CallActivities>0</CallActivities>
    </DetailActivities>
    <ExclusiveGateways>0</ExclusiveGateways>
    <ParallelGateways>0</ParallelGateways>
  </fragment>
  <fragment>
  <DetailActivities>
      <ScriptTasks>0</ScriptTasks>
      <ServiceTasks>0</ServiceTasks>
      <CallActivities>2</CallActivities>
    </DetailActivities>
    <ExclusiveGateways>0</ExclusiveGateways>
    <ParallelGateways>0</ParallelGateways>
  </fragment>
</fragments>
```

Figure 6.4: Fragment selection criteria.

Based on the provided criteria, two Process Fragments are selected out of the repository which are shown in figure 6.5.

Next is the compatibility check which in this case returns true because all the conditions mentioned in section 4.5 are met. Hence a new Process Model is synthesized and returned as shown in figure 6.6.

(a) Fragment having 2 script tasks

(b) Fragment having 2 call activities

Figure 6.5: Two Selected Fragments

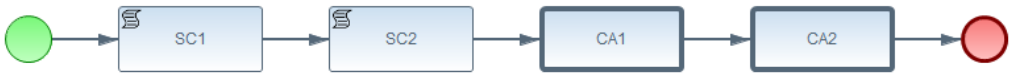


Figure 6.6: Synthesized Process Model using Fragment 6.5(a) and Fragment 6.5(b).

6.2.2 Synthesizing - Process Fragments with Gateways

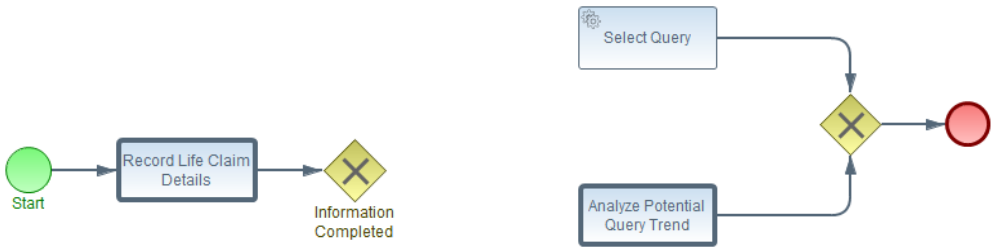
In the next few validation scenarios, Process Model is synthesized using the Process Fragments which have gateways. As mentioned in the chapter 4 gateways focused in this thesis are exclusive gateways and parallel gateways. According to the criteria shown in the figure 6.7 the first Process Fragment should have one call activity, one exclusive gateway and no script tasks, service tasks and parallel gateways. The second Process Fragment should have 1 service task, 1 call activity, one exclusive gateway and no script task and parallel gateway.

```
<fragments>
  <fragment>
    <DetailActivities>
      <ScriptTasks>0</ScriptTasks>
      <ServiceTasks>0</ServiceTasks>
      <CallActivities>1</CallActivities>
    </DetailActivities>
    <ExclusiveGateways>1</ExclusiveGateways>
    <ParallelGateways>0</ParallelGateways>
  </fragment>
  <fragment>
  <DetailActivities>
      <ScriptTasks>0</ScriptTasks>
      <ServiceTasks>1</ServiceTasks>
      <CallActivities>1</CallActivities>
    </DetailActivities>
    <ExclusiveGateways>1</ExclusiveGateways>
    <ParallelGateways>0</ParallelGateways>
  </fragment>
</fragments>
```

Figure 6.7: Fragment selection criteria.

Based on the provided criteria, two Process Fragments are selected from the repository as shown in figure 6.8

As discussed in section 5.1.1, gateway must be either split gateway or join gateway. The exclusive gateway in the Process Fragment in figure 6.8(a) has one incoming connection and no outgoing connection, therefore it must have atleast two outgoing connections in order to be considered as valid. The service task and call activity in the Process Fragment in figure 6.8(b)



(a) Fragment having 1 call activity and 1 exclusive gateway

(b) Fragment having 1 service task, 1 call activity and 1 exclusive gateway

Figure 6.8: Two Selected Fragments

also requires one incoming connection each, therefore one of the compatibility condition is fulfilled. Since the other conditions mentioned in section 4.5 are also fulfilled, therefore a new Process Model is created and returned as shown in figure 6.9.

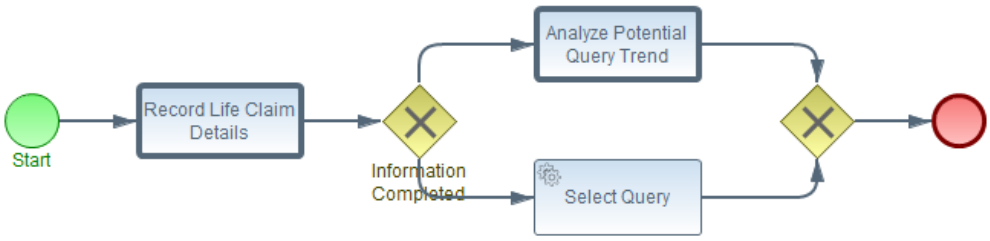


Figure 6.9: Synthesized Process Model using Fragment 6.8(a) and Fragment 6.8(b).

Now the criteria in figure 6.7 is slightly changed so that the first Process Fragment should have now 2 script tasks, 1 service task and 1 parallel gateway. The second Process Fragment should have 1 script task and 1 parallel gateway. The selected Process Fragments based on this changed criteria are shown in figure 6.10.

The compatibility of the Process Fragments in figure 6.10 is similar to the previous scenario in a way that parallel gateways are used instead of exclusive gateways. Hence a new Process Model is created and returned in this case as well as shown in figure 6.11.

The complexity of the criteria is now further increased as shown in figure 6.12.

According to the criteria in figure 6.12, first Process Fragment should have 3 script tasks, 4 service tasks and 3 parallel gateways. The second Process Fragment should have 3 script

(a) Fragment having 2 script tasks, 1 service task and 1 parallel gateway

(b) Fragment having 1 script task and 1 parallel gateway

Figure 6.10: Two Selected Fragments

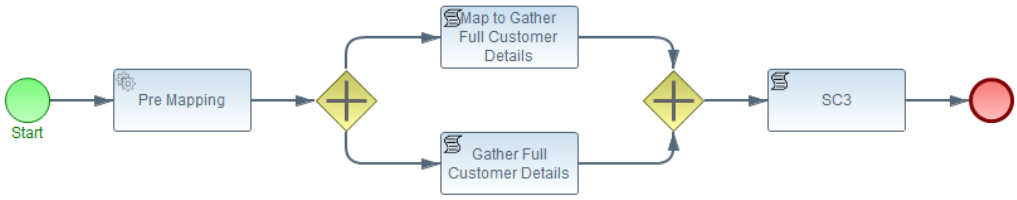


Figure 6.11: Synthesize Process using Fragment 6.10(a) and Fragment 6.10(b).

tasks and 3 parallel gateways. The Process Fragments selected based on the criteria are shown in the figure 6.13

One aspect to note in this case is that the Process Fragment selected and shown in figure 6.13(a) has a nesting as well. There are four tasks in the first Process Fragment which require one outgoing connection each. Whereas, the second Process Fragment has two parallel gateways and since both of them has one outgoing connection, therefore there must be atleast two additional incoming connections in each of them to make them valid. Hence the second Process Fragment requires four incoming connections and that would make the Process Fragments compatible. Other conditions mentioned in section 4.5 are also verified, hence a new Process Model is created and returned as shown in figure 6.14.

6.2.3 Synthesizing - Validation Parameters

As discussed in section 5.1.4 , there are some validation parameters which need to be checked before synthesizing Process Model. These parameters are validated in this subsection. Up until now, the Process Model are synthesized using two Process Fragments. The application is capable of synthesizing Process Model using N number of Process Fragments as discussed in section 4.5.

Consider the criteria shown in figure 6.15 in which the selection criteria is provided for

```xml
<fragments>
  <fragment>
    <DetailActivities>
      <ScriptTasks>3</ScriptTasks>
      <ServiceTasks>4</ServiceTasks>
      <CallActivities>0</CallActivities>
    </DetailActivities>
    <ExclusiveGateways>0</ExclusiveGateways>
    <ParallelGateways>3</ParallelGateways>
  </fragment>
  <fragment>
  <DetailActivities>
      <ScriptTasks>3</ScriptTasks>
      <ServiceTasks>0</ServiceTasks>
      <CallActivities>0</CallActivities>
    </DetailActivities>
    <ExclusiveGateways>0</ExclusiveGateways>
    <ParallelGateways>3</ParallelGateways>
  </fragment>
</fragments>
```

Figure 6.12: Fragment selection criteria.

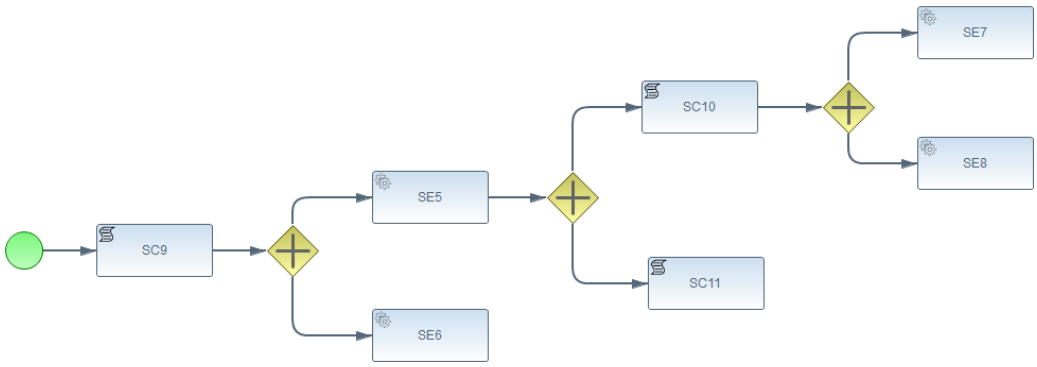
three Process Fragments. According to this defined criteria, first Process Fragment should have 1 script task, 3 service tasks, 2 call activities and 2 exclusive gateways. The second Process Fragment should have 2 script tasks, 3 service tasks, 4 call activities and 3 exclusive gateways. And lastly, the third Process Fragment should have 1 scrip task, 1 service task, 1 call activity and 1 exclusive gateway. The corresponding Process Fragments are selected from the repository by the application which fulfill the above mentioned defined criteria. The selected Process Fragments are shown in figure 6.16.

The following conditions are checked for compatibility:

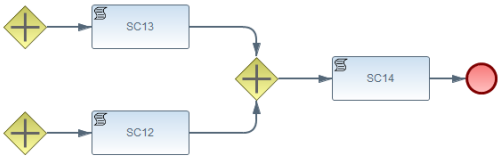
- First Process Fragment has a start event or not. In this case, it does not have any start event and the incoming connection required by the first Process Fragment is one, therefore first compatible condition is true and a flag is set which tells the synthesizer that there is a need to add start event in the first Process Fragment.

- There is no end event in the last Process Fragment which in this case is third. The outgoing connection required by the last (third in this case) Process Fragment is also one, therefore the compatible condition for this case is also true and an another flag is set in this case which tells the synthesizer that there is a need to add end event.

- Only one component (right most exclusive gateway) in the figure 6.16(a) is not according to the BPMN 2.0 specification and it requires atleast one outgoing connection in order to be considered as valid, therefore first Process Fragment requires one outgoing connection. The second Process Fragment requires one incoming connection as shown



(a) Fragment having 3 script tasks, 4 service tasks and 3 parallel gateways



(b) Fragment having 3 script tasks and 3 parallel gateways

Figure 6.13: Two Selected Fragments

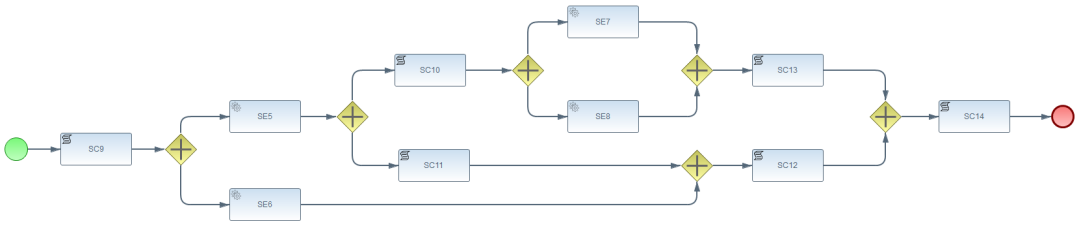
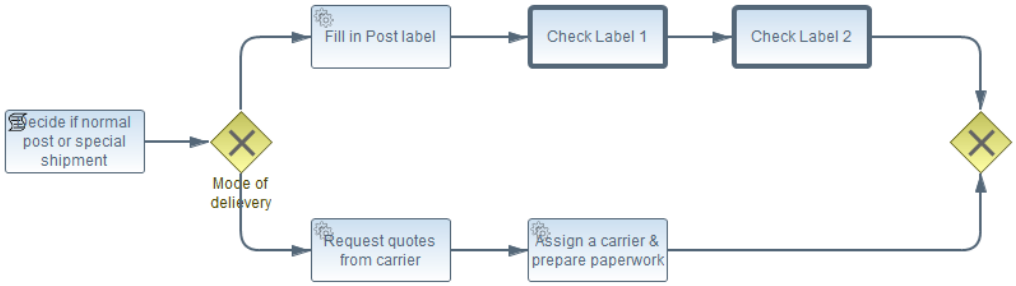


Figure 6.14: Synthesize Process using Fragment 6.13(a) and Fragment 6.13(b).

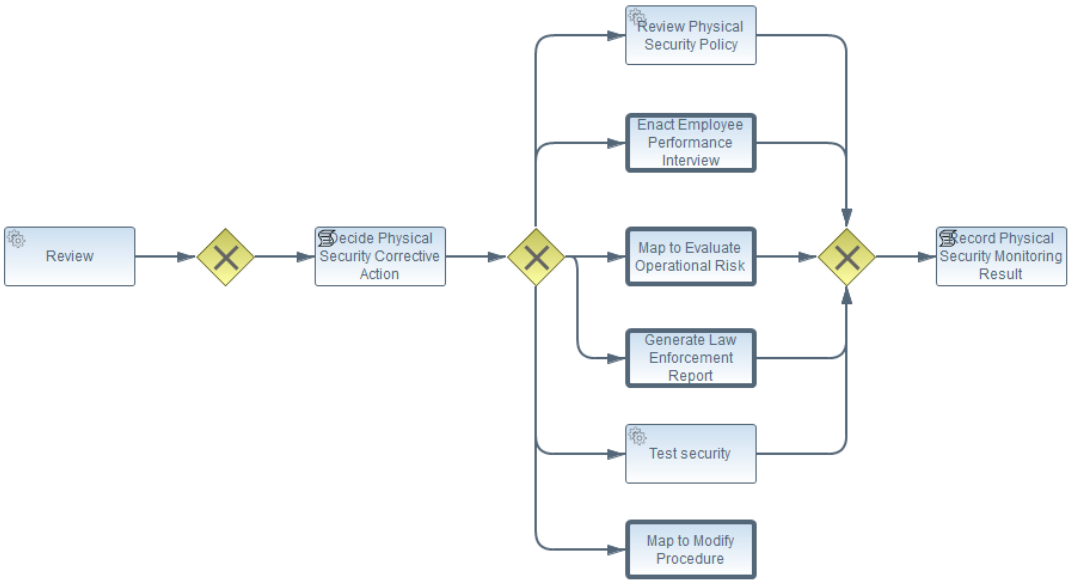
```
<fragments>
  <fragment>
    <DetailActivities>
      <ScriptTasks>1</ScriptTasks>
      <ServiceTasks>3</ServiceTasks>
      <CallActivities>2</CallActivities>
    </DetailActivities>
    <ExclusiveGateways>2</ExclusiveGateways>
    <ParallelGateways>0</ParallelGateways>
  </fragment>
  <fragment>
  <DetailActivities>
      <ScriptTasks>2</ScriptTasks>
      <ServiceTasks>3</ServiceTasks>
      <CallActivities>4</CallActivities>
    </DetailActivities>
    <ExclusiveGateways>3</ExclusiveGateways>
    <ParallelGateways>0</ParallelGateways>
  </fragment>
  <fragment>
  <DetailActivities>
      <ScriptTasks>1</ScriptTasks>
      <ServiceTasks>1</ServiceTasks>
      <CallActivities>1</CallActivities>
    </DetailActivities>
    <ExclusiveGateways>1</ExclusiveGateways>
    <ParallelGateways>0</ParallelGateways>
  </fragment>
</fragments>
```

Figure 6.15: Fragment selection criteria.

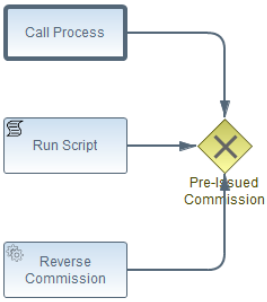
in figure 6.16(b) and three outgoing connections. The third Process Fragment requires three incoming connection. As discussed in section 4.5, since for every subset $\{N1, N2\}$ and $\{N2, N3\}$ in the set of N1,N2,N3 Process Fragments, x should be equal to y and in this case x is equal to y, therefore this compatible condition is also true.



(a) Fragment having 1 script task, 3 service tasks, 2 call activities and 2 exclusive gateways



(b) Fragment having 2 script tasks, 3 service tasks, 4 call activities and 3 exclusive gateways



(c) Fragment having 1 script task, 1 service task, 1 call activity and 1 exclusive gateways

Figure 6.16: Three Selected Fragments

- Next is to check whether all the split exclusive gateways are equal to the join exclusive gateways and similarly for parallel gateways. In this scenario, number of split exclusive gateways in all three Process Fragments are three and number of join exclusive gateways are also three. For parallel gateway, the value of split and join is both zero. Hence this condition is also satisfied.

Since all the conditions are satisfied, therefore a new Process Model is created using all three Process Fragments of figure 6.16. The start event and end event are also automatically added as required. The synthesized Process Model is shown in figure 6.17.

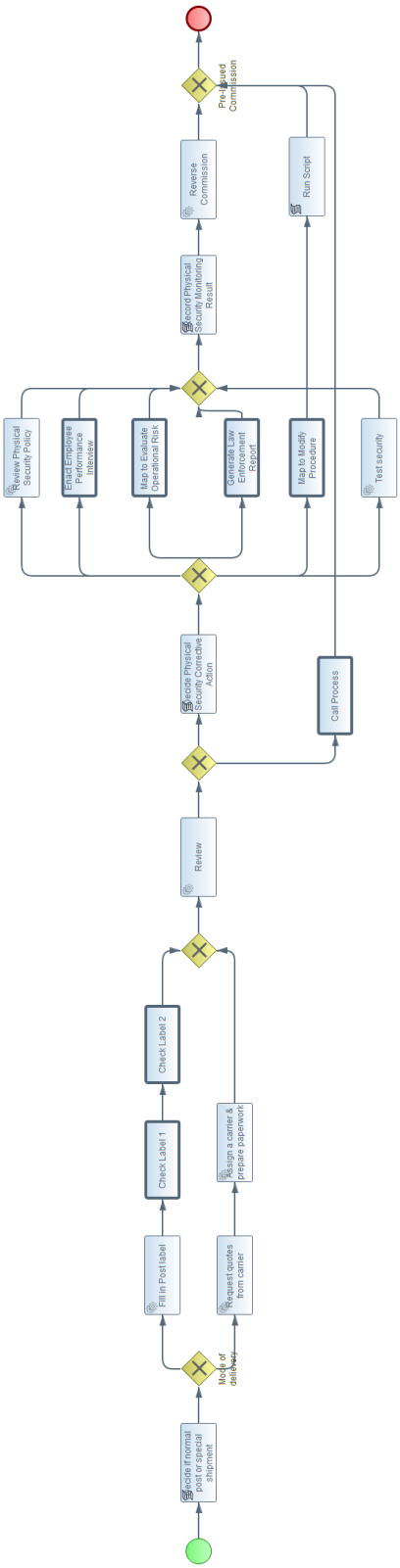


Figure 6.17: Synthesize Process using Fragment 6.16(a), Fragment 6.16(b) and Fragment 6.16(c).

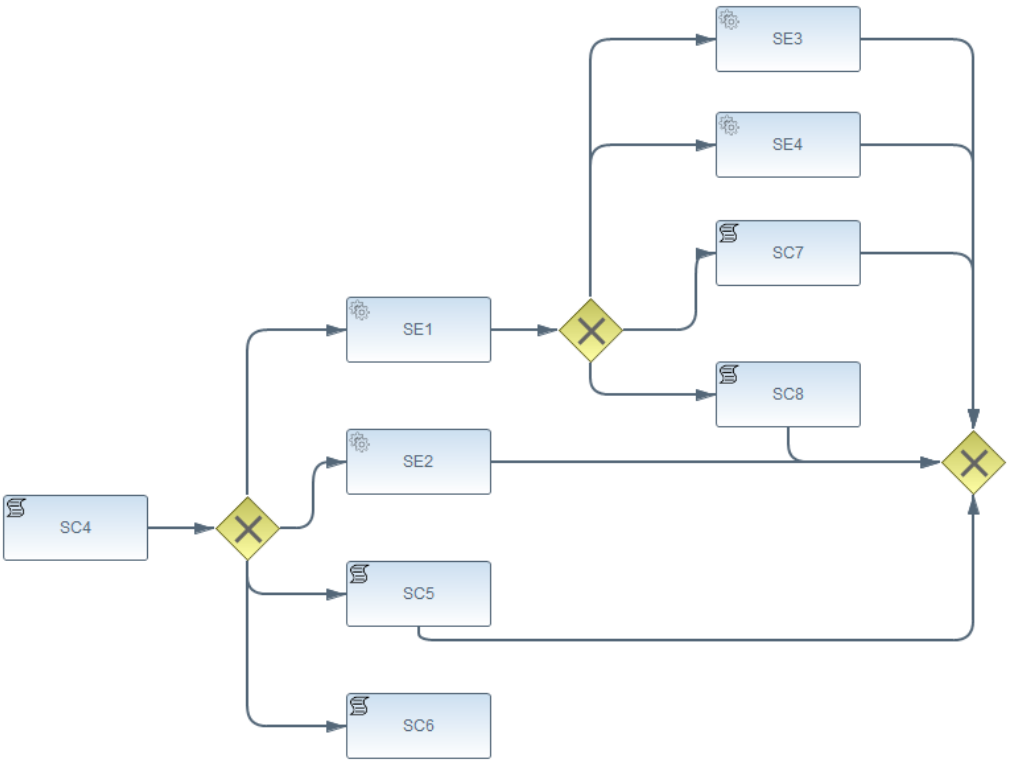
After synthesizing Process Model, a message is shown to the user which says that validation parameters are failed. These are basically the parameters which are derived from [Iva14] work and are discussed in section 5.1.4. As mentioned earlier the reason of enforcing these parameters is to synthesize realistic Process Model. Among those parameters, one of the parameter states that the maximum Fanout value of any gateway in a Process Model should not be greater than 5 which in case of figure 6.17 the value is 6, hence a violation. There is one more violation and that is the total number of tasks in a Process Model should not be equal to or more than 18, but in this case it is 18.

Another criteria is shown in figure 6.18, according to which the first Process Fragment should have 5 script tasks, 4 service tasks and 3 exclusive gateways. The second Process Fragment should have 1 service task, 1 call activity and 1 exclusive gateway. The Process Fragments selected based on this criteria are shown in figure 6.19.

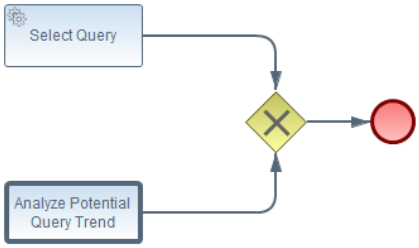
```
<fragments>
  <fragment>
    <DetailActivities>
      <ScriptTasks>5</ScriptTasks>
      <ServiceTasks>4</ServiceTasks>
      <CallActivities>0</CallActivities>
    </DetailActivities>
    <ExclusiveGateways>3</ExclusiveGateways>
    <ParallelGateways>0</ParallelGateways>
  </fragment>
  <fragment>
  <DetailActivities>
      <ScriptTasks>0</ScriptTasks>
      <ServiceTasks>1</ServiceTasks>
      <CallActivities>1</CallActivities>
    </DetailActivities>
    <ExclusiveGateways>1</ExclusiveGateways>
    <ParallelGateways>0</ParallelGateways>
  </fragment>
</fragments>
<SynthesizedProcess>
  <CFC Min="0" Max="20">true</CFC>
  <ProcessEngine>Jbpm</ProcessEngine>
</SynthesizedProcess>
```

Figure 6.18: Fragment selection criteria.

The Process Fragments in figure 6.19 also fulfill all the compatible conditions mentioned in section 4.5, hence a new Process Model is created and returned as shown in figure 6.20. But the synthesized Process Model in figure 6.20 also violates the validation parameters threshold which states that the maximum Fanin value of a gateway in a Process Model should not be more than 5 and in this case it is 6. As a consequence, a message is displayed to the user which states that it violates the threshold.



(a) Fragment having 5 script tasks, 4 service tasks and 3 exclusive gateways



(b) Fragment having 1 service task, 1 call activity and 1 exclusive gateway

Figure 6.19: Two Selected Fragments

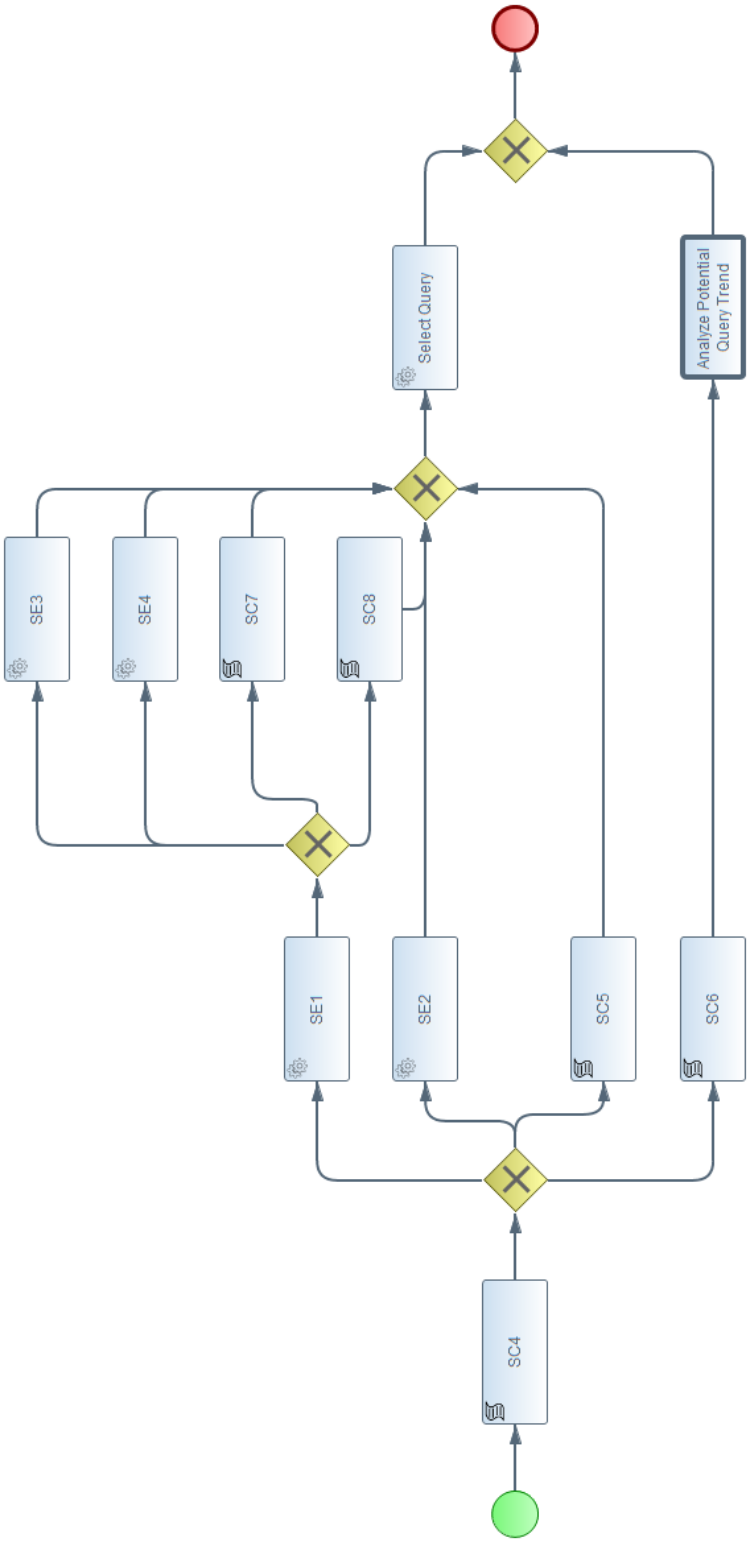


Figure 6.20: Synthesize Process using Fragment 6.19(a) and Fragment 6.19(b).

6.2.4 Synthesizing - Not Compatible Process Fragments

The Process Fragments are compatible in all the scenarios which have been discussed so far. Consider the same criteria as shown in figure 6.18 but with a slight variation in the second Process Fragment. Now the second Process Fragment should have 1 script task and 1 parallel gateway. The Process Fragments selected based on this criteria are shown in figure 6.21.

A new Process Model is not created in this case because according to the rules mentioned in section 4.5, the number of split exclusive gateways should be equal to the number of join exclusive gateways and similarly, for parallel gateways. Whereas, in case of Process Fragments in figure 6.21, value of split exclusive gateway is 2 and value of join exclusive gateway is 1.

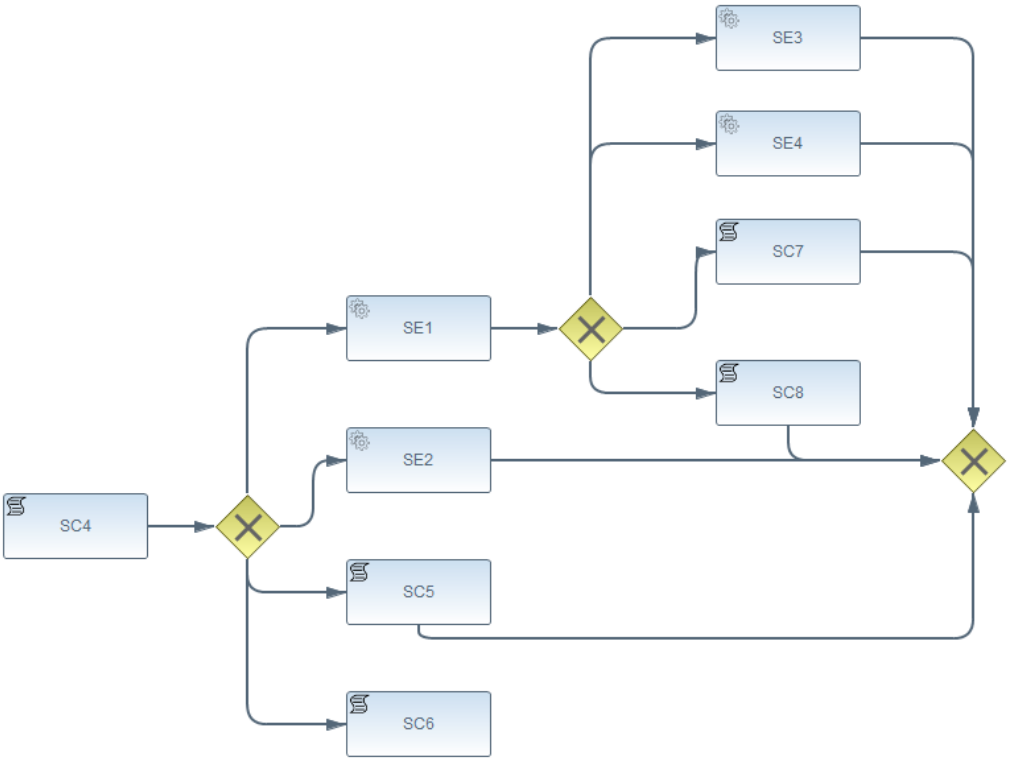
Now again consider the same criteria as shown in figure 6.18 but with different CFC value. Change the maximum CFC value to 7 instead of 20 as shown in figure 6.18. The Process Model is not synthesized in this case as well because as shown in figure 6.20, there are two split exclusive gateways having four outgoing connections each which makes the CFC value for the synthesized Process Model equals to 8. The user on the other hand defined the restriction and asked for a synthesized Process Model having CFC value not more than 7.

6.3 Process Engine Executable Process Model - Validation

According to the requirements mentioned in section 4.1, the synthesized Process Model should be automatically made executable with respect to atleast two Process Engines (Camunda and jBPM). After synthesizing, changes are made in the Process Model as discussed in section 5.2.4 and section 5.3.1. The following subsections validate whether the changes made in the synthesized Process Models for Camunda and jBPM are correct or not.

6.3.1 jBPM - Validation

To verify the changes made in the synthesized Process Model with respect to jBPM, jUnit test cases have been written. In one of the test case user can define ID of any node in the synthesized Process Model and see if that node executes by the Process Engine. Since all the nodes are not long running, therefore the result can be seen and observed immediately. In the second test case, the ID of the end event is provided by default and the test case always check whether the Process reaches the end event node or not. One important thing to remember is that before executing the test case, one has to provide the parameters to the jUnit test case which determine the path after the split exclusive gateway. Those conditions and variables have been randomly generated as discussed in section 5.3.1.



(a) Fragment having 5 script tasks, 4 service tasks and 3 exclusive gateways

(b) Fragment having 1 script task and 1 parallel gateway

Figure 6.21: Two Selected Fragments

6.3.2 Camunda - Validation

For Camunda, consider the synthesized Process Model in figure 6.20. The XML content of the synthesized Process Model in figure 6.20 is shown in listing 6.1.

```
<bpmn2:process id="YJIqWYlHbd" isExecutable="true" name="
    YJIqWYlHbd">
<bpmn2:scriptTask id="node7" name="SC4" scriptFormat="javascript"
    >
<bpmn2:incoming>_gL4ZYZLrEeWjlcnFMIV5pg</bpmn2:incoming>
<bpmn2:outgoing>sid-696E51C1-0153-4D80-97AD-420F91458E5A
</bpmn2:outgoing>
<bpmn2:script>;</bpmn2:script>
</bpmn2:scriptTask>
<bpmn2:exclusiveGateway id="gate1" name="">
<bpmn2:incoming>sid-696E51C1-0153-4D80-97AD-420F91458E5A
</bpmn2:incoming>
<bpmn2:outgoing>sid-785230FB-C3A2-4B8F-943D-DB5FA56FD0B3
</bpmn2:outgoing>
<bpmn2:outgoing>sid-399E3F3A-5E89-4518-BB09-1E61FDA80E7A
</bpmn2:outgoing>
<bpmn2:outgoing>sid-F20BB8A9-CDB8-464A-8F99-90B6A1850553
</bpmn2:outgoing>
<bpmn2:outgoing>sid-6DA200EF-C045-4B43-A561-37B7986B7A31
</bpmn2:outgoing>
</bpmn2:exclusiveGateway>
<bpmn2:serviceTask id="node2" name="SE1">
<bpmn2:extensionElements>
<camunda:connector>
<camunda:connectorId>http-connector</camunda:connectorId>
<camunda:inputOutput>
<camunda:inputParameter name="url">http://localhost:8080/engine-
    rest/deployment</camunda:inputParameter>
<camunda:inputParameter name="method">GET
</camunda:inputParameter>
<camunda:outputParameter name="statusCode">${statusCode}
</camunda:outputParameter>
</camunda:inputOutput>
</camunda:connector>
</bpmn2:extensionElements>
<bpmn2:incoming>sid-785230FB-C3A2-4B8F-943D-DB5FA56FD0B3
</bpmn2:incoming>
<bpmn2:outgoing>sid-1F031708-167D-4406-A257-AC16481C9DFB
</bpmn2:outgoing>
```

```
37 </bpmn2:serviceTask>
38 <bpmn2:sequenceFlow id="sid-785230FB-C3A2-4B8F-943D-DB5FA56FD0B3"
       name="" sourceRef="gate1" targetRef="node2">
39 <bpmn2:conditionExpression xsi:type="bpmn2:tFormalExpression"><![
      CDATA[${input1.equals("2")}]]></bpmn2:conditionExpression>
40 </bpmn2:sequenceFlow>
41 <bpmn2:sequenceFlow id="sid-88C3A36B-793F-406A-A748-F88747D846BE"
       name="" sourceRef="gate2" targetRef="node4">
42 <bpmn2:conditionExpression xsi:type="bpmn2:tFormalExpression"><![
      CDATA[${input2.equals("7")}]]></bpmn2:conditionExpression>
43 </bpmn2:sequenceFlow>
```

Listing 6.1: XML content portion of the synthesized Process Model in figure 6.20

The process name of this synthesized Process Model is "YJIqWYlHbd" which is shown in listing 6.1 and it is generated randomly by the application. As discussed in section 5.2, a '.war' package for the synthesized Process Model is automatically created for Camunda by the application which is then copied on the Tomcat server to deploy the Process Model. The '.war' package for the synthesized Process Model in figure 6.20 is created and copied on the Tomcat server. Log into the web console of Camunda (`localhost:8080/camunda/app/cockpit`), the deployed Process name is listed there as shown in figure 6.22.

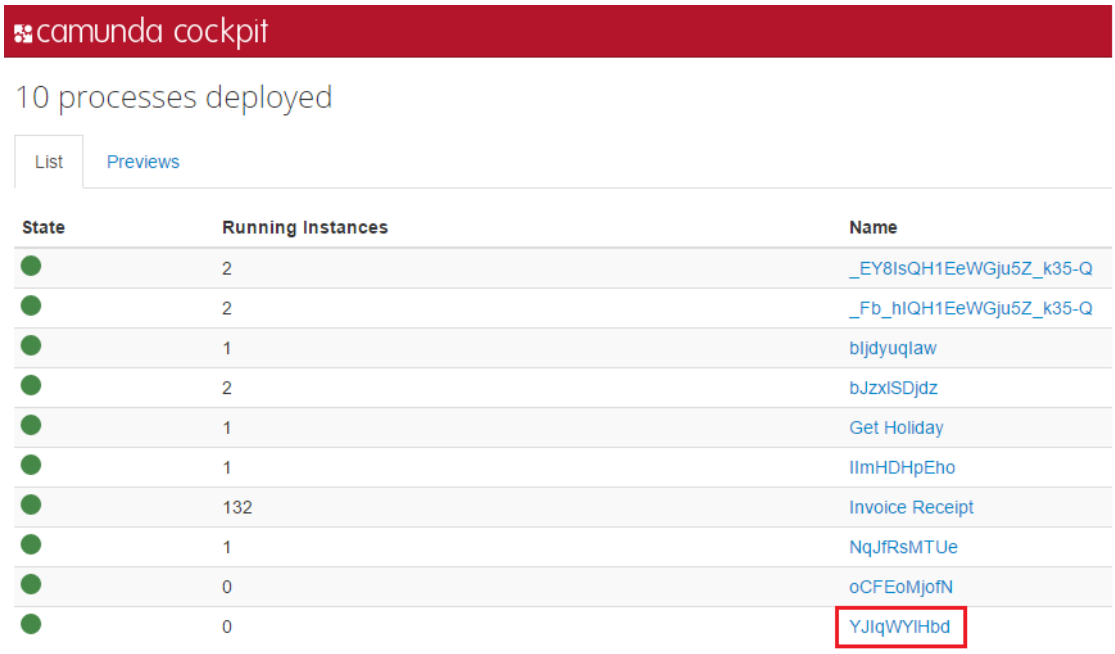


Figure 6.22: Main page of Camunda Process Engine.

There are also two randomly generated variables "input1" and "input2" for 2 split exclusive gateways as shown in listing 6.1. The value of these two variables are also randomly generated and it determines the path followed by the Process Model. For example, the path from

gateway 1 to node 2 is followed if input1 has a value equals to 2. Next step is to start the Process Model instance on Camunda. While starting the instance, the value of two variables "input1" and "input2" needs to be provided and then the instance starts running as shown in figure 6.23.

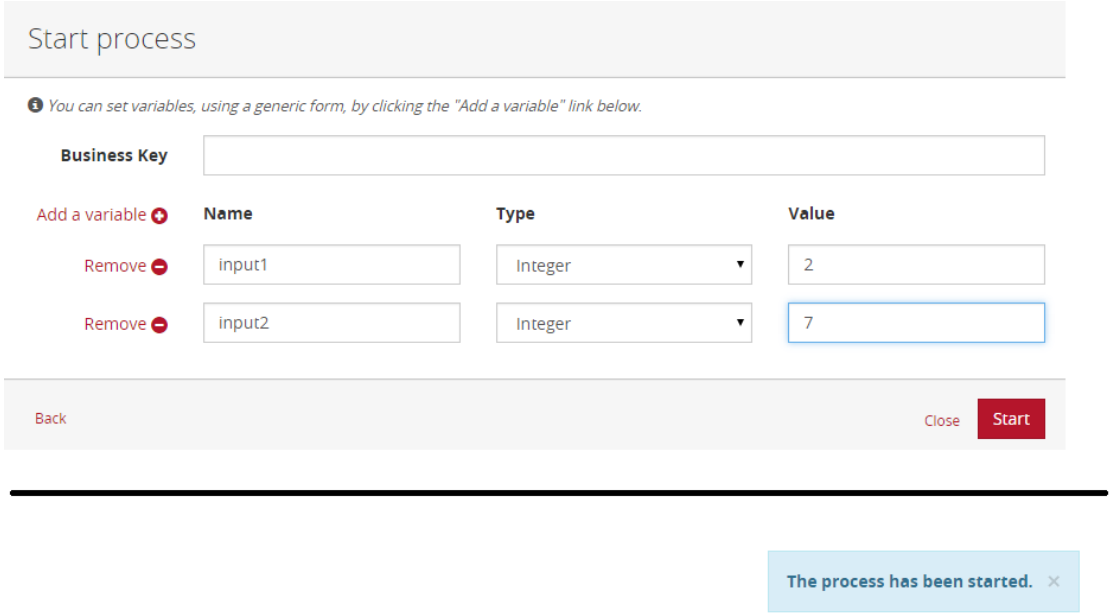


Figure 6.23: Start Process Model in Camunda.

If the input variables are not defined while starting the instance then an error is shown by Camunda and the Process instance does not start.

7 Outcome and Future Work

The main challenge in the thesis was to create a repository of Process Fragments and to identify, how a Process Model can be synthesized using the Process Fragments repository. To deploy the Process Model on the Process Engine, the technical team usually needs to do some changes in the Process Model manually. The other challenge in this thesis was to perform all the required changes automatically with respect to the Process Engines. The synthesized executable Process Models from this application will now be used to benchmark the Process Engines [Benay].

Chapter 1 describes the context of why there is a need for an application which can synthesize Process Models. Chapter 2 describes the basic concepts related to BPMN 2.0. It also describes the importance of having metadata and how metadata helps to locate the required resource. Then a distinction has been made between descriptive metadata and structural metadata. This distinction was necessary because the interest in this thesis was on structural metadata. Also, the concept of benchmarking and the architecture of the two open source WfMS (Camunda and jBPM) are also discussed because the focus in this thesis is on these two Process Engines. Chapter 3 discusses the work already done in the field of Process Fragments, the repository of Process Models and synthesizing of Process Models. We studied the design and architecture of the application already developed in the same domain as ours and identify the challenges to build the required prototype application. Chapter 4 initially describes the functional and non functional requirements of the application. Then, an architecture is proposed that can fulfill the mentioned requirements. The interaction between the different components in the architecture is also discussed. This is necessary in order to understand the flow of the application. Also, the detailed architecture of the application in the form of class diagram, database schema and XML schema is also discussed in this chapter. Finally, pseudo code of two algorithms which are used in this application are also discussed.

Using the requirements and design of the prototype application in chapter 4, we lead to the implementation of the application in chapter 5 which explains, how different Process Fragments are extracted from the repository based on the user-defined, benchmark-related criteria and then, how a Process Model is synthesized using the metadata and Process Fragments. This chapter also describes the details of how different elements in the Process Model are handled in order to make the Process Model executable. Chapter 6 discusses the validation and evaluation of the implementation. Starting from the trivial scenario to more complex scenarios, the output of the application is shown and discussed. Not only is the synthesizing of Process Models validated, but also the Process Engine executable version. The executable version is then deployed on Camunda and an instance of the Process Model is started to verify that the changes which have been made in the Process Model are correct.

As discussed in section 5.4 there are some limitations of the application. These limitations could be overcome in future work. The application has been designed to keep it flexible for

additions in future work. For example, the application could be extended for other types of tasks namely: user task, send task and so on, which are not handled. Further, two types of gateways are handled (exclusive gateway and parallel gateway). Other types of gateway: complex gateway, inclusive gateway can also be handled in future work. Further, there is also a possibility that a Process Model can have more than one start and end event. In this thesis, since the focus was on the Process Model having one start and one end event, therefore the application could be extended in this part as well. Further, the event types are not restricted to start and end event, there are also timer event, message event, signal event and so on. Those can also be made part of the application in future work. And finally in this thesis, two Process Engines are considered to make the executable version of Process Models. More Process Engines can also be included in future work for creating executable Process Models automatically.

Bibliography

[Benay] Benchflow - A Benchmark for Workflow Management Systems, 2015 (Retrieved on December 4, 2015). `http://www.iaas.uni-stuttgart.de/forschung/projects/benchflow.php`.

[BR98] John Bessant and Howard Rush. Approaches to Benchmarking: The Case of Framework Conditions and ICT-Os. In *Institute for Prospective Technological Studies European Commission Joint Research Centre*, University of Brighton, United Kingdom, 1998.

[Car08] Jorge Cardoso. Business Process Control Flow Complexity: Metric, Evaluation, and Validation. In *International Journal of Web Services Research, 5(2)*, pages 49–76, 2008.

[DES08] George Din, Klaus-Peter Eckert, and Ina Schieferdecker. A Workload Model for Benchmarking BPEL Engines. In *Software Testing Verification and Validation Workshop, ICSTW*, pages 356 – 360, Lillehammer, Norway, April 2008.

[Eli15] Mturi Elias. *Design of Business Process Model Repositories*. PhD thesis, Stockholm University, 2015.

[ELS+10] Hanna Eberle, Frank Leymann, Daniel Schleicher, David Schumm, and Tobias Unger. Process Fragment Composition Operations. In *Services Computing Conference (APSCC)*, IEEE Asia-Pacific, 2010.

[EUL10] Hanna Eberle, Tobias Unger, and Frank Leymann. Implementing Transactional Behavior in Process Fragment Execution using Two Recovery Strategies. In *Services Computing Conference (APSCC)*, IEEE Asia-Pacific, 2010.

[GmBay] Camunda Services GmBH. *Camunda BPM*, October 2015 (Retrieved on December 4, 2015). `https://docs.camunda.org/manual/7.3/guides/user-guide/`.

[Gra92] Jim Gray. *Benchmark Handbook: For Database and Transaction Processing Systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1992.

[Hatay] Red Hat. *jBPM*, October 2015 (Retrieved on December 4, 2015). `http://docs.jboss.org/jbpm/v6.2/userguide/`.

[HHM11] Jacky Holloway, Matthew Hinton, and David Mayle. Why Benchmark? Understanding the Processes of Best Practice Benchmarking. In *Business Track, British Academy of Management Conference*, pages 271–291, London, 2011.

[Hup09] Karl Huppler. The Art of Building a Good Benchmark. In *Performance Evaluation and Benchmarking, First TPC Technology Conference, TPCTC*, Lyon, France, August 2009.

[Iva14] Ana Ivanchikj. Characterising Representative Models for BPMN 2.0 Workflow Engine Performance Evaluation. Master's thesis, Università della Svizzera Italiana, 2014.

[JEay] Diane Jordan and John Evdemon. *Business Process Model And Notation (BPMN) Version 2.0. Object Management Group, Inc*, January 2011 (Retrieved on December 4, 2015). `http://www.omg.org/spec/BPMN/2.0/PDF/`.

[Ley01] Frank Leymann. Managing Business Processes via Workflow Technology. In *International Conference on Very Large Data Bases, VLDB*, Roma, Italy, 2001.

[Ley14] Frank Leymann. Lecture notes in Business Process Modeling and Notation, 2014.

[Lin08] Yun Lin. *Semantic Annotation for Process Models: Facilitating Process Knowledge Management via Semantic Interoperability*. PhD thesis, Norwegian University of Science and Technology, 2008.

[Mayay] Maybridge: The Maybridge Ro3 Fragment Library, product flyer, 2007 (Retrieved on December 4, 2015). `http://www.maybridge.com/images/pdfs/ro3frag.pdf`.

[ML08] Zhilei Ma and Frank Leymann. A Lifecycle Model for Using Process Fragment in Business Process Modeling. In *9th International Workshop on Business Process Modeling, Development and support, BPMDS*, pages 1–9, conjuction with CAiSE, France, 2008.

[MPR12] Rudolf Mayer, Stefan Proell, and Andreas Rauber. On the Applicability of Workflow Management Systems for the Preservation of Business Processes. In *9th International Conference on Preservation of Digital Objects, IPRES*, Toronto, Ontario, Canada, 2012.

[NISay] NISO. *Understanding Metadata*, 2001 (Retrieved on December 4, 2015). `http://www.niso.org/publications/press/UnderstandingMetadata.pdf`.

[RMD10] Hajo Reijers, Jan Mendling, and Remco Dijkman. On the Usefulness of Sub-Processes in Business Process Models. In *External Report, BPM center report, No. BPM-10-03*, Eindhoven: BPMcenter.org, 2010.

[RP93] Colette Rolland and Naveen Prakash. Reusable Process Chunks. In *International Conference of Database and Expert Systems Applications*, Prague, Slovakia, September 1993.

[SCG06] Valeria Seidita, Massimo Cossentino, and Salvatore Gaglio. A Repository of Fragments for Agent Systems Design. In *Workshop on Objects and Agents, WOA*, pages 130–137, Catania, Italy, 2006.

[SGHL15] Marigianna Skouradaki, Katharina Görlach, Michael Hahn, and Frank Leymann. Application of Sub-Graph Isomorphism to Extract Reoccurring Structures from BPMN 2.0 Process Models. In *9th International IEEE Symposium on Service Oriented Systems Engineering, SOSE*, San Francisco, USA, 2015.

[Sil11] Bruce Silver. *BPMN Method and Style, 2nd Edition*. Cody-Cassidy Press, 2011.

[SKK+11] David Schumm, Dimka Karastoyanova, Oliver Kopp, Frank Leymann, Mirko Sonntag, and Steve Strauch. Process Fragment Libraries for Easier and Faster Development of Process based Applications. *Journal of Systems Integration*, 2(1):39–55, 2011.

[SKLS10] David Schumm, Dimka Karastoyanova, Frank Leymann, and Steve Strauch. Fragmento: Advanced Process Fragment Library. In *19th International Conference on Information Systems Development, ISD*, Prague, Czech Republic, August 2010.

[SLM+10] David Schumm, Frank Leymann, Zhilei Ma, Thorsten Scheibler, and Steve Strauch. Integrating Compliance into Business Processes: Process Fragments as Reusable Compliance Controls. In *The Multikonferenz Wirtschaftsinformatik, MKWI*, Universitätsverlag Göttingen, 2010.

[SMS98] Xiang Yong Shi Meilin, Yang Guangxin and Wu Shangguang. Workflow Management Systems: A Survey. In *IEEE International Conference on Communication Technology*, Beijing, China, October 1998.

[SRL+15] Marigianna Skouradaki, Dieter Roller, Frank Leymann, Vincenzo Ferme, and Cesare Pautasso. On the Road to Benchmarking BPMN 2.0 Workflow Engines. In *6th ACM/SPEC International Conference on Performance Engineering, ICPE*, pages 301–304, Austin, Texas, 2015.

[Tavay] Taverna. What is Workflow Management System, 2010 (Retrieved on December 4, 2015). `http://www.taverna.org.uk/introduction/what-is-a-workflow-management-system/`.

[UEKL10] Tobias Unger, Hanna Eberle, Oliver Kopp, and Frank Leymann. The SubProcess Spectrum. In *3rd International Conference on Business Process and Services Computing, BPSC, Gesellschaft für Informatik e.V. (GI)*, 2010.

[VKL06] Jussi Vanhatalo, Jana Koehler, and Frank Leymann. Repository for Business Processes and Arbitrary Associated Metadata. In *BPM Demo Session at the Fourth International Conference on Business Process Management*, Vienna, Austria, 2006.

[YDG14] Zhiqiang Yan, Remco Dijkman, and Paul Grefen. Generating Process Model Collections. In *Asia Pacific Business Process Management : Second Asia Pacific Conference, AP-BPM*, Brisbane, QLD, Australia, July 2014.

Declaration

I hereby declare that the work presented in this thesis is entirely my own. I did not use any other sources and references other than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

Stuttgart, December 4, 2015 _____

(Name)