



Development of a Cloud Platform for Business Process Administration, Modeling, and Execution

Master Thesis at Ulm University

Author:

Stefan Büringer
stefan.bueringer@uni-ulm.de

Reviewers:

Prof. Dr. Manfred Reichert
Prof. Dr. Peter Dadam

Advisor:

Dipl.-Inf. Jens Kolb

2014

Version July 24, 2014

© 2014 Stefan B ringer

Abstract

Current business process management systems (BPMS) are laid out for large enterprises with business process management (BPM) expertise. Hence, there is a lack of tailored BPMS for small and medium-sized enterprises (SMEs) targeting at users with hardly BPM expertise. Clavii BPM cloud is a compact solution for web-based business process administration, modeling, and execution. Therefore, Clavii BPM cloud offers features to easily manage and share process models, as well as features for collaborative process modeling and execution. Moreover, Clavii BPM cloud has a unique feature set, which includes process views to reduce process model complexity and an easily extendable object-oriented data model. It also provides unique capabilities for process visualization with different notations like business process model and notation (BPMN) and a newly developed Transit Map. Created process models can be executed directly in the cloud as part of the seamlessly integrated modeling and execution environment.

Contents

1. Introduction	1
1.1. Problem Statement	2
1.2. Contribution	3
1.3. Organization of the Thesis	5
2. Fundamentals	7
2.1. Business Process Management	7
2.1.1. Process Lifecycle	8
2.1.2. Activiti BPM Platform	9
2.1.3. Elements and Structure of a Process Model	11
2.2. Web Applications with Google Web Toolkit	13
2.2.1. Overview	14
2.2.2. Fundamental GWT Technologies	16
2.3. Summary	25
3. State-of-the-art Business Process Management Systems	27
3.1. Architecture	28
3.2. User Interface	30
3.3. Summary	35
4. Requirements	37
4.1. User Stories	38
4.2. General Requirements	40
4.3. Process Visualization Requirements	43

Contents

4.4. Modeling Requirements	44
4.5. Execution Requirements	47
4.6. Summary	47
5. Overview Clavii BPM cloud	51
5.1. Architecture	52
5.2. Data Model	57
5.3. Summary	60
6. User Interface of the Web Application	61
6.1. Site Map	62
6.2. Individual Pages of the Web Application	63
6.2.1. Login Page	63
6.2.2. Registration Page	63
6.2.3. GroupOverview Page	64
6.2.4. ProcessOverview Page	66
6.2.5. ModelView Page	67
6.2.6. Settings Page	73
6.2.7. UserProfile Page	74
6.3. Summary	74
7. Implementation Aspects of the Web Application	79
7.1. General Implementation Aspects	80
7.1.1. Navigator Component	80
7.1.2. Event Bus Component	82
7.1.3. Data Storage Component	85
7.1.4. Request Management	87
7.2. User Interface Implementation Aspects	88
7.2.1. Structure of GroupOverview and ProcessOverview Page	89
7.2.2. Details on Implementation of ModelView Page	94
7.2.3. Structure of the Sidebar Component	105
7.2.4. Details on Implementation of the ViewSettings Panel	108

7.2.5. Localization	110
7.3. Summary	112
8. Summary and Outlook	113
A. Layouting Examples	117

1

Introduction

Business process management (BPM) is a management approach to align business processes of an enterprise to the economic necessities of the daily business. The increasing focus on compliance has led to the necessity to take control of business processes [EK12]. This can be achieved by capturing the as-is business processes and defining the to-be business processes in explicit process models. This also leads to a continuous optimization of existing process models, which is a major benefit of BPM [Rud07].

BPM may contribute to reduced costs, enhanced efficiency, increased productivity, and minimized risks by increasing process-awareness in the enterprise. Especially since business processes can be only analyzed and optimized, if they are documented in process models [Zai97, KH07]. Process models can then also be executed within a business process management system (BPMS).

1. Introduction

Even though there are a lot of BPMS on the market [SWKN11], the majority of them are laid out for large enterprises and do not consider requirements of small and medium-sized enterprises (SMEs). In particular, this includes the price policy and the amount of specialized knowledge necessary to operate such a system. Furthermore, few BPMS try to keep the interactions with the system as simple as possible, for the benefit of good usability for non-technical end-users. The resulting market gap is the starting point of this thesis.

1.1. Problem Statement

A major obstacle for the usage of a BPMS in an enterprise are the investments necessary for the introduction of a BPMS [Ric11]. The expense factors are the costs for the purchasing and operation of the required hardware, which is a problem for SMEs, as well as expensive licenses for a BPMS [MR13]. These problems may be solved by a cloud-based BPMS. A cloud-based BPMS is a BPMS which can be operated by an external service provider in the cloud. Therefore, it can be used on demand and no additional internal resources are required. Figure 1.1 shows that the usage of cloud services is increasing in many areas, and, thus, that cloud-based platforms are a viable option for many enterprises. Especially, such a platform can be hosted internally and externally.

Introducing a cloud-based BPMS, at first, process models can be modeled, implemented, analyzed, and executed in such a cloud-based BPMS. The coverage of business processes integrated in a cloud-based BPMS can then be increased gradually. Finally, such a BPMS may be hosted internally, since the external storage of critical data is an important security issue for enterprises [SK11].

Another obstacle for the introduction of a BPMS is the required expertise to set up and operate a BPMS. For large enterprises this is no obstruction, since they usually rely on a bigger IT budget than SMEs. Therefore, it is essential that a BPMS is as easy as possible to set up and operate for a SME.

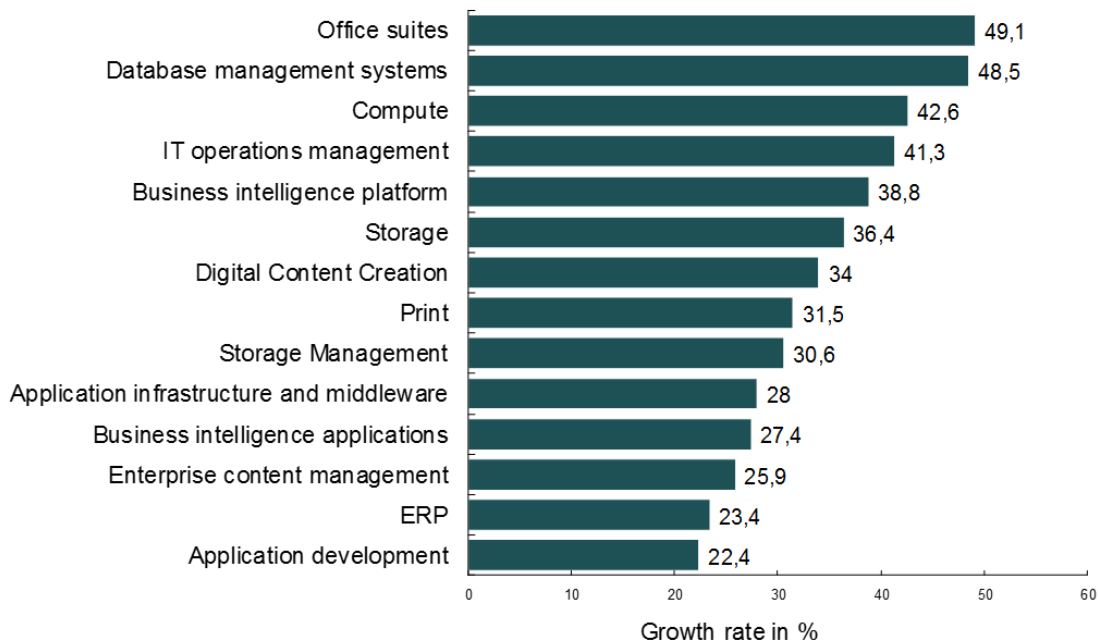


Figure 1.1.: Growth Rate of Cloud Services [Gar13]

Despite the big improvements BPM technology offers to an enterprise, there are also difficulties when a BPMS is established in an enterprise. A prerequisite for a successful BPM project is the necessary expertise for the used BPMS. Because SMEs can not rely on big IT budgets, the used BPMS should be as easy to understand as possible for users to reduce expertise-building costs. In particular, this includes an intuitive user interface and an understandable administration of process models.

The following section provides insights about the contribution of this thesis addressing this issues.

1.2. Contribution

The contribution of this thesis is the development of the web application for the Clavii BPM cloud (cf. Figure 1.2), based on the Google Web Toolkit (GWT). For the execution of process models, the Activiti BPM Platform is integrated. Additional information

1. Introduction

about server-side components and technologies of the Clavii BPM cloud is provided in [And14, Kam14]. Clavii BPM cloud allows the usage in multiple languages, currently German and English. Compatibility with all major browsers is facilitated by the usage of GWT [THET13].

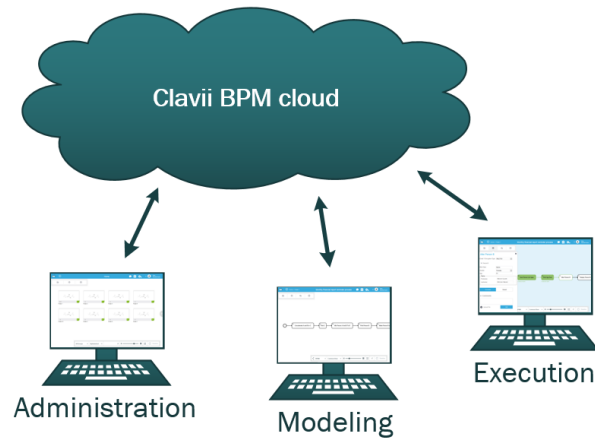


Figure 1.2.: Clavii BPM cloud

The main focus of the development of the web application is on a user-centered design, i.e., it should be as easy as possible to administrate, model, and execute process models for non-technical end-users. Clavii BPM cloud allows the creation of so-called *groups*, which can be easily shared between different users. Next, *process models* can be stored within such groups. Further, properties of groups and process models can be administrated in a sidebar, e.g., access rights and textual descriptions.

Two different process notations are provided, i.e., business process model and notation (BPMN) [OMG11] and a newly developed *Transit Map*, to assist users in understanding process models. The user interface provides capabilities for drag and drop modeling of the control and data flow of process models. Process models can be enriched with technical information, for the execution of automated tasks. Changes on process models are propagated to all users logged in, to support collaborative process modeling.

Without an explicit deployment process models can be directly started, which is enabled by a seamlessly integrated modeling and execution environment. Tasks in a process model can be executed at run-time by simply clicking on them. If necessary, users can provide input data by entering them in a sidebar, without losing the context of the

process. Thus, user always have an overview on the process model and its current execution state. Clavii BPM cloud also provides a process instance overview integrated in the sidebar to get a quick summary, which process models are being executed or which have been completed.

1.3. Organization of the Thesis

The thesis starts with a discussion of fundamentals regarding BPM and web applications with GWT (cf. Section 2). A comprehensive analysis of the capabilities of state-of-the-art BPMS on the basis of the IBM Business Process Manager (IBM BPM) and the Activiti BPM Platform (Activiti for short) is provided in Section 3. Based on the potential of the Activiti BPM Platform and GWT, and capabilities of state-of-the art BPMS, requirements are specified in Section 4.

The following sections illustrate the concrete implementation of Clavii BPM cloud (Clavii for short). Section 5 describes the high-level architecture of the Clavii BPM cloud. Section 6 gives an overview and details on the user interface. In Section 7 selected implementation aspects are singled-out and explained thoroughly. Finally, Section 8 summarizes the thesis and gives a brief outlook on possible extensions of the Clavii BPM cloud.

2

Fundamentals

This section introduces fundamentals for the design of a cloud-based BPMS. In particular, Section 2.1 describes the fundamentals of BPM, including elements and structure of a process model. Since this thesis relies on GWT, Section 2.2 presents the basic structure of a GWT application. Furthermore, some fundamental technologies of GWT are described in detail. Finally, Section 2.3 summarizes this section.

2.1. Business Process Management

In the following, fundamentals on BPM are introduced. At first, the understanding of the lifecycle of a process model is discussed. To implement this lifecycle, Clavii BPM cloud integrates the Activiti engine [Rad12]. Hence, it is essential to explain capabilities of Activiti. Activiti, and therefore Clavii BPM cloud uses process models based on the BPMN

2. Fundamentals

2.0 specification [OMG11]. BPMN 2.0 is a graphical notation for business processes, which has been standardized by the Business Process Management Initiative (BPMNI). Because Clavii BPM cloud has a few restrictions on process models, the supported elements of a process model and the block-structure of process models are discussed.

2.1.1. Process Lifecycle

The *process lifecycle* describes the evolution of a process model from its modeling, execution, monitoring, and optimization [Wes07, GT98]. It is essential for the design of a BPMS, to determine how different phases of the process lifecycle should be supported by a BPMS. Figure 2.1 visualizes the process lifecycle.

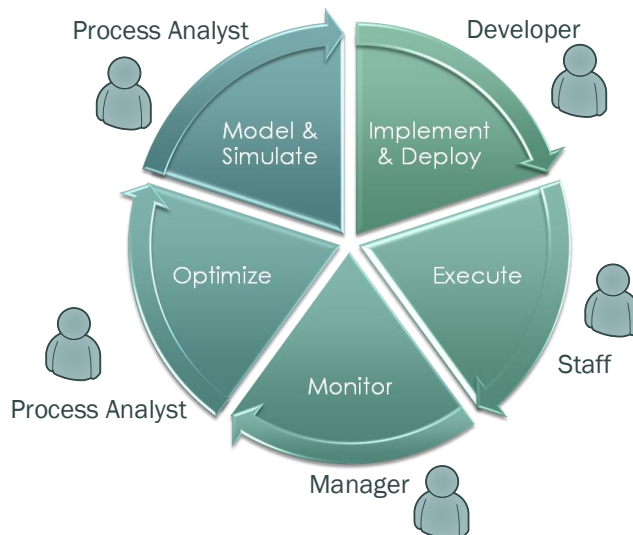


Figure 2.1.: The Process Lifecycle

Generally, modeling a business process is separated in two phases: functional and technical modeling. Whereby, functional modeling corresponds to the first phase of the lifecycle: *Model & Simulate*. In this phase, the business process is analyzed and modeled at a high abstraction level. This phase is typically conducted by a process analyst. However, it would be a substantial improvement if the process participants could model their own process models. Studies have shown when the process is modeled by

one of the process participants it results in better process models [KZWR14]. This is only possible if the used BPMS is beginner-friendly enough for inexperienced users.

Process models of the functional modeling phase are the basis for technical modeling. Thereby, a process model is enriched with implementation details by a developer. For example, users are assigned to tasks or server URLs are configured. When technical modeling is completed, which usually includes testing in a test environment, the process model is deployed on an execution environment.

After the process model is deployed, the users can execute the process model. For this, a publicly accessible internet or intranet platform is used. Such a platform usually provides work lists to users. Each work list assigns tasks to user. During the execution phase, so-called *Key Performance Indicators* (KPI) are calculated. An example for a KPI may be the average time for the completion of a task. KPIs can be used to create performance dashboards or to analyze a process model. Thus, bottlenecks and inefficiencies are made visible. Findings of the optimization phase are used for further process optimizations (cf. Figure 2.1).

The process lifecycle is repeated for continuous business process improvement. Changes in the process can be easily introduced in the optimization phase. As a result, it is possible to react quickly to evolving business demands.

2.1.2. Activiti BPM Platform

Activiti BPM Platform (Activiti for short) is selected in this thesis as starting point for extending to provide a cloud-based BPMS. To be more precise, the *Activiti engine*, without the additional environments, is integrated in Clavii BPM cloud. Hence, the focus of this section is on the functionality of the Activiti engine (cf. Figure 2.2).

Core component of the Activiti engine is the *ProcessEngine*, which is responsible for the execution of process models. Additionally to its main task, it provides access to several other components, so-called *services*. The Activiti services are explained in the following:

2. Fundamentals

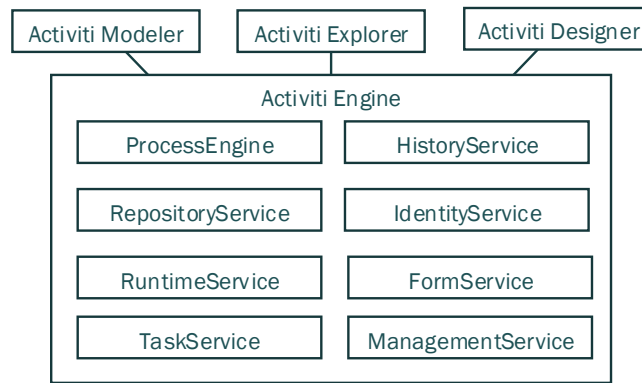


Figure 2.2.: Structure of Activiti Engine

RepositoryService. The RepositoryService offers methods for managing *deployments* of process models. A deployment contains *process definitions* and other resources. A process definition encapsulates the process model, which is represented by an XML file. Additional resources like a process model as an image can be added. These resources can be accessed through the RepositoryService. When a deployment is deployed, it is uploaded to the Activiti engine. Afterwards, contained process models are validated before they are stored in a database. From that point on, these process models are known by the process engine and can be started.

RuntimeService. The RuntimeService is responsible for starting and stopping a process instance, which is the representation of an executed process model. In addition, it provides functionality to query for running process instances. Furthermore, process variables of an instance, which contain the data of a process instance, can be retrieved and manipulated.

TaskService. The TaskService provides all methods necessary to execute so-called *user tasks*, which are tasks with user interaction (e.g., user forms). These tasks can be queried and filtered by attributes, for example, the assigned user groups or users. The most important methods are claiming and completion of tasks. Additionally, tasks can be reassigned to other users, for example, if a user is temporary not available.

HistoryService. Historical data is collected during process execution and can be accessed with the HistoryService. This data can be used for advanced monitoring services, like performance dashboards.

IdentityService. The IdentityService provides capabilities to create, update, delete, and query user groups and individual users. Therefore, it is possible to realize complex organization models.

FormService. The FormService provides support for user forms. These forms can be defined within the process definition.

ManagementService. The ManagementService provides methods to retrieve information about database tables and table meta data. For example, it is possible to retrieve the table name for a specific class, like the *HistoricProcessInstance*, which encapsulates the historic data of a process instance. Thus, custom SQL queries can be build.

2.1.3. Elements and Structure of a Process Model

Process models are typically visualized as directed graphs. In particular, the building blocks being *nodes* and *edges*. Nodes represent *tasks*, also called activities, or *gateways*. Gateways can have different control flow indicators and, thus, a wide range of process patterns is supported [RHAM06]. Edges between nodes represent the transition behavior during process execution, which is also called *control flow*. *Business objects*, which contain the process data, and *data flow* are built on top of the original process model. In the following, supported process elements and control flow constructs are discussed in detail. In this context, BPMN has been established as a de-facto process modeling language [OMG11]. Hence, we apply it in this thesis.

Figure 2.3 shows an example of a process model. We restrict ourselves on process models having exactly one *start* and exactly one *end event*. The process model in Figure 2.3 also contains a *user task* and a *service task*.

A user task can be further detailed as either a *check mark task* or a *user form*. In particular, the user task is a check mark task, when no business object is connected to it. Hence, the task is executed by just clicking on the node. In contrast, for a user form a

2. Fundamentals

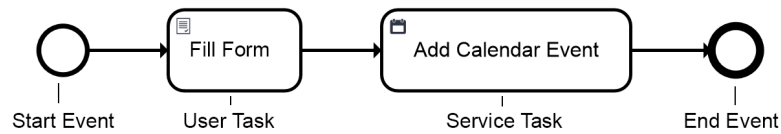


Figure 2.3.: Simple Process Model

form is generated, depending on its connected business objects, which has to be filled out by an assigned user.

In Figure 2.3, the service task is a calendar task. The task “Add Calendar Event” can be, for example, configured to add an event to a calendar. Service tasks are used to integrate external services, for example, OneDrive, Google Calendar, or Dropbox, in the process model.

A more complex process model, which contains supported gateways in Clavii BPM cloud, is illustrated in Figure 2.4. We restrict ourselves to a subset of all specified elements in BPMN [ZMR08]. The first gateway block, which is a combination of a join and a split gateway, in the process model is a *XOR gateway block*. After the gateway, the path, whose expression is evaluated to true, is followed. In Figure 2.4, task *A* is only executed if expression $Age < 9$ evaluates to true, task *B* is executed if $Age > 9$ evaluates to true, and none of them is executed if $Age == 9$. To be more precise, only one outgoing branch is selected at a time.

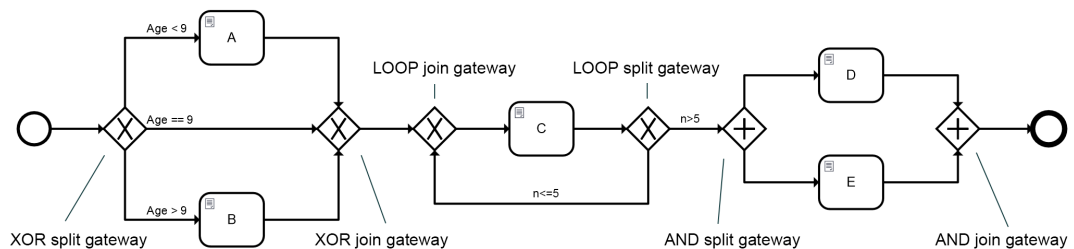


Figure 2.4.: A Process Model including Gateways

The next gateway block is a *LOOP gateway block*. Task *C* is executed as long as the expression on the edge between *LOOP join* and *LOOP split gateway* evaluates to true. Therefore, it is possible to build arbitrary loops. For example, a loop with exactly two

repetitions or a loop which is executed until the task inside the loop returns a specific result. The last gateway block is an *AND gateway block*. It expresses that both task *D* and task *E* are executed in parallel.

We restrict in this thesis to block-oriented process models [Rei00], to enforce well-structured process models [MRA10]. Block-oriented process models can be nested indefinitely (cf. Figure 2.5, ①) and disjunct gateway blocks are also allowed (cf. Figure 2.5, ②). However, it is forbidden to model overlapping gateway blocks (cf. Figure 2.5, ③).

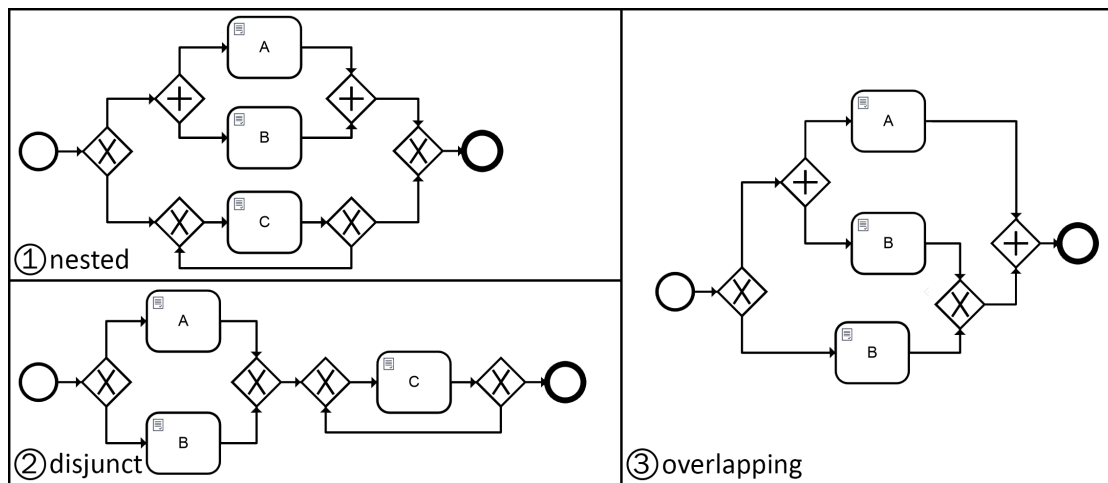


Figure 2.5.: Examples for Block-oriented Process Models

2.2. Web Applications with Google Web Toolkit

Google Web Toolkit (GWT) is a framework for creating web applications with a Java-based approach [Bur06, Dew07, THET13]. The basic concept behind GWT is that web applications can be written mostly using Java as programming language. Although, some additional CSS and HTML files are required, the major benefit of GWT is that Java developers can reuse their knowledge of Java and that no JavaScript expertise is required. Furthermore, developers can profit from Java support given by existing integration development environments (IDEs). Therefore, the productivity with GWT is higher than with plain JavaScript [Vaa13].

2. Fundamentals

In Section 2.2.1, an overview of GWT is given. Therefore, the structure of an example GWT application is broken down. In Section 2.2.2, the compilation process, which transforms the sources to a *.war* file deployable on an application server, is examined. Finally, several fundamental technologies of GWT are illustrated.

2.2.1. Overview

Figure 2.6 displays an example GWT application. The *src.main* folder is the source folder of the application, i.e., it contains all source files of the application. The *war* folder is generated by the compilation process. The content of the latter is then packaged into a *.war* file and deployed on an application server. Hence, the content of the *war* folder is the deployed structure of the GWT application.

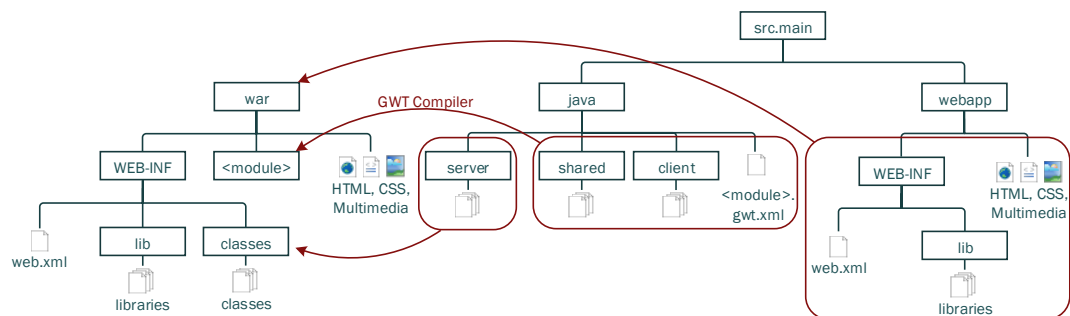


Figure 2.6.: Structure of a GWT Application

The *webapp* folder contains static resources like HTML or CSS files, libraries for Java classes on the server, and the deployment descriptor, the *web.xml* file. In the *web.xml* file, deployment parameters are specified, e.g., the welcome page, or servlets, which are explained later.

The *java* folder contains all Java packages of a GWT application and is divided into a *client*- and *server-side*. The client-side includes all files in the *client* and *shared* package. Whereas the *client* and *shared* packages, which are later compiled to JavaScript, can contain all common Java types, e.g., classes, interfaces, and enumerations. They can also contain UiBinder files and ClientBundles, which are both explained in Section 2.2.2. The *shared* package is also part of the server-side. In addition, the server-side

contains all classes of the *server* package. The *server* package usually consists of servlets, which are used to communicate with the client-side and the backend of the application.

The *.gwt.xml* file specifies necessary parameters for the compilation process (cf. Listing 2.1). Libraries can be imported by referencing them with *inherits*. Packages and their sub-packages, which should be compiled, are specified with *source*. Tag *entry-point* defines the class called at start up.

```
<module>
  <inherits name='com.google.gwt.user.User' />
  <inherits name="com.google.gwt.uibinder.UiBinder" />
  <source path='client' />
  <source path='shared' />
  <entry-point class='client.EntryPoint' />
</module>
```

Listing 2.1: *.gwt.xml* File

When compiling a GWT application, the content of the *webapp* folder is copied to the *war* folder. Next, the Java classes of the *server* package are copied to the *classes* folder. Furthermore, the content of the *client* and *shared* packages is compiled to JavaScript, as specified in the *.gwt.xml* file. JavaScript code is then copied to a folder within the *war* directory, which has the name of the so-called *module*. In particular, module is the collection of all compiled client files. Finally, the *.war* file is created.

The compilation process of the GWT compiler is illustrated in Figure 2.7. At first, the so-called *code generation* is started. The GWT compiler uses code generators based on the technologies used in the source code. For example, if the source code leverages the client-server communication technologies of GWT, the respective code is produced in this phase. This includes code to serialize, deserialize, and send the involved Java objects. The generators also optimize CSS or images for quicker loading times in the browser.

In the second phase, the final JavaScript is generated. Unused code is removed, to achieve smaller JavaScript files. Because each browser has its own quirks, GWT

2. Fundamentals

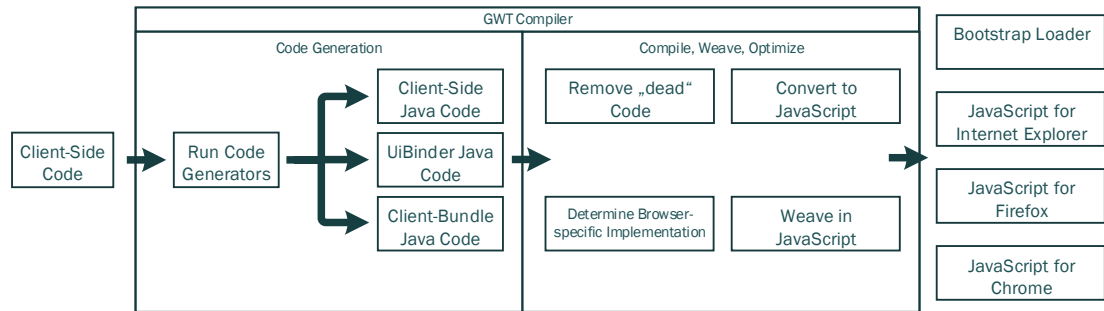


Figure 2.7.: GWT Compilation Process

provides different implementations for each browser. These implementations are used here to create customized Java files for each browser. Thereafter, the Java code is converted to JavaScript. This results in browser-specific JavaScript files and an additional *bootstrap loader*. The bootstrap loader is used to determine on start up which browser-specific file should be used.

2.2.2. Fundamental GWT Technologies

GWT is called a toolkit, because it consists of a collection of technologies, which can be leveraged to make the development of a web application as easy as possible. These technologies are the basis of all client-side Java code of Clavii BPM cloud. It is essential to gain a basic knowledge about these technologies to understand the implementation aspects singled out in Section 7. In the following, we introduce the fundamental technologies:

Remote Procedure Calls

Remote Procedure Calls (RPCs) enable the usage of *Asynchronous JavaScript and XML*, known as *AJAX* [Pau05], in GWT applications [Wal98]. RPCs are the main communication method between client- and server-side. The major difference to classic web applications is the asynchronous communication with the server-side in modern web applications with AJAX. Classic web applications use synchronous communication

by sending HTTP requests after user activity and waiting for the result. Hence, there are a lot of disruptions during the usage of a classic web application.

GWT encapsulates an AJAX engine to provide a convenient way to implement RPCs. The involved classes and interfaces are shown in Figure 2.8. To implement an RPC, a servlet on the server-side and two interfaces on the client-side have to be created. The GWT compiler creates a *RPCServiceProxy* class in the code generation phase of the compilation (cf. Figure 2.7). In the following, an example demonstrates how an RPC is implemented.

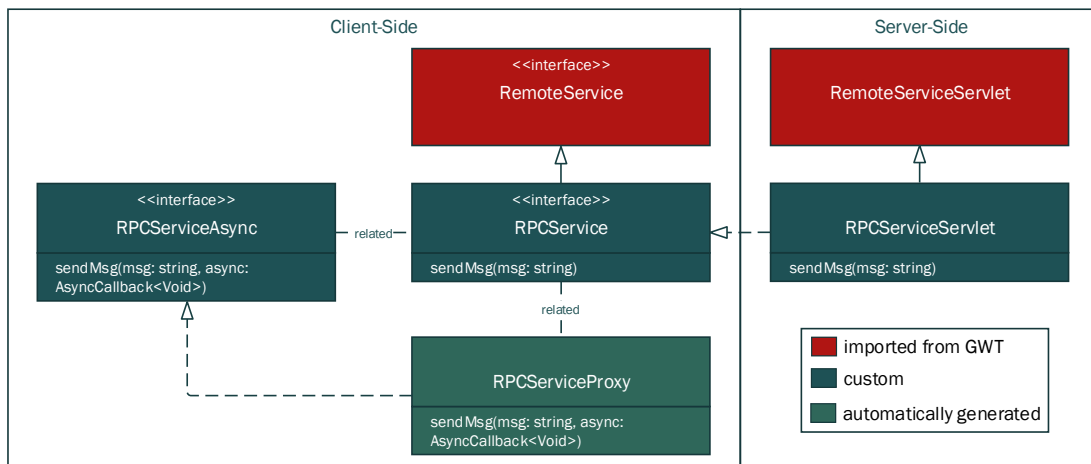


Figure 2.8.: Class Structure for RPC in GWT

First, the synchronous interface for the communication is created, to specify the methods of the RPC (cf. Listing 2.2).

```

package shared;
public interface RPCService extends RemoteService{
    public String sendMsg(String msg);
    class Impl {
        private static final RPCServiceAsync inst =
            (RPCServiceAsync) GWT.create(RPCService.class);
        public static RPCServiceAsync getInstance() { return inst; }
    }
}
    
```

Listing 2.2: RPCService Interface

2. Fundamentals

Another interface, based on *RPCService*, has to be created (cf. Listing 2.3). Each method in the synchronous interface, must have a corresponding method in the asynchronous interface. An additional parameter of the type *AsyncCallback* must be added, which enables the server to send a response to the client. The latter can then respond.

```
package shared;  
  
public interface RPCServiceAsync extends RemoteService{  
    public void sendMsg(String msg, AsyncCallback<String> async);  
}
```

Listing 2.3: RPCServiceAsync Interface

Listing 2.4 shows the servlet on the server-side. This servlet has to extend *RemoteServiceServlet*, in order to handle the communication with the client-side. The servlet also implements the previously created *RPCService* interface. When the servlet receives a method call from the client-side, the *sendMsg* method is called. With a *return* statement, an object can be send back to *AsyncCallback* on the client-side.

```
package shared;  
  
public class RPCServiceServlet extends RemoteServiceServlet implements RPCService{  
    public String sendMsg(String msg){  
        return msg;  
    }  
}
```

Listing 2.4: RPCServiceServlet Class

The application server has to be notified that the application contains servlets. Therefore, a servlet specification is provided in the deployment descriptor (cf. Listing 2.5).

```
<servlet>  
    <servlet-name>RPC Service</servlet-name>  
    <servlet-class>server.RPCServiceServlet</servlet-class>  
</servlet>  
<servlet-mapping>  
    <servlet-name>RPC Service</servlet-name>  
    <url-pattern>/RPCServiceServlet</url-pattern>  
</servlet-mapping>
```

Listing 2.5: Servlet Configuration in the web.xml File

Finally, the RPC can be executed from the client-side (cf. Listing 2.6). In this example, the static field from the *RPCService* is retrieved and method *sendMsg* is called. Thus, the RPC is sent through the *RPCServiceProxy* object, which has been automatically created. If the method call is successful, method *onSuccess* is called. If an exception is thrown on the server-side, method *onFailure* is called.

```
RPCServiceImpl.getInstance().sendMsg("Hello World",  
    new AsyncCallback<String>(){  
        public void onFailure(Throwable caught){  
        }  
  
        public void onSuccess(String result){  
        }  
    });
```

Listing 2.6: Client-Side RPC Call

RequestFactory

All data used in Clavii BPM cloud is encapsulated in entities, which are stored on the server-side. These entities can be edited through RPCs, which would require at least four RPCs for every entity: to create, to fetch, to update, and to delete it. Because every class that is used on the client-side needs to be located in the shared or client package, it is necessary to locate all entity classes in these packages. Alternatively, duplicate proxy classes on the client-side for every entity on the server-side may be created. However, a lot of effort is required to create and synchronize these entity classes as well as additional create converter, to convert the entity to its proxy class and vice versa.

GWT provides *RequestFactories* for this. We use *RequestFactory* to retrieve and modify entities from the client-side. (cf. Figure 2.9). Every entity of the server-side is represented by an *EntityProxy* on the client-side. *RequestFactory* is used to create a *RequestContext*. Every *EntityProxy* can be changed on the client-side, and changes are then sent to the *RequestFactoryServlet* on the server-side. The client-server communication is handled by GWT. To be precise, the GWT compiler automatically creates code for the communication between client and server.

2. Fundamentals

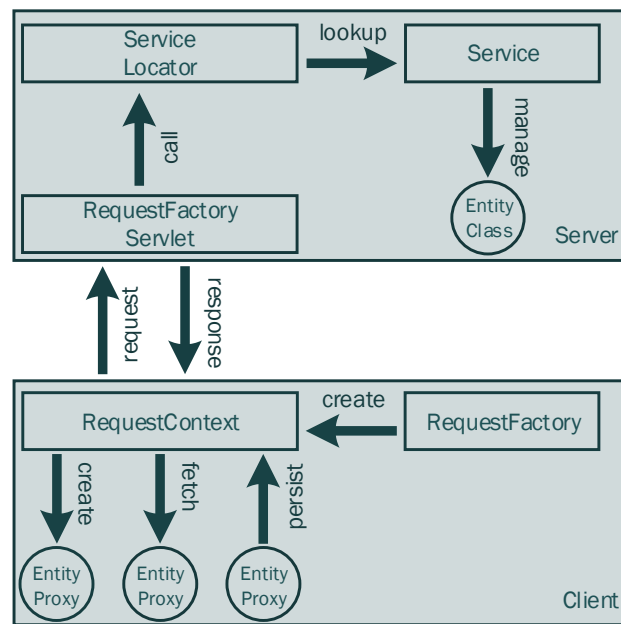


Figure 2.9.: RequestFactory

When the *RequestFactoryServlet* receives a request, it calls the *ServiceLocator* to retrieve the *Service* of the respective entity. The *Service* is then used to access the server-side entity. This mechanism has the capabilities to connect the client-side to arbitrary data storage on the server-side.

GWT Injection

GWT injection (GIN) facilitates the usage of the *inversion of control* pattern [Fow]. Inversion of control avoids that dependent components, like user interface components, create objects of general components, like components for centralized data storage. Instead, the general components are provided by the injection mechanism. GIN allows the developer to describe dependencies between components and inject components. Inversion of control increases modularity and extensibility of an application.

GIN is based on *Guice*, which is a Java dependency injection framework [Van08]. Since Guice uses Java reflection [FFI04], it can not be used on GWT client-side. Therefore, GIN has been created [THET13]. The basic principle of GIN is shown in Figure 2.10.

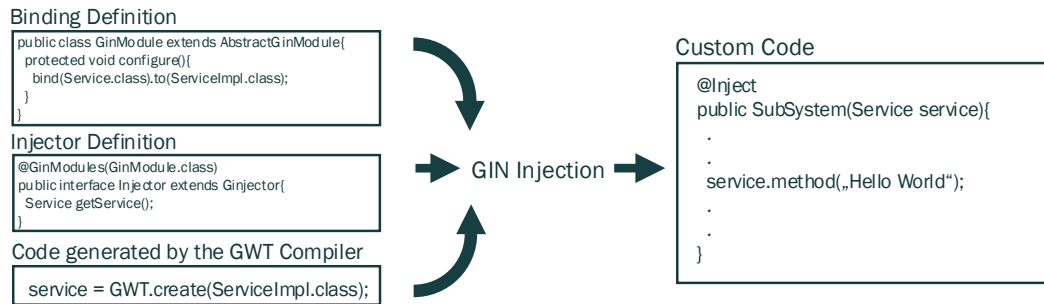


Figure 2.10.: GWT Injection

Binding definition specifies implementation classes for interfaces. Therefore, an implementation class can be exchanged by changing a binding definition. The GWT compiler generates code for the creation of dependencies. Injector definition provides access to the interfaces which have been connected to classes in binding definition. In custom code dependencies can be injected by the annotation *@Inject*. Hence, a developer only has to provide binding and injector definition and components are available everywhere without additional code.

GIN provides several types of injection: *on-demand*, *constructor*, *field*, and *method* injection (cf. Listing 2.7).

```

public class SubSystem{
    @Inject
    Service service;
    @Inject
    public SubSystem(Service service){
        GWT.create(Injector.class).getService();
    }
    @Inject
    public setService(Service service){
    }
}

```

Listing 2.7: Examples for GWT Injection

On-demand injection is implemented, for example, by creating an *Injector* object and calling method *getService*. *Gwt.create* is replaced when compiling with a concrete implementation. *On-demand* injection is useful when a component needs to be created

2. Fundamentals

at a specific time. *Constructor*, *field*, and *method* injection are all realized by the *@Inject* annotation. Therefore, the latter is placed at constructors, methods, and fields. The advantage of constructor injection is that the required component is already available in the constructor. The *field* and *method* injection is only executed after the constructor is completed.

ClientBundles

ClientBundles are used to bundle static resources, like images or CSS files. Typically, a web application requests each static resource individually. Since every request opens a new connection, loading times of the web application increase. Therefore, ClientBundles bundle images and CSS files.

Listing 2.8 displays the definition of such a ClientBundle. *ClientBundle* class has to be extended and an instance of *ClientBundle* object is stored in a static field, for centralized access to the bundle.

```
public interface StaticResources extends ClientBundle {
    StaticResources Impl =
        (StaticResources) GWT.create(StaticResources.class);
    @Source("smiley.png")
    ImageResource smiley();
    @Source("styles.css")
    BundledCssResource css();
    public interface BundledCssResource extends CssResource{
        String redbackground();
        String bluetext();
    }
}
```

Listing 2.8: Creation of a ClientBundle

Listing 2.9 shows the corresponding CSS file to Listing 2.8. Methods, defined in the *BundledCssResource* interface, refer to classes in the CSS file. It is also possible to define constants and to call static methods, which deliver property values for CSS values. The *Theme* class in Listing 2.9 contains the static method *getMargin* called in the CSS file. Hence, a central Java class can be created, which contains all constants used

in CSS files. Therefore, constants can be defined at one place and used throughout the entire application. This pattern is implemented in Clavii BPM cloud. The following section discusses how the example ClientBundle created in Listing 2.8 can be accessed in a UiBinder file.

```
@def COLOR red;
@eval MARGIN client.Theme.getMargin();
.redbackground{
  background-color: COLOR;
}
.bluetext{
  color: blue;
  margin-left: MARGIN;
}

public class Theme{
  private static int margin = 10;
  public static String getMargin(){
    return margin + "px";
  }
}
```

Listing 2.9: CSS File and CSS Theme

UiBinder

UiBinder enables developers to create a user interface in a declarative manner, which is leveraged for all user interface components in Clavii BPM cloud. A user interface defined by UiBinder consists of two files: a Java class and a corresponding *.ui.xml* file. Both are compiled to JavaScript by the GWT compiler.

An example UiBinder file is shown in Listing 2.10. At first, the namespaces are declared. The first namespace imports common *UiBinder* tags. The second namespace imports a package with a set of predefined components. It is also possible to import packages with custom components. Next, the *StaticResources* ClientBundle is imported by the *ui:with* tag. Thereafter, first components are declared. During compilation, *HTMLPanel* element is replaced by a *div* element, with the CSS class specified in the ClientBundle. *Label* element is transformed to a *div* element with the text content *HelloWorld* and

2. Fundamentals

the *bluetext* CSS class. The *Image* element is transformed to a HTML *image* element, which shows the image linked of the *StaticResources* ClientBundle.

```
<ui:UiBinder xmlns:ui='urn:ui:com.google.gwt.uibinder'                xmlns:g='
    urn:import:com.google.gwt.user.client.ui'>
  <ui:with field="res" type="client.StaticResources" />
  <g:HTMLPanel addStyleNames="{res.css.redbackground}" />
    <g:Label ui:field="Text" text="Hello World"
      addStyleNames="{res.css.bluetext}" />
    <g:Image ui:field="Image" resource="{res.smiley}" />
  </g:HTMLPanel>
</ui:UiBinder>
```

Listing 2.10: UiBinder File HelloWorld.ui.xml

To create dynamic behavior, a Java class is attached to the *.ui.xml* file (cf. Listing 2.11). In the constructor, the *initWidget* method is called, to connect the fields annotated with *@UiField* to their corresponding user interface element, declared in the *.ui.xml* file. Handlers for user interface elements can be declared with the annotation *@UiHandler*. In this example, if the *Image* element is clicked, the *Text* element is modified by calling the *setText* method on the *Text* field.

```
public class HelloWorldPanel extends Composite{
  @UiTemplate("HelloWorld.ui.xml")
  interface UIBinder extends UiBinder<Widget, HelloWorldPanel> {}
  private static UIBinder helloWorldUiBinder =
    GWT.create(UIBinder.class);
  @UiField
  Label text;
  @UiField
  Image image;
  public HelloWorldPanel(){
    initWidget(helloWorldUiBinder.createAndBindUi(this));
  }
  @UiHandler("image")
  void onClick(ClickEvent blurEvent) {
    text.setText("Bye World");
  }
}
```

Listing 2.11: Java Class for UiBinder

2.3. Summary

This section discusses the fundamentals of BPM and web applications with GWT. The process lifecycle is explained to give insights in the evolution of a process model and which capabilities a BPMS must have to support the entire process lifecycle. In the following, the functionality of the Activiti engine, which is integrated in Clavii BPM cloud, is explained. Thus, it is illustrated on which functionality of the Activiti engine Clavii BPM cloud can build to implement the process lifecycle, which is explained later in detail in Section 5.1. Subsequently, the elements and structure of the used process models are specified, including the restriction to block-oriented process models.

In the following, web applications with GWT are discussed. First, an overview on the structure is given on the basis of an example application. Next, fundamental GWT technologies, which lay the groundwork for Clavii BPM cloud, are defined. The application of the fundamental technologies in the implementation of Clavii BPM cloud is explained in detail in Section 7. RPC and RequestFactory are used in Clavii BPM cloud to ease communication between client- and server-side (cf. Section 7.1). Furthermore, modularization is facilitated by dependency injection with GIN. ClientBundles and UiBinder make it possible to create the entire user interface in a declarative manner, which is leveraged in all user interface components in Clavii BPM cloud (cf. Section 7.2).

3

State-of-the-art Business Process Management Systems

In this section, state-of-the-art BPMS are analyzed to gain the necessary knowledge for implementing a cloud-based BPMS for SMEs. To gain insights into a broad bandwidth of BPMS, IBM Business Process Manager (IBM BPM) and Activiti BPM Platform are examined. IBM BPM is a large-scale commercial BPMS from IBM, which is laid out for large enterprises. In contrast, Activiti BPM Platform is a light-weight open-source BPMS and, thus, provides insights in the architecture and the user interface of a small-scale BPMS. Activiti BPM Platform is distributed under the Apache license and can therefore be integrated free of charge in another system, which enables the integration of the Activiti engine, which is part of Activiti BPM Platform, in Clavii BPM cloud.

The core components of a BPMS typically are: *modeling environment*, *execution environment*, and *process repository* [Hol95]. Individual components may have different

3. State-of-the-art Business Process Management Systems

scopes. A process repository, for example, can be just a simple database, which stores process definitions, or it can be a complex repository with functionality for versioning of process models and access control to these process models.

The architectures of both BPMS are discussed in Section 3.1, to get insights how the architecture of a BPMS should be designed. Finally, Section 3.2 analyzes the user interface of IBM Business Process Manager and Activiti BPM Platform, to discover best practices and potential problems of an user interface for a BPMS. Section 2.3 summarizes this section.

3.1. Architecture

The architecture of IBM BPM is illustrated in Figure 3.1 [AAG⁺13]. The central environment of IBM BPM is the *Process Center*. The Process Center stores all resources, like process models and related artifacts. Process modeling and deployment is conducted in the *Process Designer* and *Integration Designer*. They are both desktop applications and have to be downloaded to a local computer and connected to the Process Center. Process models can be administrated through the *Process Center Console*.

Process execution on the backend takes place either directly in the Process Center or in an external execution environment. Therefore, external execution environments can be managed in the Process Center Console. Per default, process models are deployed to the Process Center. Subsequently, they can be deployed to external execution environments if necessary.

Process Portal and *Process Admin Console* are used as frontends for both the Process Center and the execution environments. Next, Process Portal provides access to the execution of process models for end-users. The Process Admin Console has administrative capabilities, i.e., process models can be started or stopped and process data of executed process models may be manipulated.

Individual components of the IBM BPM architecture, also called *conceptual nodes*, can be operated on different physical machines. This allows for a high redundancy and high

3.1. Architecture

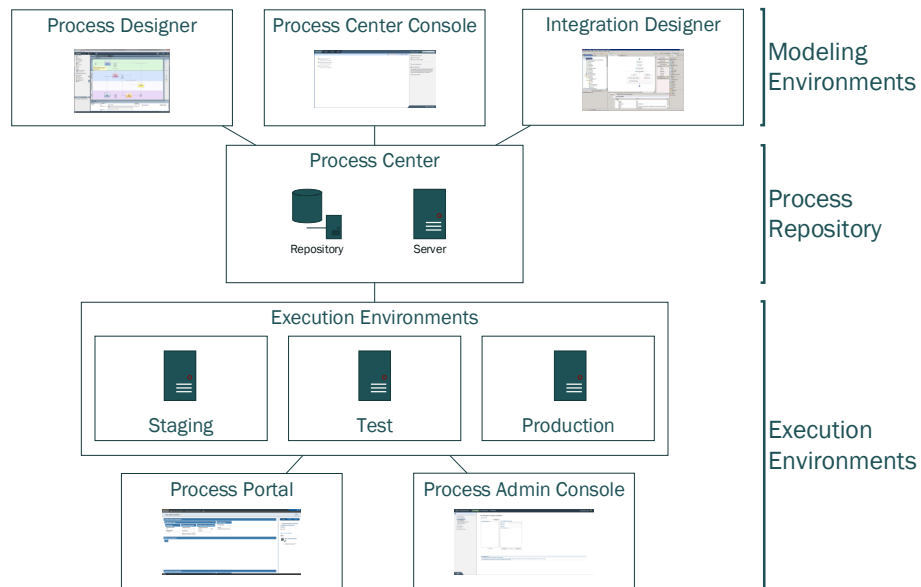


Figure 3.1.: IBM BPM Architecture

availability, but requires highly specialized knowledge for setting up and operating a production system.

In contrast to IBM BPM, Activiti has a simple architecture (cf. Figure 3.2) [Act14]. The core component of Activiti is the *Activiti engine*. The Activiti engine is responsible for storing and executing process models (cf. Section 2.1.2). The latter can be accessed by the web-based *Activiti Explorer*, which allows the user to execute and administrate process models. Furthermore, Activiti provides two modeling environments. The *Activiti Modeler* is a simple web-based modeling environment integrated in the Activiti Explorer. *Activiti Designer* is an eclipse-based environment for modeling BPMN 2.0 process models. These process models can then be deployed to the Activiti engine to execute.

Due to the simple architecture, Activiti is easy to understand and use. Hence, Activiti requires less expertise than IBM BPM to operate. This simple architecture facilitates the integration of Activiti in another system. That is why the Activiti engine is used as an integrated process engine in Clavii BPM cloud. The outcome of the discussion of both architectures is that the architecture of a cloud-based BPMS should be as simple as possible to facilitate the usage in SMEs.

3. State-of-the-art Business Process Management Systems

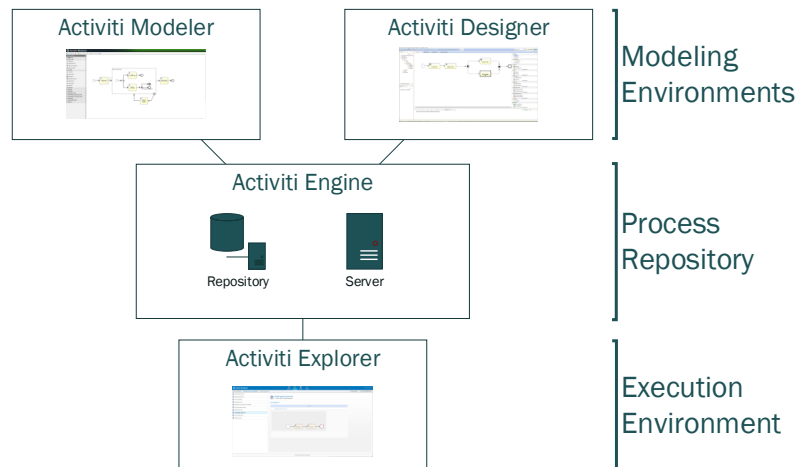


Figure 3.2.: Activiti Architecture

3.2. User Interface

After discussing architecture of BPMS, this section discusses the user interface of IBM BPM and Activiti BPM Platform. First, the user interface of IBM BPM is discussed. Figure 3.3 shows the start page of the Process Designer. On this page *Business Process Applications* (BPA) can be administrated, i.e., they can be created, removed, exported, and imported. A BPA is a collection of *Business Process Definitions* (BPD), which can be seen as process models. It is possible to configure the user access to the BPAs, imported libraries, so-called *Toolkits*, and associated execution environments.

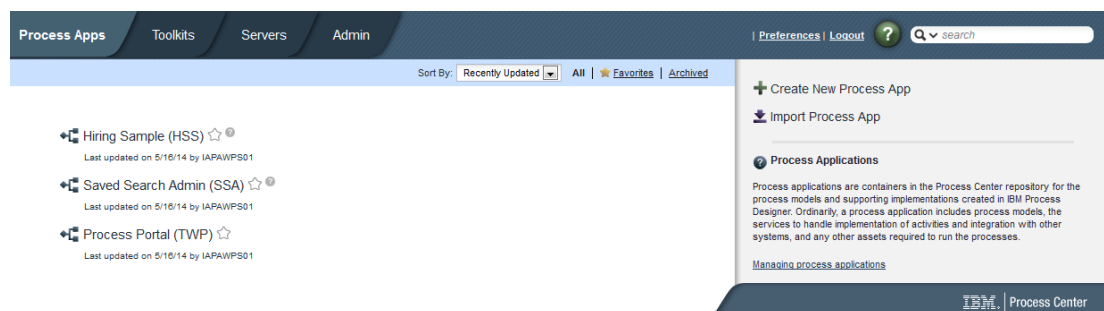


Figure 3.3.: Process Designer - Start Page

The modeling view (cf. Figure 3.4) is opened by accessing a BPA. In the modeling view new BPDs can be created and existing BPDs edited. Process elements can be

3.2. User Interface

dragged onto the process model from the palette on the right side. Basic elements are: activities, gateways, start events, end events, and intermediate events. An activity can be implemented by a user task, a system task, a script task, or a decision task. Whereby a user task consists of a sub flow, which specifies the user interaction. A system task can be, for example, a Java implementation or a web service call. The script task consists of JavaScript, which allows for modifying process variables. Decision tasks integrate rule services with the process model.

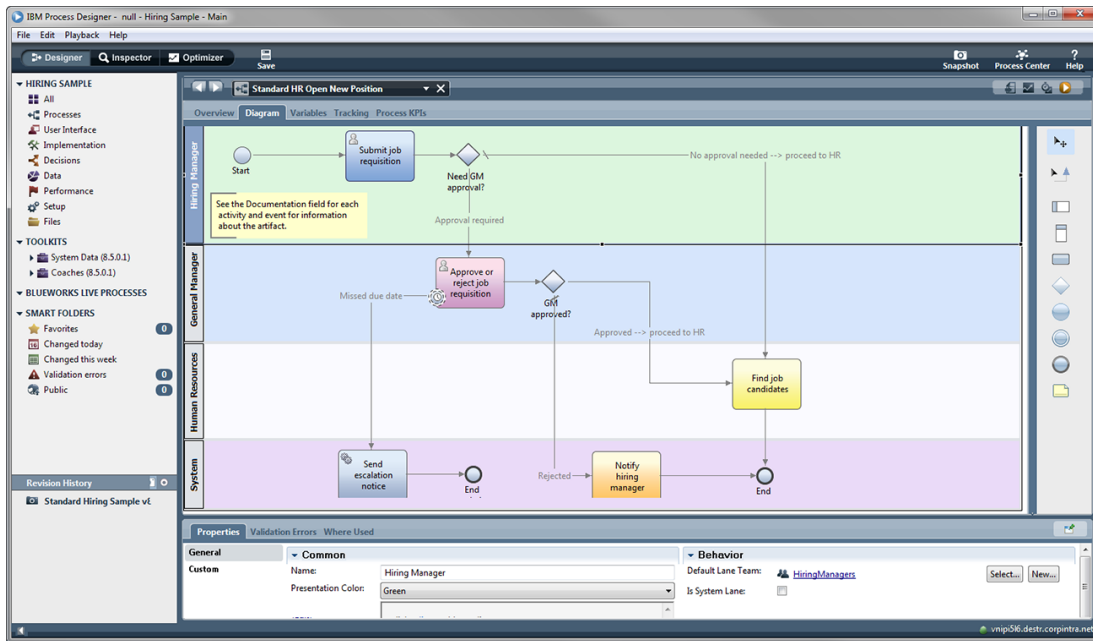


Figure 3.4.: Process Designer - Modeling View

The control flow of a process model is defined through edges between the process elements, which can be modeled using drag and drop. Process variables are created in an additional variables tab and have no graphical representation in the process model. Process variables can be connected to activities by specifying their name as an in- or output parameter.

Figure 3.5 shows how a sub flow of a user task is modeled. Individual user forms are represented by so-called *coaches*, which are represented by yellow rectangles with a user icon. In this example, at first, the *Create Requisition* coach is shown to the user. Depending on the output of the *Create Requisition* coach, the user is forwarded either to

3. State-of-the-art Business Process Management Systems

Specify Existing Position coach or to *Confirm Position Details* coach. This kind of user interaction modeling enables complex interaction scenarios.

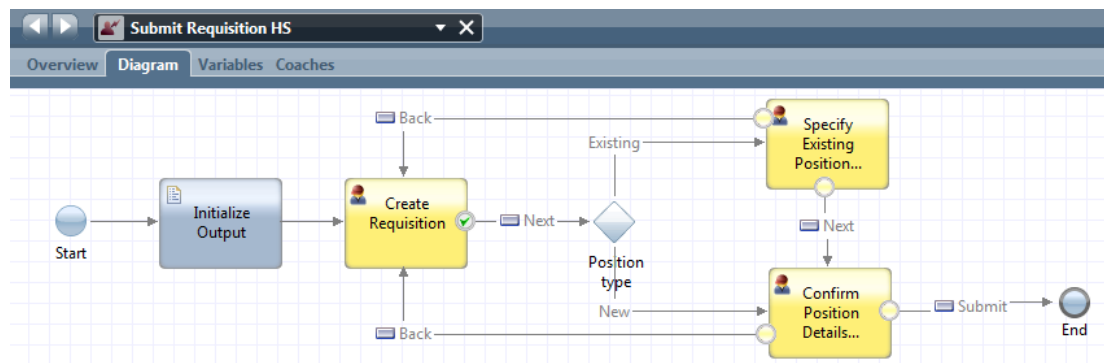


Figure 3.5.: Modeling a User Task in the Process Designer

After the process model is deployed, it can be executed in the Process Center. The user can then open the Process Portal and execute the process model. Figure 3.6 shows how the user task, modeled in Figure 3.5, is executed in the Process Portal. A panel on the right side displays additional information. This includes details, like the name of the current task, the due date of the task, and a graphical representation of the process model, the so-called process diagram. The stream tab shows when and who executed previous tasks. The experts tab provides capabilities to contact experienced users, who can help with the execution of the current task.

Requisition data			Position data	
Requester	Requested position	Position date and location	Position type	
Requisition number: 1141	Employment type: [dropdown]	Planned starting date: 5/16/2014	[dropdown]	
Hiring manager: Tom Miller	Department: [dropdown]	Location: [dropdown]	Job title: Head of Product Development	
	Number of employees required: 1			

Make your decision

Next

Task: Submit requisition

Created: May 16, 2014 8:38 AM
Due: May 16, 2014 9:38 AM

Figure 3.6.: Process Execution in the Process Portal

3.2. User Interface

In contrast to IBM BPM, Activiti provides only a rudimentary user interface. The process models can be modeled in the Eclipse-based Activiti Designer (cf. Figure 3.7). Process elements and edges can be created, by dragging them onto the process model from the palette on the right side. The palette contains all process elements and edges specified in the BPMN 2.0 specification [OMG11]. The process elements can be edited, by selecting them and changing their properties in the properties tab on the bottom. The process models are stored in *.bpmn20.xml* files according to the BPMN 2.0 specification.

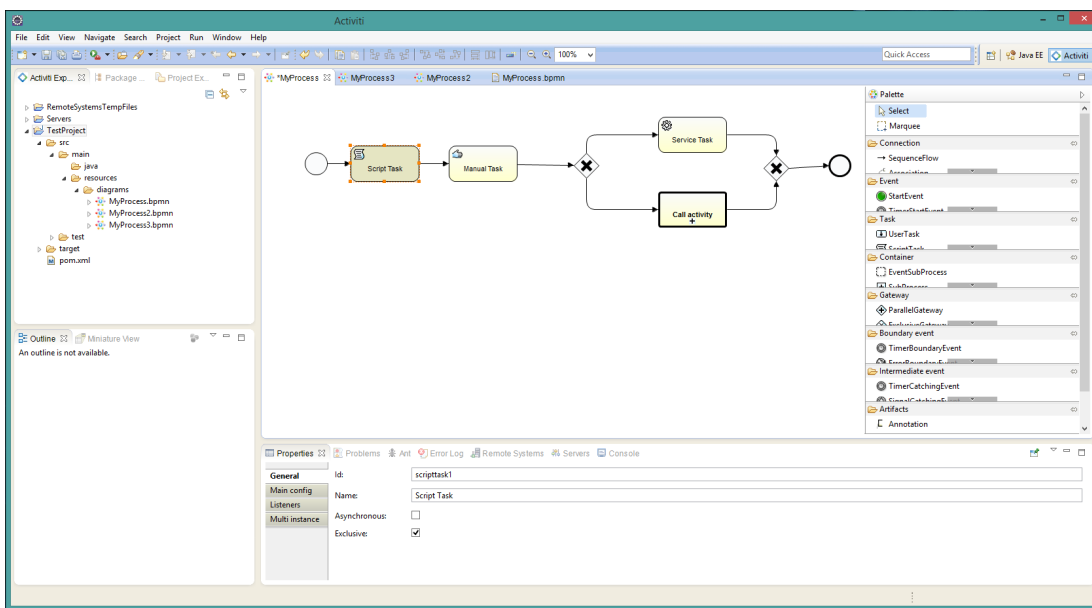


Figure 3.7.: Process Modeling with the Activiti Designer

The resulting *.bpmn20.xml* files can be deployed to the Activiti engine and are then available in the Activiti Explorer (cf. Figure 3.8). The Activiti Explorer is a web-based user interface, similar to the Process Portal of IBM BPM. An overview on all deployed process models can be found in the processes tab. The tasks tab contains a work list, which lists all tasks currently assigned to the logged in user. The user can execute the tasks by clicking on them in the work list. The task is then opened in the center and can be completed.

3. State-of-the-art Business Process Management Systems

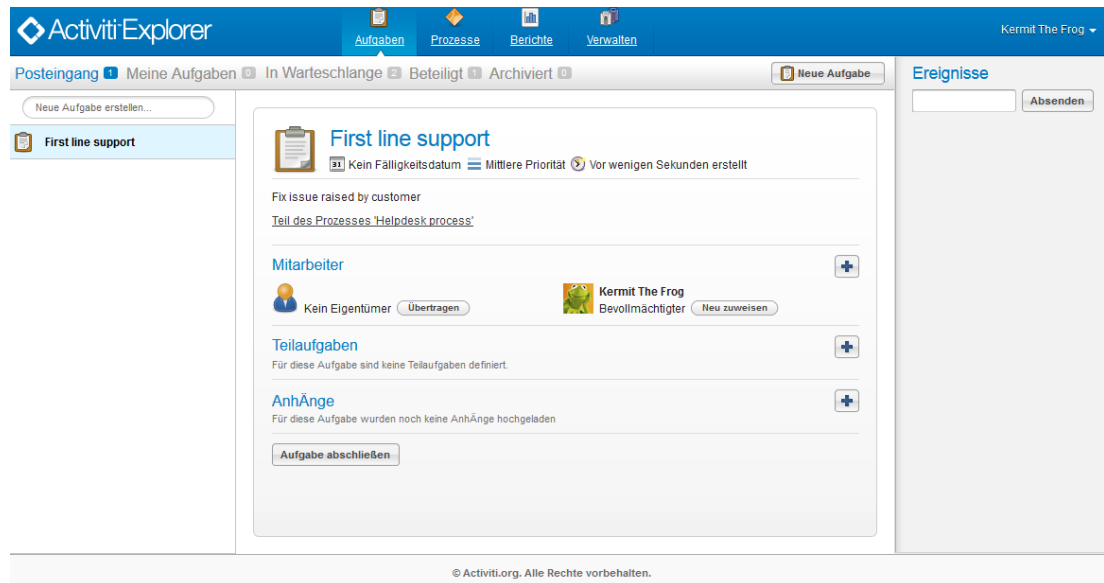


Figure 3.8.: Process Execution with the Activiti Explorer

The user interfaces of IBM BPM and Activiti BPM Platform rely both on desktop applications for process modeling. Therefore, they both require additional effort before the user can create process models. This effort can be reduced by providing one comprehensive web application with an integrated modeling environment. It is also useful to integrate the execution of process models in the web application, to avoid the necessity of process model deployment.

In both BPMS, the process elements are modeled by dragging them onto the process model. This concept has proven to be effective and should be used in a BPMS. However, the data flow can not be explicitly modeled. The data flow has to be modeled by setting properties on the process elements. It is desirable that data flow can be explicitly modeled and displayed. Thus, it is considerably easier to understand the data flow of a process model, especially for inexperienced users.

Both BPMS provide a wide range of process elements which can be used in a process model. In Activiti, it is even possible to model all elements defined in the BPMN specification. For inexperienced users it would be better to provide a restricted set of process elements in favor of better usability [ZMR08].

3.3. Summary

This section analyzes IBM BPM and Activiti regarding their architecture and user interface. The architecture of IBM BPM turned out to be relatively complex. Thus, IBM BPM requires a lot of expertise to set up and operate. Activiti has a simple architecture which also enables the integration of the Activiti engine in another BPMS. A cloud-based BPMS for SMEs should have a architecture which is as simple as possible, to reduce the effort and required expertise for set up and operation.

In the following, the user interfaces of IBM BPM and Activiti are analyzed. The modeling environments of both BPMS are desktop applications. To facilitate the roll out in an enterprise, these desktop applications could be replaced by a cloud-based BPMS with an integrated modeling environment. The process modeling approach of both BPMS is to drag new process nodes onto the process model, which is an effective concept that should be taken over. However, both BPMS lack the capabilities for graphical modeling of data flow, which could be a major improvement towards intuitive process modeling. Finally, it would be sufficient to provide only a restricted set of most used process elements, to facilitate the process modeling for inexperienced users.

The findings of this section are used to specify the requirements for a cloud-based BPMS for SMEs in the next section.

4

Requirements

According to findings of the previous section, major problems of existing BPMS can be identified. Especially for small and medium-sized enterprises, an introduction of a BPMS poses a substantial challenge. The greatest obstacles are:

- High expertise necessary to setup and operate a BPMS
- Modeling environment not tailored to inexperienced users
- Separation between process modeling and execution

Addressing these problems functions of a complex BPMS should be unified in one web application. This facilitates easy setup and operation of the BPMS through an easy deployment of the web application on an application server.

First, user stories are developed (cf. Section 4.1), to define the typical usage of a BPMS in a SME. In the following, requirements are formulated to achieve a clear understanding

4. Requirements

of the needs, to realize these stories, and other basic demands on a BPMS based on the discussion in Section 3. Requirements are separated into the following groups: *general requirements* (cf. Section 4.2), *process visualization requirements* (cf. Section 4.3), *modeling requirements* (cf. Section 4.4), and *execution requirements* (cf. Section 4.5). Section 4.6 concludes with a summary.

4.1. User Stories

The development of user stories is suitable to obtain clarification on what a BPMS should achieve [Coh04]. Therefore, stories based on the daily work with a BPMS in a SME are created (cf. Table 4.1).

Table 4.1.: User Stories

Story	Description
1	An enterprise starts a project to capture their process models. All employees, many of them without any modeling experience, should create process models for business processes of their everyday working life.
2	A complex process model has been modeled and refined by several users. Participating users are only involved in parts of it. Therefore, each respective user has its own perspective, which includes only relevant process fragments.
3	A set of process models are shared within an enterprise. This includes, for example, further education applications, vacation forms, and travel expenses accounting. All employees can file these applications and human resources department can process them.
4	A new plugin is developed to integrate an existing enterprise information system (EIS). It is imported to the BPMS and all users can use the plugin in their process models.

User Story 1 describes how a BPM project can be started in a SME. All employees create process models, which describe their view of the business processes in the enterprise. By combining these process models, process models of the real business processes, which include the perspectives of all participating employees, can be derived. This process models are more detailed and, thus, better than process models created by an outside observer [KZWR14]. To enable all employees of an enterprise, including

employees inexperienced with BPM, to create process models, the BPMS should be as intuitive and easy to use as possible.

User Story 2 shows a scenario, where a complex process model has been first modeled collaboratively and then is shared between users. For example, at first, a process model is shared within a department. Every participating user models his steps in the process model in detail. Thus a complex process model is created. Because not every user has to know every detail of the process model, custom perspectives can be created. These perspectives only show the fragments of the process model relevant to the respective user. Thus, a complex process model can be modeled and executed and every user still has an overview on his relevant parts of the process model.

In User Story 3, it is illustrated how process models, which are the same for most employees, can be leveraged for a SME. This process models are created once for an enterprise and then shared to provide uniform business processes to all users. These process models accelerate the execution of business processes in an enterprise. For example, the submission of a vacation form with a BPMS can be optimized by storing previously submitted forms and reuse them with changed values regarding the time period of the vacation. Subsequently, the vacation forms can be processed in a uniform way by the human resources department. As a result, uniform process models facilitate the reduction of unnecessary bureaucracy or at least can reduce the costs of necessary bureaucracy [IGRR09].

User Story 4 shows how existing EISs and other external systems can be integrated in a BPMS. This is necessary because business processes in an enterprise usually involve EIS [PCBV10]. Because a BPMS typically does not provide tasks types to access a specific EIS, a plugin for the BPMS has been developed. For example, this plugin provides methods to retrieve stock levels from an enterprise resource planning (ERP) system. The plugin can be uploaded to the BPMS and, thus, provided to all users, which can then integrate the ERP system in their process models.

The user stories developed in this section are the basis for the requirements specified in the following.

4.2. General Requirements

Following general requirements are identified based on the analysis in Section 3.1 and basic functionality of state-of-the-art BPMS (cf. Section 3). The architecture of IBM BPM shows how complicated it can be to set up and operate a BPMS. This complexity originates from the separation in several environments, which all have to be set up and operated individually. To reduce the effort to set up and operate a BPMS for SMEs, a BPMS should be a comprehensive platform (cf. Requirement REQ-1). Hence, all the required functionality should be bundled in one web application. Furthermore, it should be easy to deploy, i.e., it should consist of only one *.war* file and should only require little configuration effort.

Requirement REQ-1 (Comprehensive Cloud Platform). *A BPMS should be a comprehensive cloud platform comprising all functionality of the process lifecycle in one platform.*

Just like in any other state-of-the-art BPMS, multi-user operations should be supported (cf. Requirement REQ-2) [IRRG09]. Therefore, such a BPMS should be aware which users are logged in at any time. A major point of emphasis should be that actions from one user are propagated to all other logged in users. In particular, changes to process models as well as their execution. If the modeling actions would not be propagated, it would not be possible to collaboratively create a process model. The propagation of process model execution is mandatory because otherwise a task could be completed twice at the same time from different users.

Requirement REQ-2 (Multi-User Operation). *Multiple users should be able to use a BPMS at the same time. Results of actions should be propagated to all logged in users.*

For a BPMS with a large target group, it is essential that several languages are supported (cf. Requirement REQ-3). To be precise, English and German should be supported out of the box. Further, it should be possible to extend such a BPMS with additional languages, by adding new language files. This facilitates the translation of a BPMS, because no technical knowledge is necessary to translate the user interface.

Requirement REQ-3 (Localization). *A BPMS should support a wide range of languages.*

For the usage in an enterprise it is fundamental that the existing organizational model can be integrated in a BPMS (cf. Requirement REQ-4). If it can not be integrated, it has to be stored in duplicate, which results in additional effort for synchronization of these organizational models. Therefore, a BPMS should enable authentication via an external LDAP¹ directory. A BPMS should also support an organizational model with organizational units, roles, and users, to be able to implement realistic organizational structures of an enterprise.

Requirement REQ-4 (Organization Model). *It should be possible to use an existing organizational model in a BPMS.*

To make the user experience as comfortable as possible, a BPMS should, like any other state-of-the-art website, support sessions and bookmarks (cf. Requirement REQ-5), i.e., the current state of the application is represented in the uniform resource locator (URL). This could be used, for example, to represent a specific open process model in the page URL. This URL can be bookmarked in a browser and re-opened or shared to anybody allowed to access. A logged in user should be identified by setting a cookie in his browser, to avoid the necessity to log in every time the BPMS is opened.

Requirement REQ-5 (Support for Bookmarks and Sessions). *A BPMS should enable bookmarks to share URLs between users. It should also support sessions to hold the current application state.*

To provide the prerequisites for sharing process models as mentioned in User Story 3, it should be possible to group process models (cf. Requirement REQ-6). Without grouping process models, it is not possible to share process models in specific categories throughout an enterprise. The grouping also enables users to organize the process models and maintain an overview on them.

Requirement REQ-6 (Shared and Private Groups). *A BPMS should offer shared and private groups, for process grouping.*

¹LDAP is a protocol for accessing directory information services. Additional information can be found in [HSG03].

4. Requirements

Another prerequisite to share process models (cf. User Story 3) throughout an enterprise is an access control system. An access control system manages access rights for users, because not every user should have access to every group and process model (cf. Requirement REQ-7). Considering that, it should be possible restrict the access of users, to view, edit, and execute groups and process models. These access rights should be assignable to all organizational entities, which are organizational units, roles, and users. If access rights could be only assigned to users, it would be necessary to assign every user, which should have an access right to a group or process model, individually.

Requirement REQ-7 (Access Control). *An access control system should be available for groups and process models.*

A BPMS should facilitate the handling of group and process model attributes, which are basic functions of every BPMS (cf. Requirement REQ-8). To ease the entry for inexperienced users this basic functions should be as intuitive as possible. Groups should have the following attributes: name, description, icon, attached files, and access rights. Process models should have the attributes: name, description, icon, attached files, default instance name, due date time span, and access rights. Whereby, a default instance name specifies the name of the process instance, when started. The due date time span indicates how long the process is allowed to take.

Requirement REQ-8 (Administration of Group and Process Model Attributes). *A BPMS should enable users to edit the attributes of groups and process models. Furthermore,*

A BPMS should enable users to add documentation to most important artifacts (cf. Requirement REQ-9) [IGRR09]. To be precise, it should be possible to attach files and a description or a comment to every group, process model, process node, and executed task. This documentation should then be available to all users. Documentation facilitates the reuse of groups and process models throughout an enterprise, because it is easier for other users to understand the process models with additional documentation.

Requirement REQ-9 (Documentation). *A BPMS should enable users to enrich all artifacts with documentation.*

Modeling and execution of process models should be seamlessly integrated in one environment (cf. Requirement REQ-10). It is significantly easier for users if they can just

4.3. Process Visualization Requirements

execute the process models without any kind of deployment, like in IBM BPM or Activiti. It should also be possible to execute a process model in every state, even if process nodes are not completely specified, for example, if a configuration parameter of a service task is not specified. In this case, missing configuration parameter of the service task should be prompted during process execution. Thus, process models can already be tested when they are not completely modeled, which facilitates the testing of a process model.

Requirement REQ-10 (Integration of Modeling and Execution). *Modeling and execution should be integrated in one environment.*

4.3. Process Visualization Requirements

Visually appealing process visualizations should be provided, to facilitate the usage of a BPMS. For a user it is easier to understand of a process model if he already knows the used process modeling notation or it is easy to understand. Experienced users are accustomed to BPMN 2.0 [OMG11], because it is a de-facto industry-wide standard, which is also the reason that it is used in IBM BPM and Activiti [Sch08]. Hence, a process visualization based on BPMN 2.0 should be supported (cf. Requirement REQ-11) [PCBV10, IRRG09].

Requirement REQ-11 (Support of BPMN 2.0). *A BPMS should provide a process visualization based on BPMN 2.0.*

To provide an easy entry for inexperienced users simple alternative process visualizations are required (cf. Requirement REQ-12) [KRW12, LKR13, KLMR13, KFRMR⁺12]. These process visualizations should be suited especially for users not accustomed to conventional process modeling notations (cf. User Story 1). Therefore, they should provide a quick understanding of a process model with simple means.

Requirement REQ-12 (Alternative Process Visualizations). *A BPMS should deliver alternative process visualizations to provide an easy entry for inexperienced users.*

In contrast to IBM BPM and Activiti, the data flow should be visualized to facilitate a clearer understanding of a process model and especially its data flow (cf. Requirement

4. Requirements

REQ-13) [KR13a]. Without a visualized data flow it is not easily understandable to which process nodes a business object is connected and, thus, how the business objects are used in the process model. Therefore, the business objects should be listed in a sidebar and it should be possible to display data edges individually per business object, because if all data edges are displayed at once the data flow could be confusingly complex.

Requirement REQ-13 (Data Flow). *A BPMS should provide data flow visualization for all process visualizations.*

To implement User Story 2, a BPMS should support so-called *process views*, which provide abstracted views on process models (cf. *Requirement REQ-14*). For example, only the tasks assigned to the logged in user are displayed, which facilitates the understanding of the process model for the user. The process views should be available during the modeling and execution of process models. It should be also possible to define new process views [KKR12b, KR13b, KR13a, RKBB12].

Requirement REQ-14 (Process Views). *It should be possible to apply process views to process models.*

4.4. Modeling Requirements

The following requirements are related to process model updates, i.e., user interaction to model process nodes and business objects. It is essential to provide simple modeling capabilities, especially for inexperienced, but also for advanced users. Like IBM BPM and Activiti, a BPMS should provide interactions for inserting, modifying and deleting of process nodes, which includes AND, XOR, and LOOP gateways (cf. Requirement REQ-15) [PCBV10]. These interactions should include mouse interactions as well as keyboard shortcuts for advanced users.

Requirement REQ-15 (Insertion, Modification, and Deletion of Process Nodes). *A BPMS should enable users to insert, modify, and delete process nodes, including AND, XOR, and LOOP gateways.*

Furthermore, it should be possible to drag and drop process nodes, to achieve a intuitive modeling environment (cf. Requirement REQ-16). This includes dragging of new nodes

from a sidebar onto a process model. Next, users should be able to drag one or more nodes from one place in the process model to another. This requirement is also aligned to the modeling capabilities observed in IBM BPM and Activiti.

Requirement REQ-16 (Drag & Drop Modeling of Nodes). *Drag and drop should be supported for creating and moving process nodes.*

A BPMS should enable the modeling of business objects (cf. Requirement REQ-17). This comprises the creation and deletion of business objects. To model the data flow of a process model, it is mandatory that business objects can be connected as in- or output to the nodes of the process model. To improve the business object modeling capabilities seen in IBM BPM, the latter should be supported by using drag and drop interaction. This is an improvement, because drag and drop modeling is more intuitive than specifying in- and output parameters by name in the properties of a process node.

Requirement REQ-17 (Insertion, Modification, and Deletion of Business Objects). *A BPMS should support creation, modification, and deletion of business objects.*

As mentioned in Section 2.1.3, a BPMS should support check mark tasks (cf. Requirement REQ-18). If a check mark task is connect to a business object, it should become a user task. User tasks are forms, which have to be filled out by user. To determine the user that has to execute these tasks, they should be assignable to users, roles, and organizational units. Check mark tasks are necessary because not every task of a process model can be executed inside a BPMS, e.g., calls to a customer, but should be documented in a BPMS. User tasks are leveraged to integrate user input into a process model and are typically an essential part of every business process (cf. Figure 3.6).

Requirement REQ-18 (Check Mark and User Tasks). *A BPMS should support check mark tasks and user tasks.*

A process model usually contains technical tasks to integrate external systems (cf. User Story 4) [PCBV10]. These tasks should be implemented by service tasks (cf. Requirement REQ-19). Several service task types, also called plugins, should be delivered with a BPMS. For example, to connect to database or file storages. It should be possible to upload and integrate custom plugins. This requirement is mandatory,

4. Requirements

because when a BPMS can not be integrated with external systems in an enterprise, a media break is the result, which generates additional effort for the user.

Requirement REQ-19 (Service Tasks). *A BPMS should support service tasks to provide technical capabilities. Service tasks to connect to database or file storages should be predelivered.*

To enable alternative execution paths in a process model, like seen in the user task modeled in IBM BPM (cf. Figure 3.5), it should be possible to define decisions in a process model. Decisions are represented by XOR gateways (cf. Requirement REQ-20). Three different kinds of XOR gateways should be supported. The first kind, which enables great flexibility during the execution, is a run-time decision, which means a user is prompted at run-time with a question and several textual answers. His answer decides which succeeding path is selected. The second and third kind are decisions based on business objects. Whereby, the second kind is limited to one variable, but in return provides additional validation, which is especially suitable for inexperienced users. The third kind is an arbitrary condition, which is primarily intended for advanced users, because it can implement a wide range of conditions at expenses of a complicated syntax.

Requirement REQ-20 (XOR Gateways). *XOR gateways should be supported to model alternative execution paths based on decisions in a process model.*

In addition, to manually started process models, it should be possible to start process models by a defined trigger, without further input from a user (cf. Requirement REQ-21). For example, a time-based trigger, which starts a process model once a day. The trigger should be configurable at the start event of a process. This feature enables the user to execute process models automatically, when a specific event occurs. Thus it is possible, for example, to automate entire process models, without any user involvement.

Requirement REQ-21 (Trigger). *It should be possible to define a trigger, which starts a process model after a specific event.*

4.5. Execution Requirements

Beside the requirements for business process management, visualization, and modeling, there are requirements regarding the execution of process models. First of all, it is important to provide a good overview on currently executed process instances (cf. Requirement REQ-22). Thus, a user can quickly find out which tasks he has to complete and does not waste time searching for open tasks.

Requirement REQ-22 (Overview Executed Process Instances). *It should be possible to get an overview on executed process instances.*

A BPMS should provide an execution view of the process model, i.e., the current state of a process instance is shown (cf. Requirement REQ-23) [PCBV10]. To be more precise, the state of a process instance is described by the execution states of individual process nodes. This feature is necessary to make the current state of a process model traceable to the user. Furthermore, this can be used to detect errors during the process execution.

Requirement REQ-23 (Monitoring). *A BPMS should support monitoring of process instances.*

When the current state of a process instance is displayed, it should be possible to execute tasks assigned to the current user (cf. Requirement REQ-24). Therefore, it should be supported to start tasks with only one mouse click on the respective process model. The task should then be opened next to the process model, without losing the context of the process model. This is an improvement of the stream tab of IBM BPM (cf. Figure 3.6), which only showed previous tasks. Hence, the user always has the overview on executed and succeeding tasks.

Requirement REQ-24 (Simple Task Execution). *It should be possible to execute tasks easily, when a process instance is opened.*

4.6. Summary

The requirements for a BPMS (cf. Table 4.2) are presented in detail in this section. They are identified on the basis of state-of-the-art BPMS, common demands on BPMS, and

4. Requirements

user stories. It is crucial to realize on which aspects a BPMS has to focus, to provide additional value for SMEs, compared to current BPMS. Hence, the focus lies on an intuitive and easy to use user interface, especially for an inexperienced user. A point of emphasis is also that it is more important to provide a compact set of functionality, which is easy to manage, instead of an extensive feature list resulting in a low level of user-friendliness.

Table 4.2.: Requirements

#	Title	Description
REQ-1	Comprehensive Cloud Platform	A BPMS should be a comprehensive cloud platform comprising all functionality of the process lifecycle in one platform.
REQ-2	Multi-User Operation	Multiple users should be able to use a BPMS at the same time. Results of actions should be propagated to all logged in users
REQ-3	Localization	A BPMS should support a wide range of languages.
REQ-4	Organization Model	It should be possible to use an existing organizational model in a BPMS.
REQ-5	Support for Bookmarks and Sessions	A BPMS should enable bookmarks to share URLs between users. It should also support sessions to hold the current application state.
REQ-6	Shared and Private Groups	A BPMS should offer shared and private groups, for process grouping.
REQ-7	Access Control	An access control system should be available for groups and process models.
REQ-8	Administration of Group and Process Model Attributes	A BPMS should enable users to edit the attributes of groups and process models.
REQ-9	Documentation	A BPMS should enable users to enrich all artifacts with documentation.

REQ-10	Integration of Modeling and Execution	Modeling and execution should be integrated in one environment.
REQ-11	Support of BPMN 2.0	A BPMS should provide a process visualization based on BPMN 2.0.
REQ-12	Alternative Process Visualizations	A BPMS should deliver alternative process visualizations to provide an easy entry for inexperienced users.
REQ-13	Data Flow	A BPMS should provide data flow visualization for all process visualizations.
REQ-14	Process Views	It should be possible to apply process views to process models.
REQ-15	Insertion, Modification, and Deletion of Process Nodes	A BPMS should enable users to insert, modify, and delete process nodes, including AND, XOR, and LOOP gateways.
REQ-16	Drag & Drop Modeling of Nodes	Drag and drop should be supported for creating and moving process nodes.
REQ-17	Insertion, Modification, and Deletion of Business Objects	A BPMS should support creation, modification, and deletion of business objects.
REQ-18	Check Mark and User Tasks	A BPMS should support check mark tasks and user tasks.
REQ-19	Service Tasks	A BPMS should support service tasks to provide technical capabilities. Service tasks to connect to database or file storages should be predelivered.
REQ-20	XOR Gateways	XOR gateways should be supported to model alternative execution paths based on decisions in a process model.
REQ-21	Trigger	It should be possible to define a trigger, which starts a process model after a specific event.

4. Requirements

REQ-22	Overview Executed Process Instances	It should be possible to get an overview on executed process instances.
REQ-23	Monitoring	A BPMS should support monitoring of process instances.
REQ-24	Simple Task Execution	It should be possible to execute tasks easily, when a process instance is opened.

5

Overview Clavii BPM cloud

On the basis of the knowledge gained in Section 2 and Section 3, requirements for a BPMS are specified in Section 4. Originating from these requirements, the Clavii BPM cloud was developed. The Clavii BPM cloud is a cloud-based BPMS for SMEs, which has the main focus on a user-friendly user interface. This section discusses the architecture and data model of Clavii BPM cloud. Section 6 provides an overview on the user interface of Clavii BPM cloud, by discussing the individual pages of the web application. Section 7 singles out implementation aspects to give insights in the implementation of the web application of Clavii BPM cloud.

As already mentioned, the architecture and data model of Clavii BPM cloud, which are introduced in this section, were developed on the basis of the previously specified requirements and the capabilities of GWT and Activiti. Whereby, the web application, which is the contribution of this thesis, is only one part of the overall architecture. Section

5. Overview Clavii BPM cloud

5.1 gives an overview on the architecture and its components. In Section 5.2 the most important data entities and their relations are discussed. Finally, Section 5.3 summarizes this section.

5.1. Architecture

The architecture of Clavii BPM cloud comprises four tiers (cf. Figure 5.1). The four-tier architecture is applied, because it facilitates large flexibility [CK03]. Each tier is represented by individual components, which are connected by interfaces. Therefore, it is possible to replace single tiers, for example, to replace the web application with a desktop application.

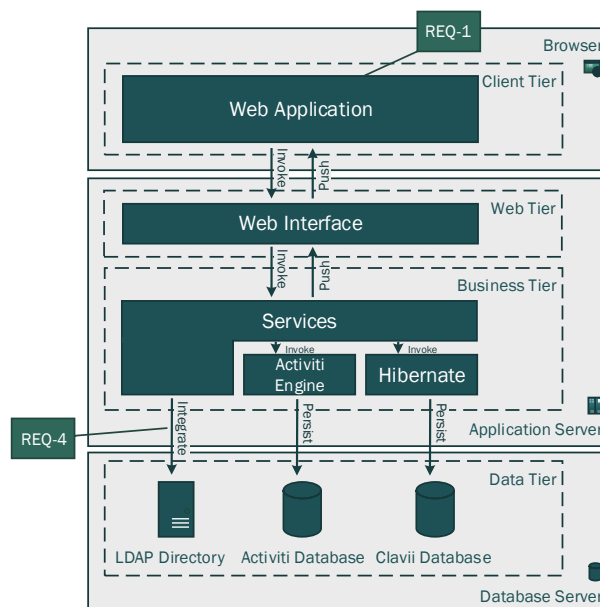


Figure 5.1.: Four-Tier Architecture of Clavii BPM cloud

The four tiers of the Clavii BPM cloud are: *Data tier*, *Business tier*, *Web tier*, and *Client tier*. The data tier contains the data storage of Clavii BPM cloud. The business and web tier are deployed together on an application server. Whereby, the business tier accesses the data tier and provides additional methods for data modification. The web tier provides an interface for the web application to the business tier. Finally, the web

application, which runs in the browser of the user, leverages the underlying tiers to provide a cloud-based BPMS. In the following, the individual tiers are described in detail:

Data tier. The *Data* tier contains the data storage of Clavii BPM cloud. To be more precise, data tier comprises the following components: *LDAP directory*, *Activiti database*, and *Clavii database*. LDAP directory stores the organizational model of Clavii BPM cloud. Activiti database contains data of the *Activiti engine*, for example, deployed process models and their execution data. Next, Clavii database stores all Clavii internal data. The latter is explained in more detail in Section 5.2. Obviously individual databases can be located on different physical machines than the application server to enable for scalability. Furthermore, it is possible to integrate an already existing LDAP directory of an enterprise.

Business tier. The *Business* tier, which is deployed on an application server, comprises the business logic. The *Activiti engine* (cf. Section 2.1.2) is the foundation of the process execution. *Hibernate*¹ encapsulates the Clavii database, to provide access for the *service* components. Service components contain interfaces, which the *web interface* of the Web tier uses to access the *Activiti engine* and the underlying data tier. The service components include additional functionality, for example, methods to modify process models.

Web tier. The *Web* tier provides access to the *business tier* for the web application. It consists mainly of *servlets* and helper classes. This tier enables the integration of new front-ends, without the need to change the business tier. Hence, to create a desktop application for Clavii BPM cloud, only a new *client tier* has to be developed.

Client tier. The *Client* tier consists of the *web application*, created using GWT (cf. Section 2.2). All functionality is available through that single web application (cf. Requirement REQ-1). The web application is delivered from an application server to the browser of the end-user. The communication between the web application and the application server is implemented through RPCs, RequestFactory, WebSockets, and HTTP requests.

In Figure 5.2 a more detailed view of the *service* components is provided. The components may be seen as the backend of Clavii BPM cloud. Since the web application

¹ See www.hibernate.org for more information about Hibernate.

5. Overview Clavii BPM cloud

component leverages the *service* components, their range of methods is explained briefly.

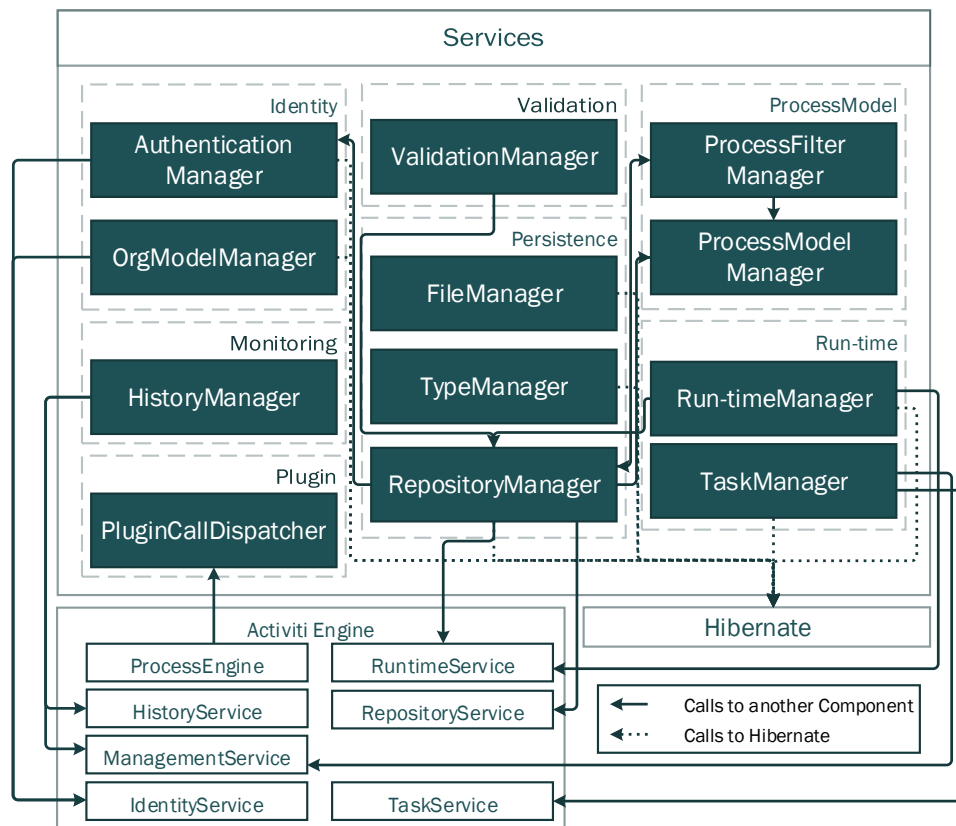


Figure 5.2.: Overview on Service Components

Particularly, *service* components is divided into single components, which are introduced in the following:

Identity Component. This component is responsible for authenticating a user. Furthermore, it comprises methods to access and modify the organizational model. This is used by the web application to offer an editor for the organization model.

Monitoring Component. The *monitoring* component encapsulates the **HistoryService** of the Activiti engine. Hence, execution information of process instances can be accessed. To be more precise, the instances of a process model can be retrieved. Moreover, the states of individual nodes of a process instance are provided.

Plugin Component. The *plugin* component is used to add new plugins as XML files. This component is also used at run-time to execute plugins. Therefore, the Activiti engine calls the *PluginCallDispatcher*.

Persistence Component. The *persistence* component provides access to all entities of the underlying data tier. For example, it is used to save files when they are uploaded to the web application as attachments. *RepositoryManager* is the central interface for managing process models. All process modification methods and the methods to create process views can be invoked by the *RepositoryManager*.

Validation Component. The *validation* component comprises methods to validate branching conditions. If such conditions are invalid, meaningful responses are delivered, which may then be presented to the user.

ProcessModel Component. The *processmodel* component is used internally by the *RepositoryManager* to modify process models. Therefore, this component provides low-level methods for process model modification. For example, methods to add nodes to a process model. This component is externally accessible, but it is recommended to use the *RepositoryManager*.

Run-time Component. The *run-time* component is the central component for process execution. Particularly, it allows to start and stop process instances. In addition, it has the capabilities to complete tasks and to deliver additional information about tasks.

The presented service components are accessed by the web application component to provide a cloud-based BPMS. Figure 5.3 shows components of the web application component and the web interface component. In principle, the web interface is a bridge between the web application component and the service components. Because service components cannot be accessed directly from the browser, the communication is handled by several servlets and other mechanisms like WebSockets. Individual components are described in the following:

Servlets Component. *Servlet* component can be categorized in servlets, which provide server endpoints for RPCs and the RequestFactory, and servlets, which are used for file uploads. Basically both are just throughput stations to enable the web application to access the service components.

5. Overview Clavii BPM cloud

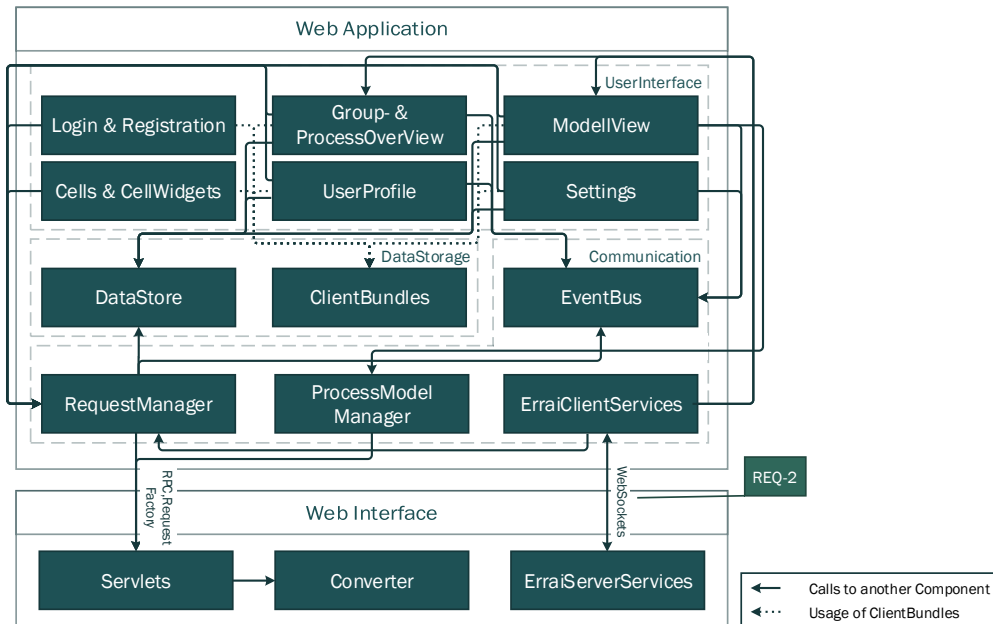


Figure 5.3.: Overview on Web Application and Web Interface

ErraiServerServices Component. *ErraiServerServices* component provide the capability to contact every client from the server-side. This is necessary to propagate updates, like modified process models, to clients (cf. Requirement REQ-2). *ErraiServerServices* are based on a framework called Errai, which provides communication capabilities through WebSockets for GWT applications [Err].

Converter Component. As previously mentioned, RequestFactory component provides an alternative to manually create proxy classes. However in some cases, it is not possible to use the capabilities of RequestFactory. For example, the BpmnModel, the Java representation of a process model in the Activiti engine, can not be sent to the client using RequestFactory. In this case, the issue is that the BpmnModel uses 88 different classes to represent a process model. This class structure can not simply be used with the RequestFactory. Therefore, converter classes are necessary to convert server entity classes to client entity classes. Based on RPCs client entity classes can then be sent from the server- to the client-side and vice versa.

Communication Component. *Communication* component contains all classes used for communication between server- and client-side as well as between client components.

The *RequestManager* encapsulates server communication based on RPC and Request-Factory. As an exception the *ProcessModelManager* comprises all methods to modify process models. *ErraiClientServices* are client-side endpoints for the communication with the *ErraiServerServices* component.

DataStorage Component. *DataStorage* component contains all components, to store cache data on the client-side. *DataStore* component caches all entities retrieved from the server, for example, groups and process models. The *ClientBundles* provide access to all static resources, e.g., CSS files and images (cf. Section 2.2.2).

UserInterface Component. The *User Interface* component contains classes, directly used by the user interface. The majority of the user interface is created utilizing UiBinder (cf. Section 2.2.2). Visualization of process models is implemented based on a graphic framework called Lienzo [Emi]. These aspects are explained in Section 6 and Section 7 in more detail.

5.2. Data Model

Besides the architecture, an appropriate data model is a prerequisite to develop an application. The data model in the context of this thesis is separated into an *organizational model* and a *general model*. The general model connects an organizational entity with groups and process models.

Figure 5.4 shows the organizational model (cf. Requirement REQ-4). A user is represented by the *agent* entity. An agent has a first name, a last name, an e-mail, and a string containing the avatar as an image URL [Mas98]. The *orgunit*, *organization*, and *role* entities comprise a name and an icon. Whereby, the *orgunit* entity represents a department of an enterprise, the *organization* entity represents the enterprise itself, and the *role* entity represents the role of an agent, which can be for example secretary.

An agent can be assigned to one or more roles, exactly one organization and exactly one orgunit. A new agent can also be created without assigning any orgunit, organization, or role. An orgunit may contain any number of agents. Furthermore, it is possible that

5. Overview Clavii BPM cloud

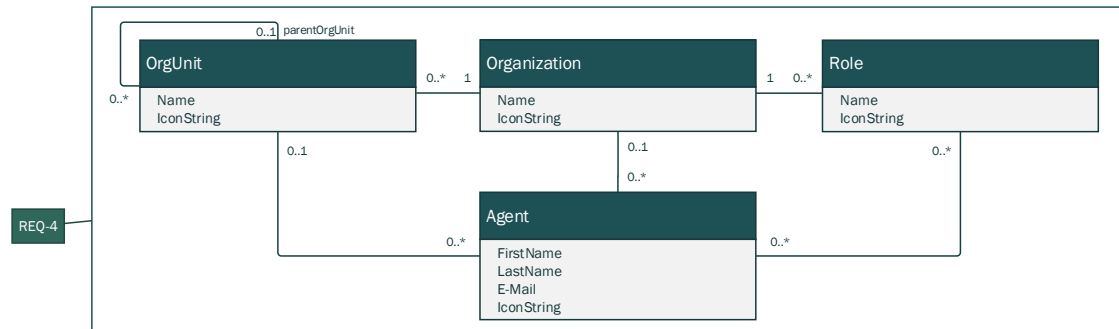


Figure 5.4.: Organizational Model Entities

orgunits contain orgunits. Orgunits are always linked to exactly one organization, and an organization can have any number of orgunits. The organization can have several roles. However, a role has always to be assigned to one specific organization.

An organizational model of a fictional enterprise which can be implemented by Clavii BPM cloud is shown in Figure 5.5. The enterprise *Contoso Ltd.* contains the departments: *Accounting*, *Marketing*, and *Product Development*. The employees of the enterprise, represented by agents, are assigned to their respective department. Independent of their organizational unit, they are also assigned to roles. For example, *Ellen Adams* is a *Secretary*.

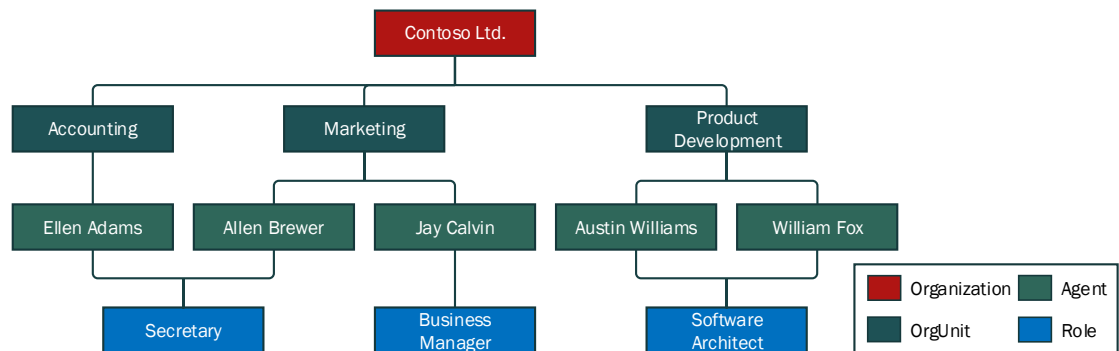


Figure 5.5.: Example Organizational Model

The general model is shown in Figure 5.6. An *organizationalentity* may be an agent, an orgunit, an organization, or a role. Access rights can be assigned to any kind of organizational entity (cf. Requirement REQ-7). For example, when an orgunit has access rights on a group, every agent within this orgunit has the same rights.

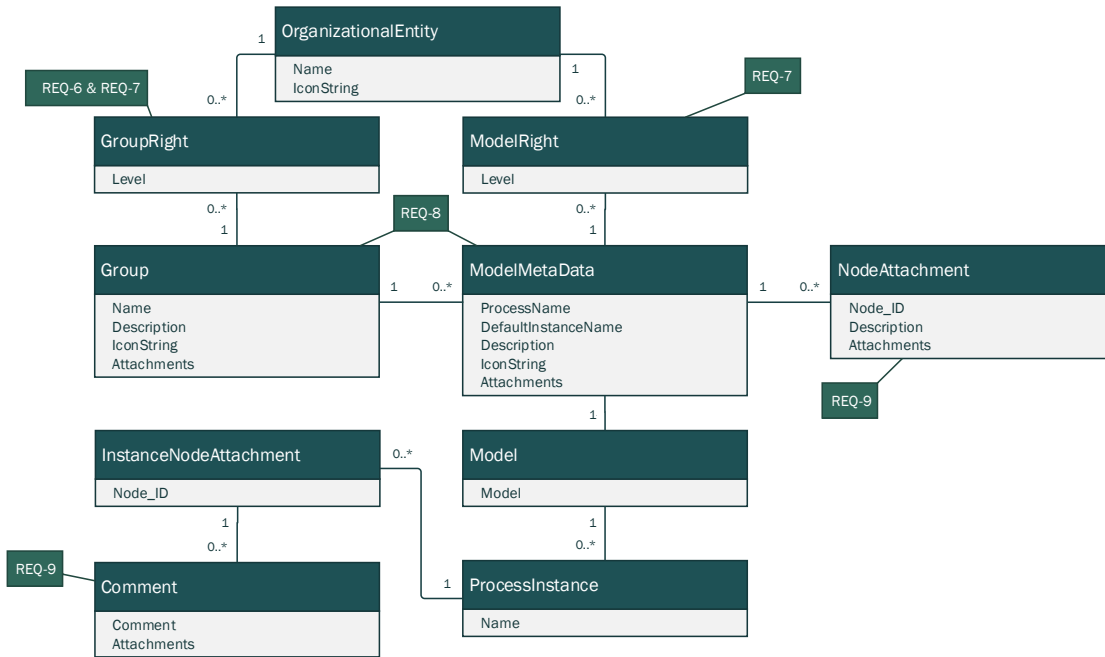


Figure 5.6.: General Model Entities

Groupright and *modelright* are used to assign access rights to an organizationalentity (cf. Requirement REQ-7). Whereby, the groupright represents the access right to a group. The *level* specifies if the user can only view the group or if he can also edit the group. Therefore, it is possible to share groups between organizationalentities (cf. Requirement REQ-6). The modelright represents the access right to a process model. The level attribute of the modelright specifies, if the user can only view the process model, if the user can only edit the process model, or if he can also execute the process model. For example, if a user should be able to execute the process model *FinancialReport*, a modelright entity is created and linked to the user and to the process model. The level of the modelright would be *Run*.

A *group* entity contains attributes *name*, *description*, an *icon*, and *attachments* (cf. Requirement REQ-8). The group entity is linked to a set of process models, which are encapsulated by the *modelmetadata* entity. The modelmetadata entity contains any meta data related to the process model. To fulfill Requirement REQ-9, the modelmetadata is linked to a *nodeattachment* entity, which contains a *description* and *attachments*.

5. Overview Clavii BPM cloud

A *processinstance* entity is created when a process model is executed. Through the *processinstance* entity, *instancenodeattachments* are linked to a process instance to enable the user to add comments to a process instance.

5.3. Summary

This section provides insights in the architecture and data model of Clavii BPM cloud. First, the four-tier architecture is discussed in detail. In the following, the individual components of the business, web, and client tier are explained. Subsequently, the data model, which consists of an organizational and a general model, is illustrated.

The architecture and data model fulfill several general requirements. Requirement REQ-1 is met by delivering the entire functionality of Clavii BPM cloud in a single web application. The ErraiServerServices and ErraiClientServices fulfill Requirement REQ-2 by propagating actions of one user to all users. Therefore, collaborative modeling is facilitated. In order to meet Requirement REQ-4, the organizational model is provided. By connecting the organizational entities to modelright and groupright, shared and private groups and the access system is implemented (cf. Requirements REQ-6 & REQ-7). The general model enables the modification of group and process model attributes and the addition of documentation (cf. Requirements REQ-8 & REQ-9).

6

User Interface of the Web Application

After the previous section discussed the architecture of Clavii BPM cloud, this section focuses on the user interface of Clavii BPM cloud. Requirements, discussed in Section 4, are the basis for the user interface design of Clavii BPM cloud. Whereby, the main goal is a simple and intuitive user interface (cf. Section 4), which should be easy to use, in particular, for inexperienced users. Furthermore, a user interface should also provide as much assistance as possible, during every step of the process modeling and execution phase.

Section 6.1 gives an overview on the user interface of the web application on the basis of a site map. Subsequently, Section 6.2 explains the entire user interface in detail. Finally, Section 6.3 concludes this section with a summary and a discussion on how the requirements specified in Section 4 are fulfilled by the individual pages of the web application.

6.1. Site Map

Figure 6.1 displays the site map of Clavii BPM cloud. Thereby, filled out rectangles represent the individual pages. A user can navigate from one page to another if the pages are connected by an arrow. Pages with bold printed borders (e.g., *Login* page) are the starting pages of the region, which is the surrounding rectangle. This implies that *Login* page is the start page of the entire web application. When the user logs in the first page displayed is the *GroupOverview* page. When a process model is opened in the *ModelView* page, the *BuildtimeView* page is shown per default.

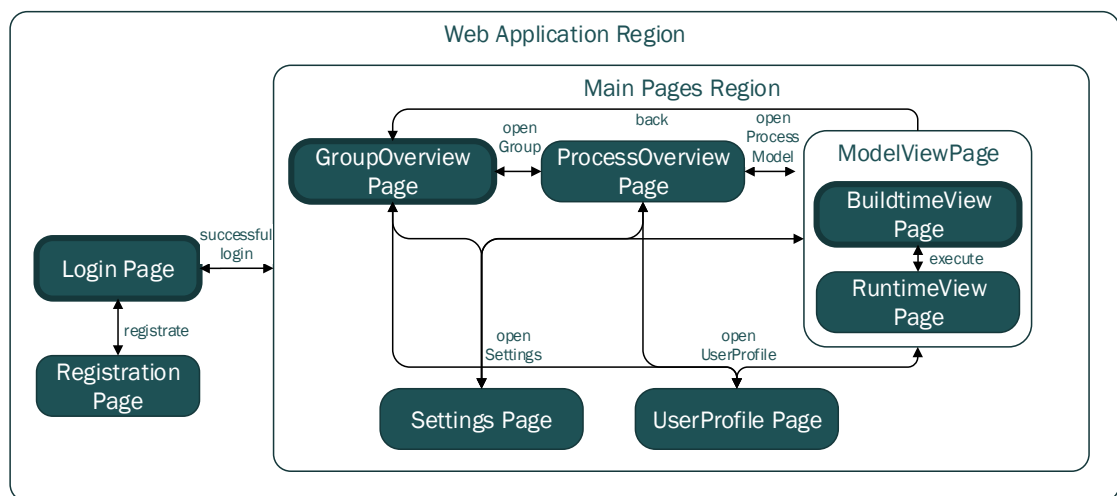


Figure 6.1.: Site Map

When a user is not logged in, he is only able to access the *Registration* and the *Login* page. After the user is logged in, he is able to access all pages in the *Main Pages* region. A *GroupOverview* page shows all groups accessible by the user. Furthermore, he may open a group. As a result, he switches to the *ProcessOverview* page. On this page he can switch back to the *GroupOverview* page and is able to open a process model and, thus, gets to the *ModelView* page. The *ModelView* page is separated into two pages: *BuildtimeView* page and *RuntimeView* page. The *Settings* page and the *UserProfile* page can be opened on all *Main Pages*. The user can also always log out and, thereby, leave the *Main Pages* region to the *Login* page. The following section introduces individual pages in detail, to provide an impression of the user interface.

6.2. Individual Pages of the Web Application

The individual pages shown in this section cover the most important parts of the user interface of the web application. Therefore, this section provides a walkthrough through the pages necessary to administrate, model, and execute a process model with Clavii BPM cloud.

6.2.1. Login Page

The first page that is displayed when a user loads the web application of Clavii BPM cloud is the *Login* page (cf. Figure 6.2). The *Login* page prompts the user for his e-mail and his password. When the user is authenticated he is navigated automatically to the *GroupOverview* page (cf. Figure 6.4).

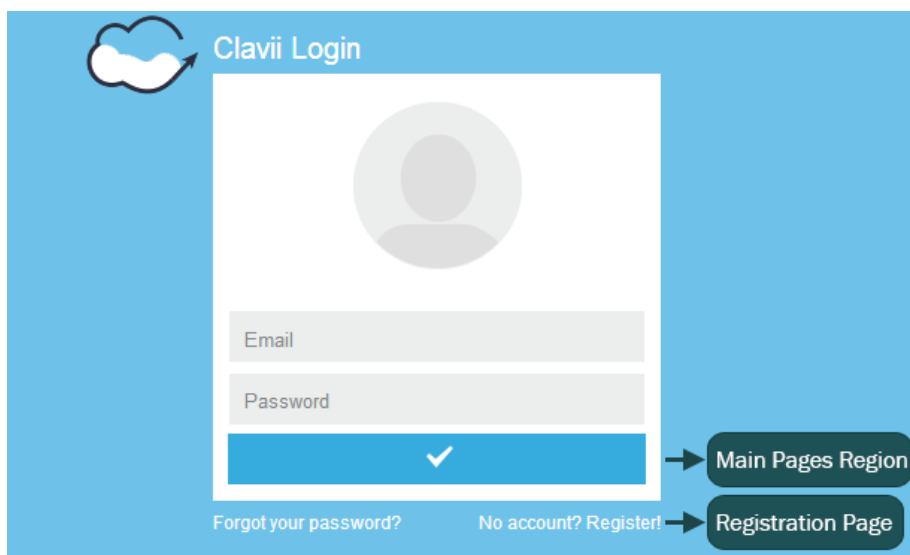


Figure 6.2.: Login Page

6.2.2. Registration Page

In case a user has no account, he can navigate to the *Registration* page to create a new account (cf. Figure 6.3). Like specified in Requirement REQ-4, a user account

6. User Interface of the Web Application

has the attributes first name, last name, and e-mail, which have to be provided when registering. The avatar (i.e., user icon) can be uploaded later in the *UserProfile* page. When a new user account is created at the *Registration* page, it is not assigned to any organization, organizational unit, or role. If a user is a member of any organizational entity, for example, organizational unit or organization, and wants to benefit from their access rights, the user has to be assigned to this organizational entity in the *Settings* page.

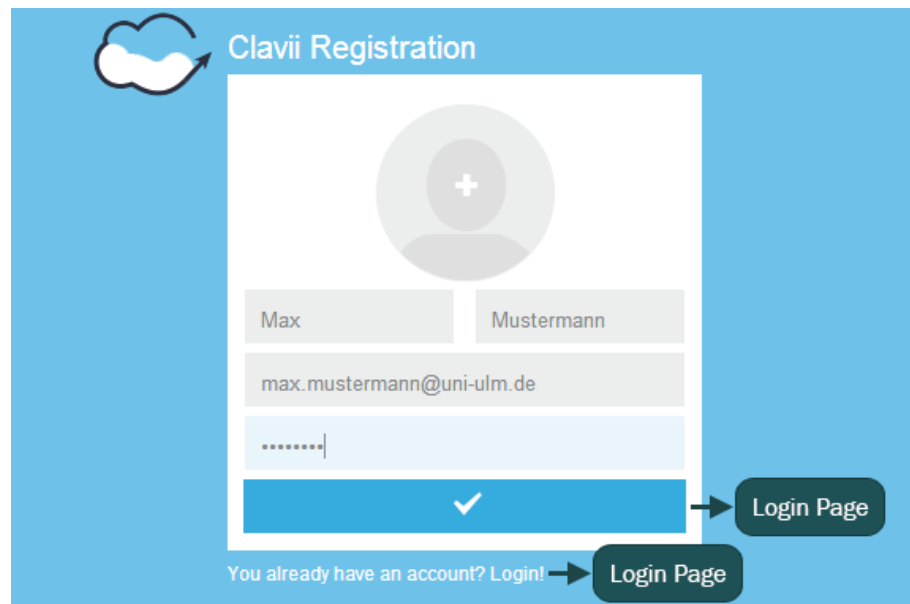


Figure 6.3.: Registration Page

After a user account is created, the user is able to go back to the *Login* page and, utilizing his new account, he is now able to access the *Main Pages* region. Furthermore, users are able to create new user accounts in the *Settings* page. This enables administrators in enterprises to create accounts for all employees.

6.2.3. GroupOverview Page

After a successful log in, the *GroupOverview* page is shown (cf. Figure 6.4). The *TopBar* panel on the top provides navigation features, to assist the user in navigating through the pages of the web application. For example, the *Back* button enables the user to

6.2. Individual Pages of the Web Application

go back to the previous page. The *Title* label in the middle shows the name of the currently opened group or process model, whereby the *GroupOverview* page is always represented by the name *Home*. Icons on the right side enable the user to access *Settings* page, *UserProfile* page, and to log out. The *TopBar* panel is visible throughout pages *GroupOverview*, *ProcessOverview*, and *ModelView*.

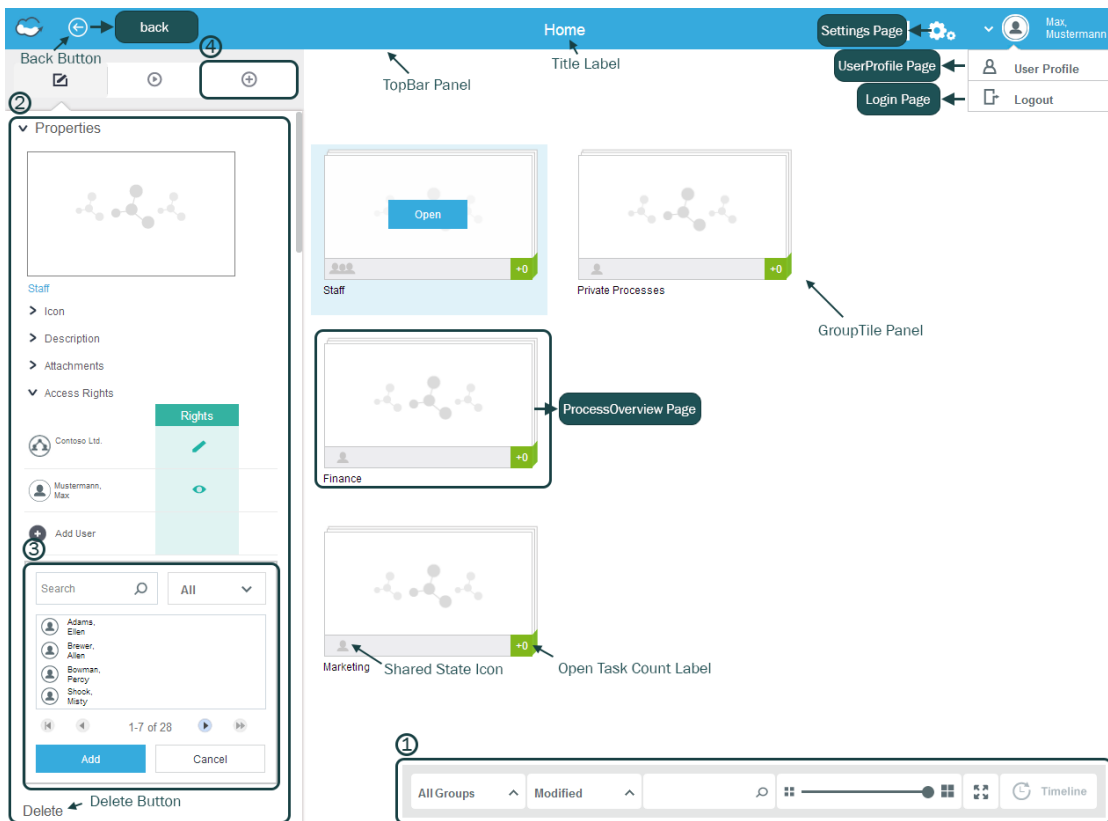


Figure 6.4.: GroupOverview Page

In the center of the page, the groups of process models are displayed by *GroupTile* panels with their icon and name. The *Shared State* icon describes if the group is shared or private (cf. Requirement REQ-6). The *Open Task Count* label displays the number of open tasks in this group of process models the current user has to do, to give a quick overview on open tasks. The user can navigate to the *ProcessOverview* page (cf. Figure 6.5) by double clicking on a group.

6. User Interface of the Web Application

The *ViewSettings* panel ① on the *GroupOverview* page provides advanced view options. Groups can be filtered by access status, i.e., private or shared groups. Groups can be sorted by last modified date, number of open tasks, and in an alphabetical manner. Furthermore, the user can search for a specific group name. If the number of groups is still too large, the list of groups can be zoomed out or fit-to-screen button can be pressed. Hence, the *ViewSettings* panel is particularly useful for large numbers of groups.

If the user clicks on a group, properties of that group are shown in a sidebar ②, to enable users to quickly modify attributes of a group. If a user has only a *View* right for a group, the sidebar opens in read-only mode. Otherwise, all properties can be edited (cf. Requirement REQ-8). Thus, name and description can be changed. Furthermore, it is possible to upload a new icon and to choose between an uploaded icon or an snapshot of a contained process model as group icon. Additional documentation can be uploaded as an attachment (cf. Requirement REQ-9).

To configure access to a group, access rights can be assigned to arbitrary organizational entities by the *AssignRights* panel ③, which is a form in the sidebar (cf. Requirement REQ-7). In the *AssignRights* panel, is also possible to search for specific names and filter by type (i.e., user, role, or organizational unit).

Groups can be deleted both by pressing the *Delete* key or by clicking the *Delete* button in the sidebar. Furthermore, multiple groups can be selected and deleted at once. New groups can be created in the *Add* tab ④, by providing a group name.

6.2.4. ProcessOverview Page

As already mentioned, opening a group, displays the *ProcessOverview* page, which shows the process models of the group (cf. Figure 6.5). This page has the same structure as the *GroupOverview* page. However, in the *TopBar* panel a *Breadcrumb* is shown. A click on the *Breadcrumb*, brings the user back to the *GroupOverview* page. The user can not only open a process model by clicking on the *Open* button, but also directly execute a process model, by clicking on the *Run* button. As a result, the started process model is shown in the *RuntimeView* page of the *ModelView* page.

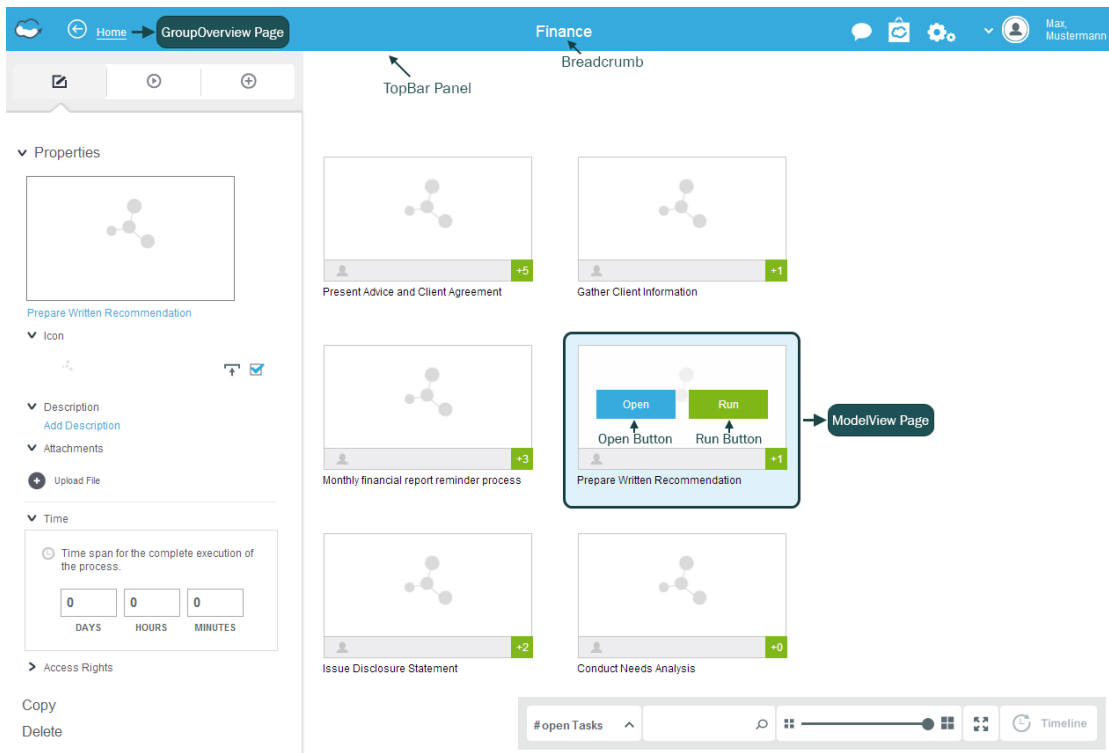


Figure 6.5.: ProcessOverview Page

6.2.5. ModelView Page

The *ModelView* page is displayed when a process model is opened, showing the *BuildtimeView* page per default (cf. Figure 6.6). The *TopBar* panel shows a *Breadcrumb*, which is similar to the breadcrumbs in the *ProcessOverview* page. The opened process model is placed in the center of the page. The default notation is *BPMN* (cf. Requirement REQ-11), a *Transit Map* (cf. Requirement REQ-12) can be chosen in the *ViewSettings* panel [Mey14]. In contrast to BPMN, TransitMap layouts the process model vertically and with simple means (cf. Figure A.2). Therefore, it is especially suitable for inexperienced users. Additional examples for the layouting of process models in Clavii BPM cloud are provided in Figure A.1 and Figure A.2.

Process views (cf. Requirement REQ-14) can be applied through the *ViewSettings* panel [KKR12a, KR13c]. Whereby, the default option *CommonView* corresponds to the original process model. The slider in the *ViewSettings* panel can be used for zooming of

6. User Interface of the Web Application

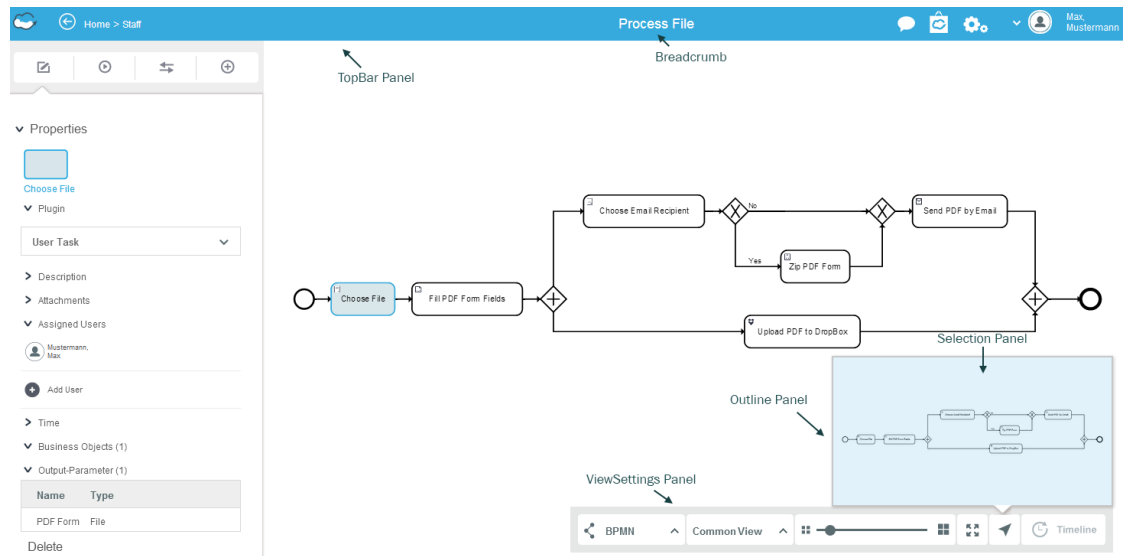


Figure 6.6.: ModelView Page

the process model. The *Outline* panel gives an overview on the currently opened process model. This is particularly helpful for large and complex process models [WRMR11]. The *Selection* panel shows the current viewport, which can be moved with the mouse to navigate the process model.

Figure 6.7 displays the creation of a new process node. A new process node can be created by dragging it from the *Add* tab of the sidebar onto a process model.

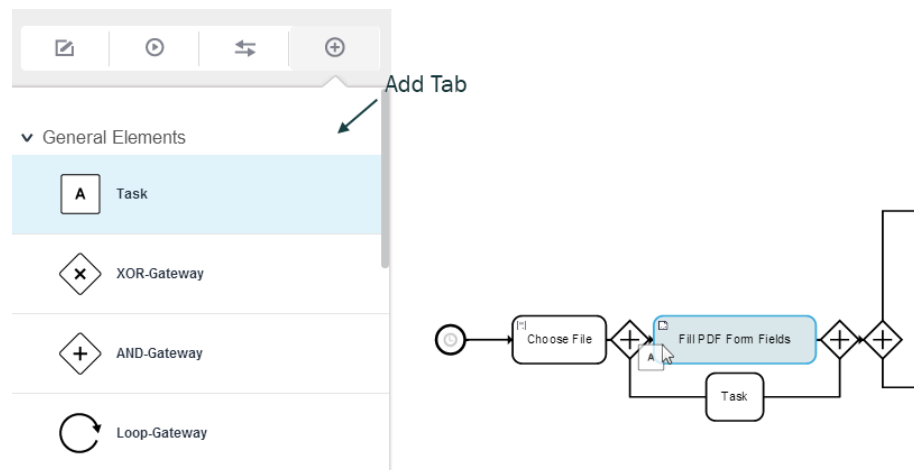


Figure 6.7.: Adding a Process Node via Drag and Drop

6.2. Individual Pages of the Web Application

The new node can be dragged onto an edge or onto a node of a process model. Dragging the task onto a node in the process model results in an enclosing AND block, i.e., an AND join gateway is inserted before and an AND split gateway is inserted after the node. XOR and LOOP blocks can be also created by dragging the corresponding nodes from the sidebar onto the process model (cf. Requirement REQ-20,). Multiple nodes of the process model can be moved from one place in the process model to another (cf. Requirement REQ-16) by simply dragging them to another location in the process model.

Nodes of a process model can be further configured using the *Properties* tab of the sidebar (cf. Requirement REQ-15), which opens automatically after selecting a node. This tab allows the user to configure trigger, user tasks, service tasks, and gateways. The *Properties* tabs of the sidebar for various node types are shown in Figure 6.8.

The figure displays four panels representing the configuration tabs for different BPMN node types in a web application sidebar. Each panel has a 'Properties' section and expandable sections for 'Description', 'Attachments', 'Assigned Users', 'Business Objects', 'Input-Parameter', and 'Output-Parameter'.

- Trigger:** Features a 'Start' icon, a 'Timer' dropdown, and an 'Activate Trigger' button. It includes input and output parameter tables with columns for Name, Type, and Path.
- User Task:** Includes a 'Choose File' button, a 'User Task' dropdown, and a table for input/output parameters.
- Service Task:** Features a 'Zip PDF Form' button, a 'Plugin' dropdown, and a table for input/output parameters.
- XOR Gateway:** Includes an 'XOR-Split-Gateway' icon, a 'Manual' dropdown, and a table for input/output parameters.

Figure 6.8.: Node Configuration Properties Tabs of the Sidebar

Common attributes (i.e., description and attachments) are editable for every node in a process model. A trigger can be configured on the start event and allows for scheduling a process execution, e.g., by configuring a timer (cf. Requirement REQ-21). In this case, a mapping to a time span business object is necessary [And14]. A time span specifies in which time period a process model should be repeatedly started. Output of a trigger (i.e., the current time) can be mapped to a business object in the process model.

A user task has an in- and output parameter mapping, which specifies the business objects which are the in- and output of the user task. Since in- and output parameters

6. User Interface of the Web Application

are variable, parameters can not be mapped for a user task. However, it is possible to remove existing business object mappings, which were created with the *Data* tab, which is explained later. As previously mentioned, a user task without a mapping to business objects is a check mark task (cf. Requirement REQ-18).

In contrast to a user task, a service task has predefined in- and output parameters, which can be mapped to business objects (cf. Requirement REQ-19).

A XOR gateway allows for configuration of three different kinds of decision types, as specified in Requirement REQ-20. To be more precise, a decision can be configured either by specifying a textual question and respective answers, by specifying a condition based on a business object, or by specifying an arbitrary condition.

The *Data* tab allows for creating and modifying business objects (cf. Figure 6.9). The business objects are listed in the *Data* tab of the sidebar. Edges represent the in- and output mapping to nodes in a process model (cf. Requirement REQ-13). Process nodes, not connected with a business object, are drawn slightly transparent.

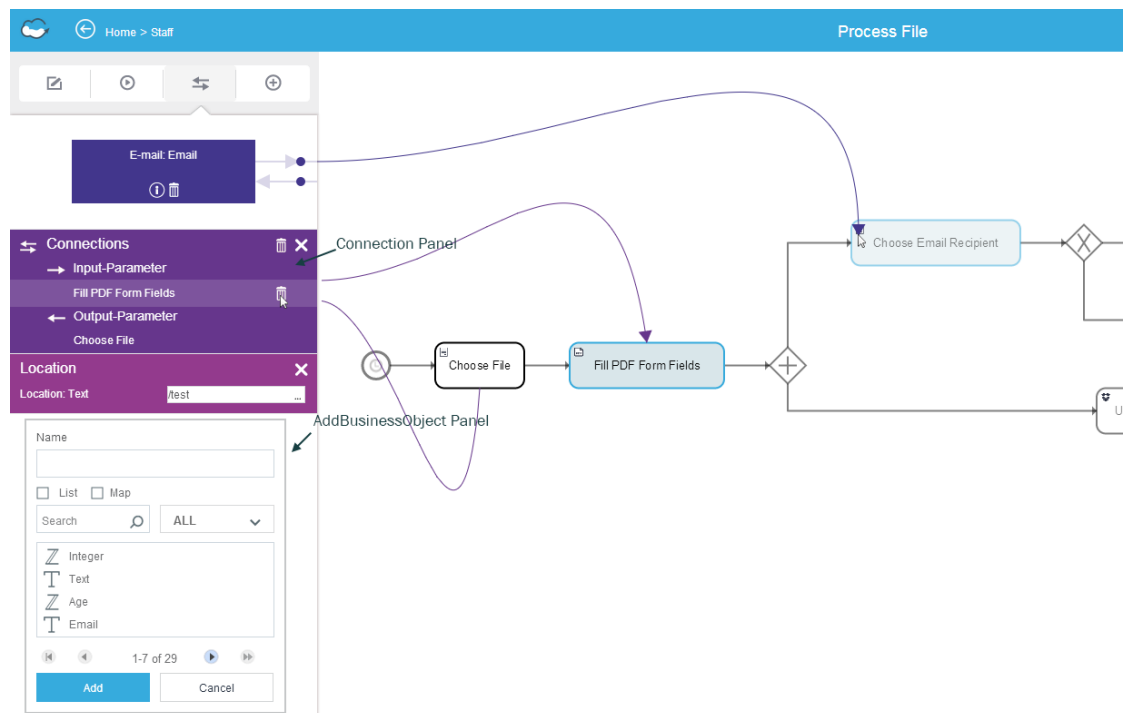


Figure 6.9.: Modeling Business Objects in the BuildtimeView Page

Furthermore, business objects can be added through the *AddBusinessObject* panel by choosing a business object type and a name. Further, lists and maps of business objects can be created. By connecting business objects to nodes of a process model, new data mappings are created. They can be created by dragging an edge from the circle right next to the business object onto a process node (cf. Requirement REQ-17).

To remove data mappings, the user has to open the *Connection* panel of a business object, by clicking on the trash can. In this panel, all data mappings and their connected nodes are listed and can be removed. Furthermore, start values for business objects, which are prefilled at the start of the execution of the process model, can be set.

Figure 6.10 shows the *RuntimeView* page, which illustrates a process instance. Process models can be started by only clicking the *Create* button in the *Run* tab of the sidebar (cf. Requirement REQ-10).

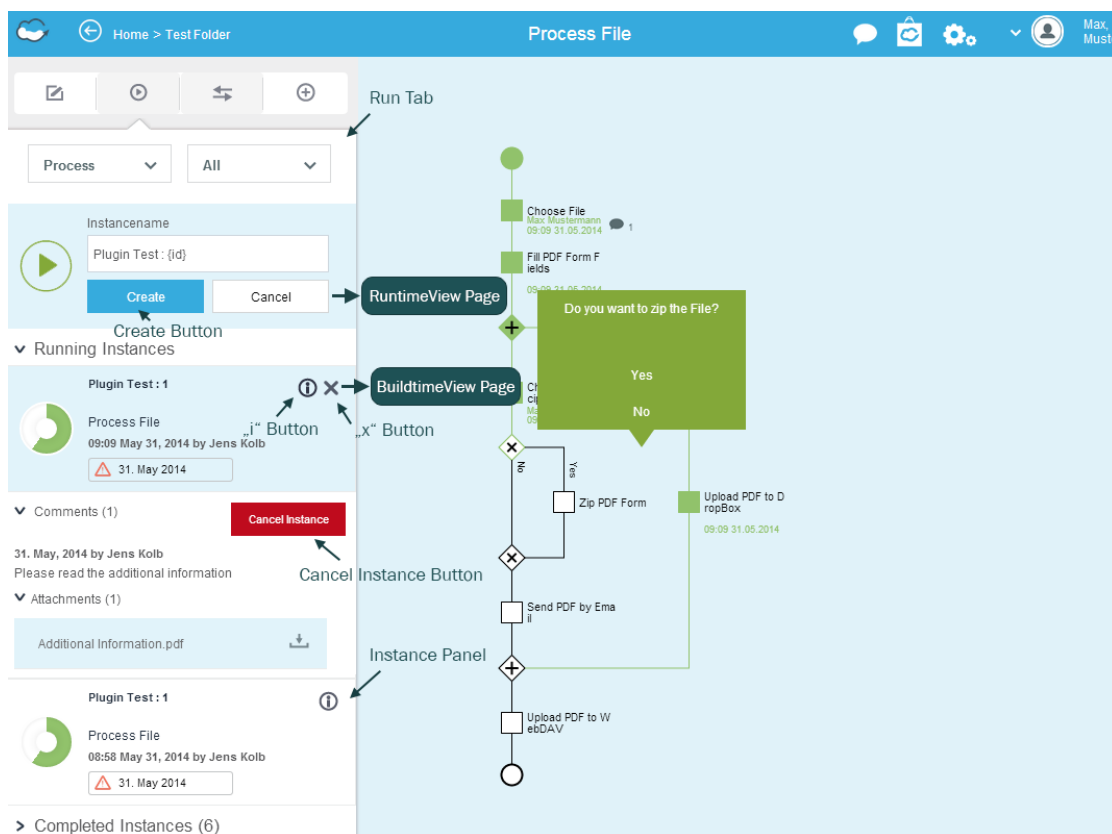


Figure 6.10.: RuntimeView Page

6. User Interface of the Web Application

The *Run* tab also lists all running instances (cf. Requirement REQ-22). They can be filtered by currently opened process models or the current group. The latter is the group to which the current process model is assigned to. Furthermore, they can be filtered by user participation, for example, only process models can be displayed with open tasks of the current user.

All details of a process instance can be accessed by clicking on the “i” button on the top right of the *Instance* panel. This includes instance name, process name, date, initiator of the process model, due date, comments, and attachments. A instance can be canceled using the *Cancel Instance* button. Moreover, user can switch back to the *BuildtimeView* page by clicking on the “x” button on the top right (cf. Requirement REQ-10).

In the center of the page, the state of the current process instance is shown. In Figure 6.10 the process instance is drawn in the *Transit Map* notation (cf. Requirement REQ-23). Already executed tasks are annotated with the user name, which executed the task, and the corresponding time. Furthermore, executed tasks are colored in green. Executable tasks, which can be executed by a simple click, have a green border. In Figure 6.10, the user is asked by a popup window which path of the XOR block should be executed.

In case a task is executed, a user form is displayed in the *Run* tab of the sidebar (cf. Figure 6.11). Whereby, the generation of the forms is not part of the contribution of this thesis. The *Add* button allows the user to add comments. Finally, the task can be completed with the *Complete* button (cf. Requirement REQ-24).

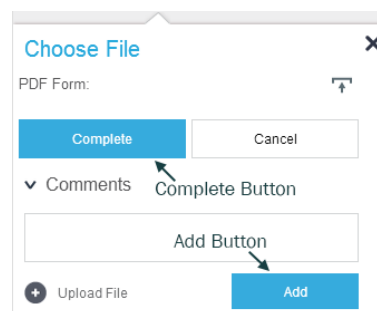
A screenshot of a web application dialog box titled "Choose File" with a close button (X) in the top right corner. The dialog contains a "PDF Form:" label with a small icon to its right. Below this are two buttons: "Complete" (highlighted in blue) and "Cancel" (white with a grey border). Underneath these is a section labeled "Comments" with a downward arrow icon. To the right of "Comments" is the text "Complete Button" with an arrow pointing to the "Complete" button. Below the "Comments" section is a text input field with the placeholder text "Add Button". At the bottom left is a button with a plus icon and the text "Upload File". At the bottom right is a blue button with the text "Add".

Figure 6.11.: User Form generated for the Task Execution

6.2.6. Settings Page

On the *Settings* page, all important settings of Clavii BPM cloud can be configured (cf. Figure 6.12). The organization model can be specified and modified in the *OrgModel* tab (cf. Requirement REQ-4). In particular, this includes modification of users, roles, organizational units, and organizations. Furthermore, an overview on assigned access rights on groups and process models is provided.

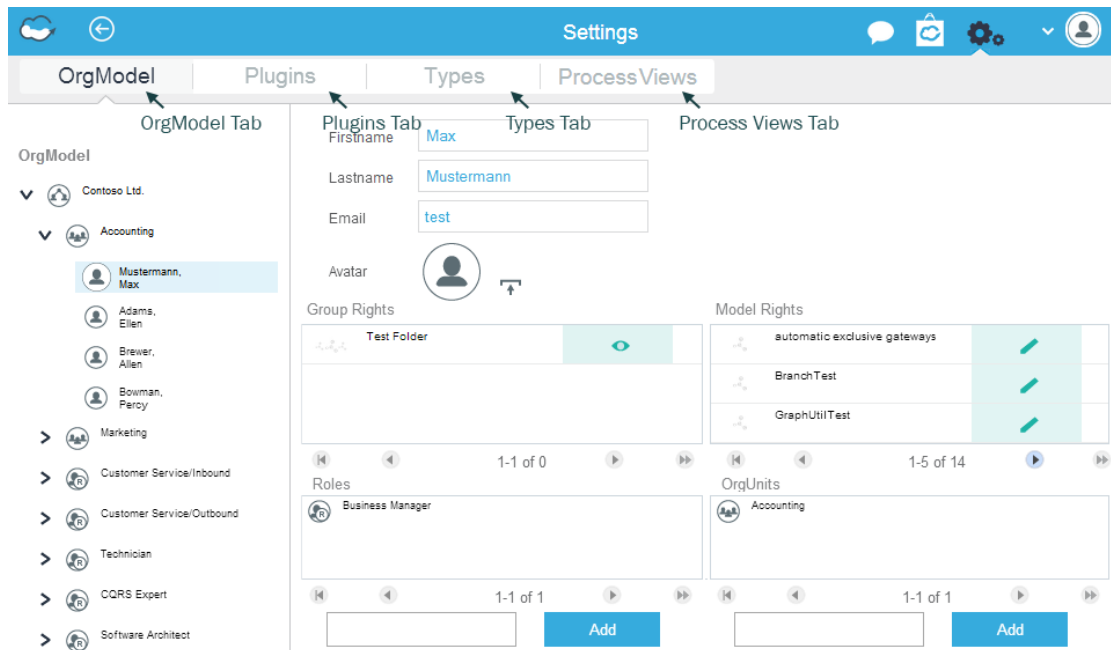


Figure 6.12.: Settings Page

The *Plugin* tab contains a plugin catalogue. The plugins and their in- and output parameters can be inspected. Further, new plugins can be uploaded in the form of a *.jar* file, containing the implementation, and an XML file, which describes the plugin (cf. Requirement REQ-19).

The *Types* tab lists all available business object types. New business object types can be created either by extending existing business object types or by creating a new simple, enum, or complex business object type [And14].

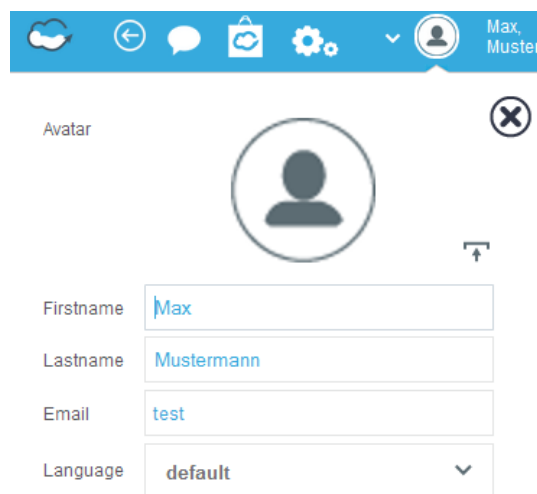
The *Process View* tab displays predefined process views. It also allows the creation of custom process views (cf. Requirement REQ-14) [Kam14]. Details of the plugins,

6. User Interface of the Web Application

business object types, and process views are not discussed in the following, because they are not part of the contribution of this thesis.

6.2.7. UserProfile Page

Figure 6.13 shows the *UserProfile* page. The first name, last name, and e-mail of the current user can be changed. Furthermore, a new avatar may be uploaded and the user language can be specified (cf. Requirement REQ-3). The user interface is always shown in the language of the logged in user.



Avatar

Firstname Max

Lastname Mustermann

Email test

Language default

Figure 6.13.: UserProfile Page

6.3. Summary

In summary, this section discusses the individual pages of the web application of Clavii BPM cloud. At first, an overview on the entire web application is given on the basis of a site map. The site map explains the two regions of the web application, which are the *Web Application* region and the *Main Pages* region. Whereby, the *Main Pages* region can only be accessed by logged in users. In the following, the individual pages of the web application are discussed in detail. The *Login* and *Registration* page enable the

user to log in to the web application and create new accounts. The *GroupOverview* page provides access to the groups and enables the user to administrate them. The contained process models of a group can be managed in the *ProcessOverview* page. Process models can be modeled and executed in the *ModelView* page, which is separated into *BuildtimeView* and *RuntimeView* page. Finally, additional capabilities for the modification of settings and the user profile are provided by the *Settings* and *UserProfile* page. In the following, the requirements which are met by the pages of the web application are discussed.

The pages shown in this section fulfill several requirements, previously defined in Section 4. The localization requirement specified in Requirement REQ-3 is met by providing a language setting to the user in the *UserProfile* page (cf. Figure 6.13). More details about the implementation of this requirement are given in Section 7.2.5.

The organizational model (cf. Requirement REQ-4) can be edited on the *Settings* page (cf. Figure 6.12). New accounts can be created both on the *Registration* and on the *Settings* page (cf. Figure 6.3). The *Settings* page allows further modification of the organizational model, e.g., adding new roles or organizational units.

The *GroupOverview* page complies with Requirement REQ-6, which specifies private and shared groups. The groups are displayed by *GroupTile* panels with their icon and name. Furthermore, the *Shared State* icon facilitates the distinction between private and shared groups (cf. Figure 6.4). Moreover, the displayed groups can be filtered by access status, for example, it is possible to show only private groups.

The access control system (cf. Requirement REQ-7) is implemented by *GroupOverview* and *ProcessOverview* page. In both pages the *Properties* tab of the sidebar, for a selected group or process model, contains the *AssignRights* panel (cf. Figure 6.4, ②). In the *AssignRights* panel it is possible to assign access rights to every organizational entity, i.e., organizational unit, role, or agent. An overview on all assigned access rights is given in the *OrgModel* tab of the *Settings* page.

Requirements REQ-8 and REQ-9 state that attributes of groups and process models should be editable by the user and it should be possible to add additional documentation to all artifacts. The *GroupOverview* and *ProcessOverview* page provide a *Properties*

6. User Interface of the Web Application

tab in the sidebar, which allows the modification of all attributes of the selected group or process model (cf. Figure 6.4, ②). Additional documentation can also be added in this sidebar. Furthermore, it is possible to add comments, including attachments, during the execution of a process model (cf. Figure 6.11).

Requirement REQ-10 states that modeling and execution environment should be integrated. This requirement is met by integrating both environments in the *ModelView* page. The user can switch to the *RuntimeView* page by executing a process model or by clicking on a process instance in the *Run* tab of the sidebar (cf. Figure 6.10). Furthermore, the user can switch back to the *BuildtimeView* page by just clicking on the “x” button of the *Instance* panel of the currently opened process model.

Process visualization Requirements REQ-11, REQ-12, REQ-13, and REQ-14 are fulfilled by the *ModelView* page. The user can select the process modeling notation by choosing between BPMN and Transit Map on the *ViewSettings* panel (cf. Figure 6.6). The implementation of these process modeling notations is discussed in detail in Section 7.2.2. The data flow is visualized as shown in Figure 6.9. The data edges can be displayed individually by clicking on the corresponding business object in the *Data* tab of the sidebar. Process views can be applied by choosing the respective view in the *ViewSettings* panel (cf. Figure 6.6). These process views can be applied during the modeling and execution of the process model. New process views can be defined in the *Process View* tab of the *Settings* page.

The requirements regarding modeling of process models (cf. Section 4.4) are met by the *ModelView* page. New process nodes can be added by dragging them from the *Add* tab of the sidebar onto the process model (cf. Figure 6.7). To fulfill Requirement REQ-15, process nodes can be edited as well as deleted through the *Properties* tab of the sidebar (cf. Figure 6.8). Process nodes can be moved from one place in the process model to another by drag and drop (cf. Requirement REQ-16). The *Data* tab of the sidebar (cf. Figure 6.9) provides all capabilities to satisfy Requirement REQ-17, i.e., business objects can be created, modified, and deleted. The process node types specified in Requirements REQ-18, REQ-19, REQ-20, and REQ-21 can be added to a process model, as already mentioned, by dragging them onto a process model (cf.

Figure 6.7). Furthermore, all specified types of process nodes can be configured through the *Properties* tab of the sidebar (cf. Figure 6.8).

The execution requirements, which are specified in Section 4.5, are fulfilled by the *GroupOverview*, *ProcessOverview*, and *ModelView* page. The overview on process instances (cf. Requirement REQ-22) is available in the *GroupOverview*, *ProcessOverview*, and *ModelView* page. The overview can be accessed by opening the *Run* tab of the sidebar. The listed process instances are filtered according to the current context. For example, if the overview is opened in the *ModelView* page, all instances of the currently shown process model are listed. A specific process instance can be displayed by clicking on the corresponding *Instance* panel (cf. Figure 6.10), which leads to Requirement REQ-23. Figure 6.10 shows how the *ModelView* page displays a process instance (cf. Requirement REQ-23). A task can be executed by simply clicking on a process node (cf. Requirement REQ-24). Subsequently, the user form for the task is shown in the sidebar and the task can be completed (cf. Figure 6.11).

Section 5 and Section 6 show how the requirements specified in Section 4 (except Requirement REQ-5, which is discussed in Section 7.1.1) are implemented by Clavii BPM cloud. The following section singles out implementation aspects, which play an important role in fulfilling the requirements.

7

Implementation Aspects of the Web Application

Previous sections introduce general insights into the web application of Clavii BPM cloud by discussing its architecture and user interface. In addition, it is analyzed how the requirements specified in Section 4 are met by Clavii BPM cloud. In the following, aspects of the implementation of the web application of Clavii BPM cloud are introduced. General aspects in the context of navigation between pages, event handling, request management, and data storage are addressed (cf. Section 7.1). Next, user interface aspects are discussed in Section 7.2. In particular, insights into the implementation of the Group- and ProcessOverview page, ModelView page, Sidebar component, ViewSettings panel, as well as the localization are given. Finally, Section 7.3 summarizes this section.

7.1. General Implementation Aspects

As already discussed, the web application of Clavii BPM cloud is built utilizing GWT (cf. Section 2.2). In particular, GWT injection is leveraged to achieve a high degree of modularization. Therefore, general components, explained in the following, are injected into user interface components. As a result, user interface components are decoupled. This leads to cleaner code and higher maintainability.

Section 7.1.1 discusses the implementation of the Navigator component, which is responsible for the navigation between web pages. Subsequently, the propagation of events between components with the Event Bus component is explained in Section 7.1.2. Section 7.1.3 shows how the data on the client-side of the web application is stored. Finally, Section 7.1.4 discusses the request management, which bundles the requests to the server-side on the client-side of the web application.

7.1.1. Navigator Component

The *Navigator* component, which consists of the *Navigator* class, is responsible for the navigation between pages of the web application of Clavii BPM cloud. As stated in Requirement REQ-5, a URL referencing a BPMS should always reflect the current state of a BPMS, to enable the users to share this URL and, thus, share the currently opened group or process model. Because in AJAX applications new pages are loaded in the background, the URL is not automatically adjusted. Therefore, it is necessary to adjust the URL every time a user navigates to another page. Furthermore, if a user opens a bookmark in the browser, it is necessary to open the corresponding page. For example, Figure 7.1 shows the internal chain of events to open the URL:



First of all, method *onModuleLoad* of the *Clavii* class is invoked. The *Clavii* class is specified as entrypoint in the *.gwt.xml* file (cf. Section 2.2.1). The *onModuleLoad*

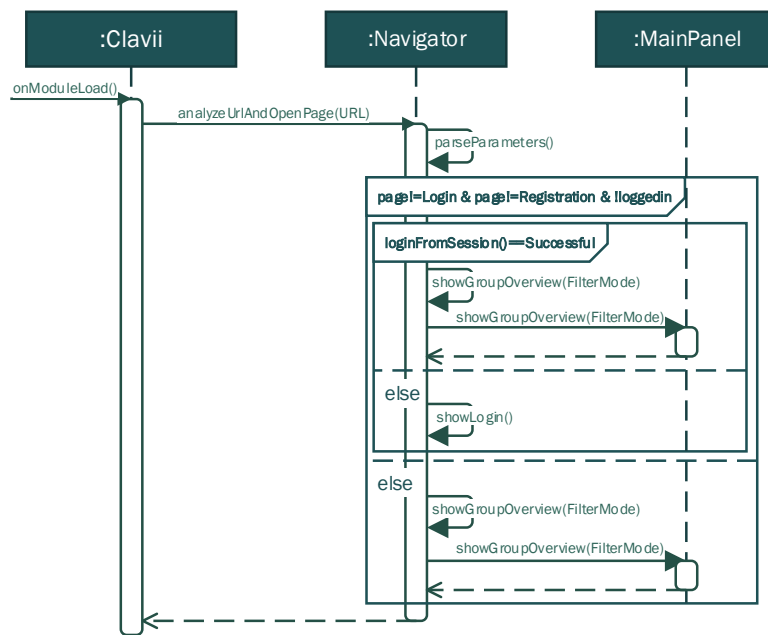


Figure 7.1.: Opening a Page based on a specific URL

method calls the *analyzeUrlAndOpenPage* method in the *Navigator* class. This method is also called if a URL changes after a user input. Next, the *Navigator* class parses parameters of the current URL. In Figure 7.1, the only parameter is *filter* with value *Private* (i.e., *filter=Private*). Particularly, *filter* specifies the filter of the *GroupOverview* page and defines if private, shared, or all groups are displayed. The page identifier is placed in front of the parameters in the URL, i.e., identifier *Main* corresponds to *GroupOverview* page (cf. Section 6.2.3).

Afterwards, it is checked if the requested page is a page of the *Main Pages* region. If the user is not logged in, the *Navigator* class looks for a cookie in the browser cache and tries to log in based on the credentials in the cache. After a successful login, the *GroupOverview* page is shown. Otherwise, the user is redirected to the *Login* page. If the user only wants to open the *Registration* or *Login* page, the requested page is opened directly, because no authentication is required.

Figure 7.1 presents a simplified view of the real procedure. For example, during the login some basic data, like organizational model and plugins, are loaded. Furthermore, it is

7. Implementation Aspects of the Web Application

checked if the language in the URL is the same as the language specified for the user. If this is not the case, the page is reloaded with the right language. Furthermore, several clean-up methods are executed. For instance, potentially opened sidebars and popup windows are closed.

The navigation between web pages is centralized in one component, because otherwise redundant code would be necessary in the implementation of individual web pages. The implementations of the web pages just have to analyze the user input and call the respective method on the *Navigator* class to switch to another page. Furthermore, the methods in the *Navigator* class contain common code, which can be reused for every page switch. Another advantage of the *Navigator* component, is that the methods can be reused when a web page should be opened based on a specific URL (cf. Figure 7.1).

7.1.2. Event Bus Component

Event propagation between components of the client-side is implemented by the *Event Bus* component. The *Event Bus* component facilitates the decoupling of components, i.e., the components do not have to hold references to each other to exchange information, because they can communicate through events over the *Event Bus* component. The *Event Bus* component uses the *publish/subscribe* pattern for event propagation [EFGK03], i.e., several components can subscribe to a specific event. The latter may be invoked by a component and, subsequently, all subscribed components are notified. Figure 7.2 displays the basic principle of the implemented *Event Bus* component in the context of this thesis.

Each user interface component, which visualizes an entity, e.g., a group or a process model, has a subscription for corresponding events. To keep the visualization of the entities up-to-date, *RequestManager* class publishes the respective event when an entity is changed. Thereby, the user interface is updated to represent the new state of the entity. For more details it may be referred to Section 7.2.

The implementation of the *Event Bus* component is provided by GWT. Several *Event* and *EventHandler* classes are created to leverage the capabilities of the *Event Bus*

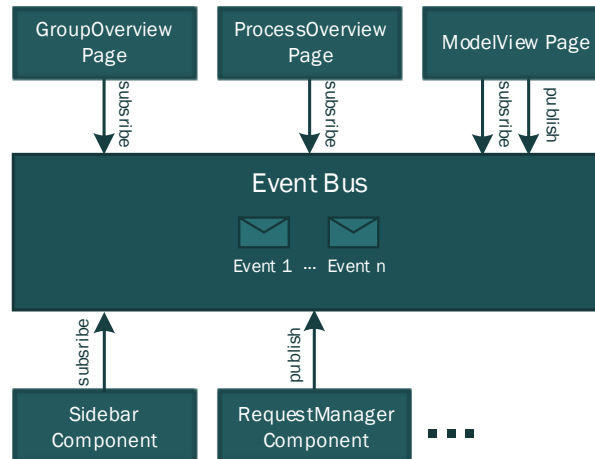


Figure 7.2.: Event Bus Component

component. Table 7.1 lists the events, which are published when the corresponding entity is changed. The only exception is that the *NodeSelectionUpdate* event is published if the selection of nodes in the *ModelView* page changes.

Table 7.1.: Events for Propagation of Entity Updates and changed Node Selection of a Process Model

Event Name	Invoked if
AgentUpdate	an <i>Agent</i> entity is changed
ModelMetaDataUpdate	a <i>ModelMetaData</i> entity is changed
ModelUpdate	a <i>Model</i> entity is changed
FilterUpdate	a <i>Filter</i> entity is changed
NodeSelectionUpdate	selected nodes of a process model are changed
GroupUpdate	a <i>Group</i> entity is changed
OrganizationalEntityUpdate	an organizational entity, e.g., a <i>Group</i> , is changed
PluginUpdate	a <i>Plugin</i> entity is changed
ProcessInstanceUpdate	a <i>ProcessInstance</i> entity is changed
TypeUpdate	a <i>Type</i> entity is changed
TypeInstanceUpdate	a <i>TypeInstance</i> entity is changed

A typical use case of *Event Bus* component is the propagation of the *NodeSelection* event (cf. Figure 7.3). First of all, the *LienzoWidget*, which is responsible for process visualization, receives a *NodeClick* event. The latter is invoked when a user clicks on a node of a process model. Thereafter, the *notifyFlowNodeSelectionHandler* method is called. This method creates a new *NodeSelectionUpdate* event, comprising references

7. Implementation Aspects of the Web Application

to all selected nodes. Subsequently, this event is published on the *Event Bus* component using the *fireEvent* method.

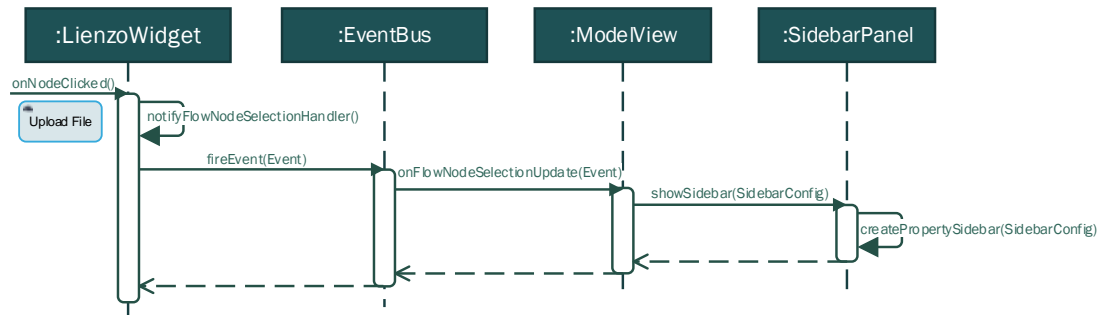


Figure 7.3.: NodeSelection Event Propagation

Since the *ModelView* page has a subscription on *NodeSelection* events, it is notified that a new event has been published. As a result, the *ModelView* page builds a *SidebarConfig* object and calls the *showSidebar* method of the *SidebarPanel* class. The *SidebarConfig* object contains information about the desired sidebar content. The *SidebarPanel* class analyzes the configuration in *SidebarConfig* object, and decides that a *PropertySidebar* sidebar should be opened (cf. Section 7.2.3). A more complex example, showing the capabilities of the integration of *DataStore* class, *ProcessModelManager* class and *Event Bus* component, is given in Section 7.1.4.

The event propagation between components is implemented by the *Event Bus* component, to facilitate decoupling of components. Otherwise, a component which wants to be notified when an event occurs has to register an event handler on every component which invokes the event. Whereby, it is difficult to find out all components which invoke a specific event. For example, if a component wants to be notified when an *AgentUpdate* event is invoked, it is necessary to find out which components invoke this event. Furthermore, if a component is updated that it newly invokes the *AgentUpdate* event, all components which are interested in the event have to be determined and updated.

This problem is solved by a central *Event Bus* component, through which a component can invoke events and register event handlers in a decoupled way. The *Event Bus* component also provides methods to retrieve all components which are registered to a specific event, to facilitate the usage of the *Event Bus* component for the developer.

7.1.3. Data Storage Component

The data storage on the client-side is centralized in the *Data Storage* component. Whereby, the *DataStore* class is the main class of the *Data Storage* component. All data, retrieved from the server-side, is stored and, thereby, cached in the *DataStore* class. As a result, the *DataStore* class provides uniform data access for all components and redundant requests to the server-side can be avoided. Figure 7.4 visualizes how the *DataStore* class is leveraged by user interface components.

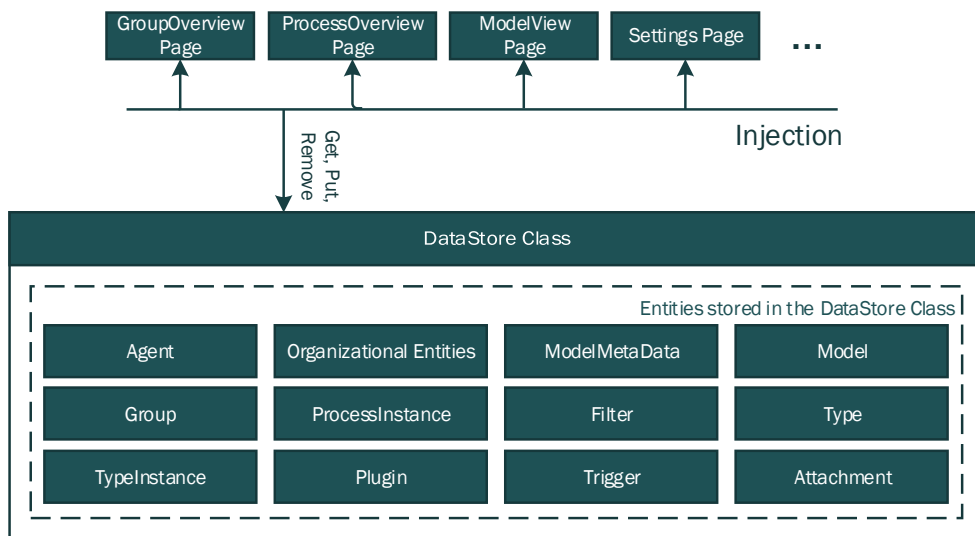


Figure 7.4.: Usage of the DataStore Class

DataStore class stores all entities utilizing hash maps, to provide an efficient access to the entities by an unique identifier. *DataStore* class also stores the current state of the web application, for example, which process model a user has opened. The *DataStore* class is injected through GIN into the user interface components to provide uniform access to all entities on the client-side (cf. Section 2.2.2). Furthermore, *DataStore* class provides methods to store, retrieve, and remove entities. For example, *GroupOverview* page can inject the *DataStore* class and read all groups.

In addition, the web storage of the browser is used to store simple settings [Hic13]. This includes the zoom factor of the *GroupOverview* page, the *ProcessOverview* page, and the *ModelView* page, and preferred process notation. Otherwise, a user has to set his

7. Implementation Aspects of the Web Application

preferred zoom factor and process modeling notation every time he opens a process model. The access to the web storage is encapsulated in the *LocalStorage* class. Listing 7.1 demonstrates the storage of a property in the web storage utilizing GWT. If a browser does not support web storage, default values are used.

```
public class LocalStorage {
    private static final double MODELL_SCALE = 1;
    public static void setModelIScale(double scale) {
        if (localStorage == null)
            localStorage = Storage.getLocalStorageIfSupported();
        if (localStorage != null) {
            localStorage.setItem("modelscale", scale + "");
        }
    }
    public static double getModelIScale() {
        if (localStorage == null)
            localStorage = Storage.getLocalStorageIfSupported();
        if (localStorage == null)
            return MODELL_SCALE;
        return Double.parseDouble(localStorage.getItem("modelscale"));
    }
}
```

Listing 7.1: Accessing the Web Storage of a Browser utilizing GWT

In summary, all data retrieved from the server-side are stored in the *DataStore* class. In addition, simple settings for the convenience of the user are stored in the web storage of the users browser.

The reason for the centralized storage in the *DataStore* class is that it is difficult to maintain an overview on all stored entities, when they are stored distributed in user interface components. For example, if the *ModelMetaData* entity is stored in the *ProcessOverview* page, a component which wants to access a *ModelMetaData* entity, e.g., the *ModelView* page, has to retrieve the entity from the *ProcessOverview* page. This forces the developer to access the entity through the respective user interface component, even if the component is not available, because it has not been created yet. Another alternative to the centralized data storage would be to store the entities redundant in multiple user interface components. This is not feasible because additional effort would be necessary to synchronize these entities if one of them is changed, to keep all entities up-to-date.

7.1.4. Request Management

Client-server communication is implemented based on RPCs and RequestFactory. Utility classes *RequestManager* and *ProcessModelManager* encapsulate the implementation for the communication between client and server. Details about the implementation of individual requests can be found in Section 2.2.2. Figure 7.5 lists the entities that can be modified and the operations that can be executed with these classes.

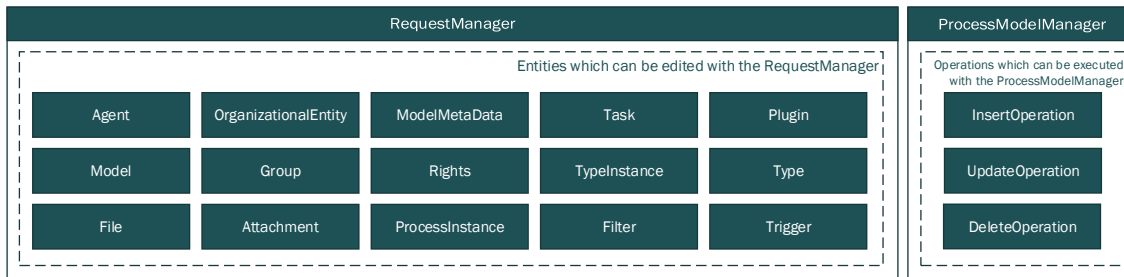


Figure 7.5.: RequestManager and ProcessModelManager Classes

To explain how the utility classes are leveraged, Figure 7.6 shows the cooperation between *DataStore* class, *Event Bus* component, and *ProcessModelManager* class on the basis of the *insertNode* method. The *insertNode* method of the *ProcessModelManager* class is invoked when a new process node is dragged onto a process model.

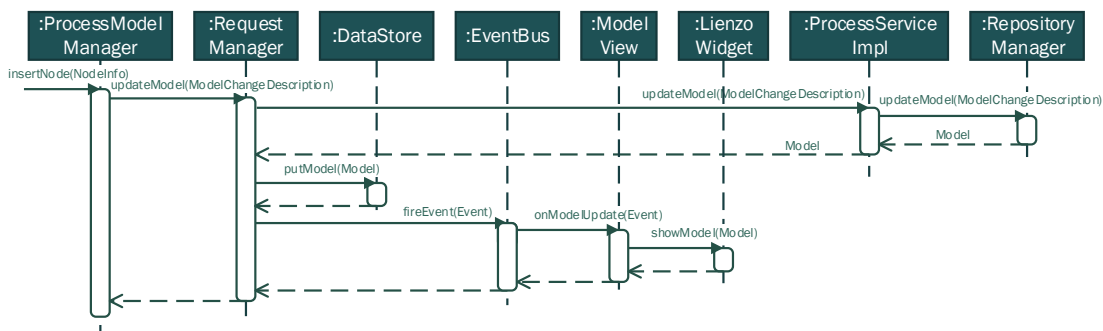


Figure 7.6.: Integration of DataStore Class, ProcessModelManager Class, and Event Bus Component

When the *insertNode* method is called, the *ProcessModelManager* class calls the *updateModel* method of the *RequestManager* class. Whereby, required data to insert a

7. Implementation Aspects of the Web Application

new process node is handed over to the *RequestManager* class. This includes, particularly, predecessor and successor node, node type, and node name of the process node to be inserted. The *RequestManager* class invokes an RPC on the server-side of Clavii BPM cloud (cf. Section 2.2.2). Then, *RepositoryManager* class updates the process model, saves the updated process model to the database, and delivers it back to the client-side of Clavii BPM cloud. Subsequently, the *RequestManager* class stores the updated version of the process model in the *DataStore* class. Next, a *ModelUpdate* event is fired on the *Event Bus* component. Thereby, the *ModelView* page is notified that an updated version of the process model is available. Therefore, it calls the *LienzoWidget* to update the process visualization.

In summary, the classes *RequestManager* and *ProcessModelManager* encapsulate all required methods to modify entities on the server-side. The advantage of centralized requests is that they can be accessed and maintained in one central place. Clavii BPM cloud uses various requests and it is easier to keep an overview on all requests for the developer, if they are bundled in one component. Furthermore, if every user interface component contains code to communicate with the server, there would be a lot of redundant code, which makes the user interface components unnecessarily complex.

The components for centralized navigation, event propagation, data storage, and request management, which have been discussed in this section, are injected in user interface components to provide all general functionality required for the implementation of the pages of the web application.

7.2. User Interface Implementation Aspects

Based on the user interface of the web application shown in Section 6, this section describes implementation details of selected user interface components. The user interface is implemented by combining UiBinder (cf. Section 2.2.2) and CSS, which are integrated through ClientBundles (cf. Section 2.2.2).

First, Section 7.2.1 discusses the structure of the *GroupOverview* and *ProcessOverview* page. Subsequently, the implementation of the *ModelView* page including the process

visualization is explained in Section 7.2.2. Subsequently, Section 7.2.3 illustrates the structure of the *Sidebar* component. Thereafter, *ViewSettings* panel is discussed in Section 7.2.4. Finally, the implementation of the localization is explained in Section 7.2.5.

7.2.1. Structure of GroupOverview and ProcessOverview Page

The *GroupOverview* page is implemented by the *GroupOverviewPanel* class. Since *GroupOverview* and *ProcessOverview* page have the same structure, only the implementation of the *GroupOverview* page is exemplarily described.

GroupOverview, *ProcessOverview*, and *ModelView* page are all embedded in the so-called *Main* panel. The *Main* panel UiBinder file is shown in Listing 7.2 and Figure 7.7 displays the corresponding user interface.

```
<g:HTMLPanel>
  <t:TopBarPanel ui:field="Topbar" /> (1)
  <s:SidebarPanel ui:field="Sidebar" /> (2)
  <g:HTMLPanel ui:field="SidebarLeftContent" visible="false" /> (3)
  <g:HTMLPanel ui:field="MainContent" /> (4)
  <v:ViewSettingsPanel ui:field="ViewSettings" /> (5)
</g:HTMLPanel>
```

Listing 7.2: Main Panel UiBinder file

Main panel arranges the omnipresent panels *TopBar* panel ①, *Sidebar* panel ②, *SidebarLeftContent* panel ③, and *ViewSettings* panel ⑤. The *MainContent* panel ④ is reserved for *GroupOverview*, *ProcessOverview*, and *ModelView* page. Because the *Main* panel is the basis of all these pages, they share all the same structure. Therefore, omnipresent panels like the *TopBar* panel can be provided to all pages without any redundant code.

TopBar panel ① describes the bar at the top of a page, which contains the breadcrumb, the name of the logged in user, and additional navigation options. *Sidebar* panel ② is the header of the sidebar and is always visible. The content of the *Sidebar* panel is updated depending on the current page displayed and access rights of the current user. When a sidebar is opened, it is shown in the *SidebarLeftContent* panel ③. The

7. Implementation Aspects of the Web Application

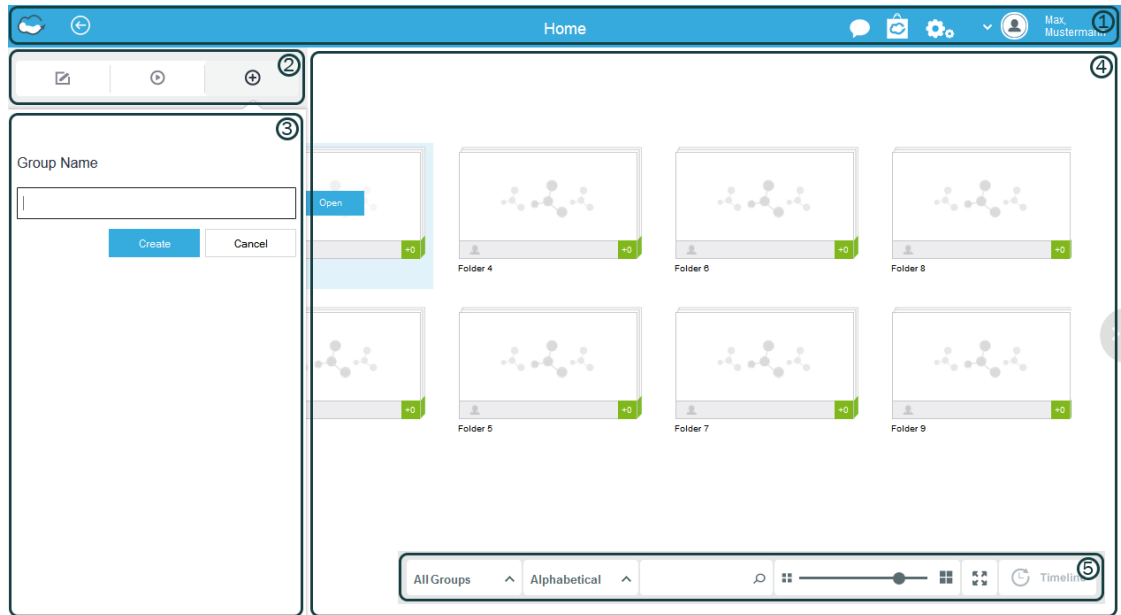


Figure 7.7.: User Interface of the Main Panel

ViewSettings panel ⑤ provides view options for the current page, e.g., filtering, sorting, or zooming.

These four panels are available in all pages of the *Main Pages* region. The *MainContent* panel ④ is filled depending on the current page (cf. Figure 7.7). When a user navigates, for example, from the *ProcessOverview* page to the *ModelView* page the *Navigator* class replaces the content of the *MainContent* panel. Other panels are updated accordingly. In this case, the breadcrumb in the *TopBar* panel is updated, the *Sidebar* panel gets an additional *Data* tab, and *ViewSettings* panel adds the possibility to switch between process modeling notations. Because only affected parts of pages are replaced or updated instead of entire pages, faster loading times of web pages are achieved.

Listing 7.3 shows the internal structure of the *GroupOverview* page and Figure 7.8 displays the user interface of the *GroupOverview* page. Groups are visualized through *GroupTile* panels ⑥, which are arranged on the *Inner* panel ②. *GroupTile* panels are encapsulated in an extra *UiBinder* file because it is not possible to specify dynamic components in a *UiBinder* file, which is necessary because the amount and types of groups which are shown to a user are dynamic. The *Inner* panel takes as much space

7.2. User Interface Implementation Aspects

as necessary to visualize all groups. The *Outer* panel represents the visible viewport of the *Inner* panel. Therefore, the overlapping parts of the *Inner* panel are clipped. The viewport itself can be moved either by clicking on the arrows on the side ③④ or by scrolling using the mouse wheel.

```
<g:HTMLPanel>
  <g:FocusPanel ui:field="OuterPanel"> (1)
    <g:AbsolutePanel ui:field="Inner" /> (2)
  </g:FocusPanel>
  <c:Image ui:field="LeftArrow" resource="{i.leftarrow}" /> (3)
  <c:Image ui:field="RightArrow" resource="{i.rightarrow}" /> (4)
  <g:HTMLPanel ui:field="Selection" /> (5)
</g:HTMLPanel>
```

Listing 7.3: GroupOverview UiBinder File



Figure 7.8.: User Interface of the GroupOverview Page

Groups can be selected by clicking on the *GroupTile* panels. In addition, the *Selection* panel ⑤ can be used to select several groups at once analog to the selection frame in desktop applications. The *Selection* panel facilitates the administration of multiple groups for user which are accustomed to desktop applications.

In the following, the composition of the *GroupTile* panels is broken down. The *GroupTile* panel is defined with UiBinder (cf. Listing 7.4) and dynamically instantiated by the *GroupOverviewPanel* class. Figure 7.9 illustrates the user interface of the *GroupTile* panel.

7. Implementation Aspects of the Web Application

```
<g:HTMLPanel>
  <g:HTMLPanel ui:field="Options"> (1)
    <g:Button ui:field="Open" />
  </g:HTMLPanel>
  <g:HTMLPanel ui:field="Icon"> (2)
    <c:Image ui:field="GroupIcon" />
  </g:HTMLPanel>
  <g:HTMLPanel ui:field="Bottom" > (3)
    <c:Image ui:field="SharedStateIcon" />
    <g:HTMLPanel ui:field="GroupCount" >
      <c:Label ui:field="GroupCountLabel" text="+0" />
    </g:HTMLPanel>
  </g:HTMLPanel>
  <c:Label ui:field="GroupName" /> (4)
</g:HTMLPanel>
```

Listing 7.4: GroupTile Panel UiBinder File

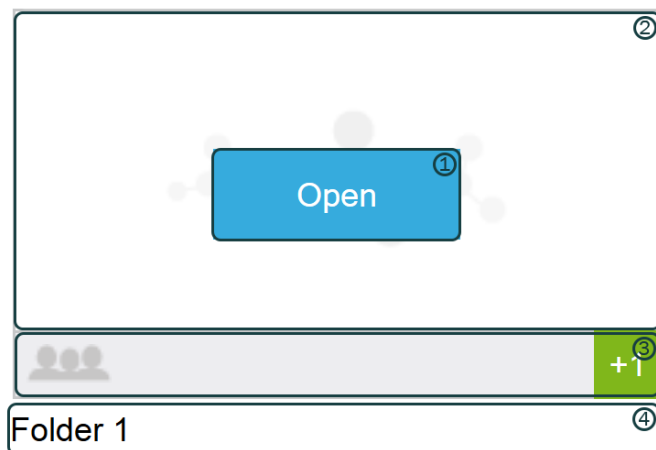


Figure 7.9.: User Interface of the GroupTile Panel

The *Options* panel ① is only visible when the mouse is moved over a group on the *GroupOverview* page or when a group is selected. A click on the *Open* button results in invoking the *Navigator* class, which then opens the *ProcessOverview* page. The *Icon* panel ② contains the icon of a group. The *Bottom* panel ③ contains the *SharedStateIcon* panel, which signals whether a group is private or shared. It also encloses the *GroupCount* panel, to report the count of opened tasks for the current user in this group. The *GroupName* label ④ displays the group name.

Previous paragraphs introduce the building blocks of *GroupOverview* page. Figure 7.10 illustrates the chain of events when a new group is created through the *Add* tab of the sidebar (cf. Figure 7.7). The corresponding panel for the content of this tab in the *GroupOverview* page is the *NewGroupContent* panel. At first, the *NewGroupContent* panel calls the *createGroup* method of the *RequestManager* class. Then, the group is created on the server-side of Clavii BPM cloud and returned to the client. Next, the created group is stored in the *DataStore* class and a *GroupUpdate* event is invoked.

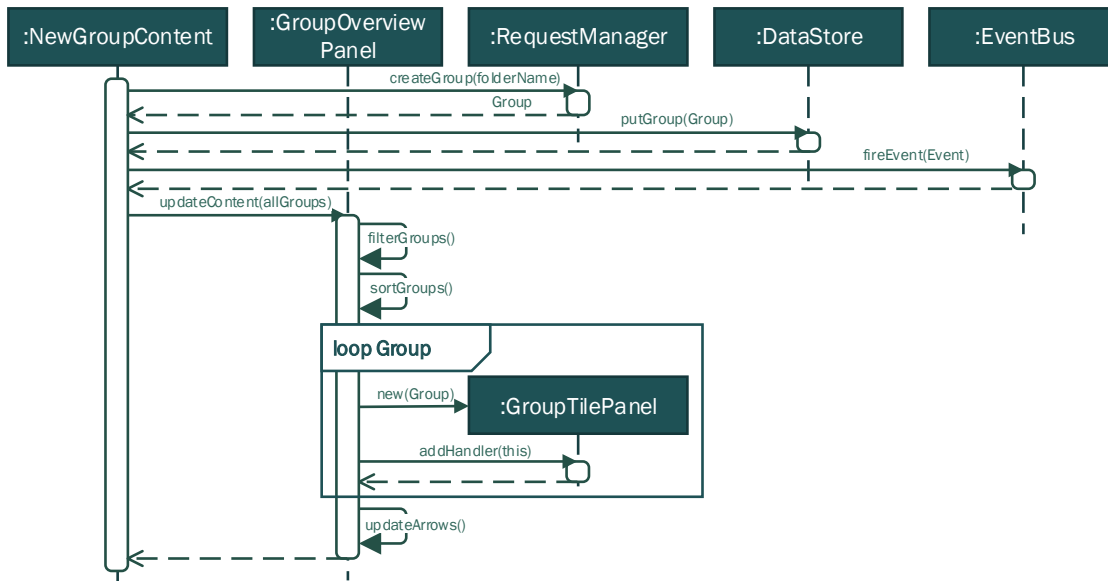


Figure 7.10.: Creating a New Group of Process Models

To show the created group to the user, the *GroupOverviewPanel* is updated. Therefore, the *updateContent* method is called containing a list of all groups as parameter. In the following, groups are sorted and filtered according to selected policies of the *ViewSettings* panel. Afterwards, groups are arranged in the *Inner* panel. Therefore, *GroupTile* panels are instantiated and stored in a cache. However, if they already exist in the cache, they are reused and updated to optimize the time necessary to display all groups. Finally, arrows on the right and left are updated, because they should be only visible, if the *Inner* panel is wider than the *Outer* panel.

This section explains the structure of the *GroupOverview* page. The *GroupOverview* page is embedded in the *Main* panel to avoid redundant code for panels which are

7. Implementation Aspects of the Web Application

displayed on all pages of the *MainPages* region, e.g. *TopBar* panel. The individual groups are represented by *GroupTile* panels, which can be dynamically created by the *GroupOverviewPanel* class. Finally, it is shown how a new group is created by leveraging *RequestManager* class, *DataStore* class, and *Event Bus* component.

7.2.2. Details on Implementation of ModelView Page

The *ModelView* page is like the overview panels, enclosed by the *Main* panel. This allows the reuse of *TopBar* panel, *Sidebar* panel, and *ViewSettings* panel, which avoids redundant code. The *ModelView* page consists of the panels *LienzoWidget* ① and *Outline* ②. The structure of the *ModelView* page is shown in Listing 7.5 and the corresponding user interface in Figure 7.11.

```
<g:HTMLPanel>
  <l:LienzoWidget ui:field="LienzoWidget" /> (1)
  <g:HTMLPanel ui:field="Outline" > (2)
    <c:Image ui:field="OutlineImage" />
    <g:FocusPanel ui:field="Selection" /> (3)
  </g:HTMLPanel>
</g:HTMLPanel>
```

Listing 7.5: ModelView UiBinder File

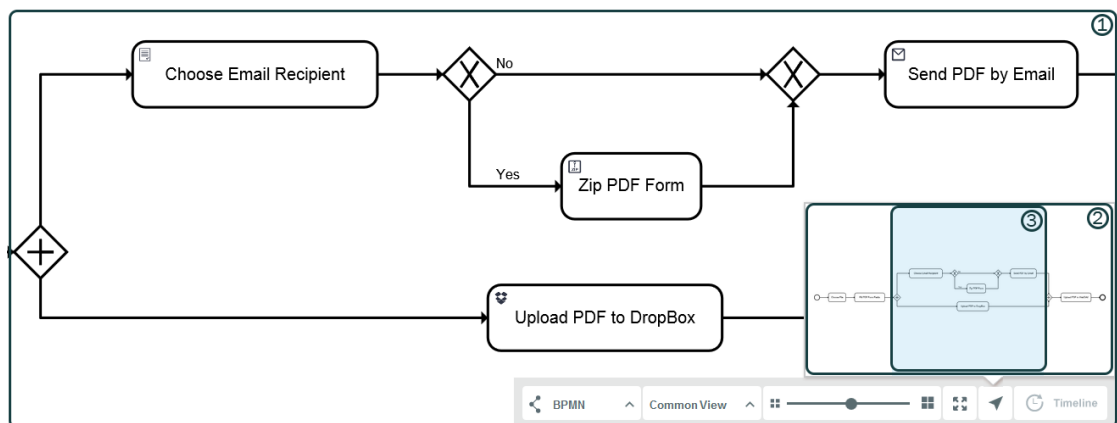


Figure 7.11.: User Interface of the ModelView Page

The *LienzoWidget* panel ① is used for visualizing process models and is discussed later in this section. The *Outline* panel ②, which can be accessed through the *ViewSettings* panel, contains a preview of the currently shown process model. Therefore, the *Outline* panel is particularly convenient for complex process models. The *Selection* panel ③ displays the current viewport of the process model shown in the *ModelView* page. Furthermore, the *Selection* panel can be moved to adjust the viewport.

In the following, it is introduced how the *LienzoWidget* panel visualizes process models and handles events. First of all, when drawing a process model in the *LienzoWidget* panel, the layout algorithms calculate the layout for the requested process modeling notation (cf. Section 7.2.2). Next, the process model is drawn in several layers (cf. Section 7.2.2). For example, the *NodeLayer* draws all nodes of a process model. The separation into layers facilitates faster drawing times, since not all elements have to be redrawn when only one node is added. The layers panels leverage so-called graphic factories (cf. Section 7.2.2) to receive notation-specific representations of process elements. When the process model is drawn, all occurring events are routed to the event controller (cf. Section 7.2.2). The event controller is the central component for internal event processing in the *ModelView* page. The previously discussed *Event Bus* component is not used here, because the events should not be propagated between multiple components and the event handlers are all attached to only one central event controller. The individual aspects of the process visualization are explained in the following.

Algorithms for Layouting Process Models

Layout algorithms are responsible for calculating the layout of a process model, based on a *Model* entity. The *Model* entity, is a Java object representing a process model, which contains its nodes and edges. Two very similar layout algorithms have been implemented to calculate the layout for the BPMN and the Transit Map notation. The main difference between them is that the BPMN layout algorithm arranges process elements from left to right and the layout algorithm for the Transit Map notation from top to bottom. Furthermore, branches of gateway blocks are layouted centered in BPMN and

7. Implementation Aspects of the Web Application

left-justified in the Transit Map. The parts of the layout algorithms which are explained in the following are the same for both notations.

Generally, the layout algorithms are divided into two phases. First, the structure of the process model is discovered. Second, the positions of process elements on the layout of a process model are calculated. The layout algorithms are explained based on the example in Figure 7.12.

Discovering the structure of the process model. During the first phase, the model is divided in *blocks* and *branches*. The discovered *blocks* and *branches* are stored in the so-called *root block*, which begins at the start event ① and ends at the end event ⑪. The layout algorithms create a *branch* for every direct successor of the start node of a *block*, i.e., the *root block* contains a branch, which starts at ② and ends at ⑩. Then, the next successor ③ is examined. Since the successor is a gateway, a new block is created. The following branches are stored in the previously created block and are examined subsequently. The layout algorithms terminate when the entire process model is discovered. Additionally, the width and height of every block and branch is calculated and stored.

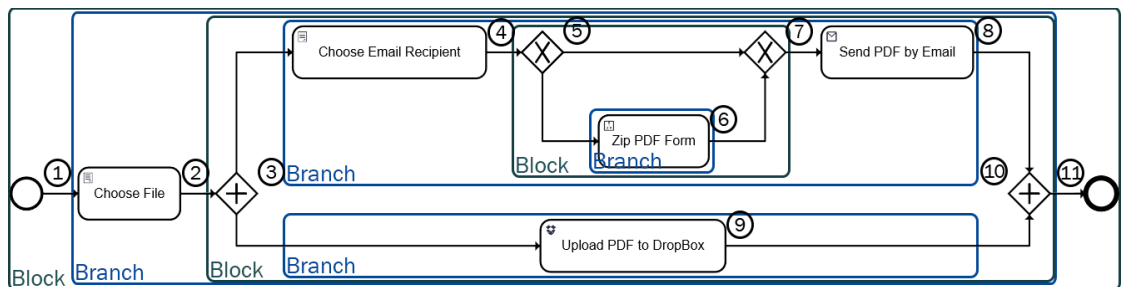


Figure 7.12.: Process Model Structure Discovery

Arranging the process elements. Figure 7.13 presents the resulting data structure of phase one of the layout algorithms (cf. Figure 7.12). The second phase arranges process elements recursively, starting at the *root block*. First, the *arrangeBlock* method sets the position of the start node ① and end node ⑪ of the *root block*. These coordinates are calculated from the start point of the process model and the width and height of the *root block*, which have been determined in the first phase. Second, the starting points of the branches, contained in the *root block*, are calculated and the *arrangeBranch* method

is called for each branch in the *root block* with the branch and the starting point of the branch as parameters.

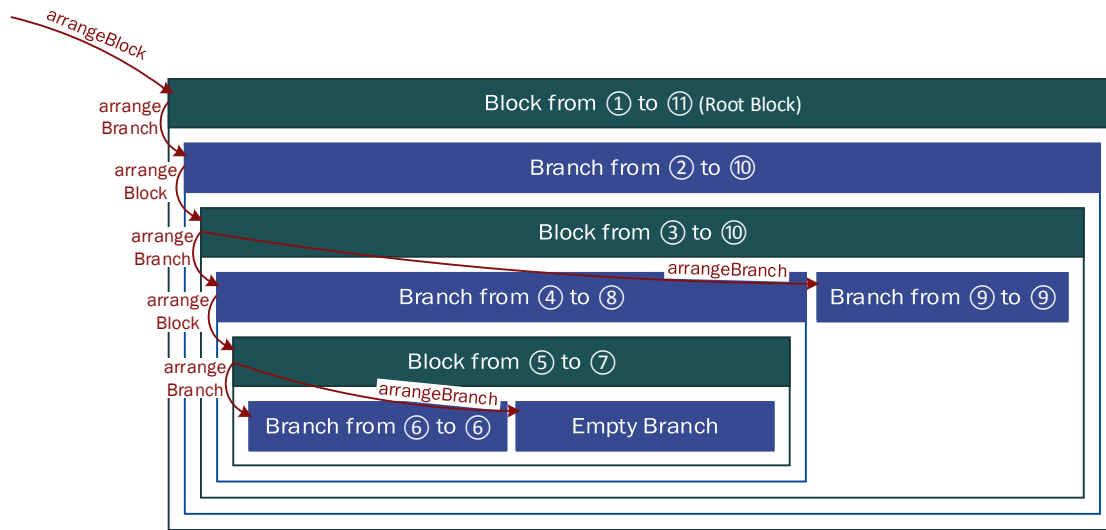


Figure 7.13.: Arranging the Process Elements

The *arrangeBranch* method positions all process nodes in a branch. The first node is positioned directly on the coordinates of the start point of the branch. Following process nodes are placed next to the first node depending on the process modeling notation, with a predefined margin. When a contained block is detected, e.g., block from ③ to ⑩ in branch ② to ⑩, the *arrangeBlock* method is called recursively.

The data structure, which is the result of the first phase, is processed until the entire layout of the process model is calculated. As already stated, the difference between the BPMN and Transit Map algorithm lies only in the recursive placement of the blocks, branches, and nodes. The calculated process model layout is stored in *ProcessElement* objects. To be precise, coordinates, width, and height of every node is stored. This data is used to draw the process model on the layers.

Drawing the Process Model on the Layers

After the layout has been calculated, the process model is drawn originating from the *ProcessElement* objects. The process model is drawn on a *LienzoPanel* panel, which is

7. Implementation Aspects of the Web Application

part of the Lienz graphics library [Emi]. By utilizing Lienz it is possible to separate the visualization of a process model into several layers. The advantage of this separation is that layers can be individually redrawn by the *LienzoPanel* panel, which results in shorter drawing times for a process model. Figure 7.14 illustrates the overlay of the layers, which assembles the process visualization. This example displays the *NodeLayer*, *SequenceFlowLayer*, and *DataLayer*.

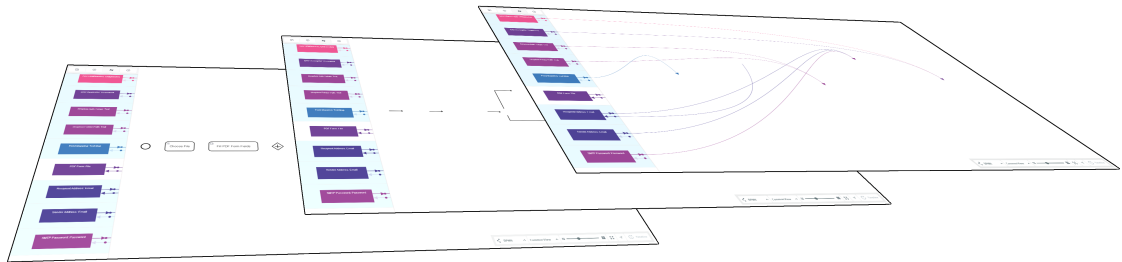


Figure 7.14.: Layers of the Process Visualization

The illustrated layers in Figure 7.14 are a subset of the layers used for process model visualization. In the following, all involved layers and their functionality are described:

NodeLayer. The *NodeLayer* draws process elements (i.e., events, tasks, and gateways). The coordinates for process elements have been calculated by the layout algorithm. The graphical representation that is drawn onto the *NodeLayer*, is produced by the graphic factories, which are described in Section 7.2.2. In addition, *NodeLayer* is capable of updating the current process model visualization when, for example, a new node is added to the process model. Therefore, *NodeLayer* calculates the difference between the old and the new layout and starts a set of animations for a fluent transition. Whereby, the animations highlight the differences between the old and the new process model.

SequenceFlowLayer. The *SequenceFlowLayer* draws edges between process elements. First, the set of edges is fetched from the *Model* entity. Thereafter, positions of source and target node are retrieved and the shape of the edge is determined and drawn. This layer also supports the transition from one layout to another, when a new node is inserted, like the *NodeLayer*.

DataLayer. The *DataLayer* is only visible when the *Data* tab in the sidebar is opened. The *DataLayer* displays the data connections between business objects and process

nodes. The data edge that is shown during the creation of new data connections is also drawn on this layer. In particular, only the *DataLayer* has to be redrawn when such a edge changes and not the complete process model.

RuntimeLayer. The *RuntimeLayer* visualizes the current state of a process instance and individual nodes when the *RuntimeView* page is active. This layer contains both process elements and edges of a process instance.

DeltaLayer. The *DeltaLayer* is used when a process view is shown and highlights differences between the original process model and a selected process view.

DragLayer. During dragging a node onto the process model from the *Add* tab of the sidebar, this node is temporarily drawn on the *DragLayer*. The *DragLayer* also draws gateways and edges, which are inserted if the dragging ends, i.e., if the user releases the mouse button. For example, when a new process node is dragged onto an existing process node a surrounding gateway is shown.

Figure 7.15 breaks down how a process model is drawn using the BPMN notation, including how the *NodeLayer* draws process elements.

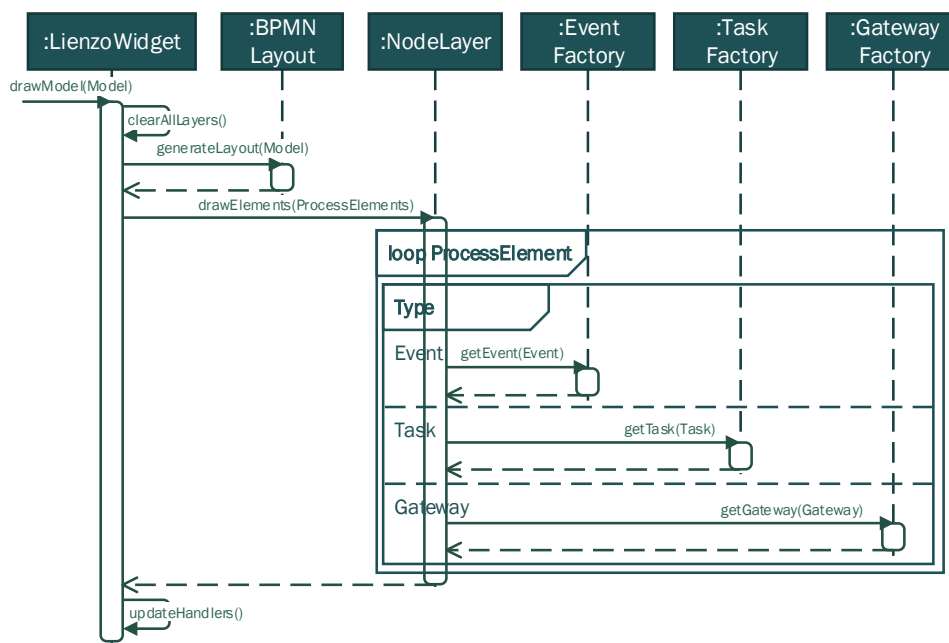


Figure 7.15.: Drawing Process Elements with the NodeLayer

7. Implementation Aspects of the Web Application

The *drawModel* method of the *LienzoWidget* class is responsible for the coordination of process model visualization. This method is called every time a user opens the *ModelView* page, a process view is activated, or the process notation is changed.

When the *drawModel* method is called the *LienzoWidget* clears all layers and, thus, removes all currently drawn objects. Subsequently, the selected process modeling notation is retrieved, in this case BPMN, and the respective layout is generated. In the following, *drawElements* methods of all layers are invoked. In Figure 7.15, only the execution of the *drawElements* method of the *NodeLayer* is displayed, for simplification reasons. The *drawElements* method iterates over all *ProcessElement* objects to retrieve the coordinates and the names of all process nodes. Depending on the type of the process element, the corresponding graphic factory is leveraged to retrieve a graphical representation of the process element. This graphical representation is used to draw the process element to the layer and, thus, to the HTML canvas. Finally, event handlers of the process elements are updated so the event controller retrieves all occurring events, which is discussed later in Section 7.2.2.

Graphic Factories

Graphic factories provide process model notation-specific representations of process elements (i.e., process nodes and edges between them). To provide centralized access to these factories, the so-called *GraphicFactory* class provides a centralized factory for the individual graphic factories. The *GraphicFactory* has methods for retrieving the *EventFactory*, *TaskFactory*, *GatewayFactory*, and *EdgeFactory*. Each factory is dedicated to the corresponding element type.

The factories build the graphical representation of process elements out of base elements, delivered by Lienzo (cf. Figure 7.16). These base elements can be categorized into two types: *Container* and *Shape* elements. *Container* elements contain zero or more *Shape* or *Group* elements. For example, the *NodeLayer*, which is also a *Container* element, contains all process nodes of a process model, whereby every process node is bundled into a single *Group* element.

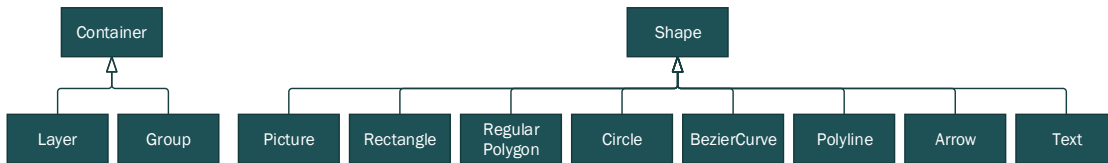


Figure 7.16.: Base Elements of Lienzo Framework

The *Group* element which represents a process node contains several *Shape* elements, which are used to assemble the visual representation of a process node. It is important to mention that event handlers can be added to every *Shape* element. The event handlers, which are used for the event controller, are attached to *Group* elements. Adding event handlers on *Shape* elements would result in hundreds of event handlers for a process model, which would have performance impacts to the entire web application. In contrast, event handlers on layers would not be able to detect which process node is affected by the event.

Figure 7.17 shows the composition of individual process elements. In general, the elements of the Transit Map are smaller to save space, which is especially useful for large process models. In return, the elements of BPMN are more detailed and the names of the elements have more space. For example, names of process nodes in BPMN are cut off at 50 characters. In contrast, names of process nodes in the Transit Map are already cut off at 30 characters.

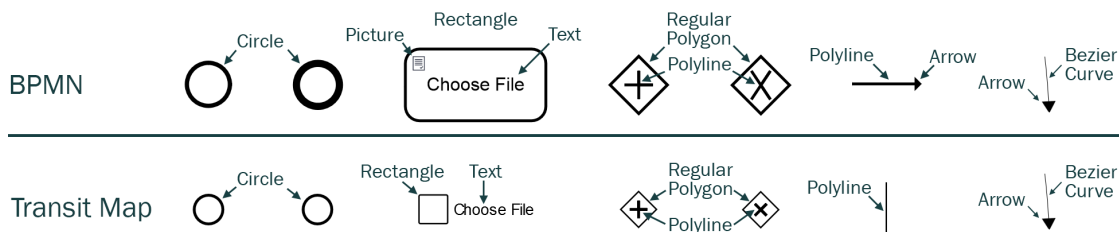


Figure 7.17.: Composition of Process Elements

Start and end events are represented by a *Circle* element. In BPMN, the circle of the end event is thicker than the circle of the start event. Tasks are represented by a *Rectangle* element and a *Text* element. In BPMN, an additional *Picture* element is used to signal the task type of the task. The AND, XOR, and LOOP gateways are composed of a

7. Implementation Aspects of the Web Application

RegularPolygon and a *Polyline* element, which has the shape of a “+” or a “x” symbol. Edges are visualized by a *Polyline* element. Whereby, edges in BPMN also contain an *Arrow* element. Data edges are in both process modeling notations composed of a *BezierCurve* and an *Arrow* element.

The graphic factories avoid redundant code by providing factories for the creation of graphical representation of process nodes and edges. Otherwise, several layer would have to implement methods to create graphical representations of process nodes and edges. For example, both the *NodeLayer* and the *DragLayer* require the capability to create graphical representations of process nodes.

Implementation of the Event Controller

As previously mentioned, all events issued by interaction with the process elements drawn on the layers are routed to the *event controller*. Figure 7.18 illustrates the interconnections of the event controller, which shows that the event controller is the central controller of the *Model/View* page, which implements the *Model-View-Controller* pattern [KP⁺88].

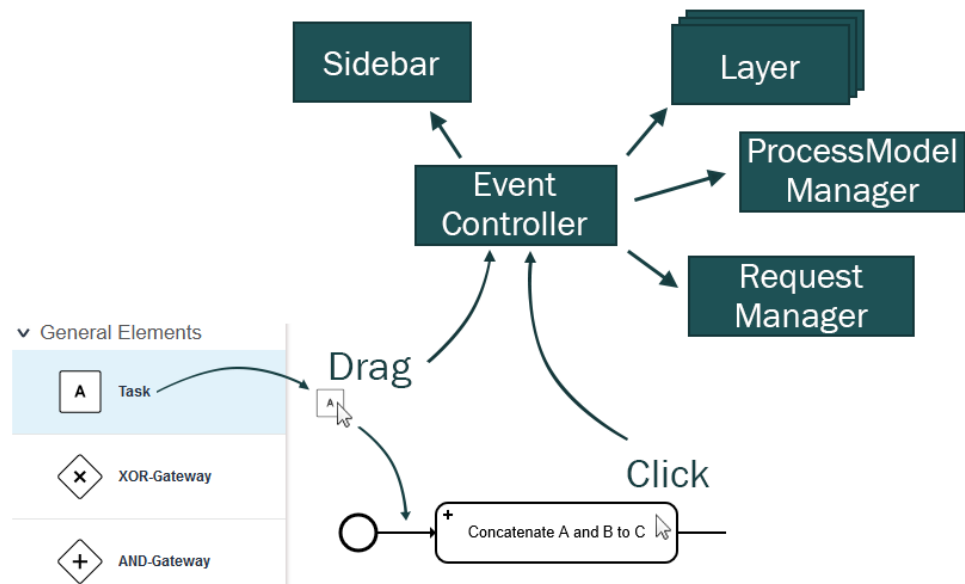


Figure 7.18.: Interconnections of the Event Controller

When events are routed to the event controller, they are processed according to their type. To be more precise, the event controller handles the event types listed in Table 7.2.

Table 7.2.: Event Types issued by Process Elements on the Layers

Type	Invoked if a user
NodeClickEvent	clicks on a process node
NodeDoubleClickEvent	double clicks on a process node
NodeDragEvent	drags a process node
NodeMouseEnterEvent	moves the mouse over an edge of the process model
NodeMouseExitEvent	moves the mouse away from an edge of the process model
DragOverEvent	drags the process node over the layers
DropEvent	drops a process node on the layers

All node events are fired by nodes and edges of the process model. Thereby, several interaction patterns are implemented. For example, clicking and double clicking on process nodes, dragging of process nodes, and fading-in the plus icon on edges between process nodes. The *DragOverEvent* and the *DropEvent* are fired when a new process node is dragged from the sidebar onto the process model.

For example, when dragging a task the *DragOverEvent* is thrown to provide context sensitive assistance to the user. When the task is dragged over an existing process node, a surrounding gateway is displayed, which represents a preview for the case the node is dropped. If the *DropEvent* occurs, the event controller determines where the user has dropped the process node. If the node is dropped on a process node the corresponding method on the *ProcessModelManager* class is called. If the node is dropped on an edge, the *insertNode* method of the *ProcessModelManager* class is called (cf. Figure 7.6).

In the following, the processing of a click on a process node by the event controller is explained. In case a user clicks on a process node on the *BuildtimeView* page the chain of events displayed in Figure 7.3 occurs. In contrast, a click on an executable task on the *RuntimeView* page leads to the chain of events illustrated in Figure 7.19.

In Figure 7.19, the *onRuntimeNodeClick* method retrieves the current state of a process instance and the task the user clicked on. If a process model is already completed,

7. Implementation Aspects of the Web Application

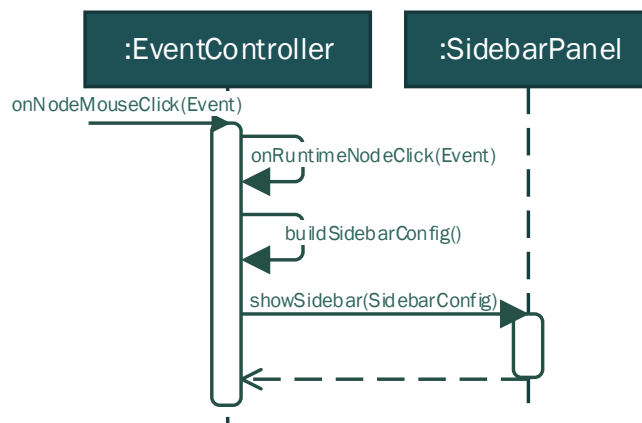


Figure 7.19.: Executing a Task

which is when the end event is executed or the process instance has been canceled, only existing comments can be read by the user. Depending on the current state of the selected node, the *SidebarConfig* object is created. The *SidebarConfig* object specifies the type of sidebar that should be opened. Finally, the *showSidebar* method is called and the sidebar is opened (cf. Section 7.2.3).

This section explains the implementation of the *ModelView* page. The *ModelView* page is like the *GroupOverview* page embedded in the *Main* panel. The *ModelView* page consists of *LienzoWidget* and *Outline* panel. The *LienzoWidget* displays the currently opened process model. The *Outline* panel contains a preview of the process model.

The visualization of the process model on the *LienzoWidget* is split in several steps. At first, the layout of the process model is calculated by the layout algorithms for BPMN and TransitMap. Next, the layout is drawn on several layers. Whereby, the most important layers are the *NodeLayer*, which draws the process nodes, and the *SequenceFlowLayer*, which draws the edges between the process nodes. The layers leverage the *graphic factories* to retrieve notation-specific graphical representation for the process nodes, which avoids redundant code in the layers. When the process model is drawn all occurring events are routed to the *event controller*. The *event controller* processes the events according to their type, e.g., if a new node should be inserted the *ProcessModelManager* class is called.

7.2.3. Structure of the Sidebar Component

The *Sidebar* component implements the sidebar on the left of all pages of the *Main Pages* region. The *Sidebar* component is build of *SidebarContent* panels (cf. Figure 7.20), which are added to the *SidebarLeftContent* panel of the *Main* panel on demand (cf. Figure 7.7). When a sidebar is opened, a *SidebarConfig* object is created and handed over to the *showSidebar* method of the *SidebarPanel* class. The *SidebarPanel* class removes any previously added content of the *SidebarLeftContent* panel and creates a new content, according to the *SidebarConfig* object. The *SidebarPanel* class is subscribed to all events on the *Event Bus* component and calls the *update* method of the current *SidebarContent* panel if necessary.

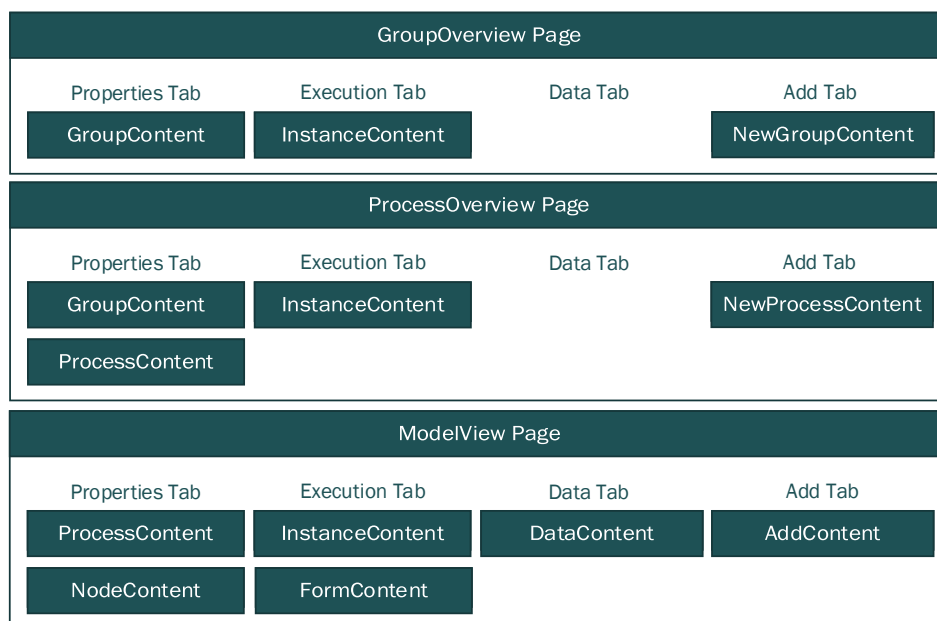


Figure 7.20.: SidebarContent Panels available in the individual Pages

SidebarContent panels are assigned to the tabs: *Properties*, *Execution*, *Data*, and *Add* (cf. Figure 7.20). The *SidebarContent* panels of the tabs are described in the following:

Properties Tab. The *Properties* tab can contain *GroupContent*, *ProcessContent*, and *NodeContent* panel. The *GroupContent* panel provides access to the properties of a group and can only be opened in the *GroupOverview* and the *ProcessOverview* page.

7. Implementation Aspects of the Web Application

ProcessContent panel shows properties of a process model and is designed analog to the *GroupContent* panel. It is only available in the *ProcessOverview* and *ModelView* page. *NodeContent* panel displays the attributes of selected process nodes in the *ModelView* page.

Execution Tab. The *Execution* tab holds *InstanceContent* and *FormContent* panel. The *InstanceContent* panel gives an overview on all running and completed process instances and is available on every page of the *Main Pages* region. Thereby, the user can always retrieve the running process instances and open them by clicking on the process instance. The *FormContent* panel is used to show forms for the task execution.

Data Tab. The *DataContent* panel is only available in the *ModelView* page, and, therefore, the *Data* tab is only shown in the *ModelView* page. The *DataContent* panel lists all business objects of the current process model. The *Data* tab enables the user to add new edges between business objects and process nodes, to remove existing edges, to set default values, and to add or remove business objects.

Add Tab. The *Add* tab has a specific content for every page of the *Main Pages* region. On the *GroupOverview* page new groups of process models can be added. On the *ProcessOverview* page new process models can be added. On the *ModelView* page new process nodes can be dragged onto the process model.

Availability of *SidebarContent* panels is also dependent on access rights of the current user. For example, if the user has only *View* access rights, the *SidebarContent* panels are always opened in read-only mode and the *Add* tab is not available.

To demonstrate how a *SidebarContent* panel is built up, Figure 7.21 illustrates the structure of the *NodeContent* panel, whereby all optional panels are surrounded by dashed lines. This example represents also the sidebar opened in Figure 7.3. *NodeContent* panel allows multiple selection of process nodes. This results in multiple instances of *NodePanel* panel; one instance for each process node. The *NodeCompositePanel* panel is shown, when more than one process node is selected and contains the common method *Delete*, for deleting all selected process nodes.

NodePanel panel contains a set of mandatory panels, which are displayed for every process node type and a few optional panels. Common panels are the *NodeIcon*, which

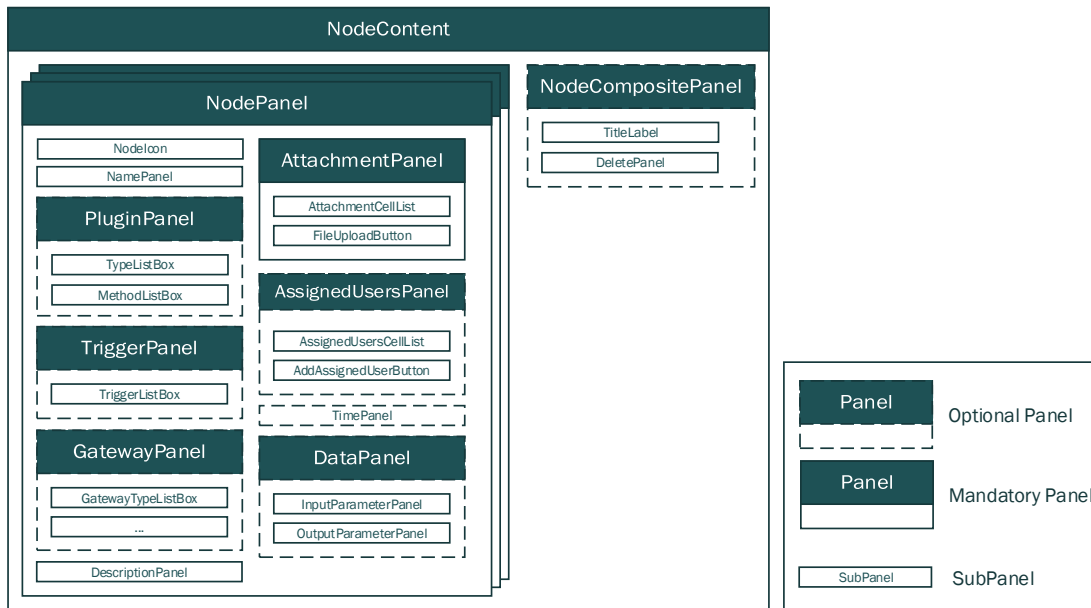


Figure 7.21.: Structure of the NodeContent Panel

signals the type of the process node, *NamePanel*, *DescriptionPanel*, and *AttachmentPanel*. The *PluginPanel* panel is displayed for service and user tasks and allows the user to switch the implementation of a process node between user forms and a specific plugin. *TriggerPanel* panel is only available for the start event. *GatewayPanel* panel is shown if a split XOR gateway is selected. Because a user task is the only task where a user assignment can be set, the *AssignedUsersPanel* panel is only visible when a user task is selected. *DataPanel* panel is displayed every time an in- or output mapping of business objects can be specified, which is the case if a user task, a service task, or a start event is selected.

This section discusses the structure of the *Sidebar* component. At first, the *SidebarContent* panels and their mapping to the tabs are explained. In the following, the structure of the *NodeContent* panel is exemplarily for all *SidebarContent* panels broken down.

The *SidebarContent* panels are implemented independent from the currently opened page to facilitate reuse in multiple pages, e.g., the *GroupContent* panel is available in the *GroupOverview* and the *ProcessOverview* page. As already mentioned, the *SidebarPanel* class is subscribed to all events on the *Event Bus* component. The

7. Implementation Aspects of the Web Application

SidebarPanel class analyzes which *SidebarContent* panel is opened and propagates the respective events if necessary. Thus, there is a central point for event handling for all *SidebarContent* panels and they do not have to subscribe to events every time they are created. As a result, the *SidebarContent* panels can be just removed when the sidebar is closed and they do not have to unsubscribe from the respective events on the *Event Bus* component. A lot of redundant code in the *SidebarContent* panels is avoided, by centralizing the interaction with the *Event Bus* component in the *SidebarPanel* class.

7.2.4. Details on Implementation of the ViewSettings Panel

The *ViewSettings* panel provides advanced view options for every page of the *Main Pages* region. The *ViewSettings* panel, which is implemented by the *ViewSettingsPanel* class, is updated, by calling the *update* method of the *ViewSettingsPanel* class every time a page of the *Main Pages* region is opened. The *update* method adjusts the *ViewSettings* panel according to the currently displayed page (cf. Figure 7.22). In addition, the *update* method retrieves the state of the page and updates the settings shown by the *ViewSettings* panel. For example, if the *GroupOverview* page is opened originating from a URL, the filter setting is adjusted according to the filter specified in the URL (cf. Figure 7.1). Individual settings of the *ViewSettings* panel have already been discussed in Section 6.

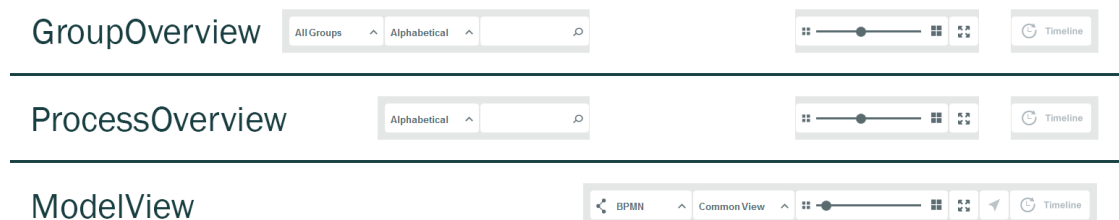


Figure 7.22.: ViewSettings Panel

Figure 7.23 shows the propagation of an updated setting to the respective user interface component. In particular, Figure 7.23 shows how a process view is applied to a process model. First of all, the user opens the context menu for process views (implemented by the *ViewContextMenuPanel* class) and selects a process view. This results in invoking

the *onClick* method. Thereafter, the active process view of the *LienzoWidget* class is set. Subsequently, the *LienzoWidget* invokes the *getModelForView* method of the *RequestManager* class. As a result, the *RequestManager* class creates an RPC, puts the process view in the *DataStore* class, and returns the process view to the *LienzoWidget* component. Subsequently, the currently shown process model is replaced with the process view.

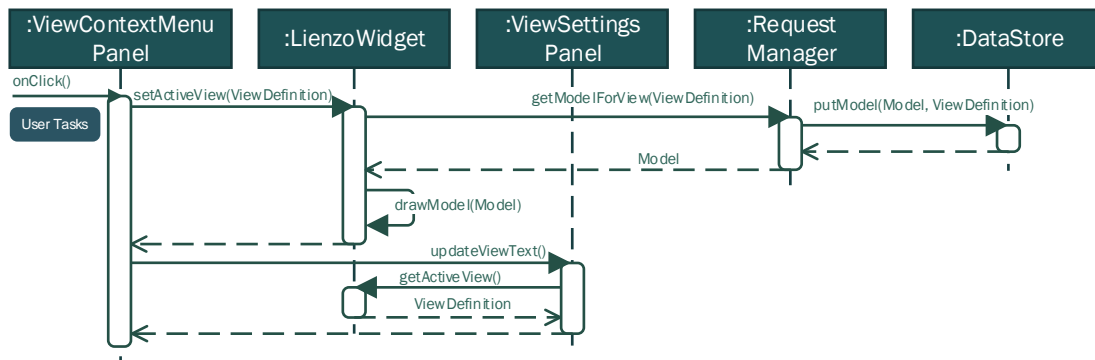


Figure 7.23.: Applying a Process View by the ViewSettings Panel

Finally, the *ViewContextMenuPanel* class calls the *updateViewText* method on the *ViewSettingsPanel* class, to reflect the changed setting on the *ViewSettings* panel. The *updateViewText* method retrieves the active process view from the *LienzoWidget* component. In this case, the process view could have been handed over from the *ViewContextMenuPanel* class. Since the *updateViewText* method is reused in the *update* method of the *ViewSettings* panel, it is necessary to retrieve the active view from the *LienzoWidget* class.

In summary, the *ViewSettings* panel has different appearances depending on the currently opened page. When the user navigates to another page, the *ViewSettings* panel is updated by calling the *update* method on the *ViewSettingsPanel* class. This results in faster loading times of the pages, because the *ViewSettings* panel does not have to be created again. Subsequently, it is discussed how a process view is applied originating from the *ViewSettings* panel.

7.2.5. Localization

The realization of the localization feature (cf. Requirement REQ-3) is illustrated in Figure 7.24. Localized text fragments are divided into the two parts: *ClaviInterface* and *ClaviHelper*. *ClaviInterface* localization contains all text fragments that are directly part of the user interface, e.g., texts of buttons or labels. *ClaviHelper* localization contains titles and descriptions of tool tips that are used to assist the user when using Clavii BPM cloud. Clavii BPM cloud delivers the German and English localizations out of the box, whereby English is the default locale.

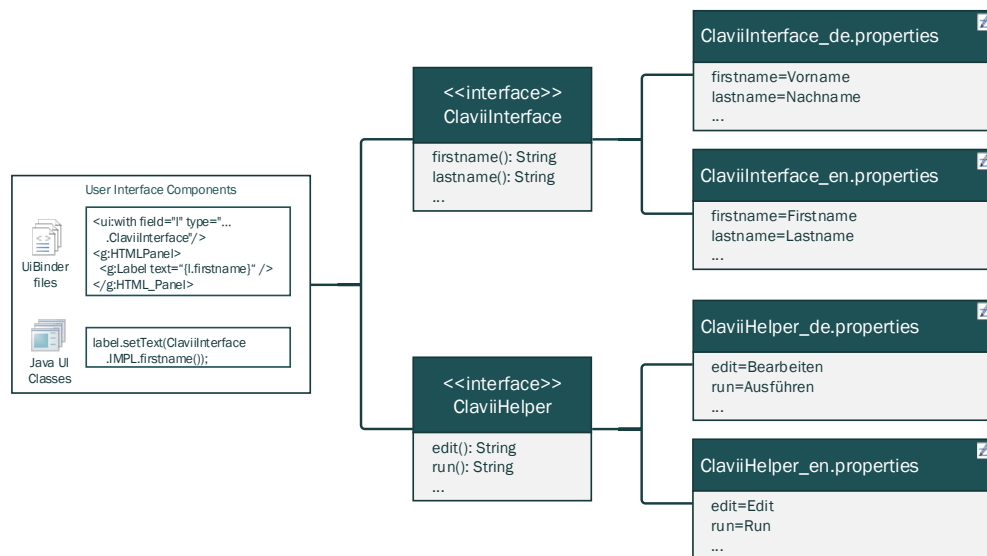


Figure 7.24.: Implementation of the Localization Feature

Localized text fragments are defined in *ClaviInterface* and *ClaviHelper* interface. Both extend the *Constants* interface of *GWT*. Every localized text fragment has a method in the interface and a translation for every language in the corresponding *.properties* file. To extend Clavii BPM cloud with additional languages, only new language files have to be provided.

Figure 7.24 demonstrates how localized text can be accessed in the user interface components. In UiBinder files, the respective interface of a text fragment has to be imported by the *ui:with* tag. Subsequently, text fragments can be accessed, for instance, by the expression *f.firstname*. The interfaces have a static *IMPL* field, similar to the

ClientBundles introduced in Listing 2.8, which contains an instance of the respective interface. In Java classes, all localized text fragments can be retrieved through this field. As already demonstrated in Figure 2.7, the GWT compiler produces browser-specific JavaScript files. If more than one language is provided, the GWT compiler also creates additional JavaScript files for each language. During the compilation, translated text fragments are retrieved from the *.properties* files and, accordingly, used in the JavaScript files. To reduce the traffic between server and client, the user receives only the JavaScript file for the selected language.

Figure 7.25 illustrates how the delivered language is determined by GWT. First of all, it is checked, if the current URL contains a query parameter that defines the user language. If not, GWT tries to find a cookie that specifies the user language. Thereafter, GWT searches for an HTML meta tag that specifies the user language. The last option are the browser headers, which can specify accepted user languages. If none of these definitions are found or the respective value is invalid, the default language is used.

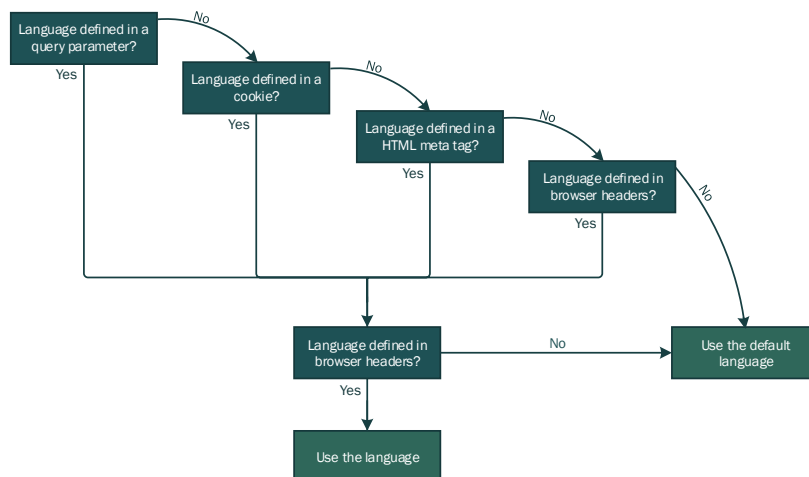


Figure 7.25.: Determining the User Language

This section describes how the localization feature is realized utilizing GWT. The text fragments are defined in the interfaces *ClaviiInterface* and *ClaviiHelper*. The *.properties* files contain the translation for the text fragments. Thus, it is possible to change the localization of Clavii BPM cloud by adjusting the *.properties* files. Furthermore, it is possible to add additional languages by providing *.properties* files without changing the

7. Implementation Aspects of the Web Application

source code of the web application which facilitates the localization of Clavii BPM cloud. Finally, it is explained how GWT determines the language which is delivered to the user.

7.3. Summary

This section gives deep insights in the implementation aspects of the web application of Clavii BPM cloud, which is introduced in Section 5 and Section 6.

At first, general components, which are used later in user interface components, are discussed. The *Navigator* component is responsible for the navigation between web pages and, therefore, enables the implementation of individual web pages to easily navigate to another page. The *Event Bus* component implements the event propagation between components, which facilitates the decoupling of components. The *Data Storage* component provides centralized data storage on the client-side for all entities retrieved from the server-side. Finally, the request management, which encapsulates the entire client-server communication, is explained.

In the following, the user interface components are discussed in detail. First, the structure of the *GroupOverview* and *ModelView* page is broken down. Next, the implementation of the *ModelView* page is illustrated, including the process visualization. Subsequently, the structure of the *Sidebar* component is explained and the individual panels of the *NodeContent* panel are illustrated. Thereafter, the implementation of the *ViewSettings* panel is discussed. Finally, the realization of the localization feature is clarified.

8

Summary and Outlook

The objective of this thesis is to develop a cloud-based platform for business process administration, modeling, and execution, which is targeted at small and medium-sized businesses. In order to build the foundation for the requirements analysis, fundamentals of business process management and building web applications are introduced. Therefore, the architecture of Activiti BPM Platform and its components is analyzed. Afterwards, general elements and structure of the process model are presented. Subsequently, it is shown how a GWT application is built as well as fundamental technologies are defined.

The analysis of the state-of-the-art BPMSs IBM Business Process Manager and Activiti BPM Platform shows the typical capabilities of a BPMS. The points of emphasis are the process lifecycle, architecture of a BPMS, and the user interface of a BPMS.

8. Summary and Outlook

Thereafter, requirements of a BPMS targeted at SMEs are defined. At the beginning, several user stories are specified. Whereby, the focus is set to a user friendly user interface, in particular, for inexperienced users. Another point of emphasis is the easy administration and operation of such a platform. Capabilities of state-of-the-art BPMS are factored into to provide the necessary basic functionality. In this context, it is important to specify a detailed access rights system to enable users to share their process models.

After requirements are introduced, the following two sections show how the requirements are implemented in Clavii BPM cloud. The architecture of Clavii BPM cloud is discussed to provide a high-level understanding of Clavii BPM cloud. Hence, the four-tier architecture, which consists of the web application, the web interface, the services, and the data tier, is discussed as well as the corresponding data model. Subsequently, the user interface is illustrated by discussing individual pages of the web application of Clavii BPM cloud. Finally, it is explained how the requirements are met by the individual pages.

In contrast, the following section gave deep insights in the implementation of Clavii BPM cloud. At first, general components like the navigator or the event bus are explained. The latter are used in several user interface components. Next, user interface components and aspects are discussed, to give a broad understanding of the implementation of the user interface. A special focus is set to GroupOverview and ProcessOverview page, the ModelView page, and the Sidebar component.

Result of this thesis is the web application of Clavii BPM cloud. There are of course restrictions compared to state-of-the-art BPMSs on the market, which offer more functionality, e.g., regarding high availability. However, Clavii BPM cloud provides a unique user-friendly process experience through a simple web-based user interface and features like several process modeling notations and drag and drop process modeling.

For the future of Clavii BPM cloud there are several extensions feasible. The next evolution of Clavii BPM cloud would be the addition of the so-called *timeline*. The timeline is already visible in the ViewSettings panel, but not implemented yet. The timeline is a dashboard, which provides a full-screen overview on all running and completed process models. It allows filtering process instances by several criteria, like start time, involved users, or current state. Such functionality is crucial to use a BPMS in enterprises.

Another possible future development step may be collaborative process modeling. Therefore, the current synchronization mechanisms could be extended to allow screen mirroring between users, i.e., it would be possible to open group sessions and see the actions of the other participating users. Related to this feature, a chat system could be implemented.

It would also be feasible to integrate a store into Clavii BPM cloud. The store could contain process models, plugins, and triggers to buy. Whereby, two different concepts are conceivable. The first is a common store concept, where users can buy and unlock store items. The second would be, that an enterprise uploads several process models and plugins into the store, which are only unlocked for users with specific roles and, thereby, can only be accessed by users with the necessary access rights.

Altogether, this thesis provides the web application of Clavii BPM cloud, which is a cloud-based business process management system targeted at small and medium-sized enterprises.



Layouting Examples

A. Layouting Examples

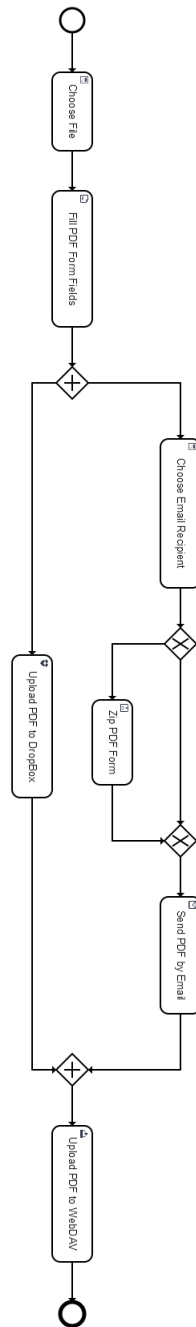


Figure A.1.: BPMN

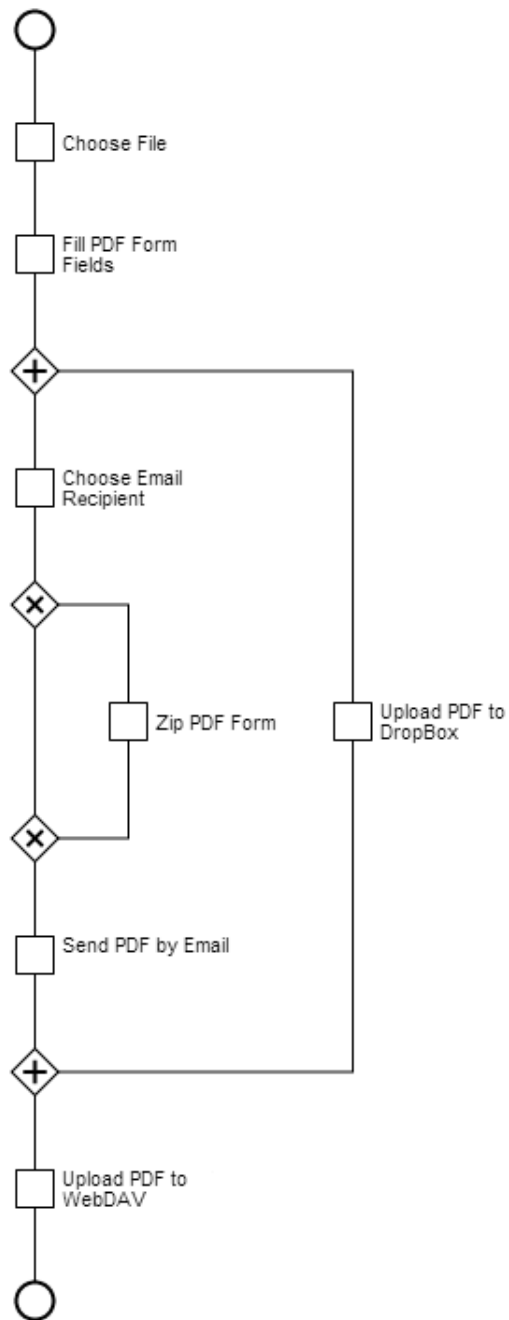


Figure A.2.: Transit Map

List of Figures

1.1. Growth Rate of Cloud Services [Gar13]	3
1.2. Clavii BPM cloud	4
2.1. The Process Lifecycle	8
2.2. Structure of Activiti Engine	10
2.3. Simple Process Model	12
2.4. A Process Model including Gateways	12
2.5. Examples for Block-oriented Process Models	13
2.6. Structure of a GWT Application	14
2.7. GWT Compilation Process	16
2.8. Class Structure for RPC in GWT	17
2.9. RequestFactory	20
2.10. GWT Injection	21
3.1. IBM BPM Architecture	29
3.2. Activiti Architecture	30
3.3. Process Designer - Start Page	30
3.4. Process Designer - Modeling View	31
3.5. Modeling a User Task in the Process Designer	32
3.6. Process Execution in the Process Portal	32
3.7. Process Modeling with the Activiti Designer	33
3.8. Process Execution with the Activiti Explorer	34
5.1. Four-Tier Architecture of Clavii BPM cloud	52

List of Figures

5.2. Overview on Service Components	54
5.3. Overview on Web Application and Web Interface	56
5.4. Organizational Model Entities	58
5.5. Example Organizational Model	58
5.6. General Model Entities	59
6.1. Site Map	62
6.2. Login Page	63
6.3. Registration Page	64
6.4. GroupOverview Page	65
6.5. ProcessOverview Page	67
6.6. ModelView Page	68
6.7. Adding a Process Node via Drag and Drop	68
6.8. Node Configuration Properties Tabs of the Sidebar	69
6.9. Modeling Business Objects in the BuildtimeView Page	70
6.10. RuntimeView Page	71
6.11. User Form generated for the Task Execution	72
6.12. Settings Page	73
6.13. UserProfile Page	74
7.1. Opening a Page based on a specific URL	81
7.2. Event Bus Component	83
7.3. NodeSelection Event Propagation	84
7.4. Usage of the DataStore Class	85
7.5. RequestManager and ProcessModelManager Classes	87
7.6. Integration of DataStore Class, ProcessModelManager Class, and Event Bus Component	87
7.7. User Interface of the Main Panel	90
7.8. User Interface of the GroupOverview Page	91
7.9. User Interface of the GroupTile Panel	92
7.10. Creating a New Group of Process Models	93
7.11. User Interface of the ModelView Page	94

7.12. Process Model Structure Discovery	96
7.13. Arranging the Process Elements	97
7.14. Layers of the Process Visualization	98
7.15. Drawing Process Elements with the NodeLayer	99
7.16. Base Elements of Lienzo Framework	101
7.17. Composition of Process Elements	101
7.18. Interconnections of the Event Controller	102
7.19. Executing a Task	104
7.20. SidebarContent Panels available in the individual Pages	105
7.21. Structure of the NodeContent Panel	107
7.22. ViewSettings Panel	108
7.23. Applying a Process View by the ViewSettings Panel	109
7.24. Implementation of the Localization Feature	110
7.25. Determining the User Language	111
A.1. BPMN	118
A.2. Transit Map	119

Listings

2.1. .gwt.xml File	15
2.2. RPCService Interface	17
2.3. RPCServiceAsync Interface	18
2.4. RPCServiceServlet Class	18
2.5. Servlet Configuration in the web.xml File	18
2.6. Client-Side RPC Call	19
2.7. Examples for GWT Injection	21
2.8. Creation of a ClientBundle	22
2.9. CSS File and CSS Theme	23
2.10. UiBinder File HelloWorld.ui.xml	24
2.11. Java Class for UiBinder	24
7.1. Accessing the Web Storage of a Browser utilizing GWT	86
7.2. Main Panel UiBinder file	89
7.3. GroupOverview UiBinder File	91
7.4. GroupTile Panel UiBinder File	92
7.5. ModelView UiBinder File	94

List of Tables

4.1. User Stories	38
4.2. Requirements	48
7.1. Events for Propagation of Entity Updates and changed Node Selection of a Process Model	83
7.2. Event Types issued by Process Elements on the Layers	103

Bibliography

- [AAG⁺13] AHUKANNA, Dawn ; ALMEIDA, Victor Paulo Alves d. ; GUCER, Vasfi ; NARAIN, Shishir ; PHAM, Bobby ; SALEM, Mohamed ; WARKENTIN, Matthias ; WOOD, J.Keith ; XIE, Zhi Q. ; ZHANG, Cheng: *IBM Business Process Manager Version 8.0: Production Topologies*. Vervante, 2013
- [Act14] ACTIVITI: *Activiti Components*. 2014. – <http://activiti.org/components.html> last visited on 03/07/2014
- [And14] ANDREWS, Kevin: *Design and Development of a Runtime Object Design and Instantiation Framework for BPM Systems*, University Ulm, Master Thesis, 2014
- [Bur06] BURNETTE, Ed: *Google Web Toolkit*. 2006
- [CK03] CRAWFORD, William ; KAPLAN, Jonathan: *J2EE Design Patterns*. O'Reilly Media, Inc., 2003
- [Coh04] COHN, Mike: *User Stories Applied: For Agile Software Development*. Addison-Wesley Professional, 2004
- [Dew07] DEWSBURY, Ryan: *Google Web Toolkit Applications*. Pearson Education, 2007
- [EFGK03] EUGSTER, Patrick T. ; FELBER, Pascal A. ; GUERRAUI, Rachid ; KERMARREC, Anne-Marie: The Many Faces of Publish/Subscribe. In: *ACM Comput. Surv.* (2) 35 (2003), S. 114–131

Bibliography

- [EK12] EL KHARBILI, Marwane: Business Process Regulatory Compliance Management Solution Frameworks: A Comparative Evaluation. In: *Proceedings of the Eighth Asia-Pacific Conference on Conceptual Modelling - Volume 130*, Australian Computer Society, Inc., 2012 (APCCM '12), S. 23–32
- [Emi] EMITROM: *Lienzo Documentation*. – <https://github.com/emitrom/lienzo/wiki> last visited on 29/05/2014
- [Err] ERRAI: *Errai Reference Guide*. – http://docs.jboss.org/errai/3.0-SNAPSHOT/errai/reference/html_single last visited on 13/05/2014
- [FFI04] FORMAN, Ira R. ; FORMAN, Nate ; IBM, John V.: *Java Reflection in Action*. (2004)
- [Fow] FOWLER, Martin: *Inversion of Control Containers and the Dependency Injection Pattern*. – <http://martinfowler.com/articles/injection.html> last visited on 25/05/2014
- [Gar13] GARDNER: *Public IT Cloud Services five Year Compound Annual Growth Rate from 2011 to 2016*. 2013. – <http://www.statista.com/statistics/203578/global-forecast-of-cloud-computing-services-growth/> last visited on 11/06/2014
- [GT98] GEORGAKOPOULOS, Dimitrios ; TSALGATIDOU, Aphrodite: Technology and Tools for Comprehensive Business Process Lifecycle Management. In: *Workflow Management Systems and Interoperability*. Springer, 1998, S. 356–395
- [Hic13] HICKSON, Ian: *Web Storage / W3C*. 2013. – Technical Report
- [Hol95] HOLLINGSWORTH, David: *The Workflow Reference Model*. (1995)
- [HSG03] HOWES, Timothy A. ; SMITH, Mark C. ; GOOD, Gordon S.: *Understanding and Deploying LDAP Directory Services*. Addison-Wesley Longman Publishing Co., Inc., 2003

- [IGRR09] INDULSKA, Marta ; GREEN, Peter ; RECKER, Jan ; ROSEMAN, Michael: Business Process Modeling: Perceived Benefits. In: *Conceptual Modeling-ER 2009*. Springer, 2009, S. 458–471
- [IRRG09] INDULSKA, Marta ; RECKER, Jan ; ROSEMAN, Michael ; GREEN, Peter: Business Process Modeling: Current Issues and Future Challenges. In: *Advanced Information Systems Engineering* Springer, 2009, S. 501–514
- [Kam14] KAMMERER, Klaus: *Enabling Personalized Business Process Modeling: The Clavii BPM Platform*, University Ulm, Master Thesis, 2014
- [KFRMR⁺12] KABICHER-FUCHS, Sonja ; RINDERLE-MA, Stefanie ; RECKER, Jan ; INDULSKA, Marta ; CHAROY, Francois ; CHRISTIAANSE, Rob ; DUNKL, Reinhold ; GRAMBOW, Gregor ; KOLB, Jens ; LEOPOLD, Henrik u. a.: Human-centric Process-aware Information Systems (hc-pais). In: *arXiv preprint arXiv:1211.4986* (2012)
- [KH07] KÜNG, Peter ; HAGEN, Claus: The Fruits of Business Process Management: an Experience Report from a Swiss Bank. In: *Business Process Management Journal* (4) 13 (2007), S. 477–487
- [KKR12a] KOLB, Jens ; KAMMERER, Klaus ; REICHERT, Manfred: Updatable Process Views for Adapting Large Process Models: The proView Demonstrator. In: *Proc of the Business Process Management 2012 Demonstration Track*. Tallinn, Estonia, 2012
- [KKR12b] KOLB, Jens ; KAMMERER, Klaus ; REICHERT, Manfred: Updatable Process Views for User-centered Adaption of Large Process Models. In: *Service-Oriented Computing*. Springer, 2012, S. 484–498
- [KLMR13] KOLB, Jens ; LEOPOLD, Henrik ; MENDLING, Jan ; REICHERT, Manfred: Creating and Updating Personalized and Verbalized Business Process Descriptions. In: *The Practice of Enterprise Modeling*. Springer, 2013, S. 191–205
- [KP⁺88] KRASNER, Glenn E. ; POPE, Stephen T. u. a.: A Description of the Model-View-Controller User Interface Paradigm in the smalltalk-80 System. In:

- Journal of Object Oriented Programming* (3) 1 (1988), S. 26–49
- [KR13a] KOLB, Jens ; REICHERT, Manfred: Data Flow Abstractions and Adaptations through Updatable Process Views. In: *Proceedings of the 28th Annual ACM Symposium on Applied Computing* ACM, 2013, S. 1447–1453
- [KR13b] KOLB, Jens ; REICHERT, Manfred: A Flexible Approach for Abstracting and Personalizing Large Business Process Models. In: *ACM SIGAPP Applied Computing Review* (1) 13 (2013), S. 6–18
- [KR13c] KOLB, Jens ; REICHERT, Manfred: Supporting Business and IT through Updatable Process Views: The proView Demonstrator. In: *Demo Track of the 10th Int'l Conference on Service Oriented Computing (ICSOC'12)*. Shanghai, China, 2013, S. 460–464
- [KRW12] KOLB, Jens ; REICHERT, Manfred ; WEBER, Barbara: Using Concurrent Task Trees for Stakeholder-centered Modeling and Visualization of Business Processes. In: *S-BPM ONE-Education and Industrial Developments*. Springer, 2012, S. 237–251
- [KZWR14] KOLB, Jens ; ZIMOCH, Michael ; WEBER, Barbara ; REICHERT, Manfred: How Social Distance of Process Designers Affects the Process of Process Modeling: Insights From a Controlled Experiment. In: *29th Symposium On Applied Computing (SAC 2014), Enterprise Engineering Track*, ACM Press, 2014, 1364–1370
- [LKR13] LANZ, Andreas ; KOLB, Jens ; REICHERT, Manfred: Enabling Personalized Process Schedules with Time-aware Process Views. In: *Advanced Information Systems Engineering Workshops* Springer, 2013, S. 205–216
- [Mas98] MASINTER, Larry: *The "data" URL Scheme*. 1998. – <http://tools.ietf.org/html/rfc2397> last visited on 29/06/2014
- [Mey14] MEYER, Britta: *Conception, Design, and Evaluation of a Graphical User Interface for a Cloud Platform for Business Process Management*, University Ulm, Master Thesis, 2014

- [MR13] MUTSCHLER, Bela ; REICHERT, Manfred: Understanding the Costs of Business Process Management Technology. In: GLYKAS, Michael (Hrsg.): *Business Process Management - Theory and Applications*. Springer, 2013 (Studies in Computational Intelligence 444), S. 157–194
- [MRA10] MENDLING, Jan ; REIJERS, Hajo A. ; AALST, Wil M. d.: Seven Process Modeling Guidelines (7PMG). In: *Information and Software Technology* (2) 52 (2010), S. 127–136
- [OMG11] OMG: *Business Process Model and Notation (BPMN), Version 2.0*. <http://www.omg.org/spec/BPMN/2.0>. Version: 2011
- [Pau05] PAULSON, Linda D.: Building Rich Web Applications with AJAX. In: *Computer* 38(10) (2005), S. 14–17
- [PCBV10] PATIG, Susanne ; CASANOVA-BRITO, Vanessa ; VÖGELI, Barbara: IT Requirements of Business Process Management in Practice - an Empirical Study. In: *Business Process Management*. Springer, 2010, S. 13–28
- [Rad12] RADEMAKERS, Tijs: *Activiti in Action : Executable Business Processes in BPMN 2.0*. Shelter Island, NY : Manning Publications, 2012
- [Rei00] REICHERT, Manfred: *Dynamische Ablaufänderungen in Workflow-Management-Systemen*, University Ulm, Dissertation, 2000
- [RHAM06] RUSSELL, N. ; HOFSTEDE, A.H.M. ter ; AALST, W.M.P. van d. ; MULYAR, N.: Workflow Control-Flow Patterns: A Revised View / BPM Center. 2006. – Technical Report
- [Ric11] RICHARDSON, Clay: *The ROI Of BPM Suites*. 2011. – <http://www.forrester.com/The+ROI+Of+BPM+Suites/fulltext/-/E-RES60205> last visited on 03/07/2014
- [RKBB12] REICHERT, Manfred ; KOLB, Jens ; BOBRIK, Ralph ; BAUER, Thomas: Enabling Personalized Visualization of Large Business Processes through Parameterizable Views. (2012)

Bibliography

- [Rud07] RUDDEN, Jim: Making the Case for BPM - A Benefits Checklist. In: *BPTrends 2007* (2007)
- [Sch08] SCHMIETENDORF, Andreas: Assessment of Business Process Modeling Tools under Consideration of Business Process Management Activities. In: *Software Process and Product Measurement*. Springer, 2008, S. 141–154
- [SK11] SUBASHINI, Subashini ; KAVITHA, V: A Survey on Security Issues in Service Delivery Models of Cloud Computing. In: *Journal of Network and Computer Applications* (1) 34 (2011), S. 1–11
- [SWKN11] SPATH, Dieter ; WEISBECKER, Anette ; KOPPERGER, Dietmar ; NÄGELE, Rainer: *Business Process Management Tools 2011*. Stuttgart, Germany : Fraunhofer IAO, 2011
- [THET13] TACY, Adam ; HANSON, Robert ; ESSINGTON, Jason ; TOKKE, Anna: *GWT in Action*. 2nd. Greenwich, CT, USA : Manning Publications Co., 2013
- [Vaa13] VAADIN: *The Future of GWT Report*. 2013.
– <https://vaadin.com/documents/10187/42fbbec4-51c8-426b-8aa8-fe46129353a3/> last visited on 19/05/2014
- [Van08] VANBRABANT, Robbie: *Google Guice: Agile Lightweight Dependency Injection Framework*. Apress, 2008
- [Wal98] WALDO, Jim: Remote Procedure Calls and Java Remote Method Invocation. In: *Concurrency, IEEE* (3) 6 (1998), S. 5–7
- [Wes07] WESKE, Mathias: *Business Process Management - Concepts, Languages, Architectures*. Springer, 2007
- [WRMR11] WEBER, Barbara ; REICHERT, Manfred ; MENDLING, Jan ; REIJERS, Hajo A.: Refactoring Large Process Model Repositories. In: *Computers in Industry* (5) 62 (2011), S. 467–486

- [Zai97] ZAIRI, Mohamed: Business Process Management: a Boundaryless Approach to Modern Competitiveness. In: *Business Process Management Journal* 3(1) (1997), S. 64–80
- [ZMR08] ZUR MUEHLEN, Michael ; RECKER, Jan: How much Language is enough? Theoretical and Practical Use of the Business Process Modeling Notation. In: *Advanced Information Systems Engineering* Springer, 2008, S. 465–479

Name: Stefan Büringer

Matrikelnummer: 691309

Erklärung

Ich erkläre, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den

Stefan Büringer