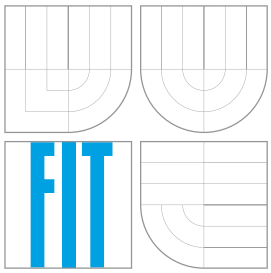
BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ



FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

BUSINESS PROCESS REPRESENTATION AS RESTFUL RESOURCES

REPREZENTACE BUSINESS PROCESŮ JAKO ZDROJŮ REST ARCHITEKTURY

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR
AUTOR PRÁCE
ALENA CHERNIKAVA

SUPERVISOR
VEDOUCÍ PRÁCE
RNDr. MAREK RYCHLÝ, Ph.D.

BRNO 2016

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2015/2016

Zadání diplomové práce

Řešitel: **Chernikava Alena, Mgr., Ph.D.**

Obor: Informační systémy

Téma: **Reprezentace business procesů jako zdrojů REST architektury**

Business Process Representation as RESTful Resources

Kategorie: Informační systémy

Pokyny:

1. Seznamte se s definicí business procesů v BPMN 2.0 a možnostmi automatizace procesních workflow (Workflow Management Systems, WfMSs). Seznamte se s architektonickým stylem REST a s reprezentací zdrojů v REST ve webových službách (RESTful API). Prozkoumejte možnosti zpřístupnění business procesů jako REST zdrojů.
2. Navrhněte způsob reprezentace business procesů a procesních workflow jako REST zdrojů přístupných pomocí RESTful API s podporou pokročilých funkcí, jako je skládání procesů a záměna jejich podprocesů za kompatibilní procesy, detekce aktuálně dostupných (spustitelných) procesů, instanciace spuštěných procesů z jejich abstraktních definic a sledování průběhu spuštěných procesů vč. požadavků na zdroje.
3. Po konzultaci s vedoucím implementujte navržené RESTful API pro vybraný BPMN stroj či WfMS (tj. BPMN/WfMS engine).
4. Výsledek otestujte a diskutujte jeho vlastnosti.

Literatura:

- *Documents Associated With Business Process Model And Notation (BPMN) Version 2.0*. OMG, 2011. [http://www.omg.org/spec/BPMN/2.0/]
- Georgakopoulos, D., Hornick, M., a Sheth, A. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and parallel Databases*, 3(2), 1995, pp. 119-153. ISSN 1573-7578. [http://www.workflowpatterns.com/documentation/documents/workflow95.pdf]
- Kumaran, S., Liu, R., Dhoolia, P., Heath, T., Nandi, P., Pinel, F. A RESTful Architecture for Service-Oriented Business Process Execution. In: 2008 IEEE 10th International Conference on e-Business Engineering, 2008, pp. 197-204. ISBN 978-0-7695-3395-7. [http://doi.ieeecomputersociety.org/10.1109/ICEBE.2008.82]
- Sepulveda, C., Alarcon, R., Bellido, J. QoS aware descriptions for RESTful service composition: security domain. World Wide Web, 18(4), 2015, pp. 767-794. ISSN 1573-1413. [http://link.springer.com/article/10.1007/s11280-014-0278-0]

Při obhajobě semestrální části projektu je požadováno:

- Body 1 a 2.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Rychlý Marek, RNDr., Ph.D.**, UIFS FIT VUT

Datum zadání: 1. listopadu 2015

Datum odevzdání: 25. května 2016

L.S.

---

doc. Dr. Ing. Dušan Kolář *vedoucí ústavu*

Brno University of Technology – Faculty of Information Technology

Department of Information Systems | Academic Year 2015/2016

Master's Thesis Assignment

Author: **Chernikava Alena, Mgr., Ph.D.**

Branch of study: Information Systems

Theme: **Business Process Representation as RESTful Resources**

Category: Information Systems

Instructions:

1. Familiarize with business process definitions in BPMN 2.0 and process workflow automatization possibilities (Workflow Management Systems, WfMSs). Familiarize with the REST architectural style and REST resource representation used in web services (RESTful API). Study possible alternatives how to make accessible business processes as REST resources.
2. Propose a way to represent a business process and process workflows as REST resources accessible through REST API with the support of advanced features such as process composition and substitution of sub-processes for compatible sub-processes, detection of currently available (executable) processes, and instantiation of executable processes from their abstract definitions, monitoring of process progress including resource requirements.
3. After a consultation with the supervisor, implement the designed RESTful API for a selected BPMN or WfMS engine.
4. Test the result and discuss its properties.

Literature:

- *Documents Associated With Business Process Model And Notation (BPMN) Version 2.0*. OMG, 2011. [http://www.omg.org/spec/BPMN/2.0/]
- Georgakopoulos, D., Hornick, M., a Sheth, A. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and parallel Databases*, 3(2), 1995, pp. 119-153. ISSN 1573-7578. [http://www.workflowpatterns.com/documentation/documents/workflow95.pdf]
- Kumaran, S., Liu, R., Dhoolia, P., Heath, T., Nandi, P., Pinel, F. A RESTful Architecture for Service-Oriented Business Process Execution. In: 2008 IEEE 10th International Conference on e-Business Engineering, 2008, pp. 197-204. ISBN 978-0-7695-3395-7. [http://doi.ieeecomputersociety.org/10.1109/ICEBE.2008.82]
- Sepulveda, C., Alarcon, R., Bellido, J. QoS aware descriptions for RESTful service composition: security domain. World Wide Web, 18(4), 2015, pp. 767-794. ISSN 1573-1413. [http://link.springer.com/article/10.1007/s11280-014-0278-0]

For the term project defense is required:

- Items 1 and 2.

Detailed obligatory instructions for master's thesis preparation you can find on the address http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Supervisor: **Rychlý Marek, RNDr., Ph.D.**, UIFS FIT VUT

Assignment date: 1 November 2015

Submission date: 25 May 2016

L.S.

doc. Dr. Ing. Dušan Kolář head of the department

Abstract

Almost every company in the world deals with business processes on a daily basis. And business can derive significant benefit from taking a formal approach. This means, that the business process is formally described (for example using Business Process Modeling Notation 2.0) and implemented in some Business Process Engine (BPE). The aim of the thesis is to design a general API (BP Orchestration Web Services) that does not depend on BPE for business process monitoring and manipulation. The main problem of current APIs is that they are not unified, do not provide enough flexibility and are too tied to the one particular BPE. This thesis includes general information about workflow, about BPMN and basic principles of REST architectural style. Based on this knowledge problesms were formally stated and as a solution BP Orchestration Web Services were designed and implemented. Web Services allow to instantiate a process from abstract definitions, monitor the state of the process and manipulate with the process (including advanced manipulations such as exchanging a sub-process in the running instance for some another compatible sub-process). The RESTful API was designed in a way to minimize the client implementation and restrict client's knowledge about internal details. As part of the thesis a connector to BonitaBPM was implemented and integration with BonitaBPM was done.

Abstrakt

Cílem diplomové práce bylo navrhnout a implementovat obecné nezávislé na workflow enginu API (BP Orchestration Web Services) pro monitorování a manipulaci s podnikovými procesy. Hlavní problém již existujících řešení je, že nejsou unifikované, neposkytují dostatek flexibility a jsou příliš vázané na používaný workflow engin. Tato diplomová práce obsahuje obecné informace o workflow, BPMN a REST architektonickém stylu. Na základě těchto znalostí byla popsána formální definice problému a byly navrhnuty RESTful webové služby Webové služby umožňují instantizace spustitelných procesů z jejich abstraktních definic, sledování průběhu spuštěných procesů včetně stavu jejich aktivit a manipulace s procesem (včetně pokročilých manipulací, takových jako záměna jejich podprocesů za jiné procesy). RESTful API bylo navrženo tak, aby implementace klienta byla co nejmenší a aby klient potřeboval co nejméně informací o vnitřní implementaci podnikového procesu. Jako část diplomové práce byl implementovan konnektor pro BoniraBPM v jazyce Java a byla udělana integrace pro BonitaBPM.

Keywords

REST, workflow, BPMN, API, design, Web Services, BPM

Klíčová slova

REST, workflow, BPMN, API, návrh, webové služby, BPM

Reference

CHERNIKAVA, Alena. *Business Process Representation as RESTful Resources.* Brno, 2016. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Rychlý Marek.

Business Process Representation as RESTful Resources

Declaration

Hereby I declare that this Master's thesis was prepared as an original author's work under the supervision of Mr. RNDr. Marek Rychlý, Ph.D. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

.....
Alena Chernikava
May 24, 2016

Acknowledgements

I would like to thank my Master's thesis superviser RNDr. Marek Rychlý, Ph.D for leading my thesis and my family for supporting and encouraging me.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

In this thesis, we provide a general description and implementation of RESTful API for *business process engines* (BPE; also known as workflow management systems), software frameworks for execution and maintenance of process workflows. Contrary to state-of-the-art approaches, the approach presented here allows to change business processes on the fly (to enable dynamic business processes executions), to execute non-linear and context-aware processes (the processes that permit different executions based on their context, such as currently available resources), and to easily switch and integrate several underlying BPE platforms to efficiently execute and maintain complex business processes decoupled on several components in distributed environments.

Each company in the world wants to overcome its competitors and one of the ways for an efficiency improvement is to optimise company's business processes. A *business process* (BP) is a collection of related, structured activities that produce a specific service or product for customers. The business can derive significant benefits from taking a formal approach. This approach consists in a formal description of the BP using well-known common tools (e.g. BPMN 2.0 [8]) and its implementation using some BPE (e.g. BonitaBPM).

Every BPE is a comprehensive application incapsulating a big variety of different tools and even though there are some limitations. For example there is no easy way to change some "part„ of the business process on the fly. That is why all business processes should be completely described and all possible required variations should be included. Even though we cannot foresee all possible variations, and some flexibility would be very useful. Usually factories are so tied to the currently used BPE, that it is too expensive to adopt new technologies or to change the current BPE for another one. There is a need to have one more abstraction level that should hide some details and provide an opportunity to change the BPE at minimal cost.

What can inspire people more than a success of some idea or its implementation? For example we can say, that the World Wide Web fulfils the REST principles [4]. Everybody knows, that WWW is a highly scalable, flexible, distributed computing platform. And its great success drives efforts of applying REST principles in Business Process Modelling (BPM). One important research paper about REST-style workflows is [10] where every activity is represented by a resource (activity-centric modelling paradigm). However, the main problem of direct applying of REST principles in the Business Process Management is an unmanageable explosion of number of resources in a process. Another approach but with information-centric modelling paradigm was taken in [13]. Authors managed to eliminate the problem of large number of resources, but introduced another not obvious one. There is an inconsistency between a visual representation of the business process (it is still activity-

centric BPMN) and the way of controlling the business process (information-centric). We are going to take completely different approach. As a resource we take a business process itself with its variations that allows users to use their favourite tool for modelling.

A delay in the business process can cause big financial losses, so it is critically important to be able to adjust the running business process to the current conditions. Some attempts in [15],[1] where done, but they both require big changes and do not allow to "reuse„ already modelled business processes. In order to fulfil the business needs we provide a description of a general RESTful API that allows to adopt the running business process on the fly to the current situation without an expensive total redesign of the business process, provides one more abstraction layer that make possible to change currently used BPE for another one, allows to model a processes with variable components.

The need to model and to implement a business process leads to the notion of *workflow* (the first workflow engines were designed in 1970s). Therefore in the second chapter a workflow and a workflow management systems are introduced.

A wide variety of companies exists throughout the world and they are made up of people with different education and experiences. Moreover, all of them need to communicate to each other in order to describe, discuss and implement business processes. Therefore, to make this process easier people should talk the same language. That is why Object Management Group has developed a standard called Business Process Model and Notation. Therefore, the third chapter describes basic ideas of BPMN and its basic elements.

As we want to apply REST principles in our design, first of all we need to familiarise with REST architectural style that was introduced by Roy Thomas Fielding in 2000 at his doctoral dissertation. It is simply a set of principles that help to develop easy and lightweight APIs and can be applied to different areas. The forth chapter describes all principles introduced by REST architectural style and advantages they can bring to the system.

The fifth chapter is all about the design of the solution and the sixth chapter describes the main points in the implementation.

As BonitaBPM was selected as target BPE the last chapter describes how business process should be modelled in the real environment in order to use designed Web Services.

The main contribution of our approach is the ability to implement dynamic business processes executions and, in general, to improve scalability and reliability of the business process engines maintaining the executions or their parts in distributed environments.

Chapter 2

Workflow

2.1 Why do companies need workflows

Nowadays almost every company deals with business processes on a daily basis. It depends on the company on how formal these processes should be. However, business can derive significant benefit from taking a formal workflow approach. A formal workflow model breaks each process into activities; providing ways to see how well activities are designed and performed. The formalisation of the business process is a way to enforce internal control and allows applying the same approach across the company, which leads to

- cost reduction of the business process implementation,
- visibility and understanding of the business processes to every employee,
- easy way to optimise the business process,
- easy way to bring newcomers up-to-date.

However, business faces some other challenges. Business has to

- have a flexibility in business process reengineering,
- rapidly develop new services and products,
- reduce a response time as much as possible,
- automate routine tasks,
- deal with a lot of different systems (a need of integration),
- be reliable,
- resist failures and recover fast,
- optimise scheduling,
- use transaction concepts,
- adapt to new market conditions,
- integrate new technologies into business processes.

To address these needs enterprises must constantly reconsider and optimise the way they do their business.

The main goal of the formal process definition is to separate activities into well-defined simple tasks, roles, rules, and procedures. This concept can be easily applied in different areas. You can find the analogy with the UNIX philosophy. McIlroy, the inventor of the UNIX pipe, summarised the UNIX philosophy as follows:

"Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface „ [14]

2.2 Process classification in the companies

We can separate out three different types of the processes in the companies [11]:

Material process is a set of human tasks related to physical world (such as moving, storing, measuring etc.) designed to deliver physical products.

Information process is a set of automated tasks (performed by programs) and partially automated tasks (performed by human by means of computer) designed to manipulate (create, process, provide, update, transform) with information.

Business process represents a higher-level abstraction. A business process treats activities as market-centred activities and it can be implemented as information process and/or material process.

2.3 Business process reengineering

One of the basic needs is that a company should follow world trends, adapt to new technologies and use them. It implies that business processes are very dynamic and business process reengineering is one of the most important activities in the company.

Among reasons for business process reengineering are [7]

- increasing customer satisfaction,
- improving efficiency of business operations,
- increasing quality of products,
- reducing cost,
- meeting new business challenges and opportunities.

2.4 Workflow management (WFM)

The first automation prototypes of workflow management systems were developed in the late 1970s. Moreover, since then a large variety have been developed. Many of the systems found their applications in coordination with document circulation and in enterprise applications.

Overall, a workflow represents a general approach for specification, automation and reengineering of business processes. We can look at the workflow from different perspectives [7]:

Conceptual level allows easy understanding, following and reengineering the business process

Requirement level is more detailed level of understanding, where requirements for functionality and human skills are described.

A *workflow management* (WFM) is a technology that includes [7]:

- A workflow definition describes a process as a set of tasks. It includes information about each task needed to control and coordinate the execution of the process, skills of individuals and resources required to perform each task).
- An easy way to reengineer and reimplement the processes as fast as possible when business needs change or current used technologies require an update.

A *workflow* is defined in [7] as a collection of tasks organised to accomplish some business process. One or more software systems, an individual or a team of people, or a combination of these can perform a task. Human tasks include interacting with computers closely (e.g., providing input commands) or loosely (e.g., using computers only to indicate task progress). In addition to a collection of tasks, a workflow defines:

- an order of task invocation,
- conditions under which tasks must be performed,
- a task synchronisation,
- an information flow (dataflow).

2.5 Workflow management system (WFMS)

A *workflow management system* (WFMS) is a system that provides the ability to specify, report on, execute and dynamically control workflows involving humans and automated systems.

A workflow technology was developed to support business needs by providing methodology and software to support [7]:

- A business process modelling (a formal business processes description in a form of the workflow specifications).
- A business process reengineering (business process optimisation and improvement).
- A workflow automation (an ability to easily generate workflow implementations from the workflow specifications).

The basic need that workflow management systems address is the need of coordination between all entities and actors that should be involved in the execution of the business process. From the coordination point of view, a workflow management system is intended to solve the following problems [3]:

- Data dependencies between activities which are managed through the control and data flows.
- Shared resources (one resource can only be used by one component at a time), which are managed through scheduling and staff resolution mechanisms.

2.6 Workflow classification

Main characteristics of the workflows are:

- Repetitiveness and predictability of workflows and tasks.
- How the workflow is initiated and controlled (human-controlled or automated).
- Requirements for WFMS functionality.

According to [9]] there are three categories of workflows:

Administrative workflows Administrative workflows typically involve repetitive, predictable tasks with simple coordination rules. The ordering and coordination of tasks in administrative workflows are static and can be automated. Such WFMS can handle simple information routing and document approval functions. Administrative workflows do not encompass a complex information process and do not require access to multiple information systems used for supporting production and customer services. Administrative WFMS are generally non-mission critical. [7]

Ad-hoc workflows Ad-hoc workflows involve a human coordination, collaboration, and co-decision that often appear in office processes. There is no pattern for passing information among people. The ordering and coordination of tasks in an ad-hoc workflow are not automated but instead are controlled by humans. Furthermore, the task ordering and coordination decisions are made while the workflow is performed. Such WFMS provides functionality for facilitating a human coordination, collaboration and co-decision. A functionality for the task ordering control typically is not provided. Users of an ad-hoc workflow need to access the WFMS to determine if work was completed. Ad-hoc WFMSs are not mission critical, i.e., periodic failures of such workflows do not significantly interfere with the overall business process. [7]

Production workflows Production workflows involve repetitive and predictable complex business processes. Production workflows typically encompass a complex information process involving an access to multiple information systems. The ordering and coordination of tasks in such workflows can be automated. However, automation is complicated because an information process is complex and the system should make a decision by itself how to proceed based on the data retrieved from other multiple information systems. Such WFMS provides facilities to define task dependencies and control a task execution with little or no human intervention. Production WFMSs are often mission critical and they must deal with the integration and interoperability of other information systems. [7]

There are other classifications. For example according to [2] workflows are divided into:

- Ad-hoc workgroup support.
- Task automation.
- Document flow.
- Process automation.

However [6] classifies workflows as:

- Mail-centric.
- Document-centric.
- Process-centric.

2.7 WFMS classification

Workflow systems can be categorised in the following categories based on their functionalities [7]:

- System-oriented workflow systems
- Human-oriented workflow systems

The detailed comparison of these two categorie you can find in the Table 2.1.

Table 2.1: Human-oriented workflow vs System-oriented workflow

Question	Human-oriented workflow	System-oriented workflow
Who is supported by the system?	Supports humans collaborating in performing tasks and coordinating tasks. Humans provide the data to the process.	Involves computer systems (and machines). In addition to being highly automated, system-oriented workflows access other information systems to retrieve data.
Who makes decisions?	Humans make decisions.	System makes decisions itself based on the data received from other information systems. A human decision-making is almost eliminated.
What is the goal?	Often control and coordinate human tasks.	Often control and coordinate software tasks (typically with little or no human intervention). Consequently, system-oriented workflow implementations must include a software for various concurrency control and recovery techniques to ensure consistency and reliability.

Continued on next page

Table 2.1 – continued from previous page

Question	Human-oriented workflow	System-oriented workflow
Who is responsible for the consistency of data?	Humans must ensure the consistency of documents and workflow results, as only humans have deep understanding of the semantics of the process. The WFMS is here only to assist people and it cannot be made responsible for maintaining data consistency, since it has no information about semantics.	A system has knowledge of an information semantic. Hence the WFMS can be given (and must be given) more responsibility for maintaining the information consistency.

2.8 Workflow components

Graphically a workflow can be represented in a natural way in a form of diagrams where a direct flow between processing components is shown (e.g. flowchart, mapping maps). A single workflow component has the following parameters:

- Set of inputs (the information, material and energy required to complete the step).
- Transformation rules or algorithm (that can be performed by human or machine).
- Set of outputs (the information, material and energy produced by the component).

These three parameters are sufficient for the local components, but for non-local (for example web service is a non-local network component) we need to specify some additional information.

A simple workflow may involve a linear path of tasks to be followed; a complex workflow may involve a graph-like organisation of tasks where some tasks may be executed in parallel or multiple tasks must complete before others can start. The complexity can be determined by the kinds of coordination/collaboration rules or constraints applied to the task execution. [7]

A workflow documentation and a business process modelling are very important aspects of the business process management.

Chapter 3

BPMN

As we want to design a solution for business process orchestration we need to be able to model a business process with common technics and tools.

3.1 Overview

A Business Process Modelling (BPM) is used to share a great deal of information with a wide variety of audiences. A business process itself involves multiple participants and coordination can turn into a complex activity. Without a standard modelling technique, it is impossible to communicate a rich set of information in the same way to different people. To address this need the Object Management Group (OMG) has developed a standard called Business Process Model and Notation (BPMN).

The main goal of BPMN is to provide an intuitive and simple way to describe, model, implement and use various business processes. We can treat BPMN as a common language for all users (business analysts, technical developers and managers). It is important that BPMN provides a visualisation of business oriented notions through the graphical elements. Such visual representation helps the viewer easily differentiate between different aspects of the process. Furthermore, the graphical notation will make for the deeper understanding of the performance collaborations and business transactions. Users will benefit from it in a similar manner as they benefit from the UML standard in software engineering.

In other words, BPMN is a standard set of diagramming conventions for describing business processes. It is intended to register enough details in order to be the source of an executable process description.

3.2 BPMN users

At high level, business people represent a BPMN user. They should be able to easily read and understand a diagram. At lower level, a process executor represents a BPMN user. They should be able to read the diagram and add further details to it in order to implement physically the process. BPMN targets all people who need to communicate business processes in a standard manner.

3.3 UML vs BPMN

First, it is important to understand what UML and BPMN have in common. Both of them are standards and they pursue a common goal to share a big amount of various information between completely different people. However, the main difference is in approach the taken. BPMN has a process-oriented approach while UML has an object-oriented approach.

3.4 BPEL

Business Process Execution Language (BPEL) is an XML-based language for describing a business process in which most of the tasks represent interactions between the process and external Web services. The BPEL process itself is represented as a Web service, and is realised by a BPEL engine that executes the process description.

Since BPEL is currently considered the most important standard for execution languages, a translation from BPMN to BPEL is specified in the BPMN standard.

3.5 BPMN elements

It is very important to design a common simple mechanism for creating business process models. In BPMN 2.0 a business process is defined as a graph consisting of objects somehow linked together. The approach is to organise the graphical aspects of the notation into specific categories. A small set of notation categories allow the user to easily recognise the basic types of elements and understand the diagram. The basic categories of elements according [8] are

Flow Objects : Events, Activities, Gateways

Data : Data Objects, Data Inputs, Data Outputs, Data Stores

Connecting Objects : Sequence Flows, Message Flows, Associations, Data Associations

Swim lanes : Pools, Lanes

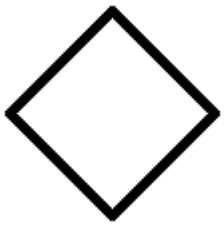
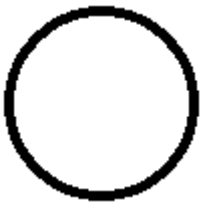
Artefacts : Group, Text Annotation, User defined artefacts

The main graphical elements describing the behaviour of a business process are the flow objects. They are connected to each other or other elements with the help of connecting objects. Swim lanes were designed for grouping the primary modelling elements to make them visible; that they are a part of some larger abstraction. In addition, you can add additional information about the process through artefacts.

Table 3.1: Basic BPMN elements

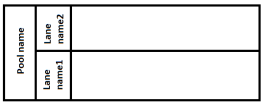
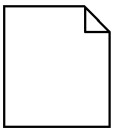
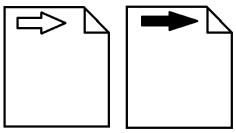
Basic Element	Description	Graphical Notation
		Continued on next page

Table 3.1 – continued from previous page

Basic Element	Description	Graphical Notation
Gateway	A Gateway is used to control the flow. It represents branching, forking, merging, and joining of paths depending on the condition.	
Event	An Event is something that happens. It affects the flow of the process and usually has a cause or an impact. We can catch and throw events. There are three common event types: Start, Intermediate, End.	
Activity	An Activity is a generic term for work that should be done. There are two types: **Sub-Process** represents a compound work. The sub-process is used to hide additional levels of business process details. **Task** represents an atomic work. The task cannot be broken down into a further level of business process details.	
Sequence Flow	A Sequence Flow shows the order that activities will be performed.	
Message Flow	A Message Flow shows the flow of messages between two participants that are prepared to send and receive them. Each participant is represented by its own pool.	

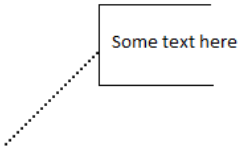
Continued on next page

Table 3.1 – continued from previous page

Basic Element	Description	Graphical Notation
Association	An Association is used to link information and artefacts with other elements.	
Pool	A Pool is the graphical representation of a participant in collaboration. It acts as a graphical container for partitioning a set of activities. A pool may have internal details or may have no internal details (black box).	
Lane	A Lane is a sub-partition within a process or pool. Lanes are used to organise and categorise activities.	
Data Object	A Data Object provides additional information for the activities: what is required by the activity and what is produced by the activity.	
Data Input/Data Output	A Data Input and A Data Output provide the same information as data object but for processes.	
Message	A Message encloses the content of a communication between two participants.	
Group	A Group does not affect anything in the process. It helps graphically highlight that elements form a group.	

Continued on next page

Table 3.1 – continued from previous page

Basic Element	Description	Graphical Notation
Text Annotation	Through Text Annotations it is possible to provide additional information to the reader of the diagram.	

Chapter 4

REST architectural style

As we want to do the design according REST principles we need to understand them first. In this chapter we briefly describe a Representational State Transfer (REST) architectural style for distributed hypermedia systems introduced by Roy Thomas Fielding in 2000 at his doctoral dissertation [5]. Generally, REST architectural style can be characterised as:

- Platform independent (there is no problem for a Windows client to connect to a UNIX server).
- Programming language independent.
- Lightweight (offers no built-in security features, encryption, session management, QoS guarantee, but they can be added by building on top of the HTTP protocol implementation).

The REST architectural style is not a standard, is not a protocol and is not an architecture. It is just a set of recommendations or principals. The REST principles are general and it is possible to apply them in the different areas, but they found their main appliance at Web Service and API design. The basic idea is to define a list of entities and their representations. Each operation in the system is a manipulation with an entity through defined list of operations that could be performed under that particular entity.

4.1 Basic principles

In this section we briefly describe each of six REST principles and point out what exactly each of them brings to the system.

4.1.1 Client-Server

The fundamental principle is a usage of a client-server communication pattern. A client-server model is the most common used pattern for distributed systems. The server offers a set of services and listens on service requests from clients. The client in turn sends a request to the server when it requires some work to be done. There are two possible options for the server to react on the service request. The server can reject or accept the request if it was accepted then server must send a reply back to the client. It is possible to say that server is a reactive component and client is proactive component. The client-server principle has the following advantages:

Scalability A client and a server can be developed independently or in parallel.

Portability A server and a client establish a contract for communication. For the client it is does not matter how data are stored at the server side, the only requirement for the client is to fulfil the established contract in the communication.

4.1.2 Stateless

A communication between a server and a client must be stateless. A client's service request should not require the server to use some additional information stored on the server's side. This implies that the service request must contain complete information for request processing on the server's side. A system client-server itself most of the time has some state and if the server cannot take care about the state then the client should. Advantages of this approach are:

Speed Because server does not store any information about the state therefore it can release resources and process service requests faster.

Scalability A server does not have any need for complex resource management between service requests that is why new service addition is a relatively easy operation.

Reliability If some error occurs on the server side, it is easy to restore, because it does not need to read a current state of the connected clients.

Understandability Everything that is needed for service request processing is already contained in the request and no other additional data is required.

4.1.3 Cache

Every server's reply on the client's service request must include information if data in the reply are cacheable or not. If data are cacheable then the client and mediators could reuse it for equivalent requests in the future. Advantages of this principle:

Effectiveness Smaller number of requests sent via the network implies smaller server load.

Performance The user feels higher client responsiveness.

4.1.4 Uniform Interface

Under the term a uniform interface, we understand the communication contract established between client and server. The client and server implementations can be different. The only one requirement is that both of them should fulfil the established contact. It is a fundamental REST principle. An established contract consists of four constraints.

Identification of resources

Every resource in the system should have a unique identifier (for example URI). This identifier is not equivalent with the resource representation. In addition, a server internal resource representation is different from the resource representation sent to clients. Furthermore, the server is able to send the same resource in different representations (JSON or XML for example).

Manipulation of resources through these representations

A representation of the resource received from the server is sufficient for the resource update, removal and any other manipulation.

Self-descriptive messages

Every message must contain enough information in order a recipient knows how to process it. It means that every message besides raw data must contain some meta-data (data describing the data).

Hypermedia as the Engine of Application State (HATEOAS)

This principle we can describe with the following four statements:

- A communication between server and client does not depend on the communication protocol. The client communicates with the server through dynamically generated hypermedia. The only thing a REST client requires in order to communicate with a server is a general ability to understand hypermedia.
- The state of the resource can be changed by the client only through the list of possible actions received from the server.
- The server not the client specifies URI namespace.
- The server should provide to the client information how to build an URI (link and media type) and a client should not surmise what hierarchy resources have.

4.1.5 Layered System

A system should consist of hierarchically ordered layers. Every component (a client or a mediator) has information only about itself and about all components it communicates directly with. Therefore, any component doesn't know if it communicates with a mediator or with an end-server. Mediators can transform data inside the message and resend them further because the messages are self-descriptive and its semantics are transparent. This approach has the following advantages:

Scalability Mediators help to balance the load on servers.

Maintainability Layers are a suitable tool for hiding old clients and old services.

Simplicity You can easily share a common functionality.

4.1.6 Code on demand (optional)

Servers can extend or modify the client's functionality through transferring executable code to the client. Among the advantages of this approach are:

Simplicity We can simplify clients by excluding some functionality obtainable from the servers as an executable code.

Extensibility From the server we can easily spread to the client's some additional functionality. This principle is optional, because such approach will make worse client's visibility.

4.2 REST principles application on Web Services

Web Services implemented according REST principles are called RESTful APIs. Usually they are implemented on top of the HTTP protocol. Four operations provided by HTTP protocol are used for the manipulation with resources through their URIs:

GET Provides read-only access to the resource. Retrieves the resource representation.

PUT Modifies the resource. The resource identified by this URI would be modified according some representation sent as request document.

DELETE Deletes the resource identified by this URI.

POST Creates a new resource according some representation sent as request document and assign a new URI to this new resource in the system.

To fulfil the HATEOAS principle every representation should contain some links for possible actions and resources nearby. A communication between a client and a server is simple. The client sends a HTTP request (URI, headers, body) and the server responds with HTTP response (headers, body, HTTP return code).

It seems, that it is a very easy task to design according REST principles, but it is not. The main problem is to identify the real resources in a system, because this step represents a basic layer for the future implementation. People tend to underestimate the importance of HATEOAS. First of all, a state of the resource can be changed by the client only through the list of possible actions received from the server. This means, that the model of the entity is complete, and client is able to select one from the provided possibilities and is not supposed to invent any new action.

Chapter 5

BP Orchestration Web Services design

In this chapter we concentrate on the problem definition and a solution design based on knowledge from previous chapters. Web Service design or API design is a crucial part for the application, because they represent a public contract. And it is not easy to change their definitions when people already started to use them. First of all in this chapter we describe a problem to solve and present a basic idea of a solution. Then we formalise the idea and briefly describe designed entities in the system. As our solution is driven by REST architectural style as next step we describe a list of resources in the system and their representations. After resources are identified we do a detailed design of Web Services over HTTP protocol.

5.1 Where is a problem

It is possible to say that a precise formal problem description is a half way to the right solution. This section formally describes a problem we are going to solve.

5.1.1 Description

Nowadays there are many possibilities to model business processes and use them through the various business process engines. In the age of effective and fast solutions, people usually think how to reuse already existing models because they were tested in the real environment. Is it possible to use the "lego principle„; meaning to take an already existing business process model and its implementation and use it as an atomic piece in a new process. This idea is not new and current modelling environments provide such an opportunity (sub-process activity in the BPMN 2.0). But usually this sub-process should also be modelled in the same BPE. So, we see this as a problem. A proposed solution represents one more abstraction layer where components can talk to each other regardless underlaying BPEs.

Problem 1 *Business process models and implementations are too tide to the underlying BPE.*

The other part of the problem is, that existing solutions are not flexible enough. Imagine that you are an operator at a factory and you have two possibilities to assemble a detail: manually or with help of robotic arms. If suddenly there were no free robotic arms available,

you would want to change a component of the running instance of the process from “robotic„ to “manual„. However, the operator is not in charge to make any changes to the running business process. There is no easy way to change something on the fly. Our solution provides tools to allow easy change on the fly without a total business process redesign.

Problem 2 *Changes on the fly in the running business process instance are too complex or impossible.*

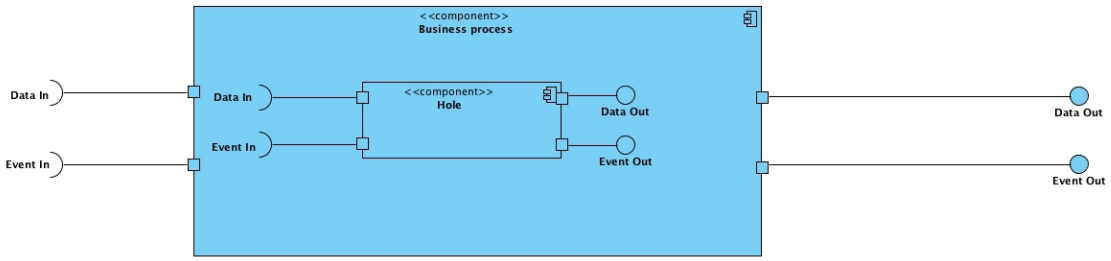
Imagine another situation, at a design phase you should already decide, how exactly activity should look like or assign one particular sub-process. What if you have multiple sub-processes, that have the same inputs and outputs, but are different inside? How can you put all this information together during the design phase? This is also a problem. The proposed solution allows to specify only boundary conditions duri ng the design phase and leave the concrete assignment to the later stages.

Problem 3 *A design of the process with unknown variable components is not supported.*

5.1.2 Basic idea

Let us represent a business process as a black box, but with missing information which we call a *hole*. It means that the process itself is mostly determined and designed by means of one existing BPE but some components are variable (and it is already known at a design phase). It is important to admit that even if a component is variable, it should satisfy some predefined boundary conditions. For the simplicity in the following picture, you can see a process with only one hole.

Figure 5.1: A process with a hole visualisation



It is possible to treat a hole as a black box sub-process, where only little information is known about it:

- What are the input data?
- What is the input event?
- What are the output data?
- What is the output event?

To the hole it is possible to assign any process that meets the specified boundary conditions.

5.2 Abstraction definitions

In this section we describe three entities required to solve all stated problems:

Template of the process is an entity, which represents a model of the process. Almost all internal details are hidden and some variable components are represented as holes. The process that does not have any holes can be also described with this template; just list of holes would be empty.

Pattern of the process (a filled template) is intended to represent a predefined set of process variations where some particular sub-processes are already assigned to the variable components.

Instance of the process is an ongoing process on some real BPE.

This structure allows generating many patterns from one template and many instances from one pattern. In addition, it provides enough flexibility for distributing the process across different servers and BPEs.

5.2.1 Abstract definitions and REST resources

Actually there were several attempts to apply REST principles to business process modelling. In the [10] an intuitive activity-centric approach was taken, where for each activity in every instance of the process a separate resource was created However, a direct applying of REST principles with this approach leads to an unmanageable explosion of resources.

Another approach but with information-centric paradigm was suggested in [13]. Authors managed to eliminate the problem of large number of resources, but introduced another problem, not obvious one. Actually the process is just a mechanism to transform some inputs to some outputs. In this approach an information is taken as a resource. Theoretical results are quite good, but as we can see the approach is not popular, because there is an inconsistency between a visual representation of the business process (it is still activity-centric BPMN) and the way of controlling the business process (information-centric).

Contrary to the both approaches we suggest to look on the problem from different angle. There are already many tools for business process modelling and we do not want to reimplement their functionality, but instead of it we want to design a tool for orchestration of business processes across different BPEs. That is why as a resource we take a business process itself with its variations and allow users to use BPMN (activity-centric approach) for modelling. The template-pattern-instance structure underlies the RESTful design of BP Orchestration Web Services.

5.2.2 A template of the process

In order to solve Problem 3 we want to introduce the notion of a process template. A template of the process is an entity which represents a model of the process. Almost all internal details are hidden and some variable components are represented as holes. Therefore, the template has the following parameters:

- The input data.
- The start event.
- The output data.

- The end event.
- The list of holes.
- The engine information.

The process that does not have any holes is also a template but with an empty list of holes. As we already said, the process itself is completely modelled in some real BPE (holes should be modelled in a special way there).

5.2.3 A pattern of the process

A pattern of the process is intended to represent one particular process variation, where other patterns could be assigned to each hole. A pattern can have only two states:

- WIP (work in progress)
- READY

A pattern in the READY state represents a ready to use model of the process. It means that some patterns are already assigned for each hole. From such a pattern, it is possible to create an instance of the process. A pattern in the WIP state represents a model of the process, where for at least one hole a sub-process was not assigned. Moreover, it is not possible to create an instance of the process from such pattern. It is important to admit that users are not supposed to manipulate the pattern state directly. The state changes automatically when a sub-process is assigned to the hole or unassigned. It is obvious that the pattern created from a template without holes would be immediately in the READY state. Such approach helps to solve Problem 1 and Problem 3.

5.2.4 An instance of the process

An instance of the process is actually generated from some pattern. It represents an ongoing process on some real BPE. When an instance of the process is created all hole assignments are copied from the parent pattern, but user is allowed to change any hole assignment in the running instance if this running instance didn't reach the hole yet. This is an elegant solution to the Problem 2. Therefore, the instance can have the following states:

NOTSTARTED An instance created in BP Orchestration Web Services, but the real process on a real BPE didn't start yet or is in some "initialising„ state.

RUNNING An instance of the process is active (at least one activity is running and does not wait for any resource to become available).

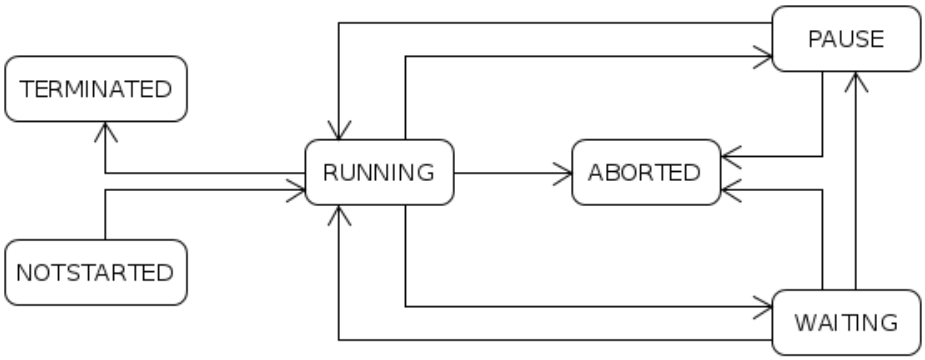
PAUSED An operator manually holds the instance of the process.

ABORTED An operator manually terminates the instance of the process before it reached the logical end.

TERMINATED The state automatically changes to TERMINATED when the instance of the process reaches its logical end.

WAITING When the instance of the process reaches the point when all current activities are waiting for some resources. But when the resources become available, the state would be automatically changed to RUNNING.

Figure 5.2: Instance states diagram



5.3 RESTful Web Services design

We want to design BP Orchestration Web Services based on REST architectural style. According its principles first of all we need to identify resources in the system, then we need to identify URI namespace for all resources, afterwards we could define resource representations and finally we should apply HTTP protocol on our resources. In this section we describe all steps listed above.

5.3.1 Resource identification

A *resource* is anything interesting enough to be a target of a hypertext link [12]. Being a user let us take a closer look at this problem. First of all, a user wants to know what business processes are already modelled in the system. Therefore, the entry point would be a list of templates. According RESTful architectural style the list of templates is a resource itself and every template is also a resource. Secondly, before a template can be used it should be customised and turned into a pattern. Therefore, a list of patterns (all patterns or some special query) is another resource and every pattern would be treated also as a resource. Only when the user finishes customisation of the process (pattern creation) it is possible to run an instance of the process. It is obvious that the instance of the process is a main resource in the system and again a list of instances is a separate resource too. Now consider the following situation: the instance was already started, but operator needs to change something urgently in the context of the possible component variations. So there is a need for every instance to have information about boundary conditions for each variable components and information what pattern (as a sub-process) was substituted there to be able to change it. Therefore, list of holes for every instance and every pattern is a resource, therefore each pattern and instance hole is a resource too.

A complete list of resources supported by BP Orchestration Web Services:

1. A list of templates.
2. A template.
3. A list of patterns.

4. A pattern.
5. A list of pattern holes.
6. A pattern hole.
7. A list of instances.
8. An instance.
9. A list of instance holes.
10. An instance hole.

5.3.2 URI namespace for resources

A RESTful Web Services introduces its data through resources and manipulation with data through the manipulation with resources. Resources are named with URIs and the URI contains all the scoping information in a resource-oriented system. Let us root a BP Orchestration Web Services for example at http://mybusinessprocess.com/. From now on only relative URIs would be used in the text (understand that they are relative to this root). Previously we have determined the list of resources and now we should define URIs for them:

/templates This resource represents a list of templates in the system and provides complete information about each template.

/templates/<identifier> This resource represents one particular template identified by <identifier> and contains complete information about it.

/patterns?template=<identifier>&state=<state> This resource represents a list of patterns and provides some details about each pattern excluding holes (as hole is represented by a separate resource). Parameters **template** and **state** are optional. The parameter **template** limits a list of patterns to the set of patterns generated from the template identified by <identifier>. The parameter **state** limits a set of patterns to the set of patterns in the state <state>.

/patterns/<identifier> This resource represents one particular pattern identified by <identifier> and contains complete information about it excluding holes (because holes are represented as separate resources).

/patterns/<identifier>/holes This resource represents a list of holes in the pattern identified by <identifier> and provides complete information about each hole.

/patterns/<identifier>/holes/<hole_name> This resource represents one particular hole identified by <hole_name> in the pattern identified by <identifier> and contains complete information about the hole.

/instances?pattern=<identifier>&state=<state1>,...,<stateN> The resource represents a list of instances and provides some details about each instance excluding holes (as hole is represented by a separate resource). Parameters **pattern** and **state** are optional. The parameter **pattern** limits a list of instances to the set of instances generated from the pattern identified by <identifier>. The parameter **state** limits a set of instances to the set of instances in the states <state1>,...,<stateN>.

/instances/<identifier> This resource represents one particular instance identified by <identifier> and contains complete information about it excluding holes (because holes are represented as separate resources).

/instances/<identifier>/holes This resource represents a list of holes in the instance identified by <identifier> and provides complete detailed information about each hole.

/instances/<identifier>/holes/<hole_name> This resource represents one particular hole identified by <hole_name> in the instance identified by <identifier> and contains complete information about the hole.

5.3.3 Resource representations

In the previous sections it was already decided which resources we are exposing through BP Orchestration Web Services and how their URIs look like. The next step is to determine how exposed resources are represented.

According to the REST architectural style, a RESTful resource representation should convey the current state of the resource (any information about the underlying resource) and link to possible new resource states or other resources nearby. The goal of resource links is connectedness (the ability to get from one resource to another by following links) [12]. Therefore, we need to think about actions under the resources and provide easy way to use them. While surfing the web, nobody types the URL to get to a resource. Usually we do so only once and after that we are just following the links. Bear in mind, that BP Orchestration Web Services are intended to be machine-usable and therefore all representations must be machine-readable.

There are many different data formats (HTML, XML, JSON and so on). We selected JSON (Java Script Object Notation) to be a primary resource representation format at the design phase as JSON is structured, lightweight, human- and machine-readable.

Template and List of templates

A template resource representation should specify a name and an identifier for the template, a list of holes in it, a list of links, the basic information about the engine (an engine type, a base URL for connection and a real process id on that particular engine, all other information is built in the particular engine connector) and some basic information about inputs and outputs. For more details see Listing 5.1.

Listing 5.1: JSON resource representation for Template

```
{
        "name": "some template name",
        "data_in": "some data1",
        "data_out": "some data2",
        "event_start": "some event1",
        "event_end": "some event2",
        "engine": {
                "baseUri": "some base uri",
                "processId": "some identifier in the real engine",
                "name": "the type of the engine"
        }
        "holes": [
                {
```

```
                        "name": "some hole name",
                        "data_in": "some data in h1",
                        "data_out": "some data out h2",
                        "event_start": "some event start h1",
                        "event_end": "some event end h2"
                },
                {...}
        ],
        "id": "template_id",
        "links": [
                {
                        "name" : "generatepattern",
                        "method" : "POST",
                        "uri": "/patterns?template=template_id",
                        "requestDocument" : null
                },
                {
                        "name" : "patterns",
                        "method" : "GET",
                        "uri": "/patterns?template=template_id",
                        "requestDocument" : null
                }
        ]
}
```

Resource representation for List of Templates is specified in the Listing 5.2.

Listing 5.2: JSON resource representation for List of Templates

```
[
        {Template resource representation},
        ...,
        {Template resource representation}
]
```

Pattern and List of patterns

A pattern resource representation should specify a name and an identifier for the pattern, a list of links, status of the pattern and a template from which this pattern was generated. For more details see Listing 5.3.

Listing 5.3: JSON resource representation for Pattern

```
{
        "name": "some pattern name",
        "status" : some_status,
        "template_id" : "template_id",
        "id": "some_pattern_id",
        "links": [

                {
                        "name" : "getTemplate",
                        "method" : "GET",
                        "uri": "/templates/template_id",
                        "requestDocument" : null
                },
```

```
                {
                        "name" : "getHoles",
                        "method" : "GET",
                        "uri": "/patterns/some_pattern_id/holes",
                        "requestDocument" : null
                },
                {
                        "name" : "generateInstance",
                        "method" : "POST",
                        "uri": "/instances",
                        "requestDocument" : "{ "pattern_id" : "
                           some_pattern_id" }"
                },
                {
                        "name" : "instances",
                        "method" : "GET",
                        "uri": "/instances?patternId=some_pattern_id",
                        "requestDocument" : null
                }
}
```

Resource representation for List of Patterns is specified in the Listing 5.4.

Listing 5.4: JSON resource representation for List of Patterns

```
[
        {Pattern resource representation},
        ...,
        {Pattern resource representation}
]
```

Pattern hole and List of pattern holes

A pattern hole resource representation should specify a name of the pattern hole (it is unique in one particular pattern), a list of links, some basic information about inputs and outputs, information to which pattern this hole belongs and which pattern is assigned to the hole. For more details see Listing 5.5.

Listing 5.5: JSON resource representation for Pattern Hole

```
{
        "name": "hole_name",
        "data_in": "some data in h1",
        "data_out": "some data out h2",
        "event_start": "some event start h1",
        "event_end": "some event end h2"
        "pattern_assigned" : "pattern_assigned_id",
        "pattern_parent":"pattern_parent_id",
        "links": [
                {
                        "name" : "assignPattern",
                        "method" : "POST",
                        "uri": "/patterns/pattern_parent_id/holes/
                           hole_name",
                        "requestDocument" : "{"assigned_pattern_id":""}"
                },
```

```
                {
                        "name" : "getParent",
                        "method" : "GET",
                        "uri": "/patterns/pattern_parent_id",
                        "requestDocument" : null
                }
}
```

Resource representation for List of Pattern Holes is specified in the Listing 5.6.

Listing 5.6: JSON resource representation for List of Pattern Holes

```
[
        {Pattern hole resource representation},
        ...,
        {Pattern hole resource representation}
]
```

Instance and List of instances

An instance resource representation should specify an identifier of the instance, a list of links, a status of the instance, start date and date of last changes and from which pattern this instance was generated. For more details see Listing 5.7.

Listing 5.7: JSON resource representation for Instance

```
{
        "state": some_instance_state,
        "id": "some_instance_id",
        "start_date": "some date1",
        "last_change_date": "some date2",
        "pattern_id": "pattern_id"
        "current_activities" :   [
             {
                "name": "Step1",
                "completed": "2",
                "ready": "1",
                "executing": "5"
             },
             {
                        "name":"Step3",
                 "completed": "10",
                 "failed": "2"
             },
         ],
        "links": [
                {
                        "name" : "getHoles",
                        "method" : "GET",
                        "uri": "/instances/some_instance_id/holes",
                        "requestDocument" : null
                },
                {
                        "name" : "getPattern",
                        "method" : "GET",
                        "uri": "/patterns/pattern_id",
```

```
                "requestDocument" : null
            }
        ]
}
```

Resource representation for List of Instances is specified in the Listing 5.8.

Listing 5.8: JSON resource representation for the List of Instances

```
[
        {Instance resource representation},
        ...,
        {Instance resource representation}
]
```

Instance hole and List of instance holes

An instance hole resource representation should specify a name of the instance hole, a list of links, basic information about input and output information, assigned pattern, instance of the assigned pattern, status of the assigned instance, and an instance where this hole belongs. For more details see Listing 5.9.

Listing 5.9: JSON resource representation for Instance Hole

```
{
        "name": "hole_name",
        "data_in": "some data in h1",
        "data_out": "some data out h2",
        "event_start": "some event start h1",
        "event_end": "some event end h2"
        "pattern_assigned_id" : "pattern_assigned_id",
        "parent_instance_id": "parent_instance_id",
        "state": some_instance_state,
        "links": [
                {
                        "name" : "assignPattern",
                        "method" : "POST",
                        "uri": "/instances/parent_instance_id/holes/
                            hole_name",
                        "requestDocument" : "{"assigned_pattern_id":""}"
                },
                {
                        "name" : "startHole",
                        "method" : "POST",
                        "uri": "/instances",
                        "requestDocument" : "{ "pattern_id" : "
                            pattern_assigned_id" }"
                },
                {
                        "name" : "getParent",
                        "method" : "GET",
                        "uri": "/instances/parent_instance_id",
                        "requestDocument" : null
                },
                {
                        "name" : "getPattern",
```

```
                    "method" : "GET",
                    "uri": "/patterns/pattern_assigned_id",
                    "requestDocument" : null
                }
        ]
}
```

Resource representation for List of Instance Holes is specified in the Listing 5.10.

Listing 5.10: JSON resource representation for List of Instance Holes

```
[
        {Instance hole resource representation},
        ...,
        {Instance hole resource representation}
]
```

5.3.4 Resource manipulation

Usually a manipulation with resource through the HTTP protocol is done through the GET, POST, PUT and DELETE methods. In this section we describe what methods are applicable to which resource URI and what is their meaning in each case.

List of templates /templates

GET Retrieve a list of existing templates.

POST Add a new template to the list. As a request document a representation of a new template is sent.

DELETE Delete all known templates (this method is supported only for testing purposes).

One particular template /templates/<identifier>

GET Retrieve a complete information about the template with the identifier <identifier>.

DELETE Delete the template with the identifier <identifier>.

A list of patterns /patterns

GET Retrieve a list of existing patterns that match searching criteria.

POST Add a new pattern to the list. As a request document a basic pattern representation is sent.

DELETE Delete all known patterns (this method is supported only for testing purposes).

One particular pattern /patterns/<identifier>

GET Retrieve a complete information about the pattern with the identifier <identifier>.

DELETE Delete the pattern with the identifier <identifier>.

List of holes in the pattern /patterns/<identifier>/holes

GET Retrieve a list of holes in pattern with identifier <identifier>.

One particular hole in the pattern /patterns/<identifier>/holes/<hole_name>

GET Retrieve a complete information about the hole with name <hole_name> in the pattern with the identifier <identifier>.

PUT Modify the hole with name <hole_name> in the pattern with the identifier <identifier>. The only thing we can modify is an assigned pattern, therefore as requested document instead of the whole hole representation a simplified version is sent (see Listing 5.11).

Listing 5.11: Request document for pattern hole modification

```
{
        "pattern": "new_asseinged_pattern_id"
}
```

A list of instances /instances

GET Retrieve a list of existing instances that match searching criteria.

POST Add a new pattern to the list. As a request document a basic pattern representation is sent.

DELETE Delete all known instances (this method is supported only for testing purposes).

One particular instance /instances/<identifier>

GET Retrieve a complete information about the instance with the identifier <identifier>.

DELETE Delete the instance with the identifier <identifier>.

PUT Modify the instance with the identifier <identifier>. The only thing we can modify is a status of the instance, therefore as requested document instead of the whole instance representation a simplified version is sent (see Listing 5.12).

Listing 5.12: Request document for instance modification

```
{
        "state": some_new_state
}
```

List of holes in the instance /instances/<identifier>/holes

GET Retrieve a list of holes in instance with identifier <identifier>..

One particular hole in the instance /instances/<identifier>/holes/<hole_name>

GET Retrieve a complete information about the hole with name <hole_name> in the instance with the identifier <identifier>.

PUT Modify the hole with name <hole_name> in the instance with the identifier <identifier>. We can modify an assigned pattern or we can start the instance for this hole, therefore as requested document instead of the whole hole representation a simplified version is sent (see Listing 5.13).

Listing 5.13: Request document for instance hole modification

```
{
        "pattern": "new_asseinged_pattern_id",
        "action" : "some_action"
}
```

5.4 Formal Web Services specification

To design some API is easy, but make its users really love is harder. To make this happen we need to have easy-to-read documentation, API should be consistent, scalable and easy-to-test. All these targets we can achieve with RESTful API Modelling Language (RAML). It is a language for API description. As RAML is human friendly format it is easy to use. As RAML is a machine-readable we can easily generate a code prototype for a server's side and some mockups for a client's side and furthermore, it is easy to generate a full documentation.

The design of the Web Services was done in RAML language with help of apiworkbench package for Atom.

Chapter 6

BP Orchestration Web Services implementation

The Web Services were designed in details in the previous chapter and the aim of this chapter is to describe an implementation.

The design of BP Orchestration Web Services was done with the assumption that underlaying BPEs would be proactive. In other words, BPE should call some BP Orchestration Web Service when the execution of the process reaches the end or reaches some hole.

If there is some BPE that is not able to call our BP Orchestration Web Service, then on the BP Orchestration Web Service side a polling of an instance state should be implemented as part of the connector implementation.

6.1 Technologies, tools, applications

6.1.1 RAML

As it was mentioned in the previous chapter formal Web Services specification was done in RAML.

6.1.2 BonitaBPM

Among a big variety of different BPEs BonitaBPM was chosen because it has open source, free to use version and it has a RESTful API and the last but not least there is a documentation for this RESTful API.

6.1.3 Groovy 2.4

BonitaBPM provides a possibility to write some functionality (e.g. connectors) in Groovy 2.4. Some integration scripts were written in the Groovy editor build-in the BonitaBPM.

6.1.4 Jersey

The Web Services were implemented in Java. Jersey RESTful Web Services framework was chosen as base framework for the implementation. It is open source framework designed to provide developers an easy way to develop RESTful Web Services in Java. It supports from the box the easy definition of the web services and it has multiple media type support. By multiple media type support in Jersey we mean on one side an automatic detection of the

media type for particular request and provision of services (but not parsers itself) to parse the request document. On the other side we mean the detection of the requested media type for a response.

In the following code example (see Listing 6.1) you can see, that the `TemplateService` class is responsible for the relative URL `/templates`. When server receives a GET request for relative URI `/templates/someid` it automatically detects the place, where it should be processed. Also it is easy to define the supported formats for request or response documents. It is important to admit, that the framework automatically converts the request document to Java object according the specification and Java object to XML, JSON or any other format for the response document.

Listing 6.1: Jersey framework usage

```
@Path("/templates")
public class TemplateService {

        @GET
        @Path("{id}")
        @Produces({MediaType.APPLICATION_XML,
                                MediaType.APPLICATION_JSON})
        public Response getTemplate(@PathParam("id") String id) {
```

6.1.5 Jackson

In order to support JSON media type a simple Java-based library Jackson was chosen. It supports a serialisation of Java objects to JSON and vice versa. The library is open source, free to use and it does not require any other library apart from JDK. The only thing you should do is to put annotations provided by the library to the right place in the code and all the rest is handled automatically by Jersey.

The following code (see Listing 6.2) demonstrates the power of the library driven by its simplicity

Listing 6.2: Jackson library usage

```
public class PatternHole extends Hole implements Serializable {

        private String patternParent = null;

        private String patternAssigned = null;

        @JsonGetter("pattern_parent")
        public String getPatternParent() {
                return patternParent;
        }

        @JsonSetter("pattern_parent")
        @XmlElement(name = "pattern_parent")
        public void setPatternParent(String patternParent) {
                this.patternParent = patternParent;
        }
```

The `@JsonGetter` annotation tells Jackson that the value of the property `pattern_parent` in JSON document should be obtained from this method. The `@JsonSetter` property works the other way. It is important to admit, that there is a build-in mechanism that detects

what methods are tied to which property in JSON. Sometimes it might detect something you don't want. The Jackson annotation `@JsonIgnore` solves this problem, as it is used to tell Jackson to ignore a certain property (or field or method) of a Java object.

6.2 BonitaBPM connector

The goal of the designed Web Services is to orchestrate the execution of the business processes under different BPEs and allow on-the-fly changes. As BonitaBPM was chosen as a target BPE in this section we would describe in details how integration was implemented. As Bonita is only one of the possible target BPEs the solution was designed in general, so other connectors could be added in the future.

6.2.1 Integration in general

The terminology for each BPE could be different, but the basic ideas would stay the same. Everywhere we would have some definition of the business process and some running instance of the process. We can start instances and read information about them.

What information do we need for our template of the process? First of all we need a `name` of the engine (whether it is BonitaBPM 7.2 or BonitaBPM 6.0 or something else). In the engine every process has to have its `processId` and we should know the `basUri` where this engine is available. What basic operations are expected to be done under the process? For example we want to start the instance of the process and method `generateInstance` is responsible for that. It is important to admit, that we expect the real BPE be proactive and signal back when instance of the process reaches some hole or the end of the process. That is why we would propagate the URL of the instance to the real instance on BPE side.

Listing 6.3: General BPE description in Java

```
public class EngineBpe implements Serializable{

        private EngineBP name = null;

        private String processId = null;

        private String baseURI = null;

        public WS getProcessResource() {
                return null;
        }

        public GeneralCase generateInstance (String instanceUrl)
                        throws InternalErrorException {
                return null;
        }
```

6.2.2 Integration in details

Let us here describe a minimal subset of RESTful API in the BonitaBPM needed. According the documentation, the only media type used for request and response documents in BonitaBPM is JSON. At the same time we describe here the counterpart in the implementation. The main information about BonitaBPM is incapsulated in the class

`BonitaConnector7_2`. Its methods and fields would be described together with appropriate RESTful API.

Read information about the process

The information about a real process on the real BPE is only informative, so we are not going to unify it from different BPEs, but instead we will just provide a direct link to it. According HATEOS principle, we are encouraged to provide links for near resources. So `links` field for the `template` will have an item with name `direct_process_information`.

Also it is important to admit, that after a process is "redeployed„ on the Bonita server it will obtain new id (even if nothing has changed).

The specification for the call you can find in the table 6.1.

Table 6.1: Read information about the process in BonitaBPM 7.2

Request url	`/API/bpm/process/{processId}`
Request method	GET
Request document	empty
Response document format	application/json

Response document	{
	"id":"1",
	"icon":"/default/process.png",
	"displayDescription":"process description",
	"deploymentDate":"2015-01-02 14:21:18.421",
	"description":"another process description",
	"activationState":"ENABLED",
	"name":"Pool1",
	"deployedBy":"2",
	"displayName":"Pool1",
	"actorinitiatorid":"2",
	"last_update_date":"2015-01-02 14:21:18.529",
	"configurationState":"RESOLVED",
	"version":"1.0"
	}

The implementation is simple (see Listing 6.4), we need to store a relative URI and return the whole URL, method and request document.

Listing 6.4: Information about the process on real BPE

```
private static final String relativeProcessURI = "/API/bpm/process/";

@Override
public WS getProcessResource() {
  if ( !isSet() ) {
    return null;
  }
  return new WS(getBaseURI() + relativeProcessURI + getProcessId(),
```

```
        "GET", null);
}
```

6.2.3 Authentication

All manipulations with processes and instances in the BonitaBPM requires authentication as user in the Bonita Server Engine database. As we do not provide a login service directly to the user, it should be handled automatically and be invisible to the user. The main surprise of this call was an undocumented behaviour in case of failures, because the service even on failure can return HTTP return code 200. So the only option, how to determine if the call was successful is to check if the response is empty.

The call generates a cookie, which must be set on each subsequent call. If the Web RESTful API is being used in an application running in a web browser, this is handled automatically by the browser. Setting the redirect parameter to false indicates that the service should not redirect to Bonita BPM Portal (after a successful login) or to the login page (after a login failure).

The brief description for the call you can find in the table 6.2.

Table 6.2: Authentication to BonitaBPM 7.2

Request url	`/loginservice`
Request method	POST
Request document format	application/x-www-form-urlencoded
Request document	username: SOMEUSERNAME password: SOMEPASSWORD redirect: false Notes: all three parameters are mandatory. The parameter `redirect` has possible values `true, false` and values are case sensitive.
Response document on success	Empty
Response document on failure	Some unpredictable format and content

The implementation (see Listing 6.5) of the authentication is quite simple. We need to fill the request according the specification, check response body if it is empty or not, and return cookies in case of success.

Listing 6.5: Authentication to the real BPE

```
private static final String loginURI = "/loginservice";
private Map<String, NewCookie> authorize() throws InternalErrorException
    {
/* How cookie should look like:
*
* JSESSIONID=86D31D2C4E8ECAE96715FE438D5196D1;
* X-Bonita-API-Token=80379964-771a-44e8-ba1f-825003a4f4ca;
* BOS_Locale=en
*/
  Client client = ClientBuilder.newClient(
```

```
    new ClientConfig().register( LoggingFilter.class ) );
  WebTarget webTargetAuth = client.target( getBaseURI() + loginURI);
  Invocation.Builder invocationBuilderAuth =  webTargetAuth.request();
  MultivaluedMap<String, String> authMap = new MultivaluedHashMap<>();
  // All three values are mandatory
  authMap.add("username", "alena");
  authMap.add("password", "11111");
  authMap.add("redirect", "false");
  Response responseAuth = invocationBuilderAuth.post(
    Entity.entity(authMap, MediaType.APPLICATION_FORM_URLENCODED));
  String responseText = responseAuth.readEntity(String.class);
  if ( !responseText.isEmpty() ) {
    throw new InternalErrorException("Cannot authenticate");
  }
  return responseAuth.getCookies();
}
```

6.2.4 Start an instance of the process

Actually it is quite simple, the only thing we need to do, is take into account, that we should pass some information from our RESTful API to the real instance of the process, as we want the real BPE be proactive. In the BonitaBPM there is a mechanism to do so and is called "case variables„ (how they should be created on the BonitaBPM side will be described in the next chapter).

The brief description for the call you can find in the table 6.3.

The implementation was done according specification.

6.2.5 Information about activities

Bonita RESTful API provides an access to the list of activities in the process. It may happen, that there is a cycle in the process and some activity can be executed more than once. Activity can have the following Bonita-states:

Initializing indicates that an activity is being initialised.

Ready indicates that a human tasks and manual tasks has been initialised but is not yet being executed.

Waiting indicates that a RECEIVE_TASK, BOUNDARY_EVENT or INTERMEDIATE_CATCH_EVENT activities is waiting for some external trigger.

Executing indicates that an activity is being executed.

Failed indicates that a task has failed because of a problem in execution, for example because of an exception that was not anticipated, a connector that fails, or bad expression design).

Skipped indicates that a task that failed because of connector execution failure is being skipped instead of re-executed. Skipping a task skips the execution of any connectors not already executed and proceeds to task completion.

Cancelled indicates that an activity is cancelled by a user.

Aborting indicates that an activity is cancelled by the system. For example, an interrupting event sub-process can trigger ABORTS for all other active paths.

Completed indicates an activity that is complete.

The brief description for the call you can find in the table 6.4.
The implementation was done according specification.

Table 6.3: Instance creation in the BonitaBPM 7.2

Request url	`/API/bpm/case`
Request method	POST
Request document format	application/json

Request document	<pre>{
	"processDefinitionId":"someid",
	"variables":[
	{
	"name":"callerurl",
	"value":"someurl"
	}
	]
	}</pre>

Response document format	application/json

Response document on success	<pre>{
	"end_date": "",
	"processDefinitionId": "6301330702208",
	"start": "2016-05-10 21:47:14.713",
	"rootCaseId": "4020",
	"id": "4020",
	"state": "started",
	"started_by": "4",
	"last_update_date":
	"2016-05-10 21:47:14.713",
	"startedBySubstitute": "4"
	}</pre>

Return code on success	200
Response document on failure	Some unpredictable format and content
Return code on failure	500

Table 6.4: Check the activities in the instance in the BonitaBPM 7.2

Request url	`/API/bpm/caseInfo/case_id`
Request method	GET
Request document	empty
Response document format	application/json

Response document on success	{
	"id": "case_id",
	"flowNodeStatesCounters": {
	"Step1": {
	"completed": "2",
	"ready": "1",
	"executing": "5"
	},
	"Step3": {
	"completed": "10",
	"failed": "2"
	}
	}
	}

Return code on success	200
Response document on failure	Some unpredictable format and content
Return code on failure	500

Chapter 7

BonitaBPM

BonitaBPM is an application platform that provides a set of tools for business process design, implementation and monitoring. The set of tools supports a business data modelling, a business process modelling using BPMN 2.0, business process execution (Bonita Service Engine), user management, pages design (UI designer) and business process execution monitoring (Bonita Portal). Such a comprehensive platform provides an opportunity to set up the business process in the company from scratch.

The process of business process modelling in BonitaBPM can be divided into several independent parts:

1. Describe data model of the process (what information is needed, what type does this information have, how these informations are organised into bigger structures)
2. Describe the process in BPMN 2.0
3. Design pages for each human activity
4. Create and assign connectors
5. Create service tasks

In order to demonstrate the functionality of the BP Orchestration Web Services we need some dummy processes. So in this chapter we will concentrate on options 2 and 4 and skip options 1, 3, 5.

In the following sections two processes would be described. In the first section "Simple process without holes„ we describe in details how process and connectors are created in BonitaBPM on the example of the process without holes. In the second section "Process with holes„ we describe the advanced version of the process.

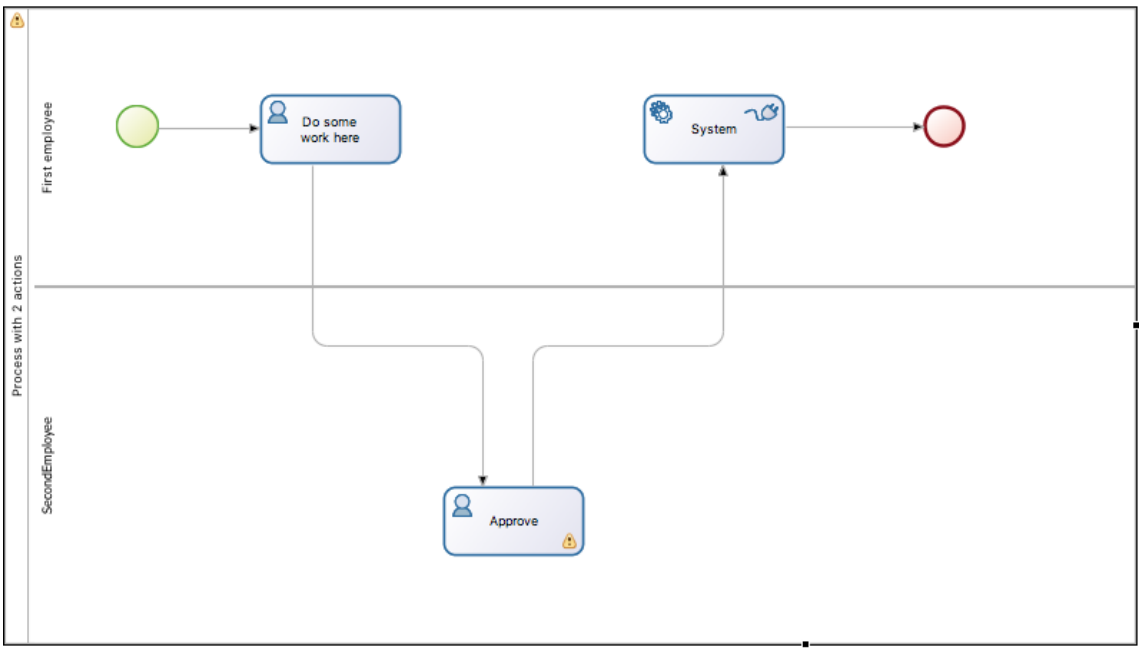
7.1 Simple process without holes

In this section we will describe step by step how to create a simple business process without holes in BonitaBPM and how to create connectors in BonitaBPM.

7.1.1 BMPN 2.0 definition of the process in BonitaBPM

We would create a BP as shown on the picture 7.1.

Figure 7.1: Simple process definition with BPMN 2.0



A business process consists of one pool, two lanes and three actions. You can see, that two actions are designed to be human tasks and one to be a service task. This service task is placed before the end of the process and should signal to BP Orchestration Web Services that an instance running on the real BPE (in our case on Bonita Engine) is finished.

7.1.2 Case variables in BonitaBPM

If we want an instance of the process running on real BPE to signal to BP Orchestration Web Services the end of the process we need to pass URL of the process instance according BP Orchestration Services (`http://somebaseURL/instances/someid`) to it. As it was already stated before the only possibility to do so in Bonita BPM is to use so called case variables. Actually "case variables„ in BonitaBMP is a synonym for all pool variables in the process. The picture 7.2 shows where in BonitaBPM we can add them.

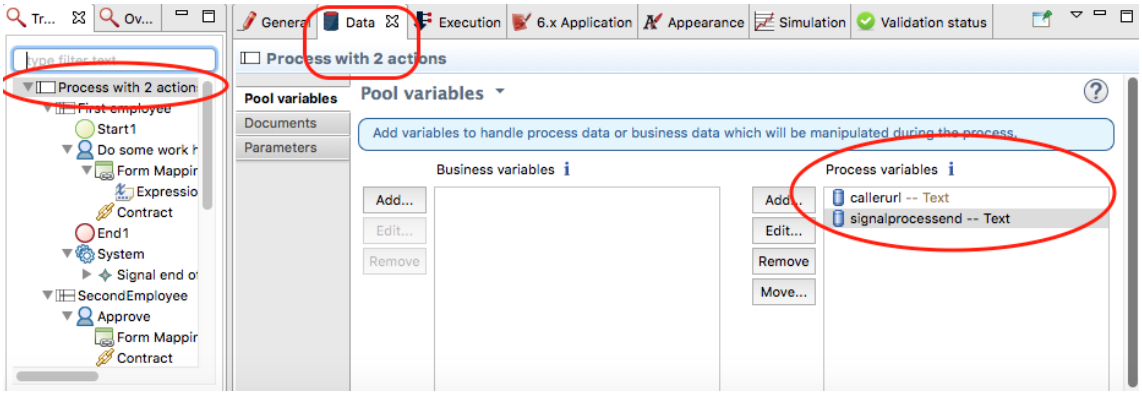
Actually we can see that there are two variables (`callerurl` and `signalprocessend`). The former is used as an input variable to pass a caller URL and the latter is used as an output variable to store the result of the `System` service task.

7.1.3 Connectors in BonitaBMP

BonitaBMP has already quite big variety of the connectors to the outside world (SOAP, SMTP, Google Calendar, Groovy 2.4 scripts and many others). RESTful Web Services are very popular, but you cannot find any "REST connector„ in the list. So we will write a Groovy 2.4 script to do the work.

Connectors in Bonita are divided into two types: in and out. "In„-connector is called when task is started and "out„-connector is called when task is finished. We will use

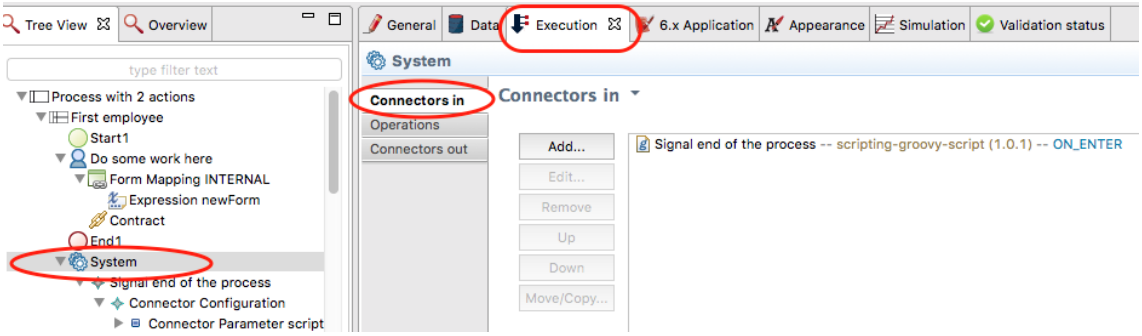
Figure 7.2: Pool variables in BonitaBPM



"in„-connectors.

The picture 7.3 shows where you can add a connector to the task.

Figure 7.3: "In„-connector for a task in BonitaBPM



7.1.4 Creating a "Signal process end„ connector in BonitaBMP

When you press "Add„ button a connector wizard starts. First of all you should chose a connector type (in our case Groovy script 2.4) from the list shown on the picture 7.4.

First of all you should enter name of the connector and select engine reaction if connector failed (see picture7.5).

As next step (see picture 7.6) you should define a script. You can select the first option to load script from the library or the second option to create a new script (with pencil icon). After you define the script you can also save it to the script library.

After you click on the pencil icon in the previous step you would see a small build-in editor for Groovy (see picture 7.7). In the script all local, global and business variables defined in the process are available (but only in read only mode). Script is quite small, so the whole listing is provided in listing 7.1:

Listing 7.1: Groovy script "Signal process end„

```
// define URL
```

Figure 7.4: List of available connectors in BonitaBPM

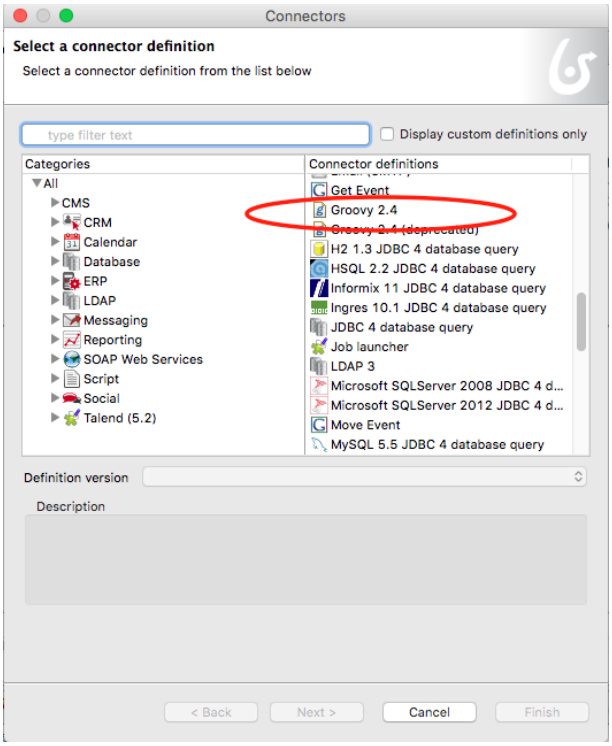
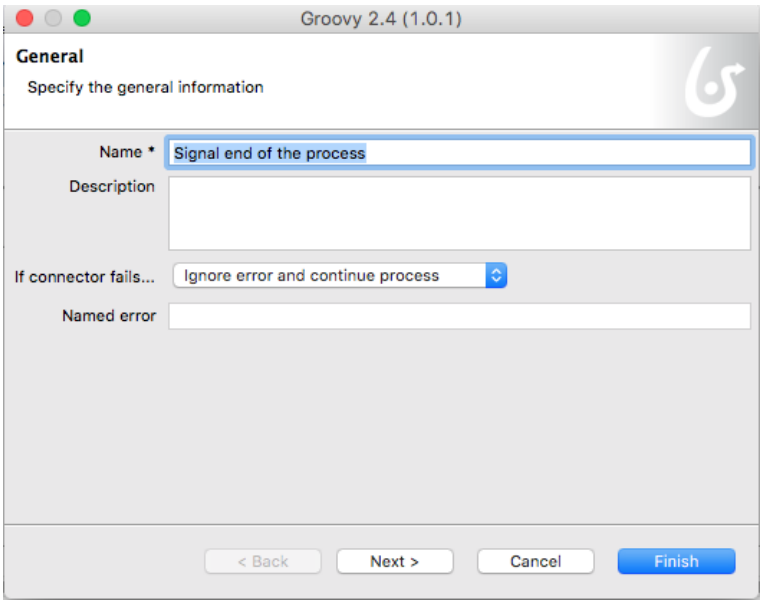
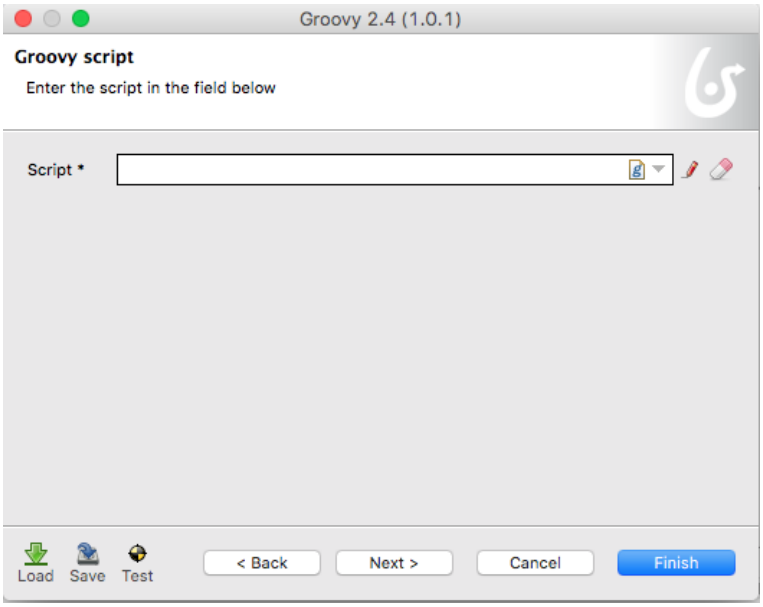


Figure 7.5: Connector creation in BonitaBPM




```
def url = new URL(callerurl)
```

Figure 7.6: Choose script for groovy connector in BonitaBPM




```
// create connection
def connection = url.openConnection()
// set up the method
connection.setRequestMethod("PUT")
// set up the type of the request document
connection.setRequestProperty("Content-Type","application/json");
// set up the type of the expected response document
connection.setRequestProperty("Accept","application/json");
// we are going to send Request document
connection.doOutput = true

def data = """{"state":"TERMINATED"}"""
new DataOutputStream(connection.outputStream).withStream { s -> s.
    writeBytes data }
def body = connection.content.text

return body;
```

Also we need to specify, what should be done with the connector output (see picture 7.8). In out case, we want to store the result in particular variable. So in the left box choose the correct variable and on the right box click pencil. In the new window fill fields according the picture 7.9.

7.2 Process with one holes

In this section we describe how we could create a process with one hole in BonitaBPM. First of all let us investigate the main problem of the integration.

In order to integrate BP Orchestration Web Services with BonitaBPM we need to decide which component would be responsible for a communication initiation. The original thought

Figure 7.7: Write a groovy script in BonitaBPM

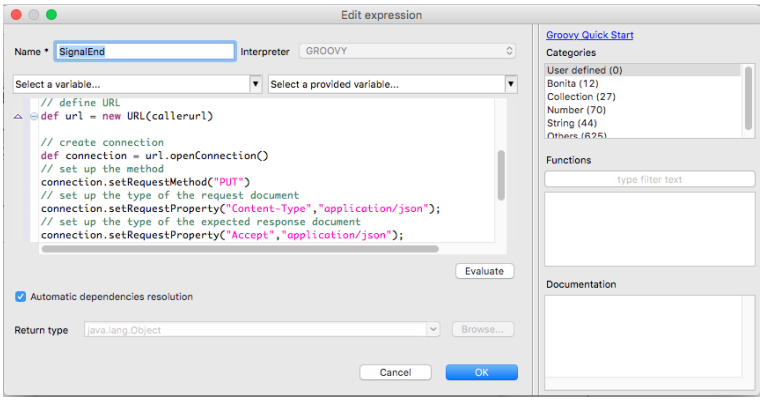
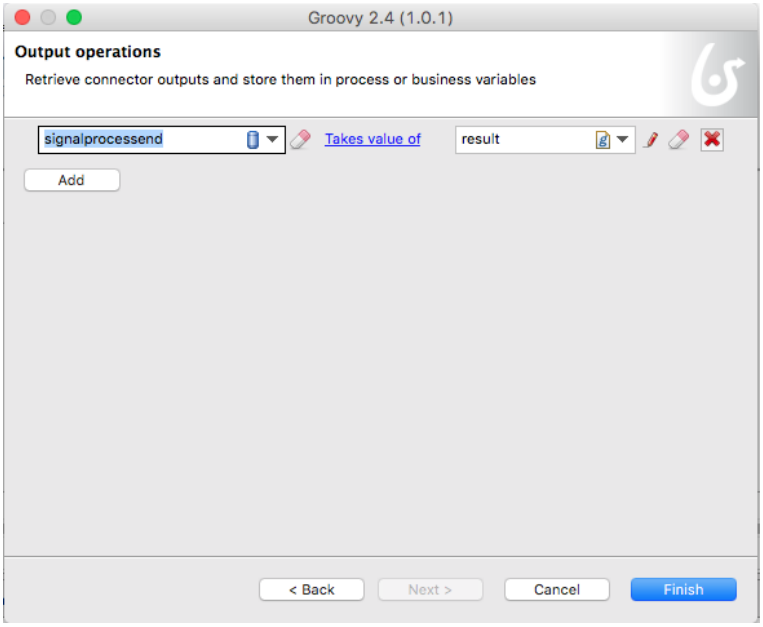


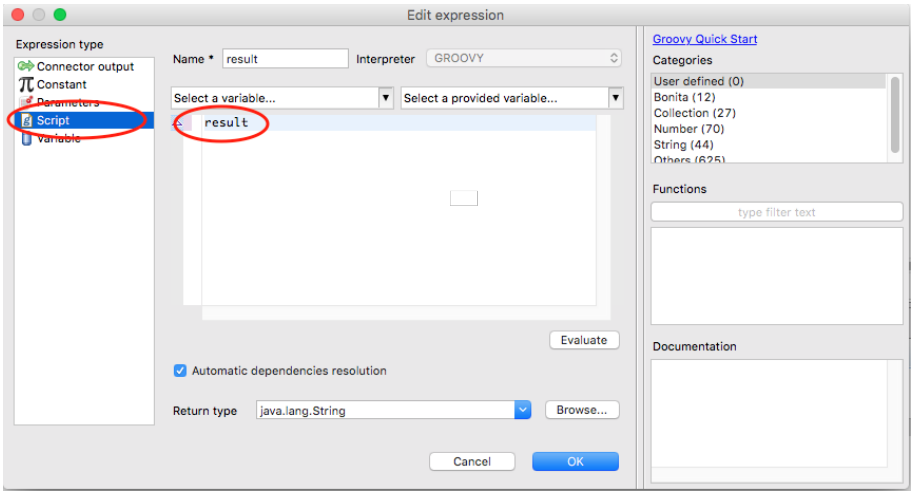
Figure 7.8: Output operations for connectors in BonitaBPM



was that all necessary actions should be done by BP Orchestration Web Services. But the main problem is that there is no easy way to signal to Bonita Engine, that something had happened in some particular instance of the process. By this we mean, that their RESTful API does not provide a possibility to "generate„ an event in the process. At this point is already clear, that BonitaBPM should signal to the BP Orchestration Web Services that instance had finished or instance has reached the hole. As BonitaBPM should continue only if hole was finished, so it also should periodically check the status of the hole.

This decision is important because it impacts the design of the Web Services. We can state, that BonitaBPM would be a first known client of BP Orchestration Web Services. So design incapsulate the solution for this client. For example there is a need to have a possibility to check the state of the hole in one call with almost no manipulations done. That

Figure 7.9: Expression that takes the result of the script in BonitaBPM



is why every instance hole has a "state„ and BP Orchestration Web Services automatically dynamically checks the status of the underlaying instance.

7.2.1 BPMN 2.0 notation for the process with one hole

Here we describe the elements of a process with one hole. The process consists of one pool, two lanes, six actions, one timer and one decision gate (see picture 7.10). Let us briefly describe each of them.

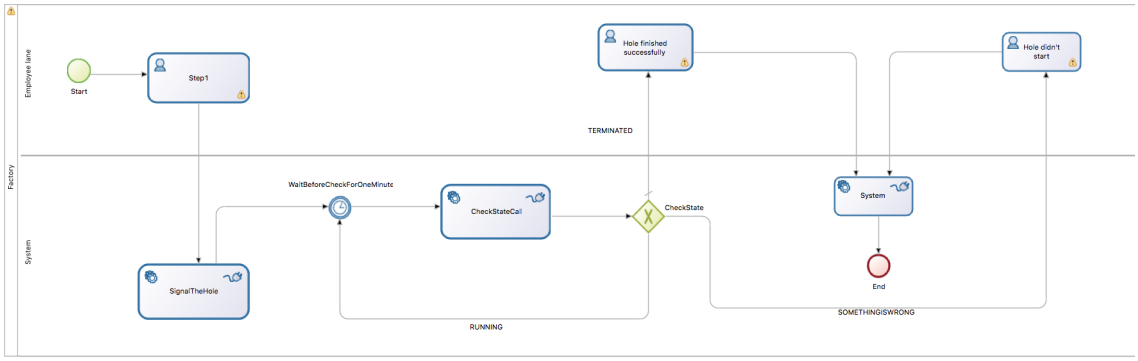
The upper lane represents a user and has three human activities inside. The first activity called "Step1„ represents a real work done. The second one "Hole finished successfully„ and the third one "Hole didn't start„ are just mockups used to demonstrate which branch actually was chosen during the run time.

The lower lane represents system activities, that do not require human interaction. The first activity "Signal the hole„ is an activity, that signal to BP Orchestration Web Services, that process has just reached the hole and the underlaying process should be started. The second system activity "CheckStateCall„ is used to get an actual status of the hole through BP Orchestration Web Services. And the last activity "System„ (the same as in the Simple process) is used to signal to BP Orchestration Web Services, that this instance of the process is finished. It doesn't make sense to overload BP Orchestration Web Services with requests, that is why the check of the hole status is done here once in minute using timer. After the status of the hole is updated, we should check the result. If result is "RUNNING„ then we will return to the timer and wait for one more minute. A hole process status "TERMINATED„ means, that hole is finished and the process can continue. And the third branch is "NOTSTARTED„ can be triggered, if a hole underlaying process wasn't started at all.

7.2.2 Process and local variables used

In this process we have to have the same process variables as for Simple process `callerurl`, `signalprocessend` and one more called `ahole` with represents the status of the hole, as it

Figure 7.10: Advanced process definition with BPMN 2.0 (process with one hole)



should be accessible from 2 elements (activity and decision gate). Local variables are used to store the name of the hole in the activities “SignalTheHole„ and “CheckStateCall„

7.2.3 Signal the hole

The activity is very similar to “System„ activity in the Simple process, but another script (see Listing 7.2) in the Groovy connector was used.

Listing 7.2: Groovy script “Signal the hole„

```
// define URL
def u = callerurl+"/holes/"+holename;
def url1 = new URL(u)
// create connection
def connection1 = url1.openConnection()
// set up the method
connection1.setRequestMethod("PUT")
// set up the type of the request document
connection1.setRequestProperty("Content-Type","application/json");
// set up the type of the expected response document
connection1.setRequestProperty("Accept","application/json");
// we are going to send Request document
connection1.doOutput = true
def data = """{"action":"START"}"""
try{
        new DataOutputStream(connection1.outputStream).withStream { s ->
            s.writeBytes data }
        def body = connection1.content.text
        if ( connection1.responseCode != 200 ) {
                return "NOTSTARTED";
        }
        else {
                return "RUNNING";
        }
}
catch ( Exception e ) {
        return "NOTSTARTED";
```

}

7.2.4 Check the status of the hole

This activity is also very similar to “System„ activity in the Simple process, but with another Groovy script used (see Listing 7.3).

Listing 7.3: Groovy script “Check status of the hole„

```
// define URL
def u = callerurl+"/holes/"+holename;
def url1 = new URL(u)
// create connection
def connection1 = url1.openConnection()
// set up the method
connection1.setRequestMethod("GET");
// set up the type of the expected response document
connection1.setRequestProperty("Accept","application/json");
// we are going to send Request document
connection1.doInput = true
try{
        def body = connection1.content.text
        if ( connection1.responseCode != 200 ) {
                return "NOTSTARTED";
        }
        else {
                def slurper = new JsonSlurper()
                def r = slurper.parseText(body)
                return r.state;
        }
}
catch ( Exception e ) {
        return "NOTSTARTED";
}
```

Chapter 8

Test the solution

8.1 Unit tests

To follow the best practises we need to write unit tests for each small part of functionality. The Web Services are based on the designed template-pattern-instance structure. That is why for testing purposes three additional classes were developed: `InstanceClient`, `PatternClient`, `TemplateClient`. They provide easy to use interface where each method represents one particular action according 5.3.4 and incapsulates the knowledge about available URIs in the system and which HTTP methods are applicable there. Media types for response and request documents are parameters in the method, that is why it is easy to test for multiple media types (was tested for XML and JSON).

Listing 8.1: InstanceClient class example“

```
public class InstanceClient {

        private final Client client = ClientBuilder.newClient(new
            ClientConfig().register( LoggingFilter.class ) );

        public Response getInstances(String mediaTypeOut) {
                WebTarget webTarget = client.target(uri).path(
                    RESOURCE_PATH);
                Invocation.Builder invocationBuilder =  webTarget.
                    request(mediaTypeOut);
                return invocationBuilder.get();
        }
```

For each method was written at least one test. All tests are a part of the source code and you can find them on CD.

8.2 Manual tests with BonitaBPM

Integration tests were done only manually, as there is no way to manipulate with human activities in the BonitaBPM from outside.

Chapter 9

Summary

In this thesis I did a research about the current business needs in an area of a business process management. Based on this knowledge I identified problems and formally state them. After a problem analysis I proposed a solution in the form of Web Services (called BP Orchestration Web Services).

Web Services were designed according the REST architectural style with completely different process-centric approach in contrast to activity-centric approach and information-centric approach. The designed Web Services allow to change the running instance of the business process on the fly, allow to distribute business processes across different business process engines and allow to model a business processes with variable components. Such approach opens an opportunity to change underlaying business process engine without extremely high expenses (usually it is a main obstacle in a way to adopt new cutting edge technologies in business process management).

In order to support business processes with variable components I introduced the new notions of a hole in the business process. Actually it is a „place-holder“ for sub-process, where only boundary information is known (data in/out and events in/out). If a sub-process meets these boundary conditions then it can be assigned to the hole.

The template-pattern-instance structure was introduced to support generation of process instances from its abstract definitions. This structure underlies the set of resources in the system. After this set of exposed resources was identified I designed the URI namespace for Web Services and designed a JSON representation for each resource and at the end I applied the HTTP protocol on top of that.

Basic implementation of designed Web Services was done in Java with help of Jersey framework. The most important part of the implementation was to implement the connector to the target BPE (BonitaBPM) and integrate BonitaBPM with designed Web Services. There is no any „REST connector“ in Bonita BPM, that is why some integration Groovy scripts were written.

The main contribution of the approach taken is the ability to implement dynamic business processes executions and, in general, to improve scalability and reliability of the business process engines maintaining the executions or their parts in distributed environments.

Bibliography

[1] Nancy Alexopoulou, Mara Nikolaidou, Yannis Chamodrakas, and Drakoulis Martakos. Enabling on-the-fly business process composition through an event-based approach, 2008.

[2] R. Alsop. *Workflow Automation Integration Requires a Large Technology Toolkit and a Structured Approach. IEEE Computer Society*, 1994.

[3] M. Becker and M. Gille. *Workflow application architectures: classification and characteristics of workflow-based information systems. Workflow Handbook*, pages 39–50, 2002.

[4] Inc. (CPI) Computer Projects of Illinois. Puting soap to rest. whitepaper, 2015.

[5] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architecture.* PhD thesis, University of California, Irvine, 2000.

[6] C. Frye. *Move to Workflow Provokes Business Process Scrutiny. Software Magaziney*, Apr 1994.

[7] D. Georgakopoulos, M. Hornick, and A. Shelh. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and parallel Databases*, 3(2):119–153, 1995. ISSN 1573-7578.

[8] Object Management Group. Documents associated with business process model and notation (bpmn) version 2.0. omg, 2011.

[9] R. McCready. *There is more than one kind of Work-flow Software*, Nov 1992.

[10] M. Muehlen, J. V. Nickerson, and K. D Swenson. Developing web services choreography standards – the case of rest vs. soap. *Decision Support Systems*, 40(1):9–29, July 2005.

[11] Medina-Mora R., Winograd T., and Flores R. *Action Workflow as the Enterprise Integration Technology. IEEE Computer Society*, 16(2), Jun 1993.

[12] Allamaraju S. *RESTful Web Services Cookbook: Solutions for Improving Scalability and Simplicity.* O'Reilly Media, Inc, 2010. IBSN: 978-0-596-80168-7.

[13] Kumaran S., Liu R., Dhoolia P., Heath T., Nandi P., and Pinel F. A restful architecture for service-oriented business process execution. In *2008 IEEE 10th International Conference on e-Business Engineering*, pages 197–204, 2008. ISBN 978-0-7695-3395-7.

[14] P.H. Salus. *A Quarter-Century of Unix*. Addison-Wesley, 1994.

[15] Sanghyun Yoo, Yo han Roh, In-Chul Song, and Joo Hyuk Jeon. Rule-based dynamic business process modification and adaptation. *International Conference on Information Networking*, pages 1–5, 01 2008.

Appendices

List of Appendices

Appendix A

Obsah CD

- tex – contains source codes for latex
- doc – contains this PDF file
- implementation – contains java source codes including used libraries
- groovy – contains groovy scripts for integration with BonitaBPM
- processes – contains example business process models for BonitaBPM
- specification – contains raml specification for web services