MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY

Service Lifecycle Management Application

DIPLOMA THESIS

Anton Giertli

Brno, autumn 2014

Declaration

Hereby, I declare that this paper is my original authorial work which I have worked out on my own. All sources, references and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Advisor: doc. RNDr. Tomáš Pitner, Ph.D.

Abstract

The main goal of this diploma thesis is to create a web based application which will allow managing service lifecycle, a concept defined in Service Oriented Architecture Governance (SOA Governance). The application is also a practical demonstration of how can various JBoss open-source projects integrate between themselves. The text of this thesis describes the context, analysis and implementation of this application and also shows how can the resultant application be used by specific representatives in the SOA Governance environment.

Keywords

Service Oriented Architecture, SOA Governance, JBoss Overlord, Business Process Management, jBPM, Service Lifecycle

Acknowledgement

I would like to thank Mgr. Štefan Bunčiak for his patient and professional guidance during the whole process of working on this thesis.
Next, I would like to thank Maciej Swiderski for his numerous advises regarding JBoss JBPM technology.
Finally, special thanks goes to my girlfriend Miška for willingly listening about SOA Governance so much.

Contents

1 Introduction

In SOA environments, it is certainly beneficial to have an overview of available or deployed services. This can be achieved by adopting a service lifecycle which is a concept introduced in a SOA Governance.

SOA Governance is trying to maximize the added value of SOA solutions and service lifecycle is an important concept in it. The Service Lifecycle Management Application (SLMA) is built around this concept, allowing users to execute specific use cases which aligns with the service lifecycle definition as such.

Service Lifecycle is simply a workflow which every service has to follow. End user of the SLMA is executing Tasks which are defined in this workflow. By doing this, every service is going through same steps with the desired output of achieving higher service quality.

This diploma thesis is describing the process of implementation of the SLMA which was done by using JBoss open source technologies. Though the important part of this thesis is an executable output in form of the web based application, there is a strong relationship between this application and various management disciplines and concepts such as Business Process Management, Service Oriented Architecture, etc.

Service Lifecycle Management Application is a practical demonstration of how these various disciplines can come together and this will be promoted throughout the whole thesis.

1.1 Objectives

The main objective of this thesis is to implement fully functional application which will act as a service lifecycle manager. It needs to allow to user to do the following:

- start a new instance of a lifecycle bound to a specific service
- execute tasks included in this lifecycle so a service can be moved from stage to stage
- display status of a given service lifecycle instance and also display list of every lifecycle instance that ever entered the system

- monitor specific policy during the service runtime - which is displaying invocations of a service which is already retired

These functional requirements were not simply made up - they align with the definition of the Service Lifecycle concept which is defined in chapter 2.2.

Another objective was to achieve the implementation by using specific JBoss open-source technologies. Majority of latest versions of open-source community projects have been used (at the time of developing the application) and the application shows how these technologies can be successfully deployed on separate servers, and what is even more important - how can these technologies communicate remotely with each other. The architecture chosen for this application was trying to copy real needs of users, adopting SOA Governance approach and that is why multiple separate servers are used in the design. Therefore, the successful integration between these servers is crucial part of this diploma thesis.

To sum up, not only the final result (the application) is important, but also the path taken in its implementation matters.

2 Theoretical background

Since the implementation of the Service Lifecycle Management Application brings many technologies and disciplines together, it is necessary to provide a reader with some background and contextual information in order to fully understand motivation and added value behind this thesis. One could think that explaining SOA Governance concept should be sufficient but that is not really possible without bringing Service Oriented Architecture into the picture. Moreover, the Service Lifecycle - key part of the SOA Governance is actually implemented as the BPMN[1] process, therefore some basic principles of Business Process Management also have to be known.

This chapter's intention is not to go too deep into these disciplines and concepts - that is not the primary motivation of this thesis. Nevertheless, this chapter will briefly introduce the most important concepts and terms used in the context of this thesis with an emphasis on the SOA Governance.

2.1 Service Oriented Architecture

This section will briefly introduce Service Oriented Architecture (SOA). It will list few SOA advantages and since SOA is revolving around the concept of the **Service**, there will also be a practical example showed for easier understanding.

There are many definitions of SOA since this concept can be viewed in many different contexts - Business, IT, technical, etc. For the purpose of this thesis I have chosen the technical view:

Definition 1 *Service Oriented Architecture Dirksen [2013, p. 4]*
Service-oriented architecture (SOA) is a flexible set of design principles used during phases of systems development and integration in computing. A system based on a SOA architecture will provide a loosely-coupled suite of services that can be used within multiple separate systems from several business domains.

1. Business Process Model and Notation

So far the term SOA has been already mentioned numerous times without defining the key part of it - **Service**. Intuitively the meaning can be clear - **waiting** in a restaurant is an example of service, **cutting hair** in a hair salon is a service too. It can be said that the service is basically any amount of work which has been done for others. Now let us try to define this term more specifically in a technical context. Thomas Erl defines Service in a very simple way as a:

Definition 2 *Service Erl [2005, p. 32]*
Small, distinct units of logic. Collectively, these units comprise a larger piece of business automation logic.

Examples of services mentioned already are still perfectly valid. However, the definition above makes it easier to understand the term Service in a technical context. This is an example of technical service which could be used in a bank:

Listing 2.1: Example of the Service

```
Name: Eligibility Calculator service
Description: This service will evaluate the
   applicant and decide whether he is eligible
   for approval or not based on the computed
   score.
URI: {serviceURI}/isEligible?id={applicantId}&
   type={applicationType}
Method: GET
Example: http://bigExampleBank.org/isEligible?id
   =98&type=mortgage
Arguments: {applicationId} is an ID of an
   applicant as stored in the corporate Database,
    it is of type Long. {applicationType}
   argument is a String which can have these
   values: mortgage, loan, newAccount.
Output: Result is in format application/json with
    two entries - true/false and score.
```

```
True if candidate is eligible and false if not.
   Score is the value computed by this service
   based on which the decision has been made.}
```

The example above could be considered as a real sample of a REST[2] based service. It can be used by various systems in a bank - i.e. Mortgage application, Loan Application - effectively achieving **reusability**. Implementation details are completely hidden from systems which will be invoking this service. The only thing which is important is a **well documented contract**. The reason why the `{applicationType}` argument is included is that every system using this service can have different conditions based on which evaluation is made - age, history of late payments, monthly income, etc. Also, since implementation details are not known, it is perfectly possible that for every application type, the different development team could take care of the implementation - preferably the one who already has some knowledge in the desired field. It is also possible that this service invokes other services - based on the applicationType. Currently, it is hidden as an implementation detail but if this was the case it would certainly achieve a **business agility**. If another application type had to be added, there would not be so much overhead - the whole structure is already up and running, the URI would not change, and the documentation would remain the same only with small changes - the only missing thing would be the support for a new application type.

The example above demonstrate possible **advantages** of SOA. In contrast with SOA, let us assume some other architectural approach, for example **client-server**. The standard client-server approach usually means that client is tightly coupled to a server. Usually, the server is implemented for a specific client. In SOA, a situation is exact opposite. Since Services are platform independent, they can be re-used by any kind of clients who follow an agreed contract, regardless of the underlying implementation. Every Service invocation is following client-server principles but not every client-server process is following SOA principles. This allows us to recognize more advantages of the SOA over the classical architectural approach. Many of these

2. Representational state transfer

have been defined in Dirksen [2013, p. 6] as follows:

- Business agility/reduced time to market - With more agility, a company can better respond to changes in the market and quickly launch new products and services.
- Reduced costs - With SOA , businesses want to reduce costs by reuse, standards-based development, and a clear view of what services are available and the functionality they provide.
- Improved reuse of services - If services are better defined and a clear inventory of the services is kept, it is s much easier to start reusing existing services.
- Improved software quality - The SOA contains a set of defined standards and best practices. It tells you how to build services, what to do, and what not to do.
- Better interoperability - you have a well-defined contract based on standards to help you in the interoperability area.

SOA certainly offers many advantages. The whole concept revolves around Services - in order to make sure that these Services are well defined and efficient it is necessary to control the process of their development from start to end. This is where the SOA Governance comes useful - it helps to get the most of the added value which SOA has to offer.

2.2 SOA Governance

Before diving into the definition of SOA Governance, it is necessary to mention and define the term around which this discipline revolves - **policy**. Intuitively, it can be understood as a constraint or condition to which some entity must comply. SOA Governance understands policy a bit more specifically - as a **service policy**. In addition to what has already been said about the policy as such, the service policy has been defined by OASIS in its SOA Reference Model as:

Definition 3 *Policy MacKenzie et al. [2006, p. 23]*
Conceptually, there are three aspects of policies: the policy assertion, the policy owner (sometimes referred to as the policy subject) and policy enforcement.

To put this in an example, let us assume that the policy assertion is "every service must pass integration tests before deployed in production", the policy owner could be "Quality Assurance lead" and policy enforcement could be done via several means, for example on a technical level or through some review.

Governance is a term which does not necessarily needs to be applied within the IT business or SOA. More generally, it can be apprehended as a set of rules, policies, laws which are applied within some organization in order to make it run more efficient. In IT context the main motivation behind Governance is to reduce risk of a project failure. As shown in the next image Dirksen [2013, p. 8] produced by Standish Group International this risk is rather high:

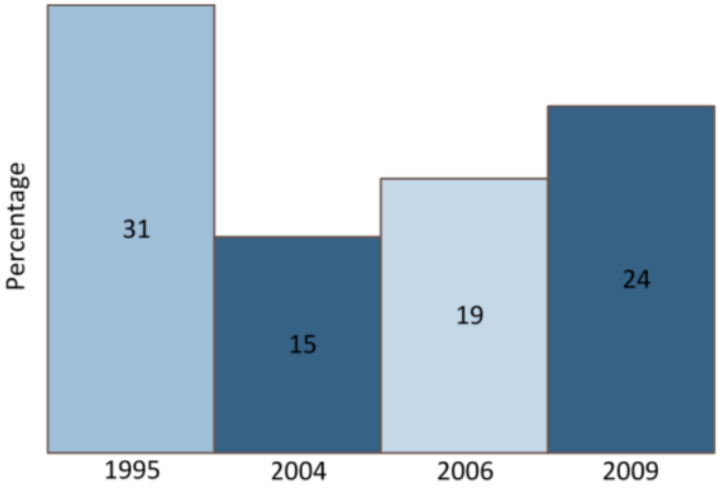

Figure 1.2
Failed projects in 1995, 2004, 2006, and 2009

To put the Governance into a SOA context, following definition is very apposite as it talks about goals of this concept:

Definition 4 *SOA Governance Dirksen [2013, p. 10]*
The goal of applying governance to SOA is to get the most out of your SOA . You do this by listening to stakeholders and, based on that information, defining a number of policies. This requires taking

the following steps:

- *Define policies you want to apply.*
- *Apply these policies during design time.*
- *Monitor and enforce policies during runtime.*

All of these three phases are equally important and omitting any of them will radically reduce the positive outcome of the SOA Governance approach.

When it comes to **defining** policies, it is important that all stakeholders comes together - e.g. CEO, Development Lead, Quality Assurance Manager, Product Support supervisor, SOA Governance specialist, etc. Each stakeholder can have different goals which means that each can consider different set of policies to be important. It is vital that the common ground will be found and an agreement will be made.

Once policies are defined, the next step is to **apply** them during the design time - this is called **Design Time Governance**. This can be done via various means but one very suitable support tool how to achieve this is to implement **Service Lifecycle**.

Definition 5 *Service Lifecycle Dirksen [2013, p. 213]*
Service lifecycle defines the stages a service goes through during its existence. A service lifecycle is important for the following reasons:

- *It helps to identify which services need to be created or updated, because it provides a complete overview of all the services currently running in production.*
- *It helps to make sure all the necessary steps are taken before a service is put in production or is made obsolete.*
- *It can be used to determine which policies need to be complied with in each stage.*

For example, if the policy assertion has been defined as "every service must be linked to a documentation", then it is possible to

model Service Lifecycle in a way that it is simply not possible to go to another stage of the Lifecycle until the resource pointing to the documentation will be entered.
Standard service lifecycle has usually these phases Dirksen [2013, p. 213]:

- model - identify the service based on the incoming requirements
- assemble - define contract, create service, test the service
- deploy - deploy service to a service container and make it available for service consumers
- manage - monitor how the service operates during runtime, if some new requirements or changes has been discovered, then start a new cycle from the model phase.

These four stages are not very fine grained, usually they serve more as a general guideline and are quite enhanced and more detailed in an actual Service Lifecycle model.

The (almost) final stage is to **monitor and enforce** policies during runtime - this is called **Runtime Governance**. This is especially important, because how useful would it be to have a set of well defined policies if there would be no way to make sure that these policies are met during runtime? For example, if policy assertion is defined as "All service invocations must be done by authenticated user" it will be rather easy to solve this on a technical level. The enforcing part could be achieved by putting a service behind some sort of authentication mechanism - i.e. Basic authentication or Form based authentication. The monitoring part could be achieved through logging all attempts of service invocations which ended up with HTTP error 401 - and making these events accessible to respective authorities. Also supporting technologies could be used in Runtime Governance such as Business Activity Monitoring Kolár [2009, chapter 5].

In previous paragraph the word **almost** was mentioned purposefully. Because once services are deployed in production it does not mean that monitoring and enforcing policies during runtime is the

last thing which needs to be done. Based on findings made in both, design time and runtime, policies can be re-defined or new can be added. Also, if Service Lifecycle is in place, it can be remodeled if necessary. And finally, techniques for runtime monitoring and enforcing of policies can be enhanced too.

2.3 Business Process Management

The reason why is Business Process Management (BPM) included in this chapter is because Service Lifecycle is actually implemented as a Business Process. The reason for this is because it is very natural for SOA and BPM to live within one ecosystem. For example, BPM is often used in the context of service orchestration. However this chapter will not focus on this part. For the purpose of this diploma thesis it is sufficient to explain the basic BPM terminology in order to understand why the BPM has been used as an approach for Service Lifecycle implementation.

Term which needs to be defined as first is Process.

Definition 6 *Process De Maio et al. [2014, p. 10]*
In the broadest sense, a process is a series of steps or transformations to achieve a specific objective in a particular context. Like any transformation, it needs to have something to transform and a well-defined desired output of the transformation.

Building a house, shopping in a supermarket, writing a diploma thesis - these are all perfectly valid examples of processes. All of these processes are following a sequence of steps with the desire to achieve an objective.

In context of BPM it is necessary to enhance the process definition with the context of "business".

Definition 7 *Business Process De Maio et al. [2014, p. 11]*
Business processes are a sequence of business activities done by business users and business applications (company or third-party systems) to achieve a business goal for the purpose of a specific increase in value from the business perspective.

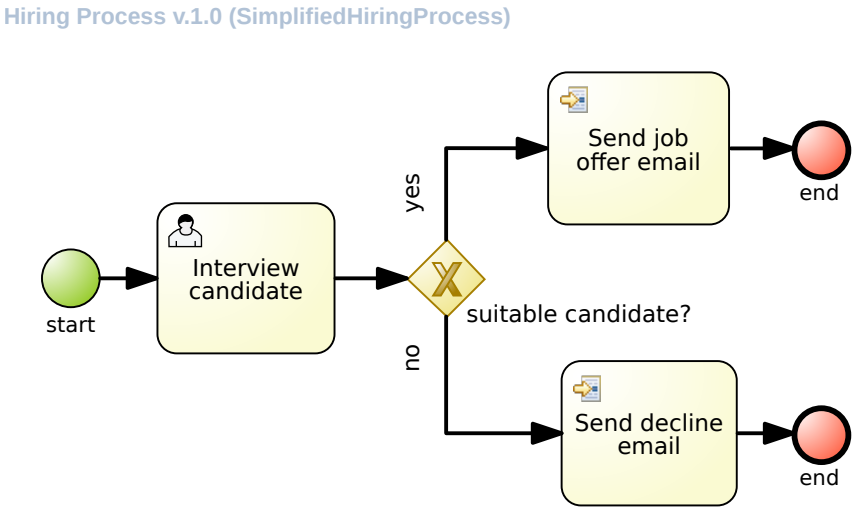

Figure 2.1: Simplified Hiring Process

The noticeable difference is that business process is trying to achieve a **business** goal. Let us take a Hiring Process as shown in a figure 2.1. Business goal is rather clear in this example - to hire a candidate. This Business Process has been visualized by using Business Process Model and Notation (BPMN)[3] which is probably the most commonly used graphical representation for business process specification nowadays.

Based on this very simple example of Business Process it is also possible to derive some of BPM's benefits.

For example, **better change management** - remodeling a specific part of the business process already in place to fulfill a new requirement is usually far less expensive than implementing a new system from scratch.

Effectiveness is another one - when monitoring a Business Process it is possible to focus on finding bottlenecks - this allows continuous process improvement, thus achieving greater effectiveness within the organization.

3. http://www.bpmn.org/

Transparency of the Business Processes is another one - all processes within the organization are known and documented. Every role knows its duties within the process. And what is even more important, if the Business Process has a graphical representation as shown above, it is easily understandable by various parties. One does not need to be a developer in order to understand business process diagram which means that every relevant stakeholder can understand and comment on the process.

Later in the text there are going to be many references to the "instance" of the Service Lifecycle. Moreover, a model of the Service Lifecycle will also be included. Therefore, it is necessary to define these two terms and understand how they differ.

Definition 8 *Process Model and Instance [Weske, 2007, p. 7]*
A business process model consists of a set of activity models and execution constraints between them. A business process instance represents a concrete case in the operational business of a company, consisting of activity instances. Each business process model acts as a blueprint for a set of business process instances, and each activity model acts as a blueprint for a set of activity instances.

Similar relationship as between model and instance exists for example between Class and Object in context of Object Oriented Programming.

Several resources are defining lifecycle of a Business Process in a different way. However, in most cases it includes five stages as shown in figure 2.2. Always have in mind that **Service Lifecycle** - the concept around which is the Service Lifecycle Management Application built is a Business Process. Therefore the figure 2.2 is very much applicable to the Service Lifecycle. It shows how the iterative approach is being applied - once the Service Lifecycle is deployed, it does not mean that it is the last step. On the contrary, it is always possible, based on the monitoring, to discover new areas for optimization which triggers another round of iteration.

Since all the supporting terminology has been defined, it is now possible to define the term Business Process Management as a man-

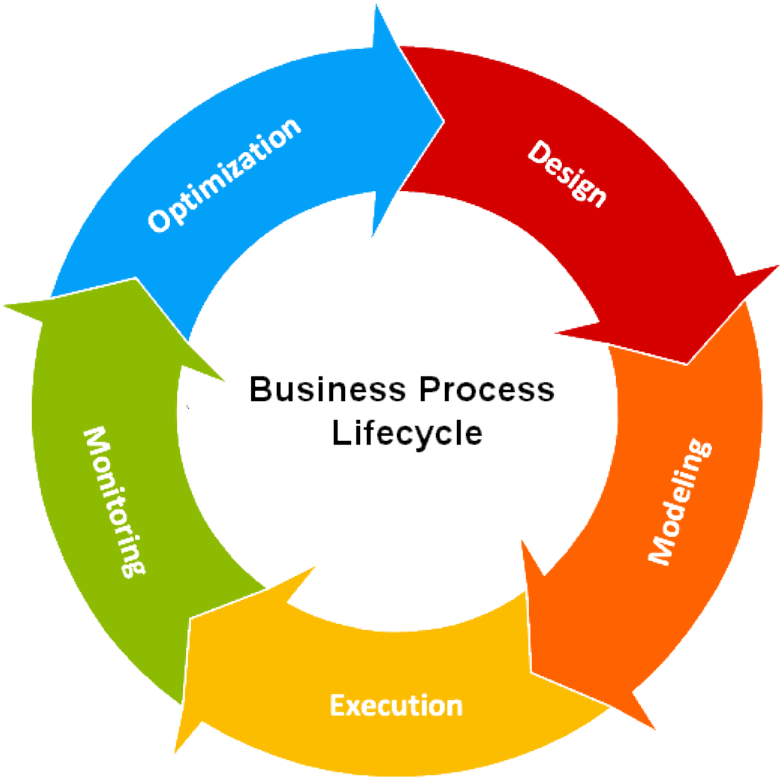

Figure 2.2: Business Process Lifecycle

agement discipline.

Definition 9 *Business process management Weske [2007, p. 5]*
Management discipline which includes concepts, methods and techniques to support the design, administration, configuration, enactment, and analysis of business processes.

Finally, if an organization wants to adopt BPM there are many systems which can help.

Definition 10 *Business Process Management System (BPMS) Weske [2007, p. 6]*
Generic software system that is driven by explicit process representations to coordinate the enactment of business processes.

BPMS is usually tightly bound to the Business Process Lifecycle. Common BPMS supports most of the stages of this lifecycle. By support it is meant that it provides the necessary **tooling** for design and modeling phase as well as the **runtime environment** for execution of the process. Sometimes, less mature BPMS are missing support for Monitoring phase.

jBPM 6[4], Bizagi[5], IBM BPM[6] are all examples of commonly used Business Process Management Systems.

This chapter defined the terminology which is practically used in the Service Lifecycle Management Application. It introduced Service Lifecycle as a key part of Design Time Governance and its benefits. It also explained the relationship between Service Lifecycle and Business Process management. The basic version of Service Lifecycle has been also mentioned - the enhanced version will be introduced later in the text. The next chapter will set many of these terms in the practical context so it will be clear how this theoretical background can be applied in the context of SLMA.

---

4. http://jbpm.org/
5. http://www.bizagi.com/
6. http://www-03.ibm.com/software/products/en/business-process-manager-family

3 Analysis

In order to successfully implement any application with extent bigger than `hello world` it is necessary to spend some time on the analysis. This chapter will discuss all analytic steps which had to be performed before the actual programming can begin. Many supporting resources such as diagrams, drafts, models will be included in order to make this chapter as much comprehensible for a reader as possible.

3.1 Overall architecture

Firstly, the overall architecture of the project had to be determined. This includes more than the SLMA itself as SLMA uses many supporting technologies which had to be taken into consideration when designing the architecture. The final proposal is shown in the figure 3.1.

The overall idea was to try to design the architecture as loosely coupled in order to achieve re-usability and ease of maintenance.

As shown in the architecture diagram servers communicate with each other via REST API[1]. SOA Server and BPM Server does not really depend on Application Server; they can be used independently. This architecture resembles client-server approach where Application Server can be understood as Client and remaining two as servers. End user of the SLMA will not have to care about remaining servers. Every communication will be done transparently and he will be presented only with the result data.
SOA Server and BPM Server could be used also by other clients - besides SLMA. This can be beneficial if organizations adopt also BPM as such (in addition to SOA Governance). Then this server could be utilized also for execution of other processes - not only for execution of Service Lifecycle. It also means that most of the libraries necessary for process execution are packaged in the BPM Server which makes the SLMA application rather lightweight because it only needs to use libraries necessary for remote communication and for graphical in-

1. http://en.wikipedia.org/wiki/Representational_state_transfer

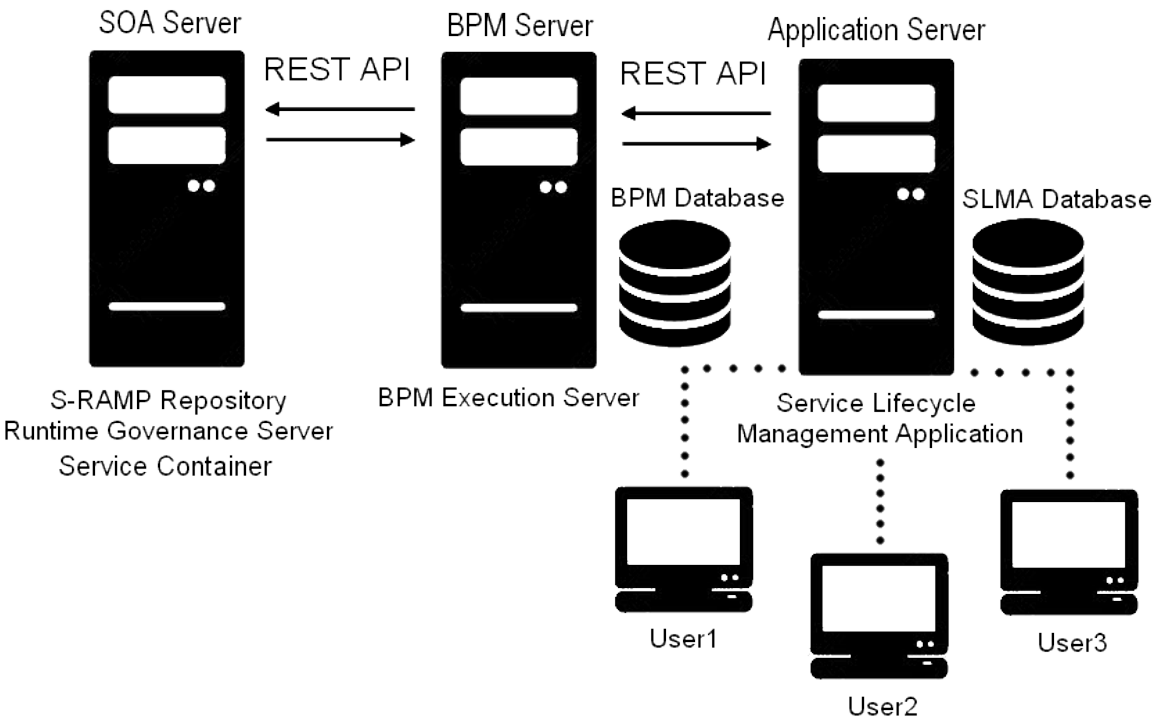

Figure 3.1: High level project architecture

terface.

Finally, this architecture - when the BPM execution server is deployed separately and the client communicates with it remotely is rather common in real organizations. I do not have any hard evidence supporting this statement but as a middleware Support Engineer I have observed this to be a very common practice. Therefore this architecture is trying to address real needs of customers adopting SOA Governance and also trying to prove that it is possible to achieve it by using JBoss open-source technologies.

3.1.1 Server descriptions

This section describes servers used in the project architecture. Each one of them is used specifically for selected components, usually hosting one or more JBoss open source technologies.

SOA Server includes SOA related components. **Service container** serves as an execution environment for a Service - this is necessary if the service is intended to be executable. The SLMA needs services to be executable for displaying invocations of retired services which is one of the use cases of the SLMA and will be discussed later.

Next, there is a **SOA Repository Artifact Model and Protocol (S-RAMP)** component, which is defined by OASIS as follows:

Definition 11 *S-RAMP Stam and Wittmann [2013, chap. 1]*
The "SOA - Repository Artifact Model and Protocol" (S-RAMP) specification defines a common data model for SOA repositories to facilitate the use of common tooling and sharing of data

SLMA needs S-RAMP as a storage for relevant SOA artifacts - such as Ontologies and Services (their deployable binary version). Ontology is a standard defined by OpenGroup as follows:

Definition 12 *Ontology Ope [2010]*
The ontology is designed for use by:

- *Business people, to give them a deeper understanding of SOA concepts and how they are used in the enterprise and its environment*
- *Architects, as metadata for architectural artifacts*
- *Architecture methodologists, as a component of SOA metamodels*
- *System and software designers for guidance in terminology and structure*

Ontology can be used for Classifying artifacts as explained in Stam and Wittmann [2013, chap. 3]. To make it even more specific, Ontology includes classes which will be applied from the Service Lifecycle to the Service artifact based on the current stage of the service - i.e.

InTest, Retired, etc.

Finally, there is also a **Runtime Governance** component which serves for monitoring the Service execution environment during runtime.

BPM Server This server includes BPM execution server and related database which is necessary if business processes are supposed to be persistent. Once again, it should be noted that the Service Lifecycle is actually a Business Process. Which means that Service Lifecycle will be deployed in this server and this Server will take care of its execution based on the incoming request from the Application Server.

Application Server Finally, the Application Server will host the SLMA itself. End user of the application will only access this server and does not need to be concerned with the rest. Due to the architectural design the SLMA can be deployed as rather lightweight since its dependencies consist merely of UI libraries and libraries necessary for remote invocation. Based on different use cases user will perform the corresponding REST request will be sent to the BPM Server - for example **Start Service Lifecycle** or **Move to the next stage of the Service Lifecycle**. BPM Server will handle these incoming requests and delegate them to the BPM execution server. Some incoming requests may trigger others, for example, when there is a need to set a Classification on the Artifact deployed in the SOA Server - however, this will all be happening transparently to the end user. For one specific use case which SLMA includes, there also has to be a database deployed on the same server.

Even though the overall architecture may seem a bit complicated it is weighed by its advantages - loose coupling, lightweight, transparency to the end user, simple maintenance.

3.2 Use cases

The use cases implemented in SLMA are derived from the official assignment. For a quick summary this is what the assignment says:

> Final application should be able to:
>
> - search S-RAMP repository for SwitchYard Services

- execute custom lifecycle for those services on demand
- monitor service lifecycle stage
- support executing notification actions once a deprecated service is used

These requirements were put together based on the Service Lifecycle definition as shown in chapter 1.

3.2.1 Use case diagram

The SLMA use cases are designed in a way that they fulfill all requirements defined in previous paragraph. The final version of the use case diagram is displayed in the figure 3.2.

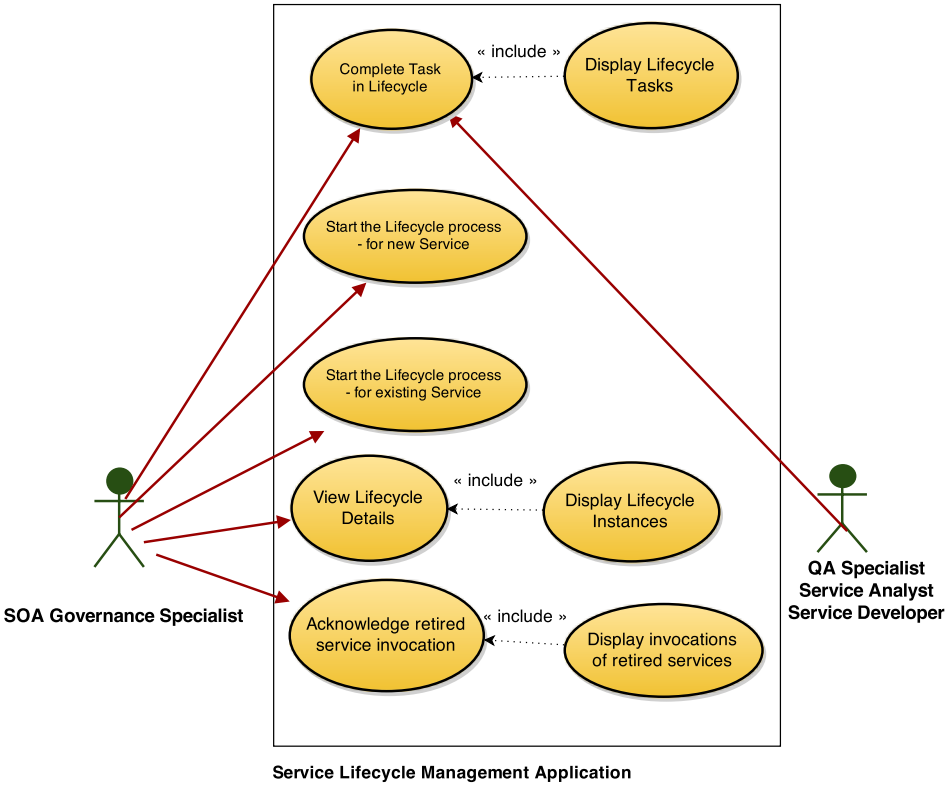

Figure 3.2: Use Case Diagram

Requirement **execute custom lifecycle for services on demand** is directly covered by these use cases:

- Complete Task in Lifecycle
- Start the Lifecycle Process - for new Service
- Start the Lifecycle process - for existing Services

These use cases are allowing user to spawn new instance of the Service Lifecycle process and work on it - by completing defined tasks in the lifecycle. The "on-demand" requirement is simply achieved by the fact that user decides **when** and **for what** services the Service Lifecycle will be activated.

Requirement **support executing notification actions once a deprecated service is used** is covered by use case "Acknowledge Retired Service Invocation".

Requirement **monitor service lifecycle stage** is included in "View Lifecycle Details" which displays the detailed information about the selected Lifecycle instance including the metadata, variables bound to the selected instance and also graphical process diagram which shows what is the current stage of a given instance of a Service Lifecycle.

Finally, the requirement **search S-RAMP repository for SwitchYard Services** is not modeled directly in the use case diagram. The reason for this is that it has been integrated directly into the process model of a Service Lifecycle and it will be discussed in the next chapter.

The use case diagram showed in 3.2 introduces also **actors** of the SLMA. Only the most basics ones were introduced but should be sufficient in most SOA Governance environments. The differentiation between them is simple.

Role **SOA Governance Specialist** is able to execute every use case of the application.

Remaining roles are only able to work on the Service Lifecycle task which has been directly assigned to them.

3.2.2 Use case details

Use cases defined in this application has been showed in the figure 3.2. This section will provide details of selected use cases which which will show how can the user of the SLMA interact with the application.

Start the Lifecycle Process - for new Service

Use case specification	Start the Lifecycle Process - for new Service
Description	This use case describes how the user can start new instance of the Service Lifecycle, specifically for a new (not yet created) Service
Actors	SOA Governance Specialist
Pre conditions	User is logged into the application
Basic Flow of Events	User clicks on the button labeled with "Start the Lifecycle Process - for new Service"
Post Conditions	New instance of Service Lifecycle process is created and user is redirected to a page which displays the first available task of the Lifecycle Process instance which he has just started

View lifecycle details

Use case specification	View lifecycle details
Description	This use case describes how can the user display details of a specific Service Lifecycle instance
Actors	SOA Governance Specialist
Pre conditions	User is logged into the application
Basic Flow of Events	1. User clicks on "Lifecycle Instances" button which displays the list of ACTIVE (by default) instances formatted in a table 2. User clicks on "Show Details" button which is located in every row of this table

Post Conditions	User is redirected to a page which displays the Service Lifecycle details of a selected instance. These details includes instance metadata, variables bound to this instance and process diagram which displays the current stage of the Service Lifecycle

Complete Task in Lifecycle

Use case specification	Complete Task in Lifecycle
Description	This use case describes how can the user complete specific task in a given instance of a Service Lifecycle
Actors	SOA Governance Specialist, QA Specialist, Service Developer, Service Analyst
Pre conditions	User is logged into the application
Basic Flow of Events	1. User clicks on "Lifecycle Tasks" button which displays the list of available Tasks for a logged user formatted in a table 2. User clicks on "Work on Task" button of a desired Task 3. User will be redirected to a Form where he needs to proceed according to the Task description 4. Once all required outputs are correctly filled in, user can click on "Complete Task" button
Post Conditions	Task is completed and user will be redirected to the "Lifecycle Tasks" page so he can choose another task to work on
Alternative Flows	If user clicked on "Complete Tasks" and some of the output fields had wrong / forbidden values he will be redirect to the same page and asked for correcting these output fields

Acknowledge retired service invocation

Use case specification	Acknowledge retired service invocation
Description	This use case describes how can the user acknowledge retired service invocation
Actors	SOA Governance Specialist
Pre conditions	User is logged into the application
Basic Flow of Events	1. User clicks on "Retired Services" button which displays the list of invocations of a retired services (for which instance of Service Lifecycle has been created) formatted in a table 2. User clicks on "send notification" button of a desired Invocation 3. User will be redirected to a another page which displays the Form with additional details of selected invocation 4. User clicks on "Process Notification" button which will "Acknowledge" retired service invocation. This specific invocation will not be displayed in the table displayed in "Retired Services" page anymore
Post Conditions	Retired Service invocation is processed and user is redirected to the remaining list of (unprocessed) retired service invocations
Alternative Flows	If user checked "Send Email" checkbox in step 3 he will be able to send email notification about this specific retired service invocation

This section talked about use cases of SLMA. It explained how they are linked to the official assignment of the thesis which actually supports the definition of the Service Lifecycle. Moreover, selected use cases were described more in depth.

3.3 Service Lifecycle Models

It has been mentioned numerous times that Service Lifecycle is actually a Business Process. Basic stages of the Service Lifecycle has been defined in chapter 2.2. The final model of a Service Lifecycle business process has used these basic stages and enhanced them significantly by adding more fine-grained steps. Two models are executable within the SLMA. One for a "new service" and the other one for "Existing Service".

The final model of **Service Lifecycle - for new Service** is displayed in the figure 3.3.

The main idea of this process is that the Service is not yet created. Lifecycle starts from the very beginning - by identifying the new service and then the service goes through different stages such as ServiceIdentification, InitialService, Registered, InTest, Available, Deprecated, Retired. Whenever service enters a different stage of lifecycle there is a Task which has to be completed by the user before moving to another. This task forces user to enter essential information regarding service - these information are always specific for the current Task. All tasks with their corresponding outputs which needs to be entered by the user are documented in the attachment A.

In figure 3.3 there are also some nodes with the "+" sign in the bottom. These nodes are actually representing sub process. The reason why all the logic is not included in the one single model is better readability of the diagram and also the re-usability - many sub processes are actually used by both models. The example of such sub process is displayed in figure 3.4

Most of sub processes has the same structure as the one shown in the figure 3.4. First, there is a Script Task used for setting a process variable with the actual stage of the Service.

Definition 13 *Script Task De Maio et al. [2014, p. 76]*
This Task allows us to execute a script that can be specified in various languages. A script basically represents a set of actions that we can code using a scripting language.

Then there is a Service task which includes a custom logic - in this

case it includes logic for setting the classification of an artifact stored in the S-RAMP repository based on the Service state.

Definition 14 *Service Task De Maio et al. [2014, p. 75]*
This task allows us to represent interactions with external automated systems. Each time our business process needs to interact with a service or procedure, we will use a service task. The service task element defines an attribute called implementation, which is used to specify the underlying implementation of the service that we are calling.

Finally, there is a User Task which will be executed by the user of the SLMA.

Definition 15 *User Task De Maio et al. [2014, p. 75]*
This task represents a human interaction. Each time we want to represent a person doing an activity, we use a User task to model this situation. Because User tasks represent a human interaction, we need to provide a way to assist the performer during this interaction.

This process model shows different paths which can the Service take. The "happy path" is the one where service will pass tests - the final stage of the Service Lifecycle is the retirement of the service. Once Service hits this stage it should not be invoked anymore - this policy will be controlled during runtime.

The other version of Service Lifecycle is called **Service Lifecycle - for existing Service**. The main difference of this version of the lifecycle is, that it allows user to work with the Service which has been **already** loaded into the S-RAMP repository. When this process is being instantiated, first the user has to fill in some service description and current service state, then he can bind this lifecycle to some specific service artifact stored in S-RAMP repository - once done, the process will continue with a corresponding branch which will be determined based on the Status entered by the user. The final version of this Lifecycle is displayed in a figure 3.5.

3.4 Service Lifecycle Ontology

As explained in definition 12 the ontology can be used for specifying some additional metadata. SLMA uses ontology for Classification of the Service artifact stored in S-RAMP repository. To be even more specific - the classifier represents the Stage of the service. This is the ontology used by the SLMA:

Listing 3.1: Ontology used for setting the Service state

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdfs="http://
www.w3.org/2000/01/rdf-schema#"
   xmlns:rdf="http://
   www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:owl="http://
   www.w3.org/2002/07/owl#"
   xmlns:xsi="http://
   www.w3.org/2001/XMLSchema-instance"
   xml:base="http://
   www.jboss.org/overlord/service-lifecycle.owl">

   <owl:Ontology rdf:ID="ServiceLifecycleStatus">
     <rdfs:label>Service Lifecycle Status
     </rdfs:label>
     <rdfs:comment>Service Lifecycle Status
     Ontology
     </rdfs:comment>
   </owl:Ontology>

   <owl:Class rdf:ID="ServiceLifecycle">
     <rdfs:label>Service lifecycle</rdfs:label>
     <rdfs:comment>Root status -
     participating in Service Lifecycle Workflow
     </rdfs:comment>
   </owl:Class>

   <!-- Basic set of possible service states -->
```

```
<owl:Class rdf:ID="ServiceIdentification">
  <rdfs:subClassOf rdf:resource="http://
  www.jboss.org/overlord/service-lifecycle.owl
  #ServiceLifecycle" />
  <rdfs:label>Service Identification
  </rdfs:label>
  <rdfs:comment>Service Identification
  </rdfs:comment>
</owl:Class>
<owl:Class rdf:ID="InitialService">
  <rdfs:subClassOf rdf:resource="http://
  www.jboss.org/overlord/service-lifecycle.owl
  #ServiceLifecycle" />
  <rdfs:label>Initial state of service
  </rdfs:label>
  <rdfs:comment>Initial state of service
  </rdfs:comment>
</owl:Class>
<owl:Class rdf:ID="Registered">
  <rdfs:subClassOf rdf:resource="http://
  www.jboss.org/overlord/service-lifecycle.owl
  #ServiceLifecycle" />
  <rdfs:label>Registered</rdfs:label>
  <rdfs:comment>Registered
  </rdfs:comment>
</owl:Class>

  <owl:Class rdf:ID="InTest">
  <rdfs:subClassOf rdf:resource="http://
  www.jboss.org/overlord/service-lifecycle.owl
  #ServiceLifecycle" />
  <rdfs:label>In Test</rdfs:label>
  <rdfs:comment>In Test
  </rdfs:comment>
</owl:Class>

    <owl:Class rdf:ID="Available">
```

```
    <rdfs:subClassOf rdf:resource="http://
    www.jboss.org/overlord/service-lifecycle.owl
    #ServiceLifecycle" />
    <rdfs:label>Available</rdfs:label>
    <rdfs:comment>Available
    </rdfs:comment>
  </owl:Class>

    <owl:Class rdf:ID="Deprecated">
    <rdfs:subClassOf rdf:resource="http://
    www.jboss.org/overlord/service-lifecycle.owl
    #ServiceLifecycle" />
    <rdfs:label>Deprecated</rdfs:label>
    <rdfs:comment>Deprecated
    </rdfs:comment>
  </owl:Class>

      <owl:Class rdf:ID="Retired">
    <rdfs:subClassOf rdf:resource="http://
    www.jboss.org/overlord/service-lifecycle.owl
    #ServiceLifecycle" />
    <rdfs:label>Retired</rdfs:label>
    <rdfs:comment>Retired
    </rdfs:comment>
  </owl:Class>

</rdf:RDF>
```

The usage is pretty straightforward - as Service moves through different states, corresponding `subClass` of the ontology above is set to the Service artifact.

3.5 User Interface Mockup

Final part of the analysis was the visual side of the SLMA. The SLMA is implemented as a web application and user can access it through his Web Browser.

The mockup of the user interface was achieved through wireframes. Wireframe is simply a visual guide representing the skeleton of the specific Web Page. Wireframe should be simple but complete. The focus of the final design was to achieve as much simplicity as possible. User Interface of the SLMA has very simple requirement - it has to support all of the use cases as listed in section 3.2. The listing of all wireframes which together create the User Interface mockup can be found in the attachment B. These wireframes has been used heavily in the development of the visual side of the Application but it is possible that the final version of the application has some differences.

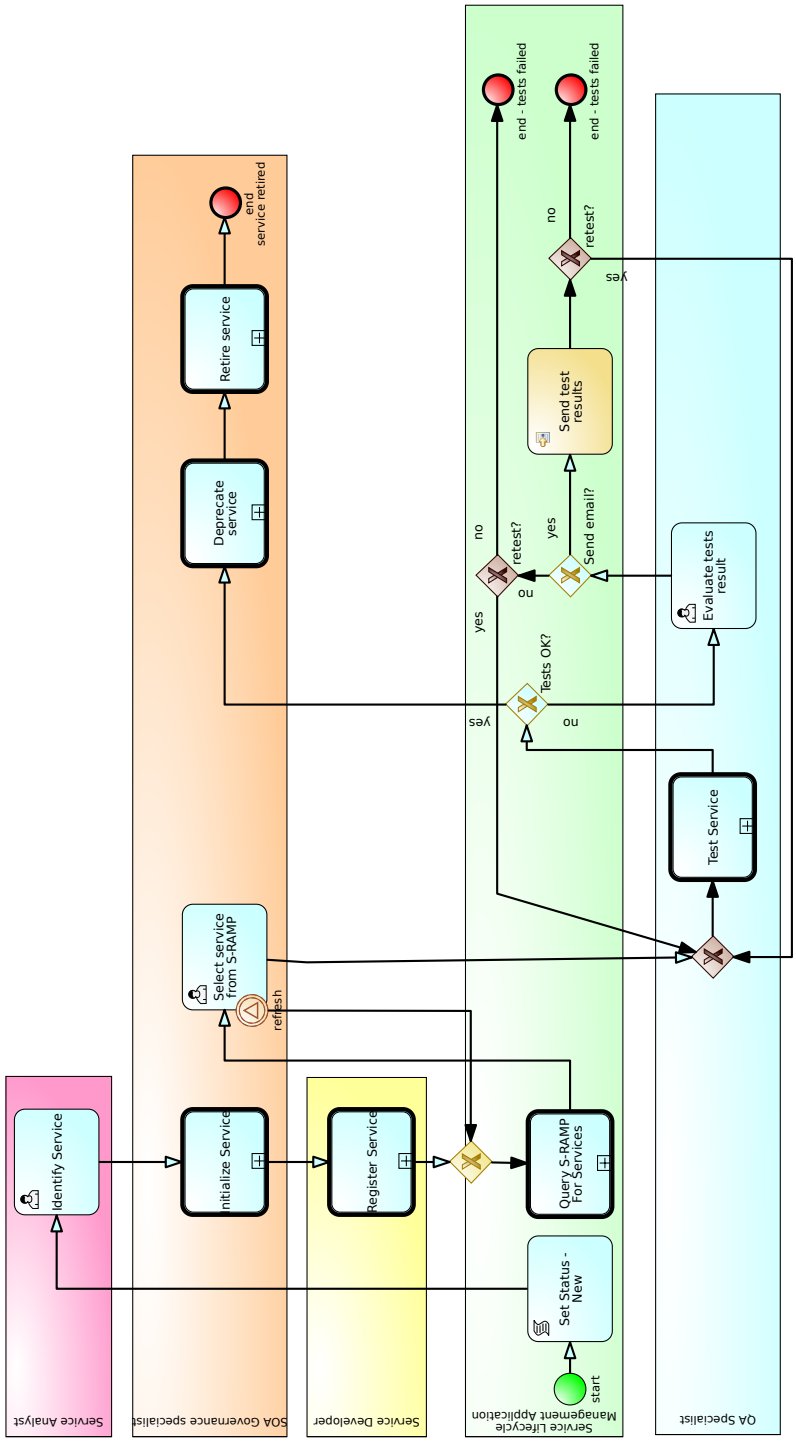

Figure 3.3: Process model of "Service Lifecycle - for new Service"

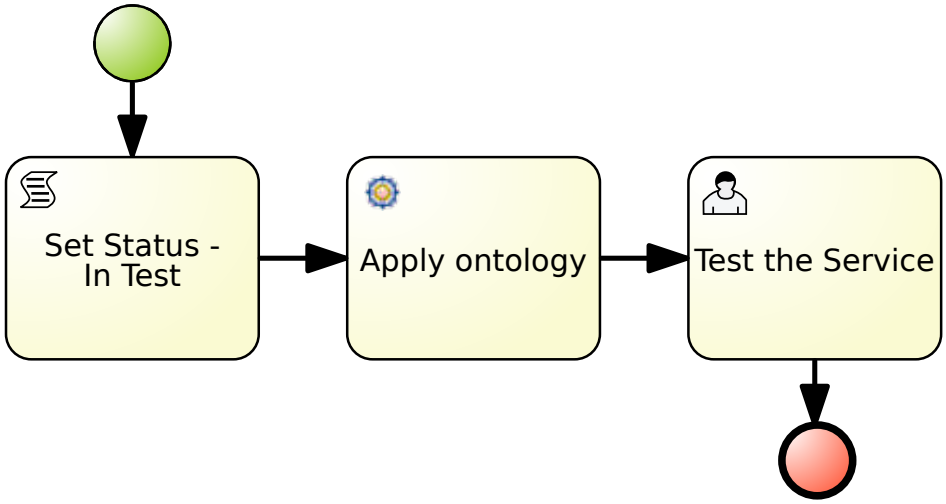

Figure 3.4: Sub Process - Test the Service

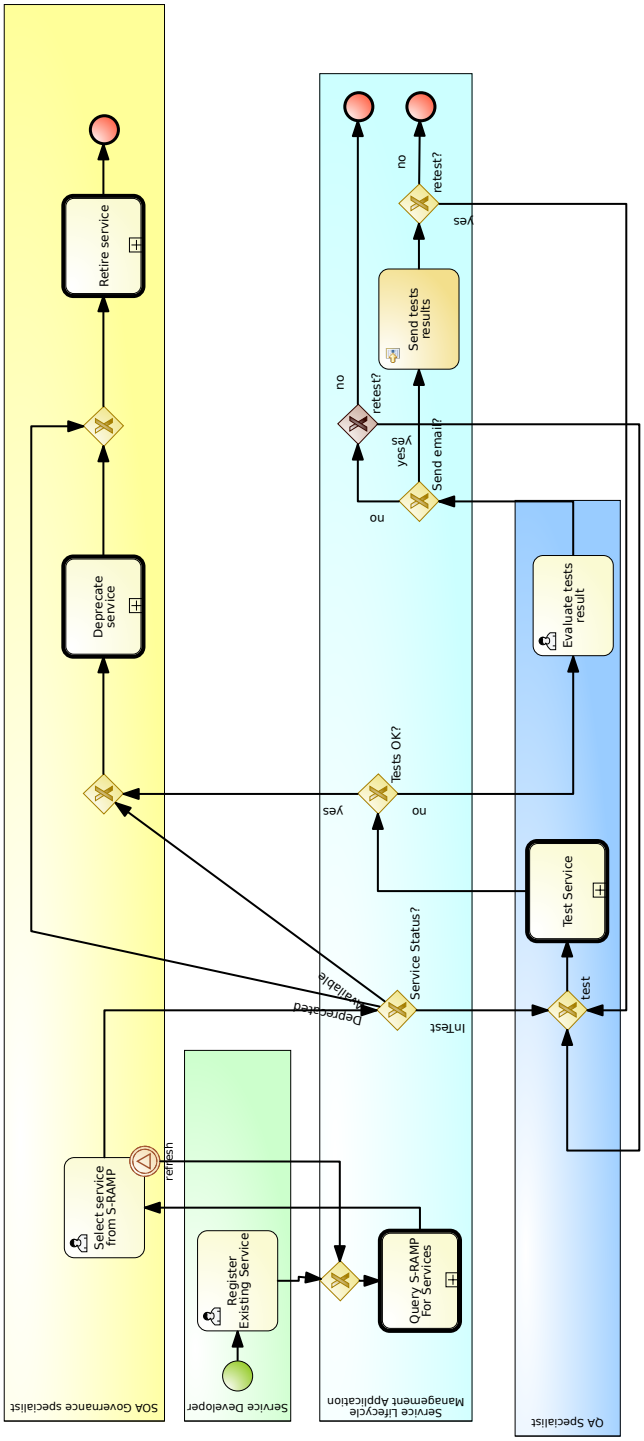

Figure 3.5: Process model of "Service Lifecycle - for existing Service"

4 Technological stack

So far, specific technologies were not really relevant. The concept of this thesis, the application design and analysis covered so far, are really technological agnostic. However, one of the objective of this thesis, as explained in section 1.1 was to achieve implementation of SLMA by using JBoss open source technologies. This chapter will introduce in few lines every technology and project used in the implementation of the SLMA and also show the role of the selected technology within the application. These technologies will also be linked to the servers defined in section 3.1.

4.1 JBoss Enterprise Application Platform

JBoss Enterprise Application Platform (JBoss EAP) is a key technology used in this diploma thesis. JBoss EAP in version 6 is used in the SLMA.

Definition 16 *JBoss EAP EAP [2014, chap. 1]*
Red Hat JBoss Enterprise Application Platform 6 (JBoss EAP 6) is a middleware platform built on open standards and compliant with the Java Enterprise Edition 6 specification. It integrates JBoss Application Server 7 with high-availability clustering, messaging, distributed caching, and other technologies.

In figure 3.1 there are multiple servers showed. These servers are actually instances of JBoss EAP with something extra deployed on top of it (depending on a server). JBoss EAP 6 is a JavaEE 6 compliant application server which is used as an underlying platform for deployment throughout the whole project.

4.2 jBPM

jBPM is another open source project from the JBoss family focused on Business Process Management. jBPM in version 6 is used in the SLMA.

Definition 17 *jBPM*
jBPM is an open source Business Process Management System. It supports creating, validating, simulating, deploying and executing Business Processes and rules.

The overall architecture of the jBPM project is displayed in the figure 4.1 jBP [2014, chap. 1]. As shown in the architecture, jBPM in-

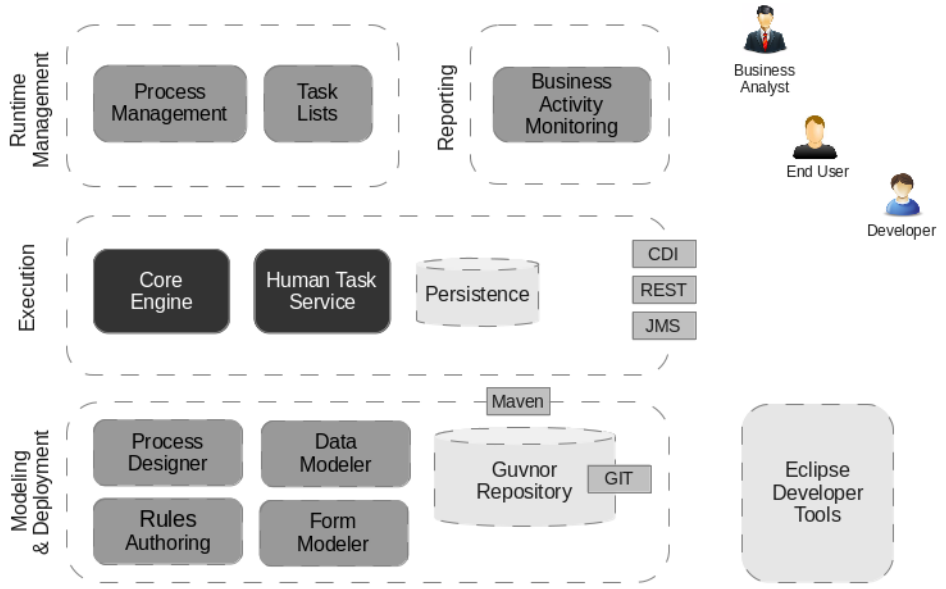

Figure 4.1: jBPM Overall architecture

cludes components for process modeling but also for their execution and monitoring. REST interface is also showed in this figure. This is exactly the integration point used by the SLMA.

jBPM has been used in many places in this diploma thesis. For example, business processes were modeled with tooling provided by jBPM - results of this modeling were included in the figure 3.3 and 3.5. Moreover jBPM is actually used as the execution environment for business processes - in this case, for execution of the Service Lifecycle itself. This is achieved by the REST API which jBPM execution server provides. SLMA sends requests to endpoints exposed

by jBPM execution server and by that effectively achieving the execution of the Service Lifecycle instance. jBPM is deployed on BPM Server.

4.3 Overlord S-RAMP

Overlord S-RAMP is the implementation of the S-RAMP specification as explained in definition 11. Overlord S-RAMP in version 0.6 is used in the SLMA.

Definition 18 *Overlord S-RAMP SRA*
Overlord S-RAMP is an artifact repository comprised of a common data model, multiple interfaces, and powerful tools. It implements the OASIS S-RAMP specification which defines the data model, query language, and an Atom REST binding.

Overlord S-RAMP is one of the SOA components used in this project. It is utilized as a Service Artifact storage - binary version of Service are being stored here. Moreover, the Ontology file which defines several classifications is also stored in here. SLMA communicates indirectly with the Overlord S-RAMP deployed on SOA Server in order to set the specific classification of the Service Artifact based on its current state.

4.4 SwitchYard

SwitchYard is another SOA component used in this diploma thesis. It is deployed on SOA Server and it serves as a Service Container.

Definition 19 *SwitchYard Swi*
A lightweight service delivery framework providing full lifecycle support for developing, deploying, and managing service-oriented applications.

SwitchYard is not directly used by the SLMA. However, it plays an important role in the project since all the Services used for demon-

stration purposes are actually SwitchYard services. Moreover, there is a Use Case as defined in section 3.2.2 which allows user to monitor all retired service invocations. In order to actually invoke a service it has to be deployed in some Service container - which is another purpose of SwitchYard.

4.5 Overlord Runtime Governance

Overlord Runtime Governance (Overlord RTGov) is a component which can be directly used in implementing the SOA Governance solution, which SLMA is. Overlord RTGov in version 2 is used in this project.

Definition 20 *Overlord RTGov*
Overlord RTGov is a framework which monitors the service execution environment and stores the activity related information from the run time.

The overall architecture mainly focused on collection and reporting features of RTGov is displayed in the figure 4.2

Overlord RTGov in context of SLMA is used for enforcing Service Policy during runtime. The one policy which is being enforced and monitored has been actually defined in the very beginning in section 1.1. RTGov is storing information regarding service invocations. As displayed in the figure 4.2 these information are exposed through the REST. SLMA is querying this API and checking whether retired service by any chance was not invoked. If so, this invocation is presented to the end user of the SLMA and he can deal with this invocation as explained in the Use Case specification defined in section 3.2.2.

4.6 Vaadin

Vaadin was selected as a framework for building the User Interface of the SLMA. It is an open source project and version 7.3.2 was used in the SLMA.

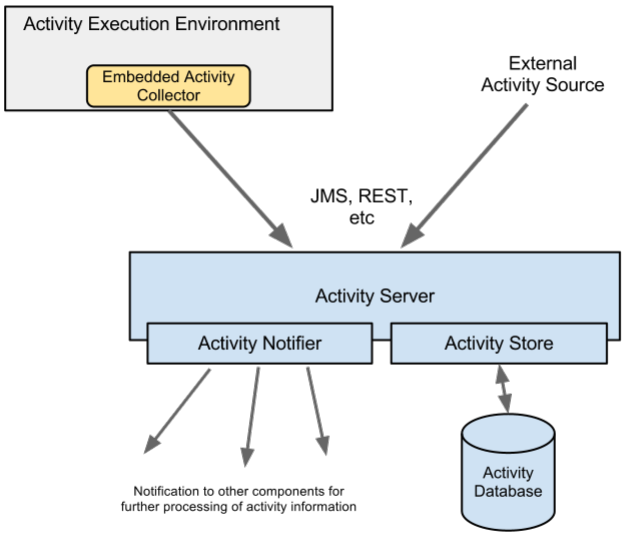

Figure 4.2: Collection and reporting architecture

Definition 21 *Vaadin Grönroos [2014]*
Vaadin is an AJAX web application development framework that enables developers to build high-quality user interfaces with Java, both on the server- and client-side. It provides a set of libraries of ready-to-use user interface components and a clean framework for creating your own components.The focus is on ease-of-use, re-usability, extensibility, and meeting the requirements of large enterprise applications.

Vaadin is used as the framework for front-end development of the SLMA. It could be said that the User Interface Mockup as defined in section 3.5 is implemented by Vaadin. In Vaadin respective web pages are implemented as java classes. From these java classes additional logic is being invoked. In case of SLMA this logic mostly represents REST API invocation - REST requests are being sent from

the Vaadin application (SLMA) to the remote BPM Server and response is then presented to the end user.

4.7 Alternative - Overlord Design Time Governance

There was not much space for alternative technologies which could be used in this project. The main reason behind this is that the official assignment of this thesis clearly stated that JBoss open source technologies are supposed to be used, and the selected ones are simply the most suitable for usage in SOA Governance solution, which SLMA is.

However, there is one other community project, Overlord Design Time Governance (Overlord DTGov) which could be used for implementing the SLMA - mainly for the execution of the Service Lifecycle.

Definition 22 *Overlord DTGov DTG [2014, chap. 1]*
The DTGov project layers Design Time Governance functionality on top of an S-RAMP repository. These two projects work together to provide the following:

- *Store and Govern artifacts*
- *Custom Governance Workflows*
- *Integrated Governance Human Task Management*

For executing Governance Workflows DTGov also uses jBPM framework.

The main reason why Overlord DTGov was not used, but instead a similar output was achieved through custom integration between jBPM and S-RAMP is due to one of the requirement from the official assignment.

As stated in section 3.2 the SLMA should execute Service Lifecycle on demand. This is not possible using Overlord DTGov as the lifecycle in DTGov is triggered every time an artifact is stored in the S-RAMP repository.

Moreover, Overlord DTGov bundles jBPM libraries for the lifecycle execution and it can happen that they might be outdated. Therefore, if one wants to use the latest jBPM libraries it is really more suitable to use jBPM as a separate execution server - just the way SLMA uses it.

5 Selected implementation details

It is not so interesting to describe the whole implementation of the SLMA. For example, the signification portion of the application's source codes consists of classes representing specific web pages which are presented to the end user. From the developer's perspective this area of the program may not appear so interesting. What is more interesting are the integration points which are implemented in the SLMA. For exampleApplication Server communicates with BPM Server - this is implemented by integrating these two servers using REST.

This chapter will briefly describe selected implementation details - focused on the integration parts. When suitable, code examples will be shared too. As already stated, not every aspect will be described and therefore it may be useful to inspect the application source codes directly for further implementation details.

5.1 jBPM Integration with SLMA

This is probably the most important integration point throughout the whole project. As showed in the figure 3.1 the SLMA and the jBPM execution server are deployed on separate servers. SLMA has to inform the jBPM server whenever user performs some action - such as "Start the process" or "Complete Task". This is achieved via integration over REST API which is exposed by the jBPM execution server.

jBPM provides library called **kie-remote-client**[1] which serves as a native java client for remote communication. This library is packaged within the SLMA. For example, if end user of SLMA press "Start new Service Lifecycle - for new Service" button, code displayed in the listing 5.1 will be executed.

Listing 5.1: Code which starts the process deployed in jBPM execution Server

```
// Start the service lifecycle instance
```

1. http://mvnrepository.com/artifact/org.kie.remote/kie-remote-client

```
        RuntimeEngineWrapper runtimeEngine;

ProcessInstance processInstance = runtimeEngine
.getInstance().getEngine()
.getKieSession().startProcess(processid);

//...corresponding part
//from the RuntimeEngineWrapper

public static RuntimeEngineWrapper getInstance
(String username, String password)
throws IOException {
// if another user is logged
//create new runtimeengine for him
if (instance == null ||
RuntimeEngineWrapper.username != username) {
instance = new RuntimeEngineWrapper();
properties = new JBPMProperties();
RemoteRestRuntimeEngineFactory factory =
RemoteRestRuntimeEngineFactory.newBuilder()
.addDeploymentId(properties.getDeploymentId())
.addUrl(properties.getUrl())
.addUserName(username).addPassword(password)
.buildFactory();
RuntimeEngineWrapper.username = username;
RuntimeEngineWrapper.password = password;
RuntimeEngineWrapper.setEngine(factory
.newRuntimeEngine());
}
return instance;
}
```

As showed in the code, the `RemoteRestRuntimeEngineFactory` returns the `RuntimeEngine` which is an object which can be used for remote interaction with the jBPM Execution Server. It is necessary to configure the `RemoteRestRuntimeEngineFactory` with

couple of properties which are shipped in the standard Property[2] file with the SLMA. What is also important that for every logged user it is necessary to instantiate new `RuntimeEngine`. This is necessary if user is supposed to see only Tasks which are assigned to him.

This integration is effectively implementing requirement "execute custom lifecycle for those services on demand" as mentioned in section 3.2. The same integration principle is used when implementing requirement "monitor service lifecycle stage" because `RemoteRuntimeEngine` provides access to `AuditLogService` which allows jBPM developer to get access to the history information regarding specific process instance. Based on these history information it is possible to determine current stage of the Service Lifecycle and display it to the end user of the application.

5.2 jBPM integration with S-RAMP

As showed in the figure 3.5 and 3.3 there is a Service task called "Query S-RAMP for Services". In figure 3.4 there is Service task called "Apply Ontology". Both of these tasks are communicating with the S-RAMP repository which is deployed on a separate SOA Server. This communication is also achieved via REST API. Service Task can be configured as explained in De Maio et al. [2014, p. 140]. The key part is the class which implements **`WorkItemHandler` interface**[3]. Every Service Task has its own implementation - once the process flow reaches the Service Task node the **`executeWorkItem`** method will be invoked. This method will include the necessary business logic which is related to the specific Service Task.

Let us assume that user is executing "Service Lifecycle - for new Service" process and he just ended task with name "Register Service". The process flow continues to the "Query S-RAMP For Services" task and engine will execute `executeWorkItem` method which includes code displayed in the listing 5.2.

2. http://docs.oracle.com/javase/7/docs/api/java/util/Properties.html
3. http://docs.jboss.org/jbpm/v6.1/javadocs/org/kie/api/runtime/process/WorkItemHandler.html

Listing 5.2: Code which starts the process deployed in jBPM execution Server

```
public void executeWorkItem(WorkItem wi,
WorkItemManager wm) {
Map<String, Object> params = wi.getParameters();

SRAMPClient srampClient = new SRAMPClient((
String) params.get("inUsername"),
(String) params.get("inPassword"),
sPort, (String) params.get("inHost"));

List<Service> serviceList = null;
serviceList = this.srampClient.getServices();

Map<String, Object> result =
new HashMap<String, Object>();
result.put("OutServiceList", serviceList);
wm.completeWorkItem(wi.getId(), result);

//corresponding code from SRAMPClient
private final static String getAllServices =
"/s-ramp/ext/SwitchYardService";

SrampAtomApiClient client = new SrampAtomApiClient
(endpoint, username, password, true);

public List<Service> getServices() {
List<Service> serviceList =
new ArrayList<Service>();

for (ArtifactSummary sum :
client.buildQuery(getAllServices).query()) {
   serviceList.add(new Service(sum.getName(),
  sum.getUuid()));
}
return serviceList;
```

```
}
```

`SRAMPClient` is a class wrapping the `SrampAtomApiClient` which is the class provided by the Overlord S-RAMP. Method `getAllServices` fetches all the SwitchYard services stored in the S-RAMP repository and returns them back to the Service Lifecycle. In task "Select Service from S-RAMP" these Services are displayed to the user where he needs to pick one in order to bound Service Lifecycle instance to a specific Service.

This integration is effectively implementing requirement "search S-RAMP repository for SwitchYard Services" as mentioned in section 3.2.

5.3 Overlord RTGov integration with SLMA

Overlord RTGov is storing information regarding service execution environment. One of these information is invocation of a service. This can be accesses at following REST API exposed by RTGov Server:
`[GET] /activity/events/?from=$from&to=$to`

Retired Services are stored in the jBPM database, so when user tries to display Retired Service invocations, the application code will be sending `GET` requests to the endpoint above for all the retired services found. The `from` parameter will be set to the time when Service was marked as a retired. The `to` parameter will be set to the current date. If the `GET` request will return some result it will be displayed to the end user in a formatted table and available for further processing.

This integration is effectively implementing requirement "support executing notification actions once a deprecated service is used" as mentioned in section 3.2.

6 Project testing

In order to make sure that the resultant application will meet the quality criteria of the diploma thesis several tests has been preformed during the development. This does not concern only the final SLMA but also the Service Lifecycle processes we well.

Workflow tests are verifying that the Service Lifecycle have been modeled properly. Especially that the process execution is following the process model as expected and every process variable which is being passed is preserved correctly. These tests are located in `org.fi.muni.diploma.service.lifecycle.tests.ProcessEngine.ServiceTest`.

Service Lifecycle includes two Service Tasks which are communicating with remote S-RAMP repository - this integration is being verified by **S-RAMP integration tests** This test is making sure that the integration is successful by verifying the functionality on a sample data. Tests can be found in "handlers" project in class `com.sample.SRAMPTest`.

RTGov integration tests are important as successful integration with Overlord RTGov is essential since it is achieving one of the official requirement of the SLMA - monitoring retired Service invocations. Functionality which is being tested is that the invocation of a specific service is recognized by the RTGov server. This test is located in project "vaadin-frontend" in a class `org.fi.muni.diploma.service.lifecycle.tests.RTGovIntegrationTest`

Email tests are verifying that Service Lifecycle is capable of sending emails - for example when the Service is being retired, the user of the SLMA can inform the Service consumers about this. Another situation is when User is dealing with Service retired invocation - he can acknowledges it and send email to the respective authorities regarding this. This functionality has been tested using FakeSMTP [1] utility. The difference between FakeSMTP and "real" SMTP server is that FakeSMTP only delivers email in form of files created on a filesystem. The test for this functionality is located in project "vaadin-frontend" in class `org.fi.muni.diploma.service.lifecycle.tests.EmailServiceTest`.

1. https://nilhcem.github.io/FakeSMTP/

User acceptance tests were final tests performed. These tests were trying to determine that the resultant application really fulfilled all necessary requirements. This was the only test which was not automated at all. It was executed in form of a live session with my technical consultant. We have gone through the every screen of SLMA and tried to think of every possibility how can the application be utilized. In total approximately 15 User Interface issues were found and 2-3 functional issues too.

7 SLMA showcase

This chapter will show Service Lifecycle Management Application in action. Its main purpose is to show all use cases implemented by SLMA and accompany these use cases by screenshots so it will be clear what web page is implementing what exact functionality. This will all be done be executing "Service Lifecycle - for existing Service" process.

If reader is interested it is possible to try all of this in local environment since all necessary source codes as well as the installation instructions are published in the GitHub repository Giertli [2014]. After successfully installing the whole project it can be started by issuing a command `./start.sh` from the installation folder. The whole showcase will assume that the git repository is cloned locally on file system and installed according to the published installation instructions.

7.1 Uploading SwitchyYard Service into S-RAMP

"Service Lifecycle - for existing Service" process assumes that the Service artifact is already uploaded into the S-RAMP repository. So first we need to execute this task - in real scenario this would probably by done by the Service developer or similar role.

- Navigate to folder `./installation/jvm3-soa-server /jboss-eap-6.3/quickstarts/bean-service`
- Execute `mvn clean package` command from the terminal
- Log into `http://localhost:8480/s-ramp-ui/`
- Navigate to `Manage Artifacts - > Import Artifacts`
- Locate `./installation/jvm3-soa-server/jboss-eap-6.3/quickstarts/bean-service/target/switchyard-bean-service.jar`
- As Artifact Type set `SwitchYardApplication` and press `Import`

That is it! Example **OrderService** has been just uploaded to the S-RAMP Repository.

Note that value `SwitchYardApplication` for Artifact Type is required. This is the value which Service Lifecycle uses for fetching Service artifacts stored in S-RAMP repository.

7.2 Execute Service Lifecycle on demand

The first action when using the SLMA initially is to start new instance of the Service Lifecycle. The On Demand requirement is achieved by the fact that user decides when he wants to start the new instance. It is not done automatically as in Overlord DTGov.

- Log into `http://localhost:8280/service-lifecycle-mgmt/`. Sample credentials are **anton/password1!**. User **anton** has role **SOAGovernanceSpecialist** assigned which has all permissions set - this role can execute every use case in the application and complete every Lifecycle task.

- In left menu bar click on `Start New Lifecycle Instance`

- Click on `Start Service Lifecycle - for existing Service`

That is it! The new instance of the Service Lifecycle has been created. You should be redirected to the first Task of this Lifecycle as shown in the figure 7.1.

7.3 Work on the Service Lifecycle Task

In order to move from between stages of the Service Lifecycle user has to work on some tasks. Their complete documentation is in the Appendix A.

- Navigate to `Lifecycle Tasks`

- Since only one lifecycle is started, there is only one available. Click on `Work on Task` button

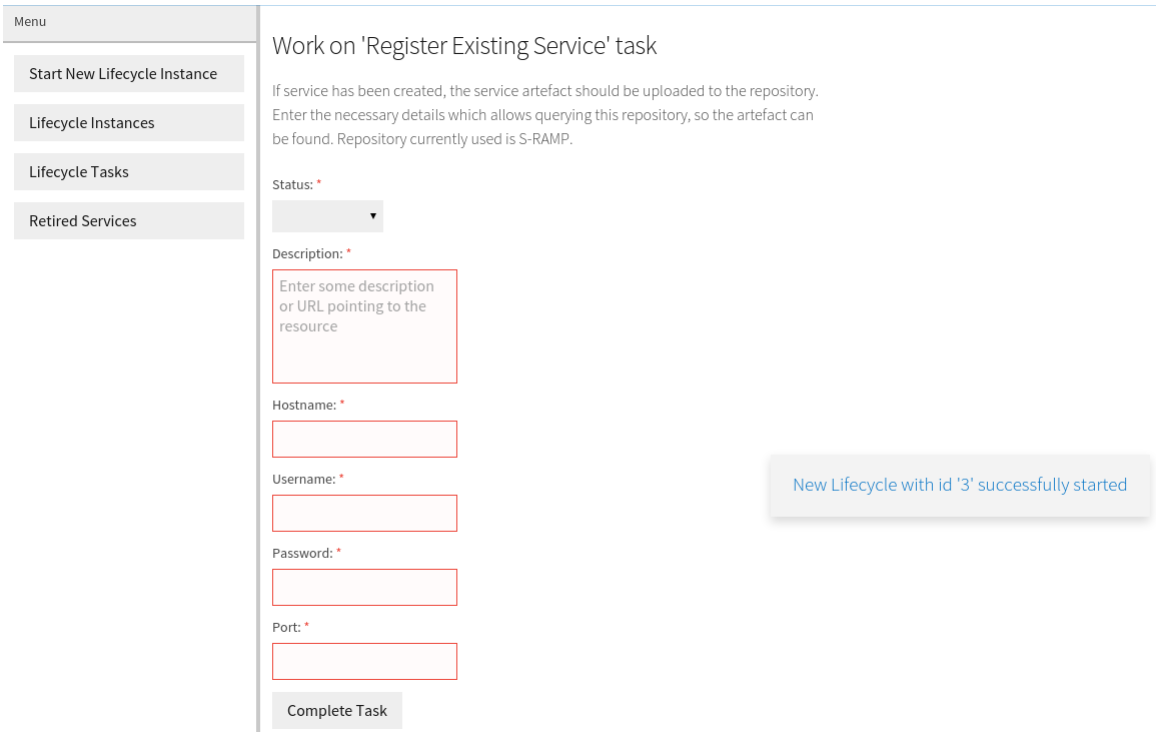

Figure 7.1: First Task of the Service Lifecycle as presented to the end user

- You will be presented with the same screen as shown in figure 7.1. Fill in appropriately, for example:

  - `Status - Deprecated`
  - `Description - SampleDescription`
  - `Hostname - localhost` (S-RAMP host)
  - `Username - admin` (S-RAMP username)
  - `Password - password1!` (S-RAMP password)
  - `Port - 8480` (S-RAMP Port)

- Press `Complete Task` button

That is it! First task have been completed and now the Service has moved to a different stage and the Service Lifecycle to a different

task. This is how the Service Lifecycle is being executed by the user. All tasks works similarly - User is presented with a form where he needs to input some values - there is always a description displayed, once done, he can press Complete button and he will be redirected to the Task List where he can pick another task to work on.

I will not be providing examples for every single Task as the principle is really the same. However there is one which is a bit more interesting as it directly implements one of the requirement - query S-RAMP repository for services.
If the "Register Existing Service" task has been finished, the next Task in this specific Service Lifecycle is called "Select Service from S-RAMP". Its corresponding form presented to the user is displayed in a figure 7.2. This form includes list of Services and user needs to select precisely one. In this case specifically only **OrderService** is being displayed. This is because this was the Service which we have uploaded in the S-RAMP Repository as explained in section 7.1.

This task specifically is implementing use case "search S-RAMP repository for SwitchYard Services" as listed in section 3.2. The data which are being presented to the user are obtained from a Service Task called "Query S-RAMP Services" - its implementation details has been explained in section 5.2.

Sections 7.2 and 7.3 are a practical example of use cases specified in section 3.2.2. These two use cases are together implementing requirement "execute custom lifecycle for those services on demand" as defined in section 3.2.

7.4 Monitor Service Lifecycle Stage

At any stage of the Service Lifecycle instance it is possible to display its detailed information including Service Lifecycle stage - in a graphical way as well as all the variables associated with the selected instance. It is possible to display these information for any Service Lifecycle Instances - Active, Aborted and even Completed.

- Open `http://localhost:8280/service-lifecycle-mgmt/`
- Navigate to `Lifecycle Instances`

- Select `Show details` on a desired Lifecycle instance and you will be redirected to a page where detailed information will be displayed

That is it! Details of selected Service Lifecycle instance are now displayed. The page includes list of Instance metadata, variables bound to this specific instances, and most importantly the Diagram which shows the current progress of the Service Lifecycle. Based on the active task in the Service Lifecycle the SLMA will dynamically generate new diagram. Example one is shown in the figure 7.3.

This section showed how the use case **View Task Details**, as defined in section 3.2.2, is implemented in the SLMA. It is an effective realization of the official thesis's assignment "monitor service lifecycle stage" as listed in section 3.2. The implementation detail behind this functionality consists of two parts - obtaining history related data of the Service Lifecycle instance which is possible due to successful integration with jBPM execution server as explained in section 5.1. The second part is of course the dynamic generation of the Service Lifecycle diagram.

7.5 Acknowledge Retired Service Invocation

This particular functionality can be fully utilized only under these two circumstances:

- There is a completed Service Lifecycle instance which ended up in making Service retired

- The Service has been invoked

Note that in order to finish the Service Lifecycle the user has to complete the last Task which is called "Retire Service". If user wants to this Task will send an email to the Service consumers regarding Service retirement. If this Email is supposed to be sent successfully the SMTP server has to run on port 1025.

This is the whole procedure how this functionality can be utilized:

- Start the SMTP server on port 1025: `java -jar fakeSMTP-1.11.jar`[1] `-s -b -p 1025 -a 127.0.0.1 -o /emailInbox`
- Navigate to `Lifecycle Tasks` and Complete the `Retire Service` Task by filling all necessary fields and pressing `Complete Task` button
- Invoke the retired Service by sending a Simple Object Access Protocol (SOAP) request to the respective endpoint:
  - Navigate to the cloned git repository and execute the following command: `./installation/jvm3-soa-server/jboss-eap-6.3/bin/jboss-cli.sh -connect controller=localhost:10399`
  - Deploy **OrderService** to the service container by executing: `/installation/jvm3-soa-server/jboss-eap-6.3/quickstarts/bean-service/target/switchyard-bean-service.jar`
  - Open `./installation/jvm3-soa-server/jboss-eap-6.3/quickstarts/bean-service/src/test/java/org/switchyard/quickstarts/bean/service/BeanClient.java` and on line 45 change the Port value so it points to the Service container. If SLMA was installed according to the README instructions in the Git repository, then the value is `8480`.
  - Navigate to the root of the `bean-service` folder and issue command which will execute `BeanClient` class: `mvn exec:java`
  - That is it! The **OrderService** was invoked by Sending an appropriate message SOAP message to the service container
- In SLMA, navigate to `Retired Services` menu and observe that the SLMA has registered this invocation.
- Acknowledge this Invocation by pressing `Send Notification` button.

1. https://nilhcem.github.io/FakeSMTP/

- The form presented to the user is displayed in figure 7.4. The "body" field is prefilled automatically with additional details regarding the Invocation.
- Finish the whole process by filling in all the necessary fields and pressing `Process Notification` button.
- You will be redirected to the List of remaining (unprocessed) invocations - the one just processed will not be displayed anymore.

That is it! You have just acknowledge retired Service invocation! This section showed how the Service Lifecycle can be completed. If it was completed successfully it really means that the Service bound to that particular Service Lifecycle instance was marked as "Retired" as that is the final stage of the Service Lifecycle. Once Service is marked as Retired the SLMA will watch out for every invocation of such service. This section explained how can the Service be invoked by sending a simple SOAP request to the Service Container and also how can the user process this retired Service invocation.

This example showed how the use case **Acknowledge retired service invocation**, as defined in section 3.2.2, was implemented by SLMA. This use case is effectively realizing the official diploma thesis's requirement "support executing notification actions once a deprecated service is used" as listed in the section 3.2. The implementation of this requirement was achieved by successful integration between Overlord RTGov and SLMA as explained in section 5.3.

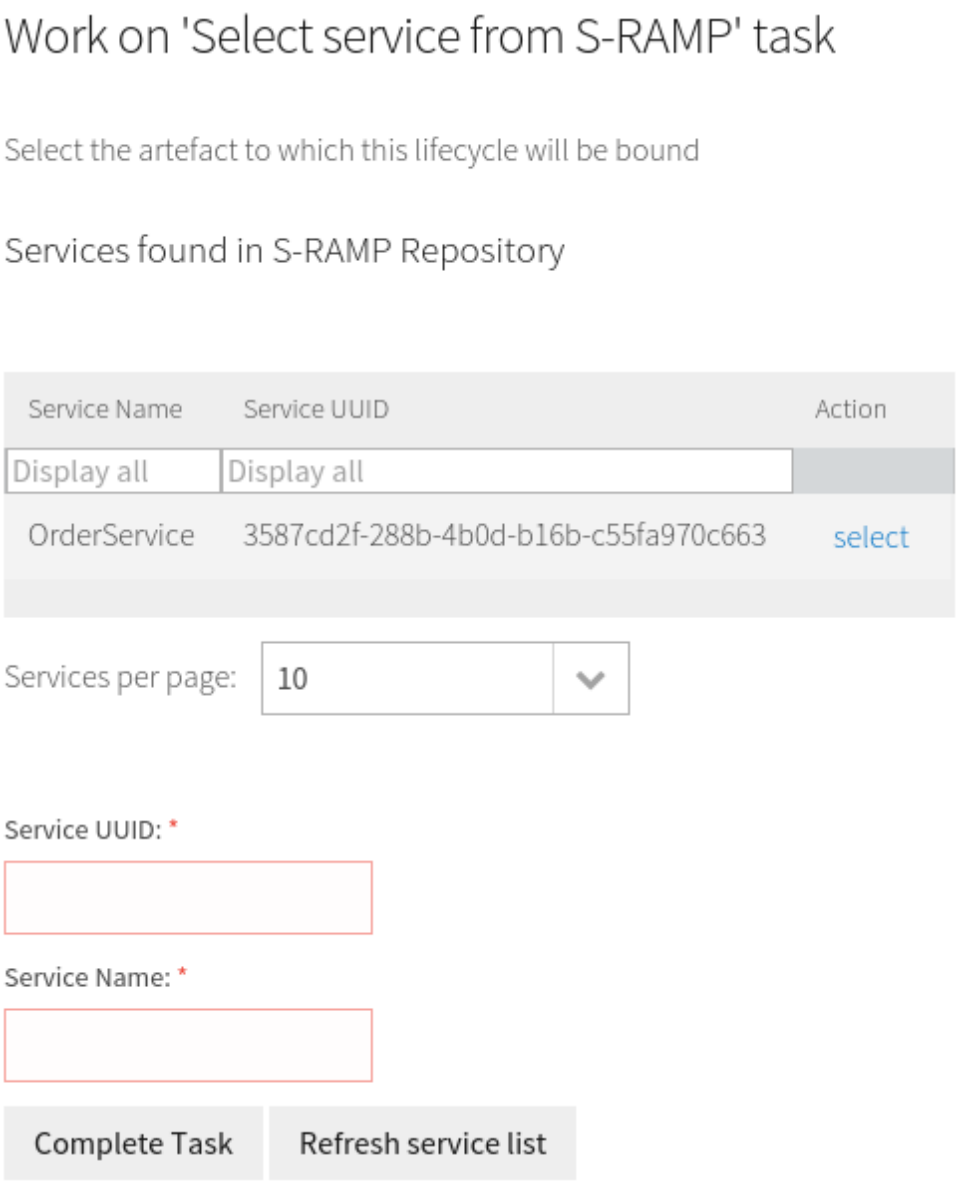

Figure 7.2: "Select Service From S-RAMP" Task Form

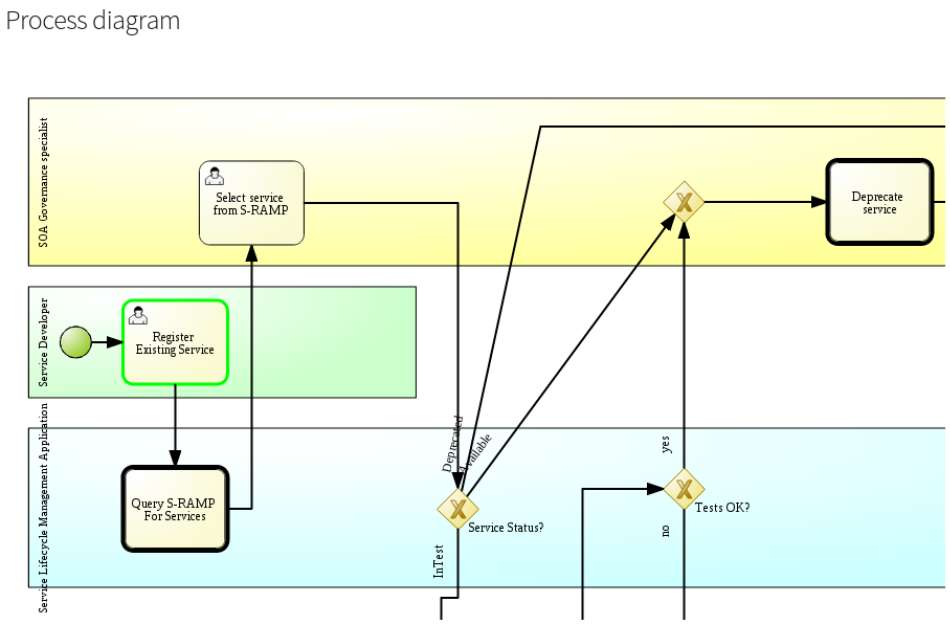

Figure 7.3: Diagram which shows the current stage of the Service Lifecycle

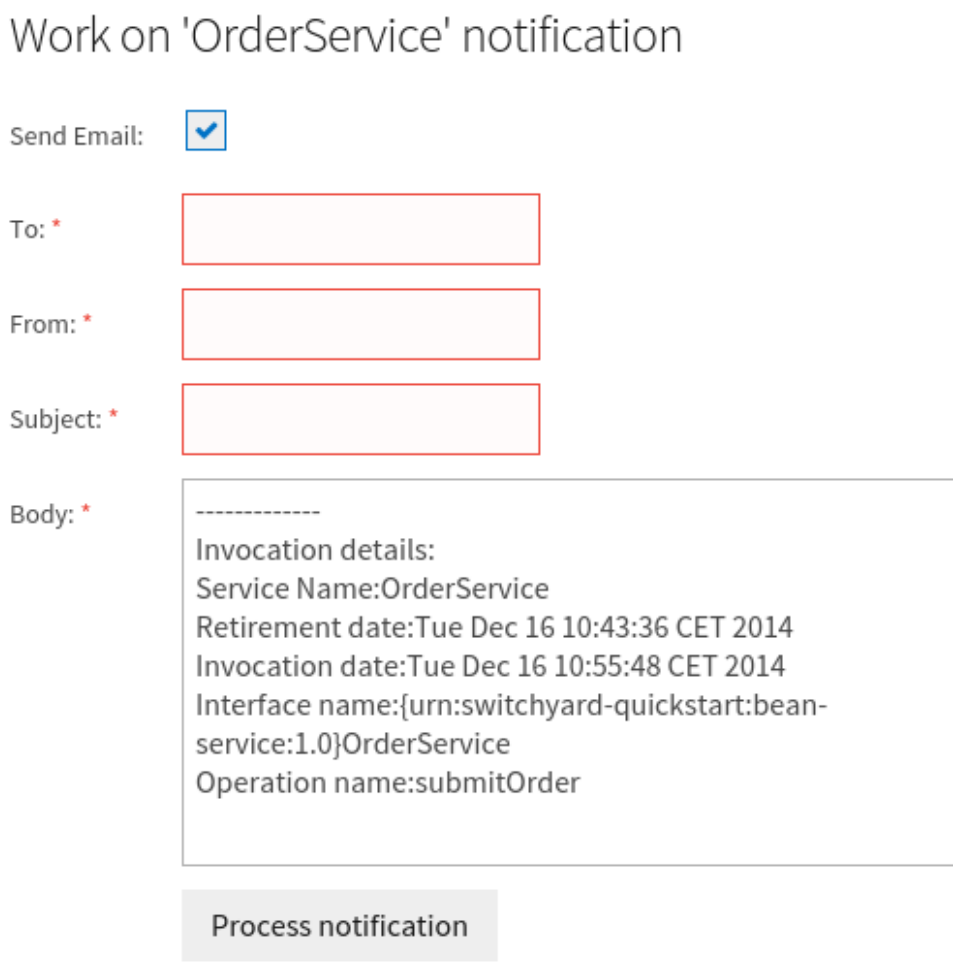

Figure 7.4: Web page allowing user to process retired Service invocation

8 Final word

8.1 Conclusion

The goal of this thesis was to implement Service Lifecycle Management Application based on agreed requirements. These requirements were derived from the definition of Service Lifecycle which is one of main concepts used in SOA Governance and was explained by Joe Dirksen Dirksen [2013, p. 213].

This goal has been met and the application source codes as well as the installation instructions have been published in the GitHub repository Giertli [2014].

The added value of this application is that it shows how can the rather complex discipline - SOA Governance - be implemented in a real application. Theoretical concepts have been propagated throughout the whole application.

Besides the fact that SLMA can be used as part of the SOA Governance solution it also shows how the numerous JBoss open source technologies can integrate between themselves through the REST API.

8.2 Future enhancements

The initial implementation of the SLMA could be enhanced in couple of areas. This section will discuss possibilities of these future enhancements.

- Strengthening the architecture by adding the Clustering support in order to achieve High Availability
- Service Lifecycle process enhancements - for example Service Lifecycle could automatically deploy Service to a Service Container and undeploy it once it is Retired.
- Add support for another type of the Lifecycle and Ontology - for example Policy Lifecycle. Initial design is showed in the attachment C.

SLMA is an open-source project published under Apache v2 license. All forks of the project, contributions and ideas for improvement are more than welcome.

Bibliography

Overlord s-ramp. [online]. [qtd. 2014-12-18]. Available at: <http://www.projectoverlord.io/s-ramp/>.

Switchyard. [online]. [qtd. 2014-12-18]. Available at: <http://switchyard.jboss.org/>.

Soa ontology v2.0. Technical report, OpenGroup, 2010. https://www2.opengroup.org/ogsys/catalog/C144.

Dtgov guide. Technical report, JBoss community, 2014. http://docs.jboss.org/overlord/dtgov/1.4.0.Final/html/index.html.

Jboss enterprise application platform 6.3 - administration and configuration guide. Technical report, Red Hat, 2014. https://access.redhat.com/documentation/en-US/JBoss_Enterprise_Application_Platform/6.3/html/Administration_and_Configuration_Guide/index.html.

jbpm documentation. Technical report, JBoss community, 2014. http://docs.jboss.org/jbpm/v6.1/userguide/.

M.N. De Maio, M. Salatino, and Esteban Aliverti. *jBPM6 Developer Guide*. Packt Publishing, 2014. ISBN 978-1-78328-661-4.

Jos Dirksen. *SOA Governance in Action: Rest and Web Service Architecture*. Manning Publications Co., 2013. ISBN 9781617290275.

Thomas Erl. *Service Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall, 1 edition, 2005. ISBN 9780131858589.

Anton Giertli. agiertli/service-lifecycle-mgmt. [online]. [qtd. 2014-12-18]. Available at: <https://github.com/agiertli/service-lifecycle-mgmt>.

Marko Grönroos. Book of vaadin. Technical report, Vaadin Ltd, 2014. https://vaadin.com/book.

Jiří Kolár. Business activity monitoring. Master's thesis, Masaryk University, 2009.

C.M. MacKenzie, K. Laskey, F. McCabe, P.F. Brown, R. Metz, and B.A. Hamilton. Reference model for service oriented architecture 1.0. Technical report, OASIS, 2006. http://docs.oasis-open.org/soa-rm/v1.0/.

K. Stam and E. Wittmann. Soa repository artifact model and protocol. Technical report, OASIS, 2013. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=s-ramp.

Mathias Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer, 2007. ISBN 978-3-540-73521-2.

A Service Lifecycle Tasks

This attachment includes documentation of all Tasks which are included in both Service Lifecycle processes.

Task name	**Task Description**	**Data which needs to be entered**
Identify Service	Every service has to have a purpose. Enter the reasoning and description of this service and any additional details which you feel are necessary.	• Service Description
Initialize	Every service implements some interface (contract). Make sure that this contract is well defined and documented. If these resources are exposed somewhere include URLs which points to them	• Contract • Documentation
Register	If service has been created the service artifact should be uploaded to the repository. Enter the necessary details which allows querying this repository so the artifact can be found. Repository currently used is S-RAMP. Note that service has to be created and policies fulfilled before moving onto the next stage of the lifecycle	• Are the service policies fulfilled? • Has the service been created? • Hostname • Username • Password • Port

Select Service from S-RAMP	Select the service artifact to which this lifecycle will be bound	• Artifact UUID in S-RAMP • Service Name
Test Service	Enter the Service Test results. If all went well service should be deployed in production with all tests passed.	• Is service deployed in production? • Have service passed the integration with other components? • Have service passed tests with service consumers?
Evaluate Tests result	Tests has not passed. It is possible to evaluate the result - re-test the service if needed and follow up on the test results through email - e.g. with QA lead.	• Send email? • Email Body • Email recipient • Email subject • Email sender • Re-test?

Deprecate Service	The best practice, when it comes to Service Deprecation is to inform service consumers about it and also update the service configuration so it reflect this change	• Inform consumers about service deprecation? • Email recipient • Email sender • Email subject • Service deprecation announcement
Retire Service	Policy regarding retired service is usually that it should not be used at all. Consumers should be informed about this.	• Inform consumers about service retirement? • Email recipient • Email sender • Email subject • Service retirement announcement

Register Existing Service	If service has been created, the service artifact should be uploaded to the repository. Enter the necessary details which allows querying this repository, so the artifact can be found. Repository currently used is S-RAMP	• Status – InTest – Available – Deprecated • Hostname • Port • Username • Password

B Wireframes

This attachment includes all the wiraframes which has been used during the design of the User Interface of the SLMA. During development these wireframes has been used as a template.

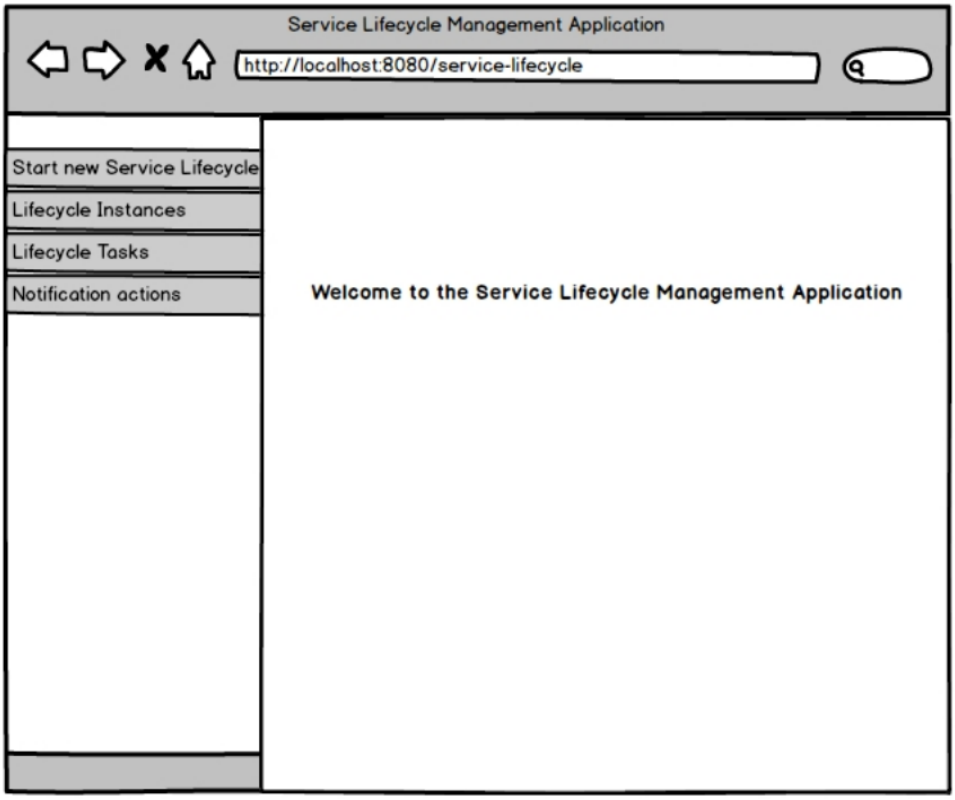

Figure B.1: Welcome page

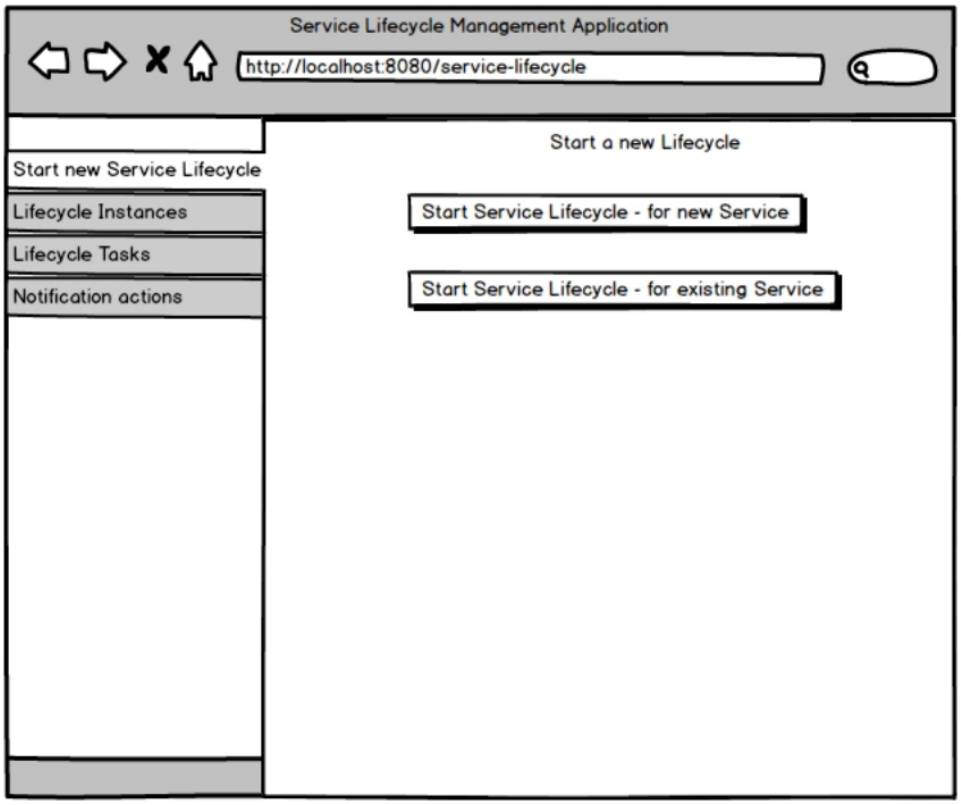

Figure B.2: Start the Lifecycle

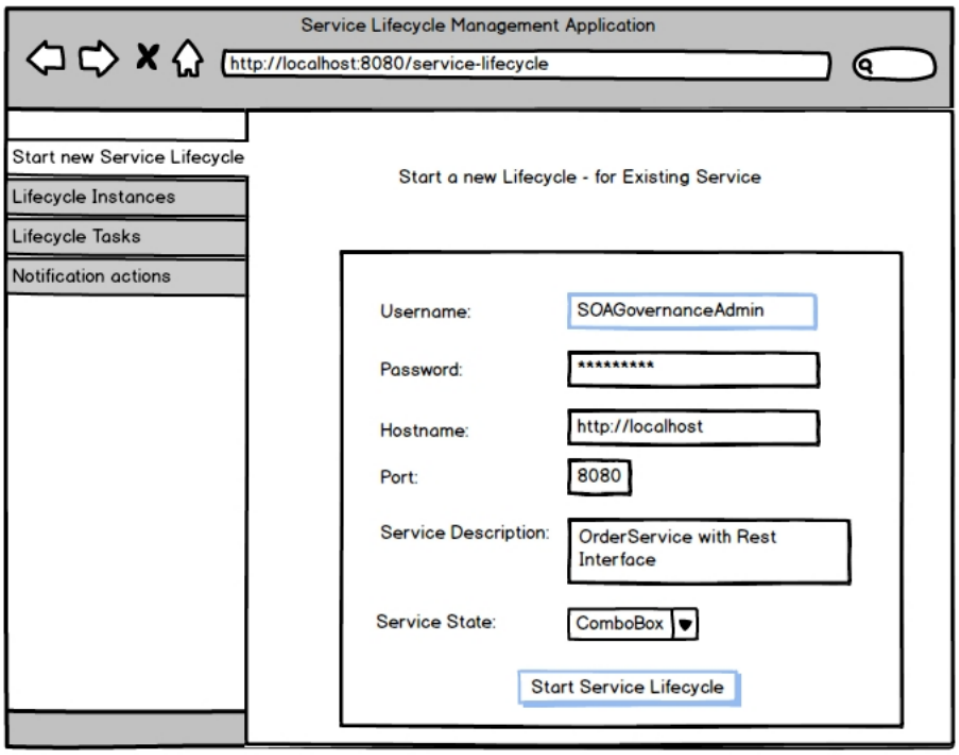

Figure B.3: First Task of the "Service Lifecycle - existing Service"

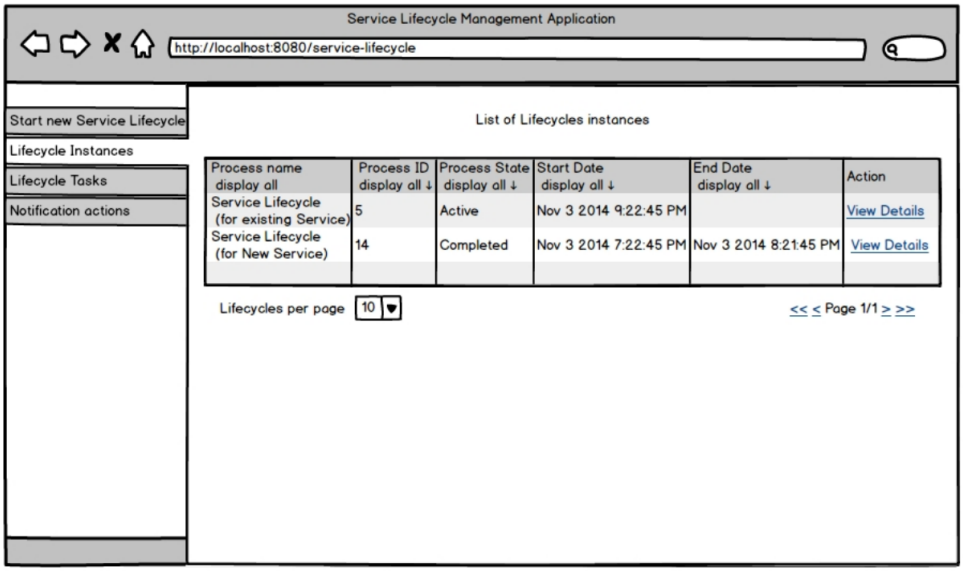

Figure B.4: List of Lifecycle instances

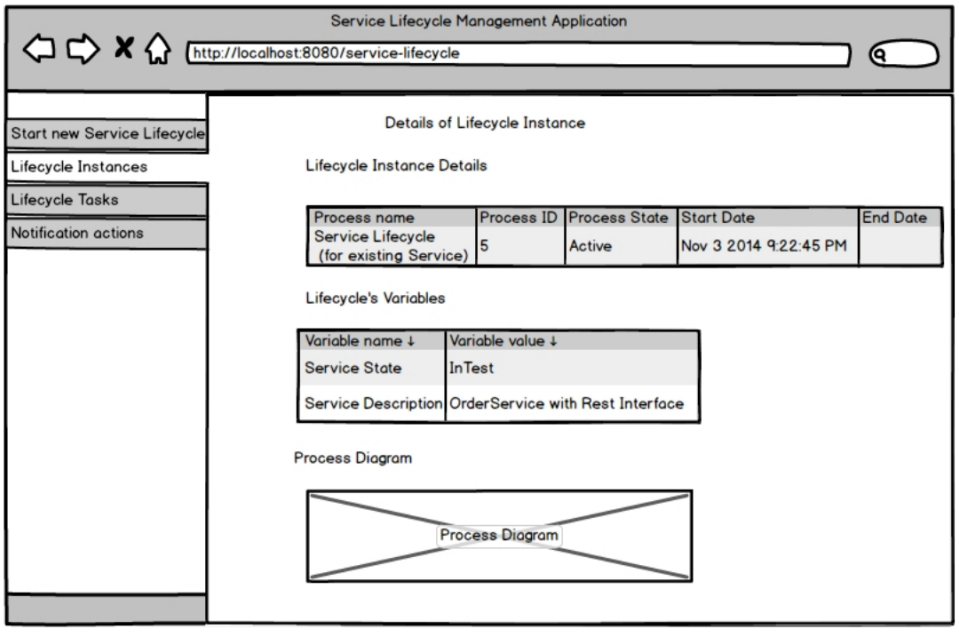

Figure B.5: Details of selected lifecycle instance

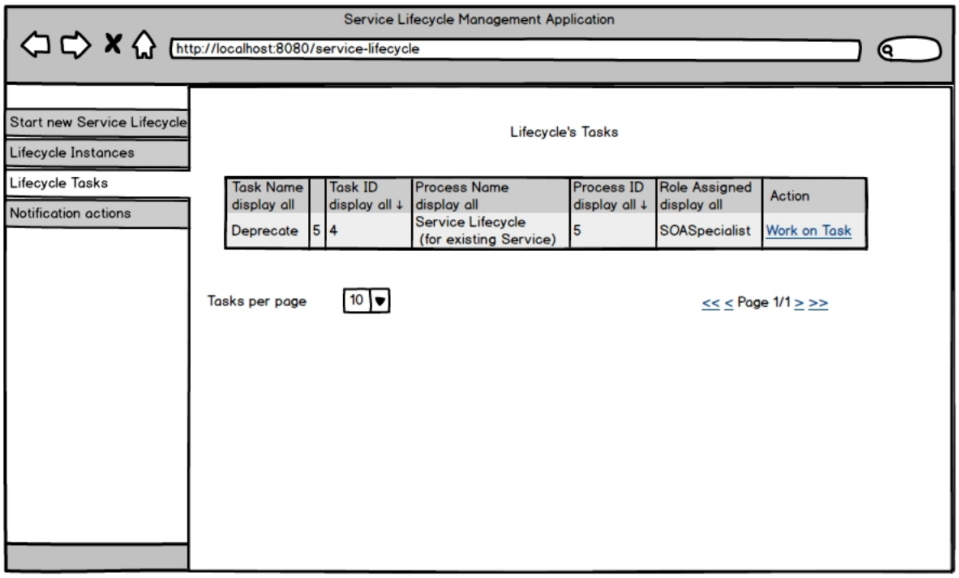

Figure B.6: List of available Tasks

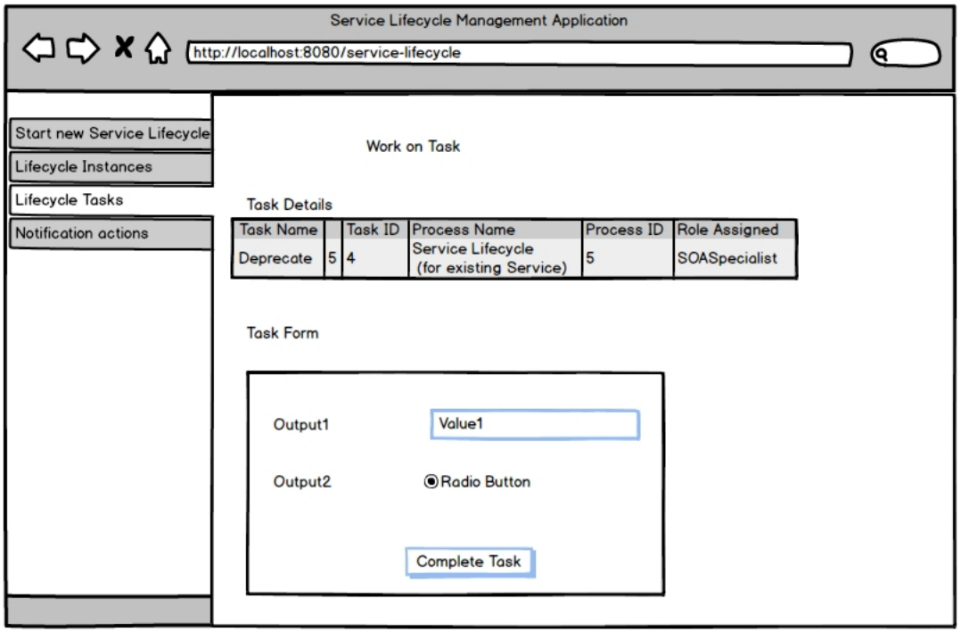

Figure B.7: Details of selected task

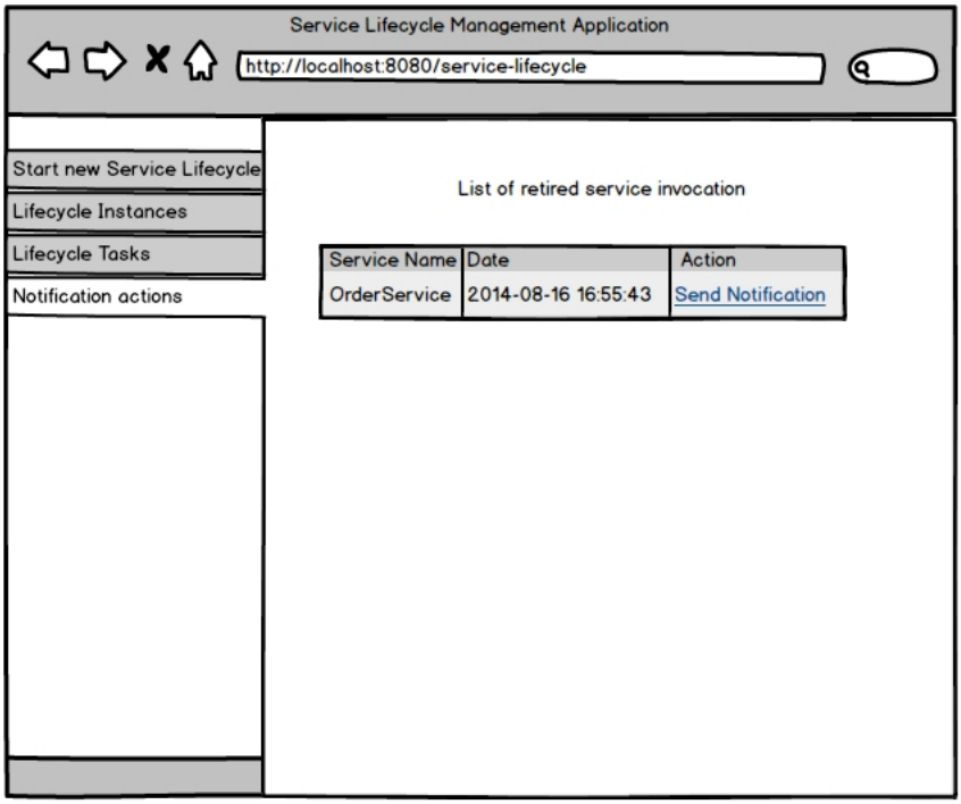

Figure B.8: List of retired service invocations

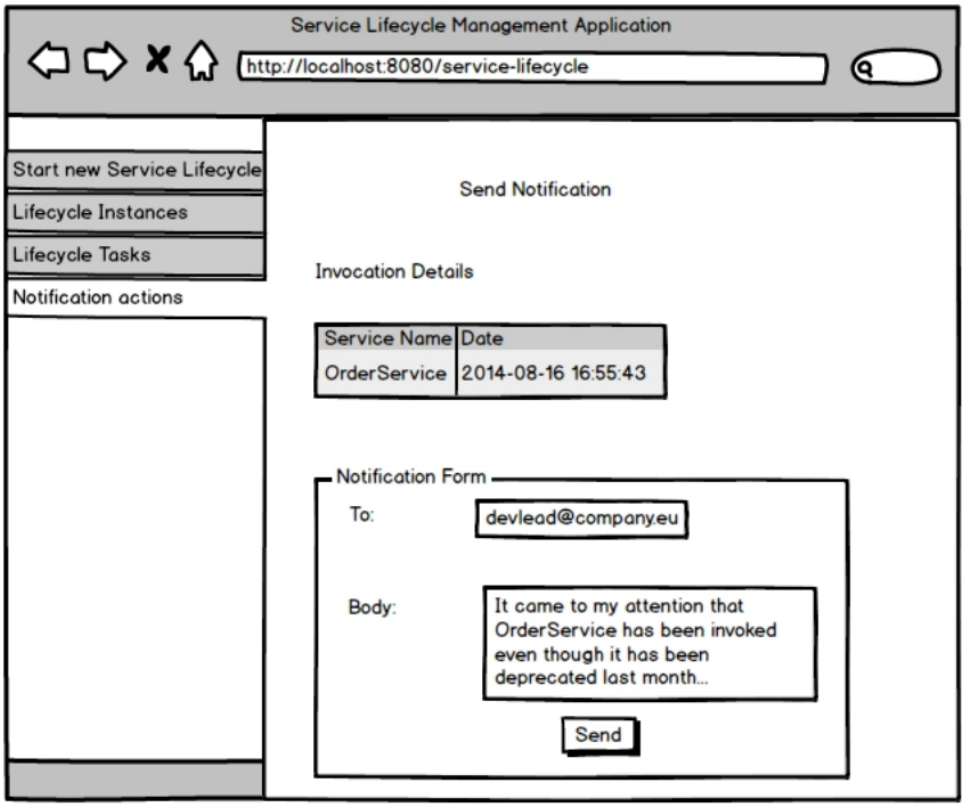

Figure B.9: Details of selected retired service invocation with possibility to acknowledge it

C Policy Lifecycle and Ontology

This attachment includes resources which could be used to implement support for Policy Lifecycle.

C.1 Ontology for Policy classification

Listing C.1: Ontology which could be used for Policy classification

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
xmlns:rdfs=
"http://www.w3.org/2000/01/rdf-schema#"
xmlns:rdf=
"http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:owl=
"http://www.w3.org/2002/07/owl#"
xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xml:base=
"http://www.jboss.org/overlord/service-
lifecycle.owl">

  <owl:Ontology rdf:ID="PolicyLifecycleStatus">
    <rdfs:label>
    Policy Lifecycle Status
    </rdfs:label>
    <rdfs:comment>
    Policy Lifecycle Status Ontology
    </rdfs:comment>
  </owl:Ontology>

  <owl:Class rdf:ID="PolicyLifecycle">
    <rdfs:label>
    policy lifecycle
    </rdfs:label>
```

```
  <rdfs:comment>
  Root status –
  participating in Policy Lifecycle Workflow.
  </rdfs:comment>
</owl:Class>

<!––
Basic set of possible policy lifecycle states
––>

<owl:Class rdf:ID="PolicyDesigned">
  <rdfs:subClassOf rdf:resource=
  "http://www.jboss.org/overlord/
  service–lifecycle.owl#ServiceLifecycle" />
  <rdfs:label>
  Policy Designed
  </rdfs:label>
  <rdfs:comment>
  Policy Designed
  </rdfs:comment>
</owl:Class>
<owl:Class rdf:ID="TrialPeriod">
  <rdfs:subClassOf rdf:resource=
  "http://www.jboss.org/overlord
  /service–lifecycle.owl#ServiceLifecycle" />
  <rdfs:label>
  Trial Period
  </rdfs:label>
  <rdfs:comment>
  Policy is in trial period,
  it has not been accepted yet
  </rdfs:comment>
</owl:Class>
<owl:Class rdf:ID="Active">
  <rdfs:subClassOf rdf:resource=
  "http://www.jboss.org/overlord/
  service–lifecycle.owl#ServiceLifecycle" />
```

```
  <rdfs:label>
  Active Policy
  </rdfs:label>
  <rdfs:comment>
  Policy is active and all services must conform
  </rdfs:comment>
</owl:Class>

  <owl:Class rdf:ID="Deprecated">
  <rdfs:subClassOf rdf:resource=
  "http://www.jboss.org/overlord/
  service-lifecycle.owl#ServiceLifecycle" />
  <rdfs:label>
  Deprecated Policy
  </rdfs:label>
  <rdfs:comment>
  Policy is deprecated and services does
  not need to conform to this policy anymore
  </rdfs:comment>
</owl:Class>

  <owl:Class rdf:ID="Failed">
  <rdfs:subClassOf rdf:resource=
  "http://www.jboss.org/overlord/
  service-lifecycle.owl#ServiceLifecycle" />
  <rdfs:label>
  Failed Policy
  </rdfs:label>
  <rdfs:comment>
  Policy did not pass the trial period,
  therefore it can not be marked as active
  </rdfs:comment>
</owl:Class>

    <owl:Class rdf:ID="Obsolete">
  <rdfs:subClassOf rdf:resource=
  "http://www.jboss.org/overlord/
```

```
    service-lifecycle.owl#ServiceLifecycle" />
    <rdfs:label>
    Obsolete Policy
    </rdfs:label>
    <rdfs:comment>
    After some period of deprecation, policy
    becomes obsolete. Services should no
    longer use it.
    </rdfs:comment>
  </owl:Class>


</rdf:RDF>
```

C.2 Policy Lifecycle Model

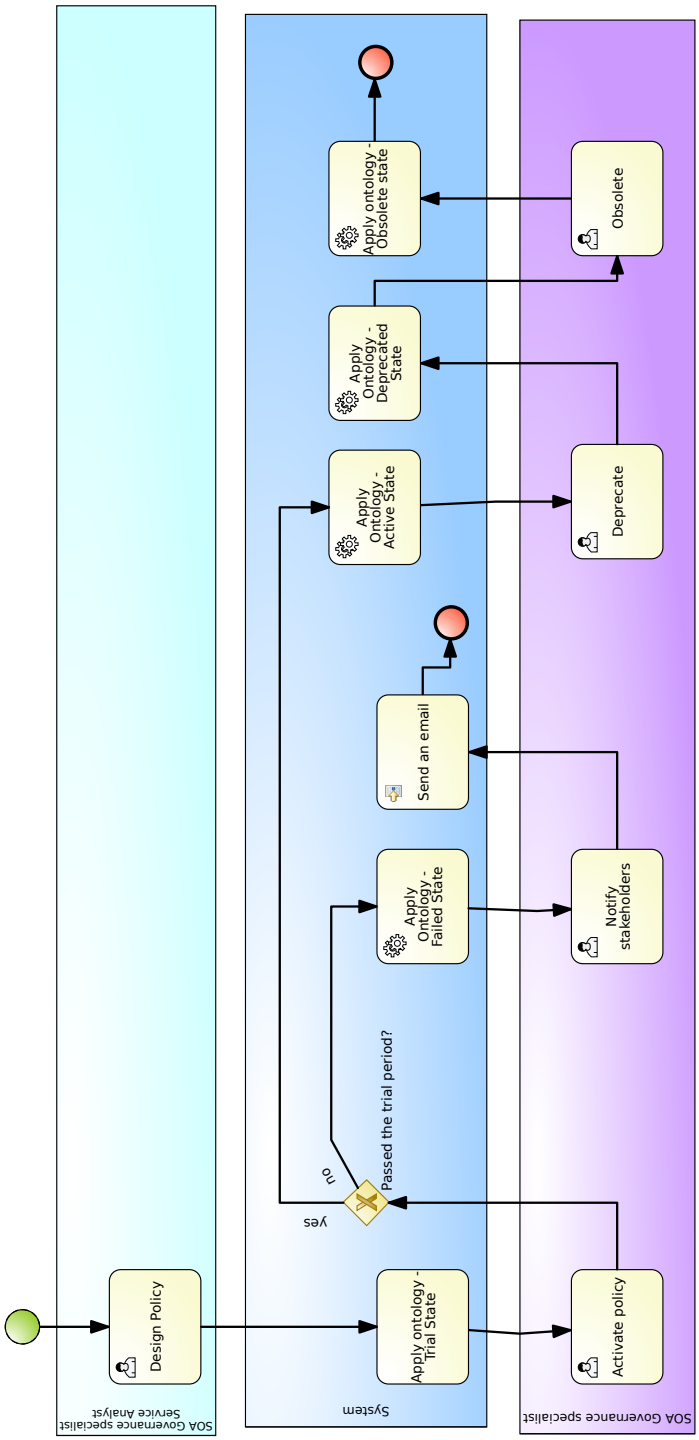

Figure C.1: Process model of Policy Lifecycle

D Content of the CD

The enclosed CD contains following items:

- `/git_repo` Clone of the git repository including source codes of the application and installation instructions
- `/support_software` All the supporting software which is used by the SLMA, such as JBoss open source technologies can be either downloaded from web, or taken from this directory
- `/text` Text of this diploma thesis