Bachelor's Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Graphics and Interaction

The usage of BPMN library to define workflow

Evgeniya Brichkova
Study program: Software Technologies and Management
Branch: Web and Multimedia

January 2017
Supervisor: Ing. Jiří Šebek

Acknowledgement

I thank Ing. Jiří Šebek for the guidance of my bachelor thesis, help and valuable advices.

Declaration

I declare, that i have done assigned the bachelor thesis alone led by supervisor. I used only literature, that is listed in work. Furthermore i declare, that i have no objections against lending or making public of my bachelor thesis or it's part with agreement of department.

In Prague 3.1.2017

. . . .

Abstrakt / Abstract

Bakalářská práce se zabývá problematikou návrhu a modelování workflow procesu a jejich integrace do logiky aplikace. Workflow procesy jsou vytvořeny v souladu s Business Process Model and Notation (BPMN) 2.0 standardem a representované ve formě BPMN diagramu. Aplikace představuje dvě horní vrstvy systému, který se zabývá problematikou přiřazení rolí uživatelům a umožňuje správu účtů uživatelů, oprávnění a úpravu organizační strukturu firmy. Diagramy, representujici workflow procesy, mají více specifický formát, než standardní BPMN diagramy. Nový formát diagramů vychází z vlastních pravidlech modelování, které vyhovují cílovému systému. První vrstva představuje BPMN modelář, který podporuje vlastní pravidla modelování. Druhá vrstva zajišťuje validátor a parser BPMN diagramu, který jej převede do stromů Plain Old Jáva objektů (POJO). Tím pádem, správcům roli je poskytován pohodlný způsob, jak vyřešit problémy správy role s velkými úspory času.

Klíčová slova: BPMN, definice workflow, řízení uživatelských rolí, vnitřní reprezentace XML

The bachelor thesis deals with the issue of the definition and modelling workflow processes and their integration into the application logics. Workflow processes are created according to the Business Process And Modelling Notation (BPMN) 2.0 standard and represented in the form of BPMN diagram. The application represents two upper layers of the system, which deals with the problem of user role assignments and allows managing users' accounts, permissions and modifying organizational structure of the company. The diagrams, which represent the workflow processes, have more specific format, than standard BPMN diagrams have. New format of diagrams is based on the custom modelling rules, which suit the aims of the target system. The first layer is represented by the BPMN modeler, which supports custom rules. The second layer provides the validator and parser of BPMN diagram, which converts it to the tree of Plain Old Java Objects (POJO) objects and saves the diagram to the database. Thereby, role administrators are provided with a convenient way to solve the role management issues with big savings of time.

Keywords: BPMN, workflow definition, user role management, internal representation of XML

Contents

Tables / Figures

Chapter 1
Introduction

Web applications are usually accessed by a large number of users, so the role management is a significant aspect of each application. A set of permissions for an authorised user is defined by the role, which the user owns. It needs to make sure, that all the important business informations can only be accessed by selected users, who dispose it in a correct way.

1.1 Motivation

The management of user roles might be a tiresome process, especially in large companies. In many cases, such companies have a huge amount of user roles and a difficult hierarchy of them. Thereby, the process of user role assignment might be long and inconvenient. Moreover, the user role administration might be time-consuming for the people, who own the particular roles in that hierarchy. The company might have thousands of different roles, thereby the role administrators are supposed to know those roles and the rights assigned to them. One of the solutions for this problem is an automation, which performs the role approval request and releases the role assignment, if the response is positive. An automation tool can be a simple web application, which has two interfaces: the first one servers for sending a request from one side and the second one serves for accepting or declining it in another side. The aim of this application is to provide the administrator with an ability to define the workflows of approval processes. With the help of a simple graphical toolkit, the administrator can define the workflow for each role approval. It means that it defines the path of this request and treats all the possible faults during this path. Thereby, each time when somebody needs to own a particular role, the appropriate request is sent and a certain workflow process is executed. As it was mentioned before, all the possible problems, which can break the workflow, are avoided. For instance, in some workflows there is defined the time limit of waiting for the answer from the target person. In case the waiting time gets too long, the request goes to another target person, who has the same rights as the previous one or higher. The target person then gets the request, containing the information needed. As soon as the decision is taken, the target person sends the response by simply choosing „approve“ or „reject“ option. In case of the positive response, the role is assigned and the confirmation is sent. In another case, the applicant only get the notification, that the role is not assigned and the reason for it.

1.2 Purposes

The purpose of this bachelor thesis is to provide the detailed way how to handle role-based management. So the administrators of the application are able to define and modify the workflow processes of the system using comprehensible and user-friendly graphical interface, which is accessible from browser.

1.2.1 Modeler

The uppermost layer of the application is a web-modeler, which is implemented in JavaScript (js) language using the BPMN-js library. The main function of this module is to provide the graphical interface for defining workflow processes for the administrator of an application. The interface allows displaying and modifying the BPMN diagrams representing workflow processes and creating new ones, saving them in BPMN 2.0 format and then passing to the lower layer. The issue is to use and customize the BPMN-js library for the clear definition of the purpose of individual objects which make up the business process elements.

The library modification included the followings:

- restriction of the variety of available elements for the diagram creation
- modification of elements' design
- restriction of the ability to add features to the elements and connections according to their type
- restriction of the elements' naming according to the type and function
- restriction of ability to connect some elements to others according their type

The module also allows:

- storing and sending the diagram to the web service (the second module) in the format of BPMN 2.0
- visualization of the server's response with the result of sending and validation of the diagram

1.2.2 XML Parser

The second module represents an (Extensible Markup Language)XML parser and validator for the workflow modeler. This module converts the diagram into java POJO objects using Java Architecture for XML Binding (JAXB). Its responsible for the diagram validation against custom rules, which have been created with the purpose of the target application. The first part of the validation is performed against an XML Schema Definition (XSD) file, which is the customisation of initial schema created according to the standard BPMN diagram rules. The second part is the validation by java functions. The module validates the tree of POJO objects, which represents the initial diagram. The module also contains the services for working with the generated objects and defines a unified interface for the lower layers of the target application

Chapter 2 Background

2.1 Business process

Each flow of actions, which is happening around us can be characterized as a process, as well as there are business processes in business, which are commonly found in business organizations. Michael Hammer, a former professor of computer science at the Massachusetts Institute of Technology, in his seminal work gave such a definition of a business process: *„A collection of activities that takes one or more kinds of input and creates an output that is of value to the customer. A business process has a goal and is affected by events occurring in the external world or in other processes“*. But quite a significant feature of each business process is also an existence of actors, roles and the collaboration between them for achieving a specified business goal.

2.2 Business process management and modeling

Business process management is the discipline, which deals with designing, modeling, execution, analyzing and optimization of business processes [1] in Appendix D. The most important component of this discipline is the business process modeling. It is a mechanism for describing the states of a business process in different intervals of time. Simple geometrical symbols and arrows are used to represent the sequence of activities. They serve for the visualization of the „if/then“ relationships in the workflow. The business process model typically defines the goal of business process, its inputs and outputs, resources, activities and events, which drive the process and other elements. This is highly used in large organizations to improve organizational efficiency by representing the workflows, analyzing this representations and modifying them. The modeling of processes, which is going on in a business, can bring an instant problem identification and it is an important tool for the simulation of efficiencies of certain processes. Some of the benefits of analyzing and modeling business processes are the following:

- clear definition of roles and responsibilities in the company
- easy detection of potential problems
- clear understanding the company activities

2.3 Business modeling lifecycle

Business modeling lifecycle is the process by which business models are created, used and maintained. It allows the organization to make changes and improve it from time to time. The modeling lifecycle is described in Figure 2.1

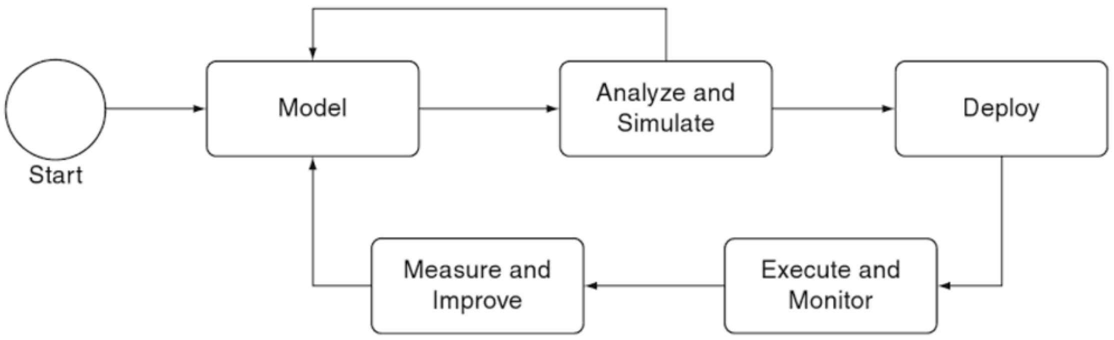


Figure 2.1. Modeling life cycle (undertook from reference 2 in Appendix D)

Phases of the modeling lifecycle [2] in Appendix D:

- **model phase**
  The first phase of the modeling life cycle is called model phase, when the diagram is being created. There is plenty of different modeling techniques and tools, which can be chosen according to the context of the business process.
- **analyzing and simulation phase**
  During this phase the diagram is examined and simulated. As the result of this phase it might be changed. In the picture such a change is represented by an arrow going back to the modeling phase.
- **deploy**
  During the deployment phase, the diagram is converting into a specified format to be used as an application or to be integrated into some application.
- **execute and monitor**
  During this phase the application, which received a newly created model, is running, according to its input (the model).
- **measure and improve**
  The output data is analyzed and compared to the expected values.

2.4 Modeling techniques overview

There is a number of standards for business process modeling and it might be an issue to choose the correct one for the defined purposes. Different modeling techniques focus on different features of processes.

- **Role activity diagrams (RAD)**
  RADs are the graphical representation of processes in terms of roles presented within these processes, their component activities and their interactions, together with external events and the logic, which determines the sequence of those activities (when and by whom) [3] in Appendix D. Thereby, the main function of RAD diagrams is to emphasize the interaction between the roles in the organization.
- **Data Flow Diagrams (DFD)**
  Data flow diagrams (DFD) are used to illustrate the system functionality with underlying processes and data flows.
- **Activity Diagrams**
  Activity diagrams are used for the workflow mapping. It is a graphical representation of the steps, actions and decisions that are made in a certain process [4] in Appendix D.

- **Business Process Model and Notation (BPMN)**
  BPMN is a graphical notation that depicts the steps in a business process [5] in Appendix D. It is the most common graphical representation tool for the business process visualization, because it contains the semantics, which is quite richer, than the other modeling techniques have. The modeling techniques comparison is shown in the Table 2.1

Modeling technique name	Advantages	Disadvantages
Role Activity diagrams (RAD)	elements of diagram are grouped into blocks according to the roles, so it is easy to understand the flow of processes: switching between the roles and the reasons for their actions; flexibility: easy behavior redefinition of some concrete roles not touching the whole diagram.	
Data Flow Diagrams (DFD)	ability to create the child diagrams for each activity, so the diagram might have multiple levels; understandability and flexibility: each activity can be decomposed into smaller processes, which allows easy redefinition	modeling is time-consuming; huge models, which might be uncertain and probably will not cover the whole system [4] in Appendix D
Activity Diagrams	understandability: built on simple symbols; parallel activities modeling	no distinguishes between activities, which can be quite confusing
Business Process Model and Notation (BPMN)	good for both: complex and simple projects	huge amount of symbols might be confusing for unskilled users

Table 2.1. Techniques comparison

2.5 BPMN 2.0 standard

2.5.1 Types of BPMN processes

- **private processes**

Private processes define the internal workflows of the organization and they never leave the organization scope [6] in Appendix D. The private processes can be executable and non-executable. Each executable process has enough details to be executed. Non-executable process does not have enough details and serves for the workflow description at the modeler defined level. An example of a private business process is in Figure 2.2. Private processes are used for the workflow approval definitions in this thesis.

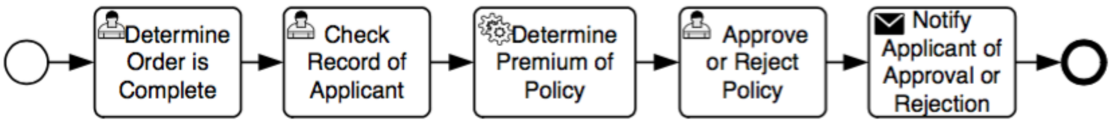

Determine Order is Complete
Check Record of Applicant
Determine Premium of Policy
Approve or Reject Policy
Notify Applicant of Approval or Rejection

Figure 2.2. Example of a private Business Process (undertook from reference 6 in Appendix D)

- **public processes**

Public processes are used to represent the interaction between some private process and the participant [6] in Appendix D. In the public process, the internal behavior of it is not shown. Only the points of interaction are shown with the help of Message Flows. An example of a public business process is shown in Figure 2.3.

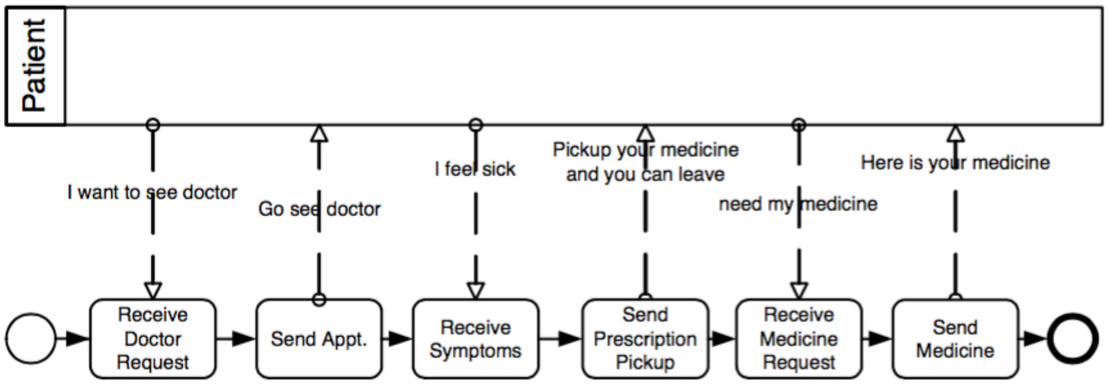


Figure 2.3. Example of a public business process (undertook from reference 6 in Appendix D)

2.6 BPMN 2.0 core structure

The BPMN specification is structured by layers, where each layer is inherited by some other layers. This kind of structure allows easily redefinition and extending of layers by creating new successors of them. Thereby, the backwards compatibility is not modified. The core of BPMN includes three sub packages: the foundation package, the service package and the common package [6] in Appendix D. The class diagram in Figure 2.4 shows the core packages

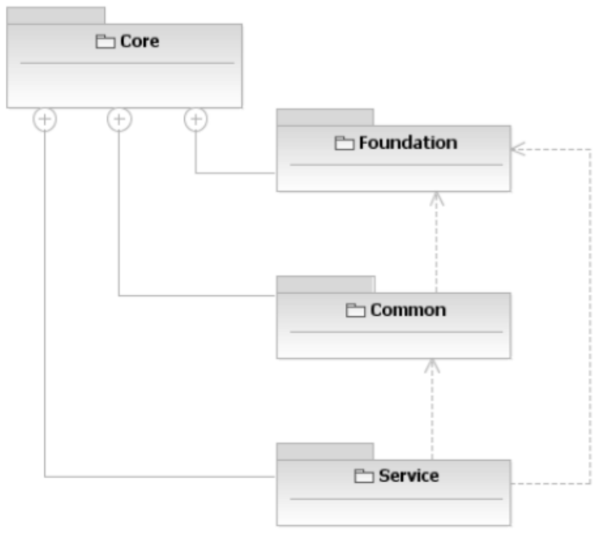


Figure 2.4. Class diagram showing the core packages (undertook from reference 6 in Appendix D)

2.6.1 Infrastructure package

The infrastructure package contains two elements, which serve for the diagram modeling and abstract syntax modeling [6] in Appendix D.

- **Definitions** is the element, which contains all other elements in the diagram. It defines the namespaces and the the scope of visibility for all the elements contained.
- **Import** element is used to reference an external BPMN element from another Definitions element or non-BPMN element.

2.6.2 Foundation package

The foundation package contains classes, which are shared by all the members of the abstract syntax model's core [6] in Appendix D.

- **BaseElement** is extended by almost all BPMN elements. It contains id and documentation attributes, which all the elements inherit.
- **Import** element is used to reference an external BPMN element from another Definitions element or non-BPMN element.
- **Documentation** is used to add one or more text descriptions to all BPMN elements.
- **Extensions** allow to extend the BPMN diagram with other elements and leave it still BPMN compliant. This set of elements allows adding custom attributes to the existing elements and the result elements will still be understood by BPMN adopters.
- **ExternalRelationships** are intended to enable BPMN artifacts to be integrated into more complex structures.
- **RootElement** is the abstract superclass of all BPMN elements, which are placed within Definitions. It has the own defined lifecycle and it is not deleted when the elements inside are deleted.

2.6.3 Common package

The common package represents the standard BPMN diagram elements, which can be used in more than one type of diagrams (for instance Collaboration, Process or Choreography) [6] in Appendix D. Following ones are used in this thesis.

- **FlowElement** is an abstract superclass for all elements that can appear in a Process flow.
- **Event** element is used to represent a real event, that happens during the flow of process or Choreography. The Event normally has a cause and the result of it. Thereby, the Event might be of different types, such as for example Start Event, End Event, Intermediate Event and others.
- **Extensions** allow to extend the BPMN diagram with other elements and leave it still BPMN compliant. This set of elements allows adding custom attributes to the existing elements and the result elements will still be understood by BPMN adopters.
- **SequenceFlow** is used to show the order of Flow Elements in the Process or a Choreography. Each Sequence Flow has only one source and target.
- **Gateway** is used for the representation of the convergence and divergence during the process flow. It might have one input and multiple output Sequence Flows.

2.6.4 Service package

The service package contains the important interfaces and operations needed for modeling services and interfaces [6] in Appendix D.

2.7 XML to POJO converting with JAXB

In many applications an XML document serves as an external representation of something, therefore it needs to have some internal form to be used in lower layers of an application. One of the most popular tools is Java Architecture for XML Binding (JAXB). JAXB is a tool, which provides a fast and convenient way to bind XML schemas and Java representations, making it easy for Java developers to incorporate XML data and processing functions in Java applications [13] in Appendix D.Binding a schema means generating a set of Java classes that represents the schema [14] in Appendix D. All JAXB implementations provide a tool called a binding compiler to bind a schema. So, the XML elements are mapped into java POJO objects.

2.8 Used technologies

2.8.1 Node Package Manager(NPM)

NPM is a technology, which allows easy sharing of the js code among developers. It works the way, that each time, when the foreign code is getting updated, the NPM allows easily downloading of a new version of that code, so a developer does not need to make changes himself. The bits of reusable code are called packages or modules. Each package also contains a file called package.json, which is used for the package management. Basically, it serves for the documentation of the packages the project depends on and for the version control.

2.8.2 Node.js

Node.js is a server-side solution for js. It is intended to run on the HTTP server and is event-based and asynchronous. It is made on the base of Google V8, which is the js engine, that compiles js into the machine code. The core functionality of Node.js is written in js. Normally, js can only run in browser, but Node.js provides an environment for js running, it can access the local files, listen to the network traffic, accept HTTP requests the machine gets, send responses and access the databases directly.

2.8.3 Grunt

Grunt is a js Task Runner for Node.js projects. It is the tool used to automatically perform frequently used tasks such as minification, compilation, unit testing and others. It uses a command-line interface to run custom tasks defined in its configuration file (known as a Gruntfile).

Chapter 3
Related work

The problem of role management, in the terms of role approval definition is quite an untouched topic in software. Until now, there were no special tools for the definition of role approval processes. However, there is a huge amount of tools for the definition of business processes in general. The BPMN-js library has been used for the application, described in this thesis. The library is still in progress, but it contains all the functionality needed to provide import, export and modifying of BPMN 2.0 diagrams. Moreover, it already contains enough features to prove, that it is one of the best BPMN modeling tools for embedding diagrams into the application logics. The library can be easily customized and extended according to the purposes of the target application. The BPMN-js toolkit does not have most of the disadvantages, which other modelers have. But on the other hand, the biggest part of the other modelers have quite wider functionality and they are not restricted only by process modeling and export/import. Some of them have an Application Programming Interface (API) for executing and testing the result diagrams.

3.0.1 TIBCO Business Studio

TIBCO Business Studio is a standard based business process modeling environment, which enables business experts and process authors to collaborate for creating process models, organization models and data models [7] in Appendix D. It is also an eclipse-based free business modeling tool that lets you model and simulate business processes. One of the major characteristics of TIBCO is that it provides single environment for process modeling, testing, simulation and deployment. One of the main disadvantage of this tool is that it is not easy to use. It also has no graphical export for BPMN-diagram.

3.0.2 Aris Express

Aris Express is one more modeling tool for business process analysis and management. It is not open source and has not open license, but it is free to use. Among the advantages of Aria Express is the idea of using model fragments: the pieces of diagram or collection of objects, which can be considered as new units of the diagram, which helps to avoid creating the same parts of the diagram. The Aris Express is written in Java and the disadvantage of it is that it requires Java Web Start every time to get started. The Java Web Start software allows to download and run Java applications from the web [9] in Appendix D. Moreover, it does not matter if it has already been installed, the internet access with the community account is needed. It also sometimes gets slow while using.

3.0.3 Yaoqiang BPMN Editor

Yaoqiang BPMN editor is one more open source graphical editor for business process diagrams, compliant with BPMN 2.0 specification. Yaoqiang BPMN Editor is a very small application written in Java. It requires no installation and makes it easy to view and modify BPMN processes. Among the main advantages of Yaoqiang BPMN Editor is the existence of LDAP browser and executable BPMN simulation in its later versions. It can also deploy directly to the existing BPMN 2.0 Engine. It has small disadvantages, such as the restriction of the amount of processes by the page size. Also the connections and lines from one symbol to another are not controlled.

3.0.4 Bizagi

Bizagi BPMN Modeler is a freeware application to graphically diagram, document and simulate processes in BPMN format. With the Bizagi Modeler, processes can be published to Word, Portable Document Format(PDF), Wiki, Web or SharePoint, or exported to Visio, image formats, such as Portable Network Graphics (png), bpmn, Scalable Vector Graphics(svg) or Joint Photographic Experts Group(jpg), and XML Process Definition Language(XPDL), to be shared and communicated across the organization. It also has Bizagi Studio, which serves for business process automation and Bizagi Engine, which takes the previously modeled and automated processes and executes them across the organization. The installation is quite easy as well as the usage. It contains some small graphical disadvantages, such as, for example, the availability of connections and associations only at the left or right side of the elements.

There are much more different BPMN tools. For the need of the current application, the main feature of a modeling tool is the ability to easily customize and extend it as much as possible for the application purposes. It is also supposed to be easy to use, install and embed into the application logics. BPMN - js library perfectly satisfies all these points.

Chapter 4
Analysis and design of the application

4.1 BPMN-js overview

BPMN-js was created for working with BPMN 2.0 standard. It can be used with any modern browser, which allows easily embedding BPMN 2.0 into the web application. BPMN-js can be used as a viewer and as a modeler. In the first case, it is used to embed and display the BPMN 2.0 diagrams in browser. In the second case, it serves for creating the own BPMN 2.0 diagrams and extending them. The library is built on the base of two main components: diagram-js and bpmn-moddle [10] in Appendix D.

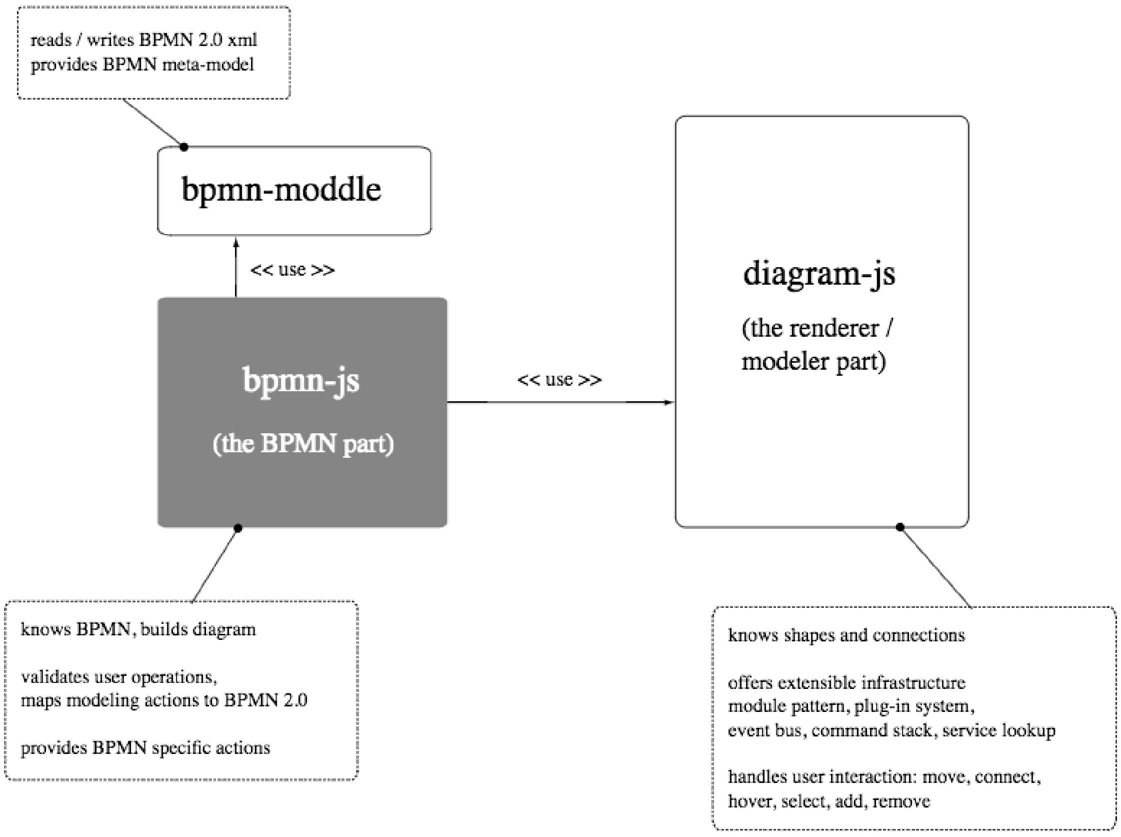

reads / writes BPMN 2.0 xml
provides BPMN meta-model
bpmn-moddle
<< use >>
bpmn-js
(the BPMN part)
<< use >>
diagram-js
(the renderer / modeler part)
knows BPMN, builds diagram
validates user operations, maps modeling actions to BPMN 2.0
provides BPMN specific actions
knows shapes and connections
offers extensible infrastructure module pattern, plug-in system, event bus, command stack, service lookup
handles user interaction: move, connect, hover, select, add, remove

Figure 4.1. bpmn-js architecture: parts and responsibilities (undertook from reference 10 in Appendix D)

For the purposes of the current application, the number of changes and extensions to the diagram-js components have been carried out.

4.1.1 Diagram-js core classes overview

The Diagram.js library is responsible for the creating, displaying and modifying BPMN 2.0 diagrams. It provides users with the interface for interaction with shapes and connections, handles user's actions like move, hover, select, add, remove. It offers the module system, which can easily be extended. It also contains own implementation of dependency injection, so each component can be accessed by the unique name it was given in the start of the program. The Diagram.js contains a huge system of modules. The core of this system consists of five main modules.

- **Canvas** provides the interface for adding and removing elements. In fact, it is responsible for the whole image of a diagram. When the application starts running, the Canvas creates an svg element which is wrapped into div - container, where the diagram is placed. It also provides a log of services for working with the diagram and its elements, such as, for example, modifying the diagram's viewport, which is the visible part of the main svg element, adding and removing elements.
- **EventBus** serves as a communication channel across the diagram instance between the actions, performed by a user or the individual components and answers to those actions by other application logics. It was created to decouple the concerns and it provides a very easy way to custom the existing application behavior by overloading existing event listeners. The Event Bus interface contains such methods as EventBus.on() and EventBus.once() to register new listeners. The EventBus.off() method is used to remove the registration. All these methods receive an event object as the first parameter, which allows them to hook into the event execution. The priority of listeners is configured by the value of an optional parameter called priority. It is used for the listeners customization, so on the purpose to overload an existing event listener, it is only needed to set the registration priority higher that the default one. The events can also be emitted by the method EventBus.fire(), which fires a named event.
- **ElementFactory** serves for the creating of shape and connection according to the Diagram.js internal data model. The data model says, that each shape has a parent, a list of children, incoming and outgoing connections. Each connection has a parent, a source and a target, which is pointing to the concrete shape. ElementFactory model contains the reference to the Model class, which follows the data model to create an element. In fact, the only work of ElementFactory is to handle the element type name, create the element's id (in case it does not exists) and the attributes, then transfer them to the Model. The Model defines all the element's components needed such as, for example, parent and children, so the result object contains the initial attributes received and the components according to the data model. The model contains the defined structures for each element type, so it only applies the received attributes to those structures. When the result element is created, it directly goes to the Graphics factory to be rendered in Canvas.
- **ElementRegistry** contains the information about all the elements added to the diagram and provides a user with the interface to access the elements by id. In fact, it represents a collection with the possibilities to add, remove, update, get elements by id and get the complete information about them.
- **GraphicsFactory** is responsible for the rendering shapes and connections into the canvas. However, the real rendering is performed by the Renderer module. The actual main work of Graphics Factory is that it creates the graphical background for elements, such as graphical containers and notifies the Renderer module about the updates, which need to be expressed in canvas. It uses the EventBus to fire the event about the updates. The module is also used for other purposes like accessing any kind of graphics in the diagram (for instance, getting element's children) or removing elements.

4.1.2 Dependency injection in Diagram-js

Dependency injection (DI) is the process of supplying a resource that a given piece of code requires [11] in Appendix D. The required resource, which is often a component of the application itself, is called a dependency. The Diagram-js library uses the Injector.js module, which is responsible for the dependency injection in the application. It provides an access to each of the application components/modules, which are needed to be instantiated once during the application lifetime. When the application starts running, the Injector is instantiated and its constructor gets a set of objects as a parameter. Each object represents a value, which is used later, or a real component with its own functionality. The Injector has two main fields: a set of instances and a set of its providers. Provider creates an instance of a class, passing all the attributes needed. During the Injector's initialization the set providers gets filled with special representations of each module/value. Each provider contains the information, saying how the object has to be used: if its a module - call the constructor, if it is just a value - return it. Each of the result instance has a unique name and then can be accessed by the get method.

4.1.3 Other customized components

- **ContextPad** and **ContextPadProvider** are responsible for the pad with available tools and actions, which belongs to each element in the diagram. It appears after clicking on the element and offers different kinds of actions to perform on the current element. It might be a connection to another element, creating a new element after the current one and others. The ContextPad module represents the pad itself. The ContextPadProvider module fills the ContextPad with concrete actions and defines its behavior.
- **Draw** module is responsible for the look and feel of BPMN elements: for each element type it has defined parameters for drawing. The module also contains all the basic methods for drawing, which use the defined parameters. The Draw module contains two classes: BpmnRenderer, which actually contains all the module functionality and PathMap, which contains the geometrical information about the elements' paths. Then this information is used in BpmnRenderer while defining the elements' look.
- **Palette** and **PaletteProvider** modules are responsible for the bar of available BPMN elements and tools. The elements from the bar can then be added to the diagram. The tools are used to modify and work with the diagram. The element bar is placed in the left side of the window. The Palette module represents the bar of elements itself and defines its behavior. The Palette provider manages the content of the bar.
- the separate library **diagram-js-direct-editing** is responsible for the text description of the diagram elements. It deals with the look and behavior of the input field for text description on each element. The input field appears after double click on the element and saves new or modified description after clicking outside it. The library contains two classes: DirectEditing class is responsible for the behavior of the element editing and the TextBox represents the input field itself.

4.1.4 Custom rules

The modeler follows specific custom rules, created on the purposes of the target application. The following elements are needed for the workflow approval process: StartEvent, EndEvent, Task, ExclusiveGateway and Timer. Some of them contain substantial information about the workflow, which is then saved to the database. This information is placed in these elements' description, so the description is an obligatory parameter them. The description is restricted into the set of values received from the server. The content of this set depends on the application context and dynamically changes. Among these elements are Task, End event and Timer. Task contains all the information about the request sent: type of it, action and responder. The End event represents the result of a workflow, thereby it contains "yes" or "no" values. The Timer element represents the waiting period and its value indicates the amount of time for waiting. Other elements play a visualization role and contain no information: they help a user to comprehend or create the diagram in a right way and the server to interpret it. ExclusiveGateway is used to evaluate the state of a business process and breaks the flow into mutually exclusive paths. StartEvent indicates the start of a workflow. The order of diagram elements is also defined in the custom rules.

4.2 Class diagrams of the Modeler part

A class diagram represents the project architecture by the visualization of classes and their relations. The following class diagrams describe the front end of the application, namely, the BPMN-js customizations. Each diagram package represents a separate module. In the diagram, the packages contain only modified units, so most of them contain more classes, than the following diagram represents. Also, most of the classes in the diagram contain instances of different library utilities, which are not represented in the diagram due to their insignificance in the current topic. All the classes, which relate to the modules' customization, are placed in the custom-modeler package. The actual customizations are places in the package custom-modeler.custom, while the package custom-modeler contains the main class, which binds the customizations to the application routine.

4.2.1 DirectEditing module customization diagram

The CustomElementEditor class inherits the library class DirectEditing. It modifies the editor behavior and substitutes the text box for editing by a select box. The SelectBox class represents a simple select box with some additional behavior.

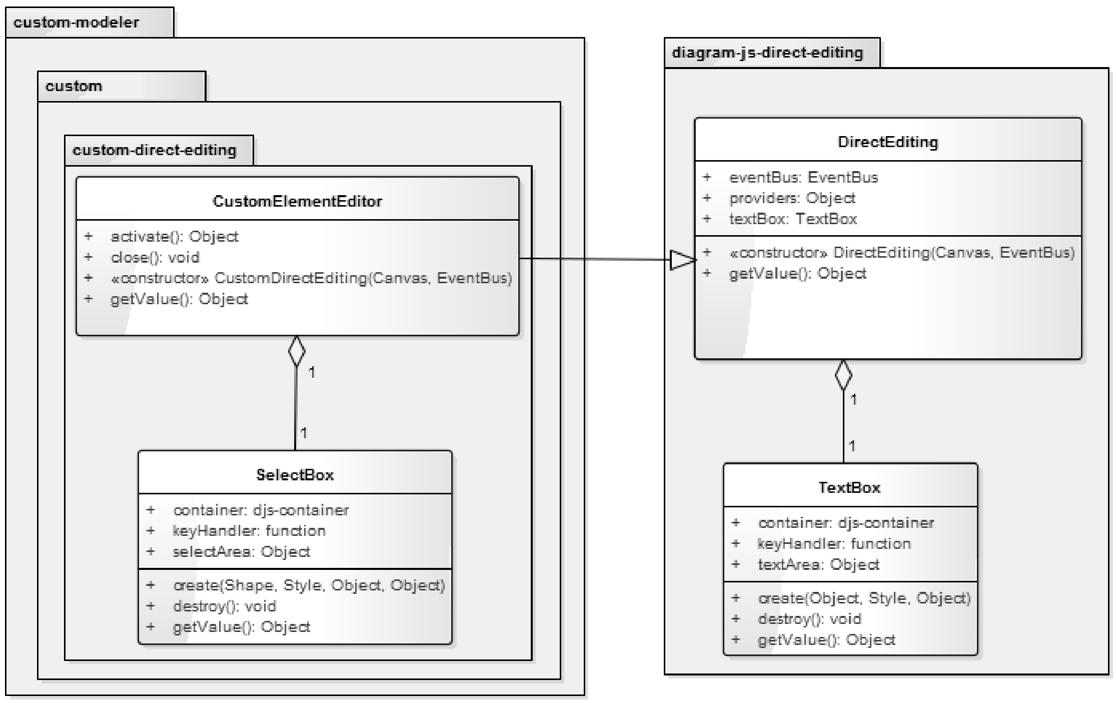


Figure 4.2. DirectEditing customization

4.2.2 Palette and ContextPad modules customization diagram

The classes CustomContextPadProvider and CustomPaletteProvider inherit the library classes ContextPadProvider and PaletteProvider respectively. They include some overridden methods and attributes from the parent classes. The CustomContextPadProvider extracts the number of elements and tools contained in the pad and modifies some of them. The CustomPaletteProvider extracts and modifies the number of elements in the standard palette bar.

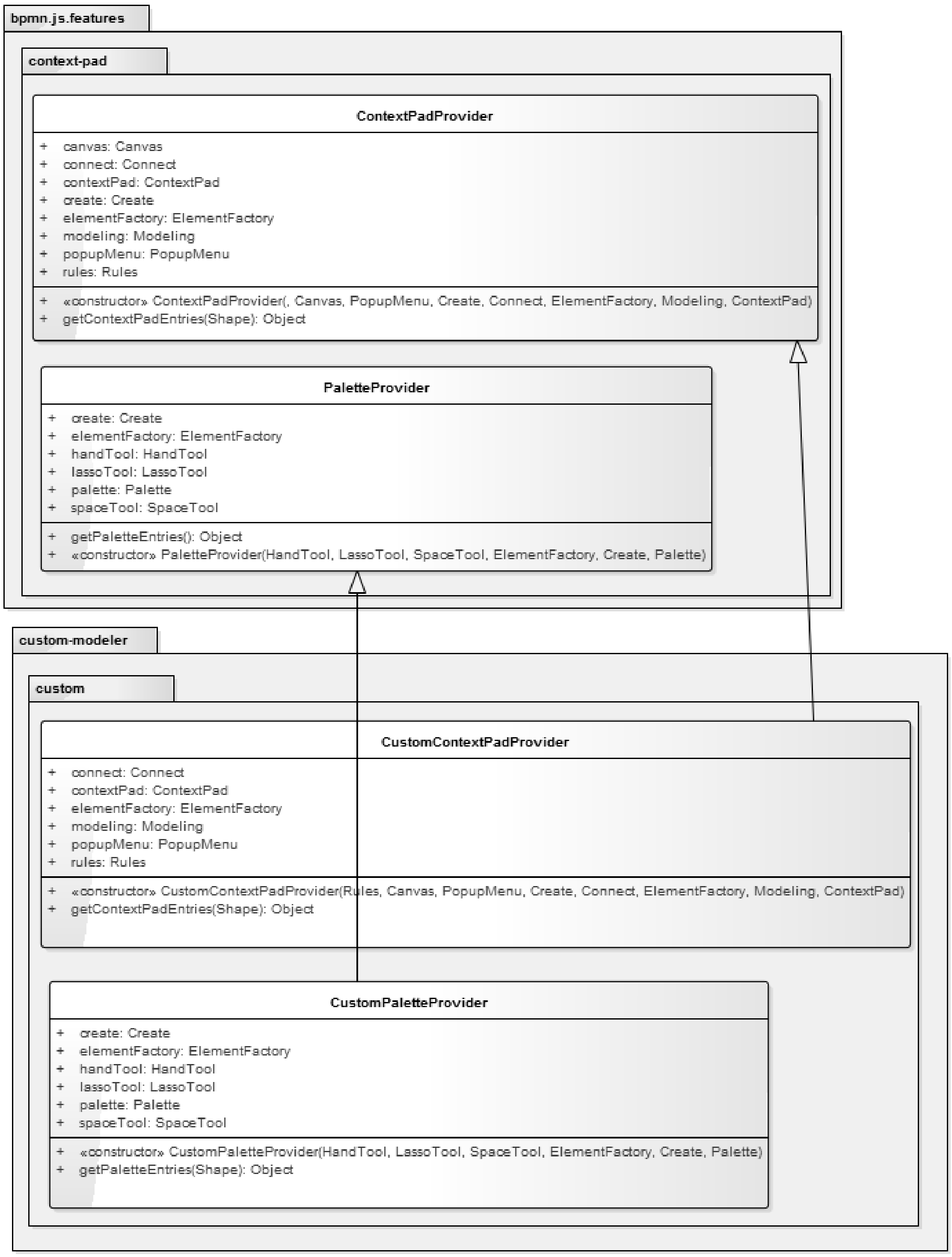

bpmn.js.features
context-pad
ContextPadProvider
+ canvas: Canvas
+ connect: Connect
+ contextPad: ContextPad
+ create: Create
+ elementFactory: ElementFactory
+ modeling: Modeling
+ popupMenu: PopupMenu
+ rules: Rules
+ «constructor» ContextPadProvider(, Canvas, PopupMenu, Create, Connect, ElementFactory, Modeling, ContextPad)
+ getContextPadEntries(Shape): Object
PaletteProvider
+ create: Create
+ elementFactory: ElementFactory
+ handTool: HandTool
+ lassoTool: LassoTool
+ palette: Palette
+ spaceTool: SpaceTool
+ getPaletteEntries(): Object
+ «constructor» PaletteProvider(HandTool, LassoTool, SpaceTool, ElementFactory, Create, Palette)
custom-modeler
custom
CustomContextPadProvider
+ connect: Connect
+ contextPad: ContextPad
+ elementFactory: ElementFactory
+ modeling: Modeling
+ popupMenu: PopupMenu
+ rules: Rules
+ «constructor» CustomContextPadProvider(Rules, Canvas, PopupMenu, Create, Connect, ElementFactory, Modeling, ContextPad)
+ getContextPadEntries(Shape): Object
CustomPaletteProvider
+ create: Create
+ elementFactory: ElementFactory
+ handTool: HandTool
+ lassoTool: LassoTool
+ palette: Palette
+ spaceTool: SpaceTool
+ «constructor» CustomPaletteProvider(HandTool, LassoTool, SpaceTool, ElementFactory, Create, Palette)
+ getPaletteEntries(Shape): Object

Figure 4.3. Palette and ContextPad customization

4.2.3 The whole customization package diagram

The following diagram represents the whole customization package. The package custom contains all new custom modules. The package custom-modeler contains the class CustomModeler which inherits the default modeler and uses the new modules instead of the default ones.

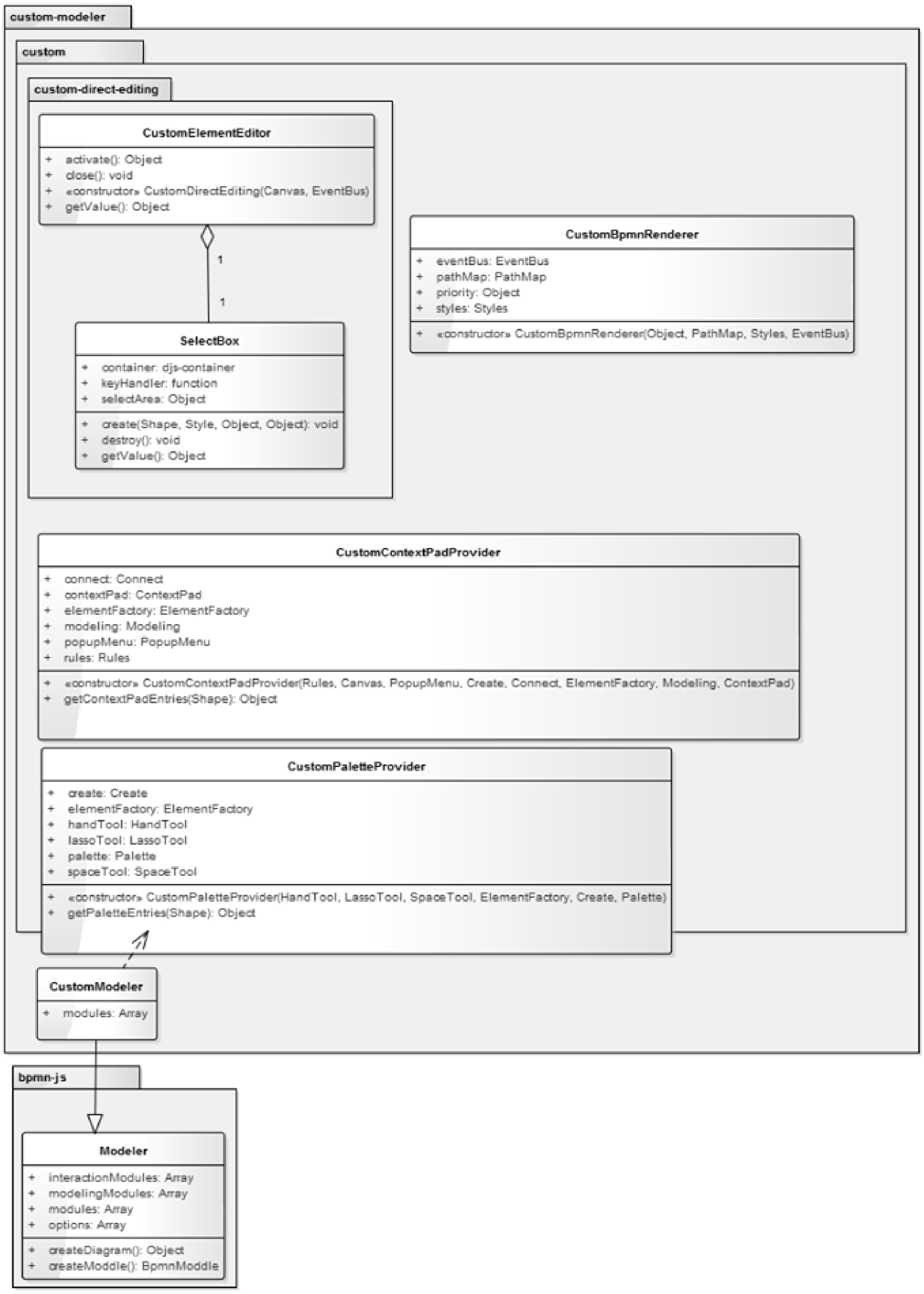


Figure 4.4. Customization package diagram

4.2.4 BpmnRenderer customization diagram

CustomBpmnRenderer class inherits the library class BpmnRenderer, which serves for changing the look of some library elements. The custom class modifies the rules for the elements' rendering, which are placed in the constructor of BpmnRenderer.

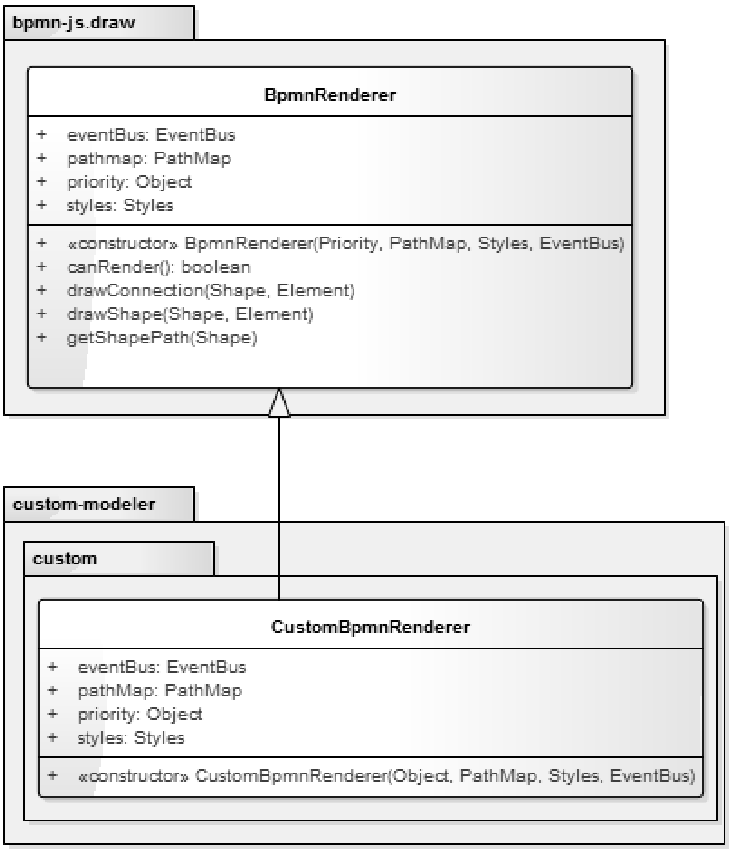


Figure 4.5. BpmnRenderer customization

4.3 Sequence diagram

Sequence diagram is used primarily to show the interactions between the objects in the sequential order that those interactions occur. The processes of sending BPMN diagram to the server, its validation and sending response to a client are represented by the following sequence diagrams. Unsuccessful validation by java functions is shown in the first diagram. Unsuccessful validation by XML schema is shown in the second diagram. The third diagram represents successful validation by both XSD and java. The diagrams represent only the substantial methods, participating in the process.

4.3.1 Unsuccessful validation by java functions

The validation by java functions can fail in two places. As it is shown in the diagram, the first fail can happen during the tree generating. The second fail can happen during validation.

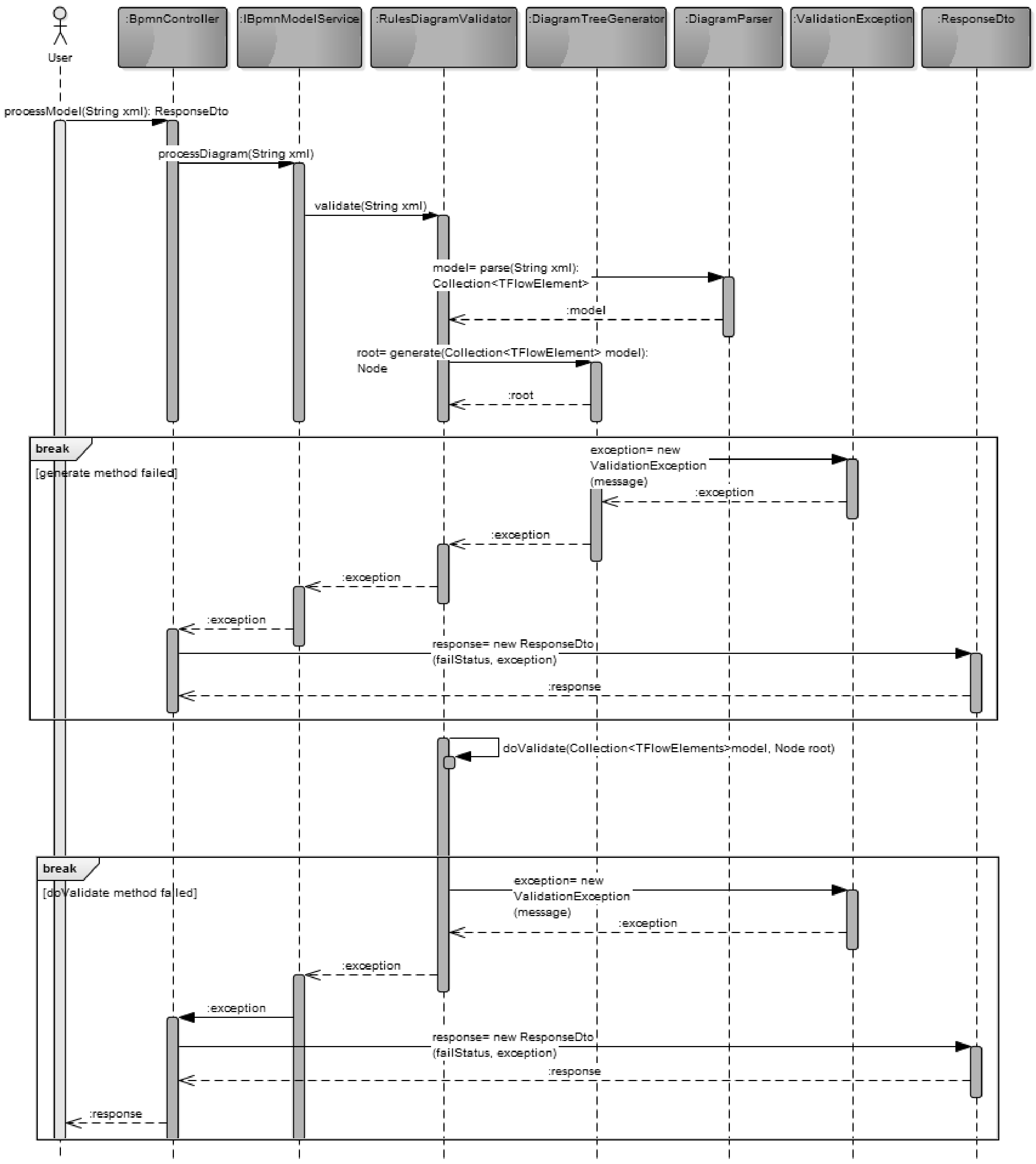

User
:BpmnController
:IBpmnModelService
:RulesDiagramValidator
:DiagramTreeGenerator
:DiagramParser
:ValidationException
:ResponseDto
processModel(String xml): ResponseDto
processDiagram(String xml)
validate(String xml)
model= parse(String xml): Collection<TFlowElement>
:model
root= generate(Collection<TFlowElement> model): Node
:root
break
[generate method failed]
exception= new ValidationException (message)
:exception
:exception
:exception
:exception
response= new ResponseDto (failStatus, exception)
:response
doValidate(Collection<TFlowElements>model, Node root)
break
[doValidate method failed]
exception= new ValidationException (message)
:exception
:exception
:exception
response= new ResponseDto (failStatus, exception)
:response
:response

Figure 4.6. Unsuccessful java validation

4.3.2 Unsuccessful validation by XSD

The validation by XML schema fails in case SAX exception is thrown.

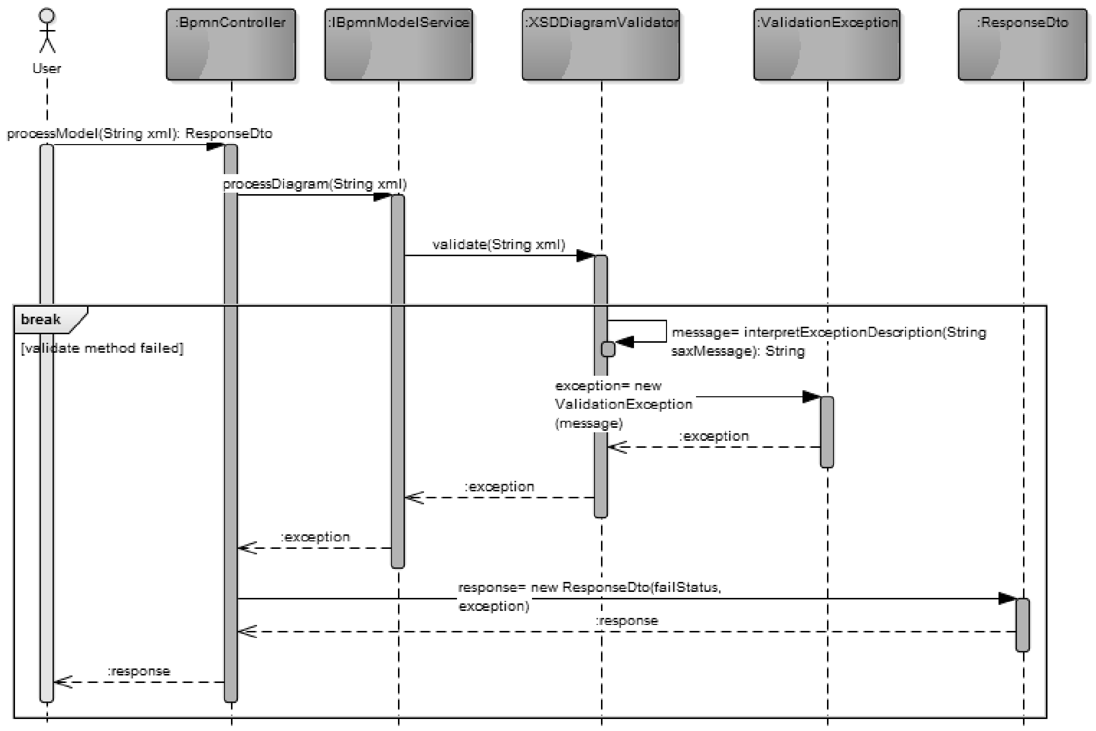

User
:BpmnController
:IBpmnModelService
:XSDDiagramValidator
:ValidationException
:ResponseDto
processModel(String xml): ResponseDto
processDiagram(String xml)
validate(String xml)
break
[validate method failed]
message= interpretExceptionDescription(String saxMessage): String
exception= new ValidationException (message)
:exception
:exception
:exception
response= new ResponseDto(failStatus, exception)
:response
:response

Figure 4.7. Unsuccessful XSD validation

4.3.3 Successful validation by XSD and java functions

In this case, the validation is successful and no exceptions are thrown.

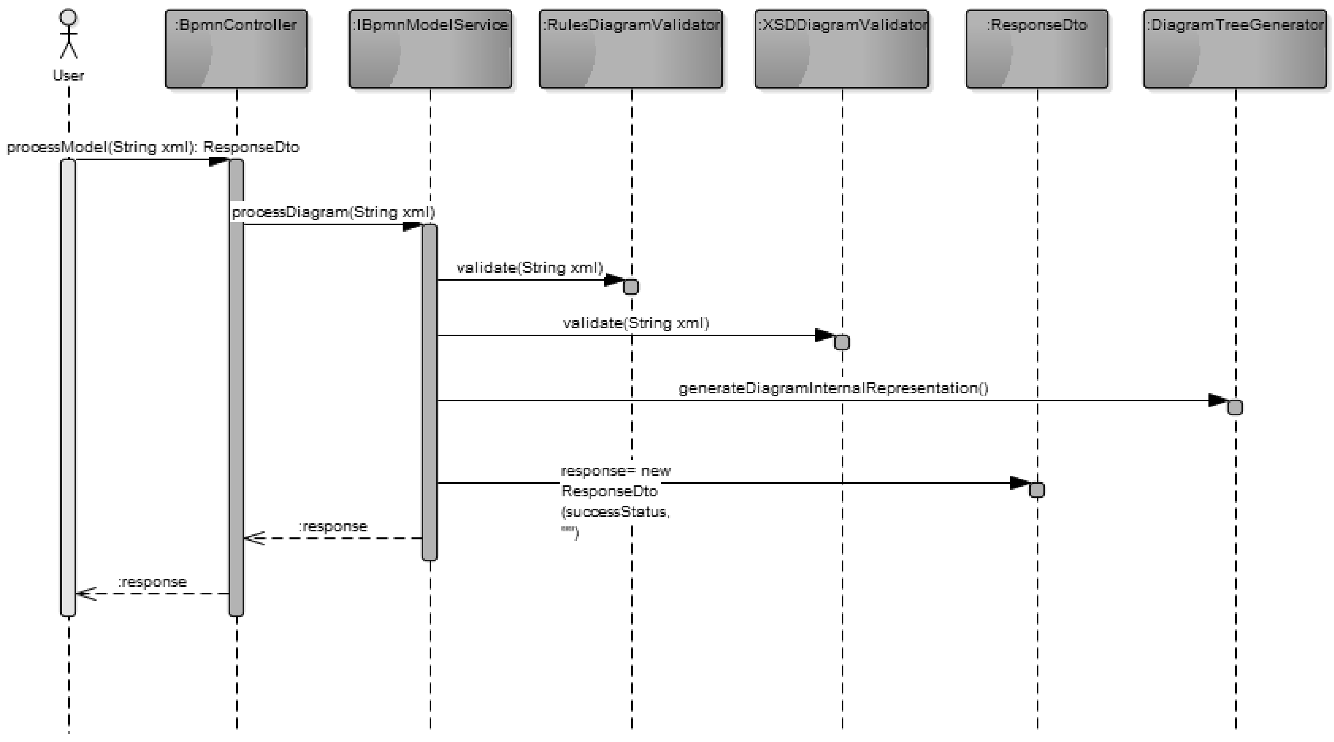


Figure 4.8. Successful validation

The following elements take part in the diagram:

- **objects**
  Objects are used to represent the classes, which are used in the process, described in the diagram. Among the diagram objects are: BpmnController, IBpmnModelService, RulesDiagramValidation, XSDDiagramValidation, ValidationResponseDto, ValidationException, DiagramTreeGenerator, DiagramParser. The sequence diagram represents their interaction and the results of it. Each object has the own lifeline, which represents the period of time that the object exists in the application.
- **actor**
  The actor is a user, who initiates the process, which is described in the diagram. In the case of current process, he creates the BPMN diagram and sends it to the server in the form of XML.
- **arrows**
  The arrows represent the messages sent between the objects. Communication is started when the user sends the diagram to the server. It is actually sent to the controller's method processModel. Following messages take place in the diagram:

  - BpmnController:: processModel - handles diagram from client and transfers it to the IBpmnModelService
  - IBpmnModelService:: processDiagram - handles a diagram from the controller and delegates it to the particular validators for validation
  - RulesDiagramValidator:: validate - validates the semantics of a diagram against java functions
  - DiagramParser:: parse - parses a diagram into java POJO objects and retrieves the objects, which are responsible for the diagram logics. Returns the collection of these objects

  - DiagramTreeGenerator:: generate - generates a tree from the collection of POJO objects, which represents the diagram, and returns the root of the tree
  - RulesDiagramValidator:: doValidate - does the diagram validation using the collection of POJO objects and tree representation of the diagram
  - XSDDiagramValidation:: validate - validates the semantics of the diagram agains XML schema
  - XSDDiagramValidation:: interpretExceptionDescription - interprets the SAX exception thrown during the XSD validation
  - new:: ValidationException(message) creates the object of ValidationException class
  - new:: ResponseDto(status, message) creates the Data Transfer Objects for the notification, that the diagram passed validation
  - DiagramTreeGenerator:: generateDiagramInternalRepresentation() - saves the particular parts of the diagram to the database

- **break**
  The break block is used for the representation of the exception handling. In the current application it represents the program flow when a diagram does not pass the validation. In this case, the ValidationException is thrown in some validator, according to the place where the validation error happened. Then it is delegated through service to the controller, where the exception is caught.

4.4 Use case diagram

The use case diagram represents the relation between a user and an application. The diagram shows the main functions, that can be used.

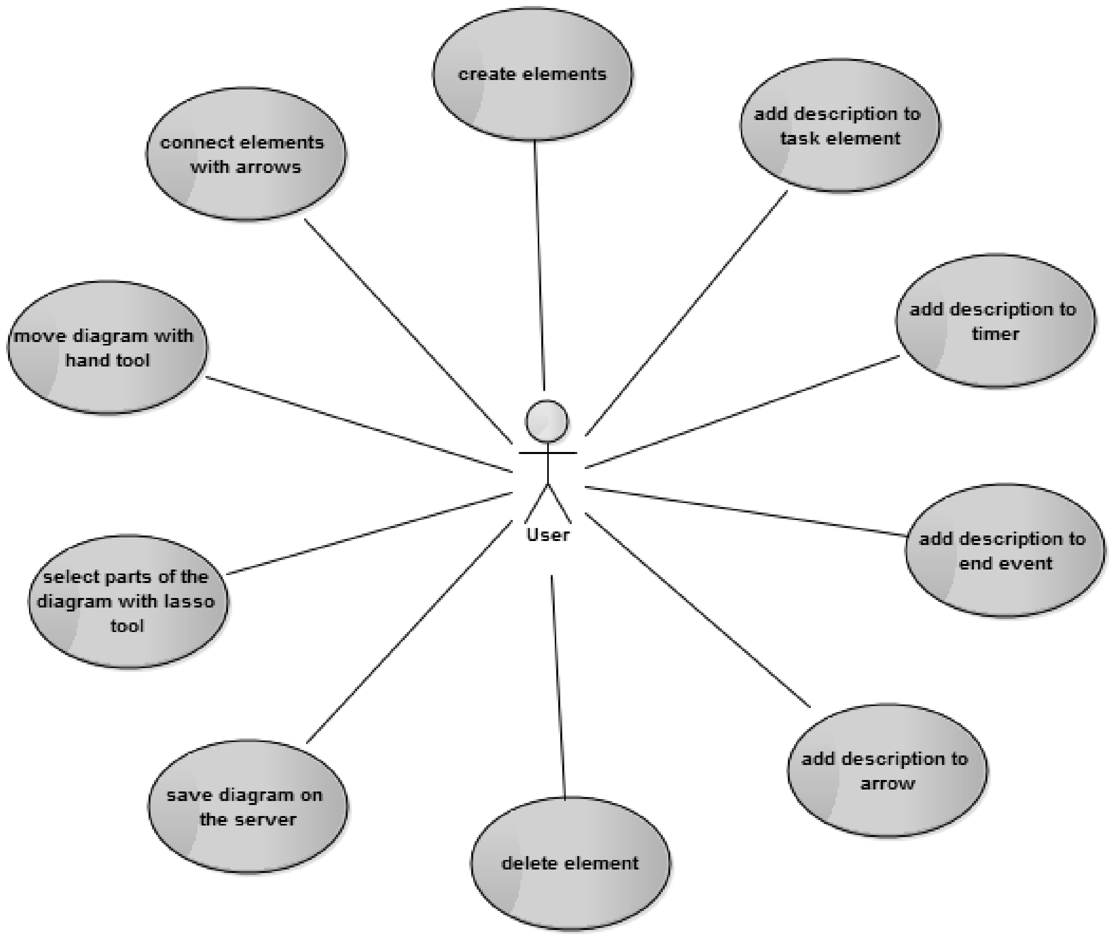

create elements
connect elements with arrows
add description to task element
add description to timer
move diagram with hand tool
User
add description to end event
select parts of the diagram with lasso tool
add description to arrow
save diagram on the server
delete element

Figure 4.9. Use case

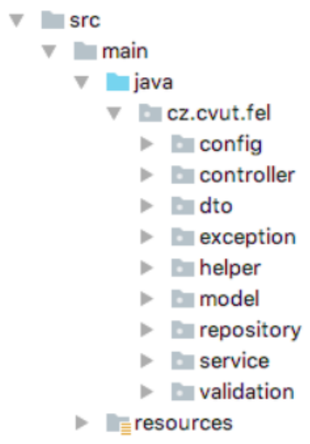
Chapter 5
Implementation

5.1 The project structure

The application is of client-server type, thereby it has the separate implementations for the client and server sides.

5.1.1 Server side packages

The server side of application is located in the directory java. Spring framework is used for its implementation. Building of the application is performed by Apache Maven. The file structure of the server-side is shown in Figure 5.1.1.



```
▼ src
  ▼ main
    ▼ java
      ▼ cz.cvut.fel
        ▶ config
        ▶ controller
        ▶ dto
        ▶ exception
        ▶ helper
        ▶ model
        ▶ repository
        ▶ service
        ▶ validation
    ▶ resources
```

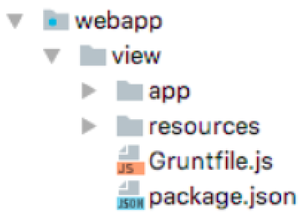
Figure 5.1. Server side package structure

It contains the following packages:

- **config** package contains the configuration of persistence layer. It contains the class, where the data source, entity manager and transaction manager are defined.
- **controller** package contains the controllers of the application. Controllers are responsible for incoming requests. Usually they invoke business logics, update the model if needed and return the view. When a new request comes, the dispatcher servlet handles it to a particular controller, according to the URL mapping configuration.
- **dto (data transfer object)** package contains the data transfer objects, which are used for data transferring between the parts of application, mainly between the client and server sides
- **exception** package contains the ValidationException class, which is an extension of the RuntimeException class. The exception is thrown if the diagram does not pass the validation against custom rules.
- **helper** package contains classes, which are responsible for the particular functionalities of the application, for example the diagram parsing services.
- **model** package contains definitions of business objects, which are mapped to the database entities.

- **service** package contains the service, where the main application logics is located. Services are used for the data handling from controllers and processing it. In the case of current application, the service class uses the validators to process the diagram and then saved it the database.
- **validation** package contains the validators for the BPMN diagrams.
- **resources** package contains the XML schemas of the BPMN specification and the resource bundle message source.

5.1.2 Client side packages



```
▼ webapp
  ▼ view
    ▶ app
    ▶ resources
      Gruntfile.js
      package.json
```

Figure 5.2. Client side package structure

The client side of the application is placed in the view directory:

- **app** package contains the main page and script of the application, all the custom classes and styles.
- **resources** package contains the default diagram, which is displayed in the BPMN editor after the application is launched.
- **Gruntifile.js** is used for the definition and configuration of tasks and loading the Grunt plugins.
- **Package.json** is the configuration file for NPM.

5.2 BPMN-js customizations

The client side of the application is implemented with the help of BPMN-js library. The implementation includes the customization of some library modules.

5.2.1 Adding custom modules

When the main script of the application starts running, the instance of CustomModeler is created. The CustomModeler class inherits the default Modeler class of BPMN-js library. Modeler contains a set of references to the default library modules. Thereby, it decides, which of the modules are used in the application. CustomModeler expands its parent with the custom modules. RequireJS is used for the module loading. RequireJS is a js file and module loader. It is optimized for in-browser use, but it can be used in other js environments, like for example Node [15] in Appendix D.

```
CustomModeler.prototype._modules = [].concat(
  CustomModeler.prototype._modules,
  [
  require('./custom')
  ]
);
```

Listing 1: The extension of the default modules collection in the CustomModeler

Each module contains the field name, which is used to access the reference to the instance of this module during the application lifetime. The name of each custom module is identical to the name of the default module, which it inherits. An example of a custom module export is in Listing 2

```
module.exports = {
paletteProvider: ['type', require('./CustomPaletteProvider')]
};
```

Listing 2: Exporting of the custom modules with corresponding names

While initializing, the Injector module receives the collection of modules, which consists of the default ones and the custom ones. The Injector creates a map of the modules' instances, where keys are the module's names and values are instances. Custom modules are in the end of the module list, because they are added the last, so the Injector put them to the map after all default ones. The key set of result map cannot contain duplicates, thereby when the Injector accesses the custom module to put it to the map, it does not create a new key-value pair - the value of already created key-value pair gets overridden by the custom one. As a result, map with custom values is generated and used during the application lifetime.

5.2.2 DirectEditing module customization

The CustomElementEditor module is created to override the DirectEditing module. It inherits the DirectEditing and changes some points of its implementation. Unlike the default module, the custom one uses select box for the element description and forbids the description on some kinds of elements. Moreover, due to the usage of select box it is possible to restrict the values, which can be passed to the element description. The element description consists of three select boxes. The first one serves for the choosing the type of a task: it might be, for instance, notification or request. The second one is responsible for the action type. The third one is responsible for the responder type. In the target application, the values for select boxes are generated dynamically, depending on the application context. In the current application, the values are hardcoded in a static JSON document.

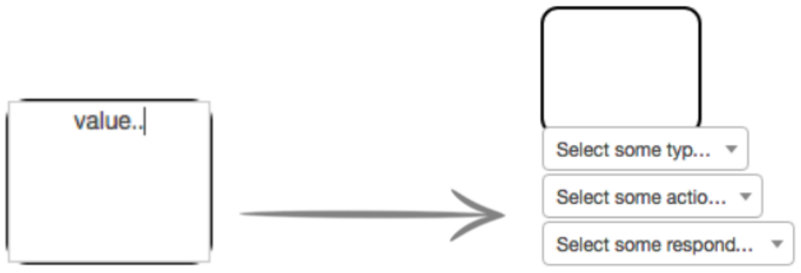


Figure 5.3. Element editing customization

The SelectBox instance is passed to the constructor of CustomElementEditor instead of the text box, as it is done in the library module. On the purpose to forbid the description of some elements, the method responsible for the description activation is overridden and the restrictions are added.

5.2.3 BpmnRenderer customization

BpmnRenderer module is customized on the purpose to create a separate Timer element. In the primary modeler version, the Timer element could be created only as an extension on an IntermediateCatchEvent. So in the custom version, Intermediate-CatchEvent is created with the type of Timer by default. The IntermediateCatchEvent types are shown in Figure 5.4

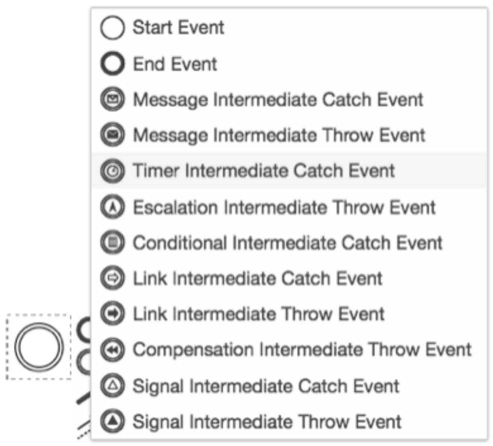

Start Event
End Event
Message Intermediate Catch Event
Message Intermediate Throw Event
Timer Intermediate Catch Event
Escalation Intermediate Throw Event
Conditional Intermediate Catch Event
Link Intermediate Catch Event
Link Intermediate Throw Event
Compensation Intermediate Throw Event
Signal Intermediate Catch Event
Signal Intermediate Throw Event

Figure 5.4. IntermediateCatchEvent types

After the customization, the new element Timer is accessible from palette and context pad as any other element. BpmnRenderer module contains the rules, which define the way each element type looks like. Each element of the BpmnRenderer's field *handlers* represents a function, which draws a particular object according to the rules. Before the actual rendering, the particular type rules are chosen and passed to the module, which is responsible for the graphics rendering. The rules for the IntermediateCatchEvent element are configured in such a way, that Timer type is assigned to it by default while initialising in the CustomBpmnRenderer module. The definition uses rules from the BpmnRenderer module.

```
handlers['bpmn:IntermediateCatchEvent'] = function(p, element){
 var outer= handlers['bpmn:Event'](p, element, {strokeWidth:1});
 handlers['bpmn:TimerEventDefinition'](p, element);
};
```

Listing 3: Redefinition of the IntermediateCatchEvent's look

Handlers['bpmn: Event'] definition represents a circle, which every Event-type element has. Handlers['bpmn: TimerEventDefinition'] definition represents a small clock figure. The result is of the customization is in Figure in Figure 5.5.

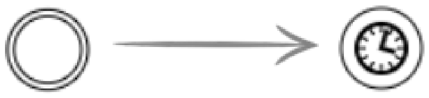


Figure 5.5. IntermediateCatchEvent transformation

5.2.4 ContextPadProvider customization

The customization of ContextPadProvider module is done on the purpose to change the content of the context pad for particular elements. Initially all the context pad of all elements had the same content. The customizations of the ContextPadProvider module are placed in the module CustomContextPadProvider, which inherits the default one. Some elements are forbidden to activate the context pad or their content is restricted. The result of ContextPad customization is shown in Figure 5.6.

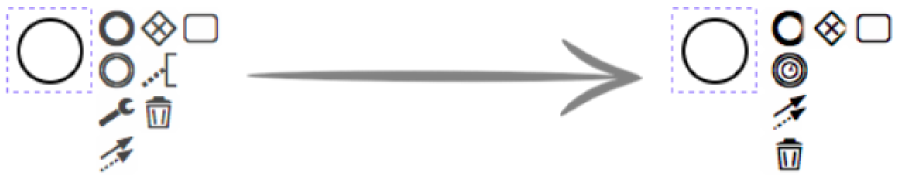


Figure 5.6. ContextPad transformation

The method getContextPadEntries() is overridden in the custom module. The restrictions for the element type are added and the set of elements in the context pad is modified.

5.2.5 PaletteProvider customization

The content of the default library palette is modified in the CustomPaletteProvider module, which inherits the default one. The method getPaletteEntries() is overridden and the content of the palette is changed the way it is shown in Figure 5.7.

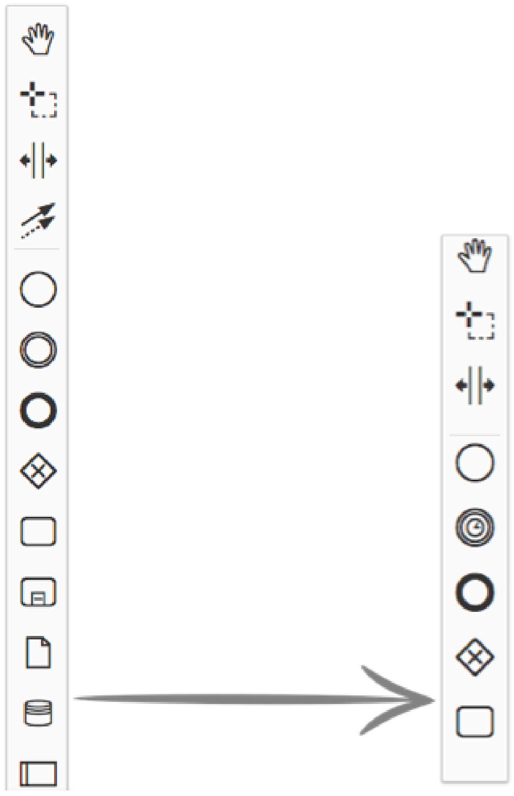


Figure 5.7. Palette transformation

5.3 Server side implementation

5.3.1 Configuration

The application is made in Spring Model View Controller (MVC) framework. The Spring MVC applications are designed around the Dispatcher Servlet, which handles all the requests coming to the server and passes them to the particular controllers, depending on the URL requested [16] in Appendix D. The configuration of the Dispatcher Servlet is placed in the web.xml file.

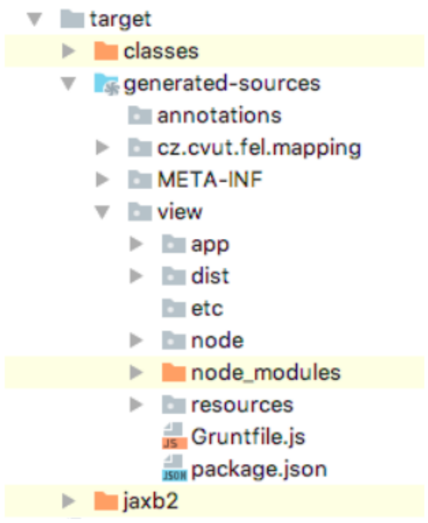
Spring-servlet.xml contains different spring-mvc configurations and configuration beans. However, there is no special need to create any configuration beans, because Spring MVC maintains a list of default beans to use. The following beans have been configured on the purposes of the current application.

- InternalResourceViewResolver bean serves for the resolving of string-based view names to View types. Thereby, the prefix and suffix are added to the view name. Since the current application needs to resolve only one simple index.html view, the suffix is set to html
- ResourceBundleMessageSource bean serves for the accessing to the resource bundles using specified base names.

Application-context.xml defines the spring container. The components of the container can be defined via xml configuration or annotations. Pom.xml is the configuration file of Maven. It is an XML file which contains information about the project and configuration details used by Maven to build the project. The result of the Maven build is generated to the target folder by default. In the current project, it performs a few additional goals during the build of the whole application. Among those goals are:

- installing Node.js and NPM locally in the project
- downloading BPMN library sources into the Target directory by NPM
- building the front end of the application into the Target directory
- generating JAXB classes from the specified XML schemas into the Target directory

All the application's outputs are shown in Figure 5.8:



```
▼ target
  ▶ classes
  ▼ generated-sources
      annotations
    ▶ cz.cvut.fel.mapping
    ▶ META-INF
    ▼ view
      ▶ app
      ▶ dist
        etc
      ▶ node
      ▶ node_modules
      ▶ resources
        Gruntfile.js
        package.json
  ▶ jaxb2
```

Figure 5.8. Target directory

The result of maven build is located in the target directory:

- **classes** package contains compiled java classes
- **cz.cvut.fel.mapping** package contains the classes generated by JAXB
- **META-INF** package contains the only one file sun-jaxb.episode. This is a binding file generated by XJC compiler, which associates schema types with existing classes.
- **view** package contains the client side files. Among them is node_modules folder, which contains the BPMN-js library sources; dist folder is the output of the Grunt build; node is the locally installed Node.js.

5.3.2 Processing request

The modeled diagram is sent from the client side by Asynchronous JavaScript and XML (AJAX) and the request is mapped onto the processing method, which is declared in the BpmnController.

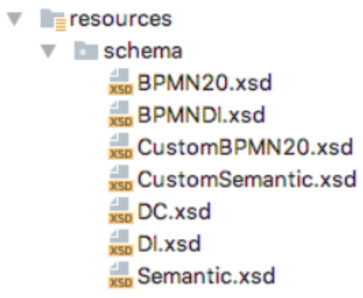
```
@RequestMapping(value = "/validate", method = "RequestMethod.POST")
@ResponseBody
public ValidationResponseDto processModel(@RequestBody String diagramXml)
```

Listing 4: Head of the function, which handles the request with the diagram

The BpmnController class transfers the diagram to BpmnModelService class, where the particular validation services are invoked for the diagram validation.

5.3.3 XSD validation

XML schema is used to define the rules for XML document. It defines the values for the elements and attributes, their order, types and the number of children of particular elements. It is a very convenient way to define the desirable look and feel of the BPMN diagram. The BPMN specification has its own XML schema, which can be customized and extended. In the current application the XML schema is changed according to the specific custom rules. Custom XSD files are created, where new rules are defined. The files are placed in the resources/schema folder, the content of it is shown in Figure 5.9.



```
resources
  schema
    BPMN20.xsd
    BPMNDI.xsd
    CustomBPMN20.xsd
    CustomSemantic.xsd
    DC.xsd
    DI.xsd
    Semantic.xsd
```

Figure 5.9. Schema directory content

Semantic rules of the diagram are placed in the Semantic.xsd file, thereby the default modeling rules are taken from this file for customization. Custom rules are placed in CustomSemantics.xsd. The XMLDiagramValidator class is responsible for the XSD validation in the current application. It uses the package javax.xml.validation, which provides an API for the validation against XML schema. The Validator class performs the validation. The path to custom XSD file is used for creating Validator instance, so the diagram is validated according to the custom rules instead of the default ones. In case the validation fails, the Simple API for XML (SAX) Exception is thrown and then interpreted by the interpretExceptionDescription() method, so the comprehensible answer is returned to the client.

5.3.4 Validation by Java functions

The RulesDiagramValidator class is responsible for the validation against Java functions. The validation process includes three steps:

- **parsing XML diagram into POJO objects**
  As it was mentioned before, the JAXB is used for the schema binding in the current application. The custom XML schema is used for binding, so the POJOs are generated according to the custom rules. The project is configured in the way, that each time when it builds, the binding compiler is launched and the classes are generated according to the XML schema. XML to Java Compiler(XJC) is a tool which performs the actual schema binding, so it accepts an xml schema and generates java classes. The DiagramParser class is responsible for the diagram parsing. When the classes are generated, the DiagramParser's method getDefinitions uses the JAXB unmarshaller to read and translate the diagram into a collection of TFlowElement objects, corresponding to the XML elements in the diagram.
- **generating the tree from diagram**
  The TreeGenerator class is responsible for the tree generating from the diagram. It uses the collection of TFlowElements, which is created by the DiagramParser class. Basically it iterates the collection and creates the instances of the Node and Transition classes for each element and sequence. Since the diagram can be represented in a form of tree, each element can have a predecessor and a collection of successors. An instance of the Node class represents an element in the diagram. Structure of this class is shown in the Listing 5

```
public class Node{
 private TFlowElement value;
 private Transition incoming;
 private Node parent;
 private String name;
 private List<Node>successors;
```

Listing 5: Node object structure

  An instance of the Transition class represents the diagram's connection between two elements. Structure of the Transition class is shown in the Listing 6

```
public class Transition{
 private TSequenceFlow value;
 private Node destination;
 private Node source;
```

Listing 6: Transition object structure

  Basically, the Transition and Node classes wrap the initial diagram elements, parsed from XML, with the references to successors and predecessors, so the tree is generated.
- **the diagram tree validation**
  As soon as the diagram tree is generated, the validation is performed by doValidate method. This method validates the order of elements, the amount of particular elements in the diagram and presence or absence of elements' descriptions. There is a special function, which is used for each of the validation aspects. An example of

elements' order validation is shown in the Listing 7. This piece of code assures, that the ExclusiveGateway and EndEvent can not be located after the ExclusiveGateway element.

```
if(isSuccessor(root, EXCLUSIVE_GATEWAY, EXCLUSIVE_GATEWAY)||
                        isSuccessor(root, EXCLUSIVE_GATEWAY, END_EVENT))
String message=bundle.getMessage('error.diagram.wrong-order');
                   throw new ValidationException(message);
```

Listing 7: Elements' order validation

Successful validation means that no exception has been thrown during it. In this case, the diagram is saved to the database and the positive response in JSON format is sent to client.

5.3.5 Saving diagram to the database

The elements of diagram can be divided into two types: the elements with significant information about the workflow and the elements which serve only for the graphical representation. Thereby, the information about the workflow must be extracted from the diagram to be saved to the database. TaskEvent, EndEvent and Timer are considered as significant parts of diagram. Figure 5.10 represents the diagram before converting, while Figure 5.11 demonstrates the result of converting.

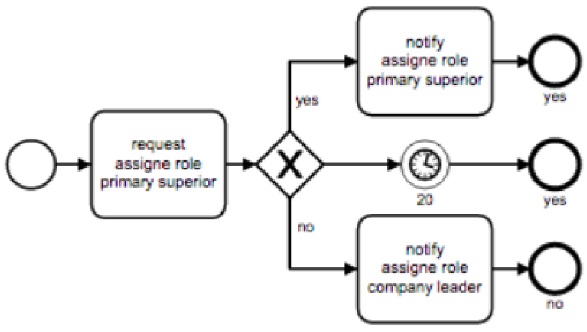


Figure 5.10. Diagram before converting

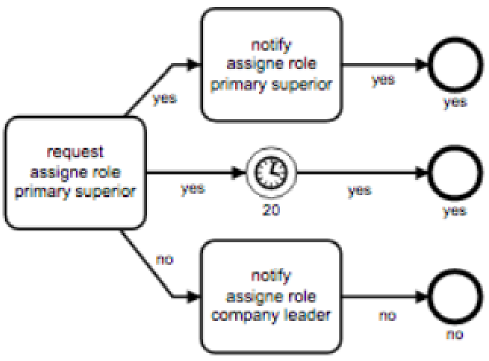


Figure 5.11. Diagram after converting

Basically, Figure 17 shows the form in which diagram is saved to the database. ExclusiveGateway and StartEvent elements are not internally represented, because they serve only for visualization. Rest of the elements is converted into business objects

(BO). Each business object is mapped to the particular entity in the database and its attributes are mapped to the particular columns. There are three types of business objects in the application:

- **Workflow**
  Workflow BO serves to identify diagram. It contains only ID field and serves for the grouping elements, which belong to the same diagram.
- **Element**
  Element BO is the internal representation of the diagram element. It contains information about the element's value (text description), workflow identification and the type of element, which can be EndEvent, TaskEvent or Timer.
- **Transition**
  Transition BO represents the connection between two elements. It contains the information about its source and destination, response and responder of the request. The responder of request is defined according to the context, where it is located in the diagram. It can be of three types:

  - **Person responder**
    If transition has exclusive gateway on its way, then the responder is a person. In this case, the result of request is defined according to the answer of a particular role representer. In target application the request is sent directly to the person who accepts or declines through some interface.
  - **Timer responder**
    The responder is Timer when the transition contains it on its way. It defines the amount of time of waiting. If the time runs out, the answer is automatically positive.
  - **System responder**
    The responder is system in the rest of cases and the answer from it is always positive.

Chapter 6
Evaluation

The purpose of the evaluation is to check the correct behavior of the client-server communication in the application. It means, that the validation of the diagram which is sent from the client side is done in a right way and all the possible mistakes are treated correctly. For the purposes of this thesis, both modules are combined into a separate application, so no lower layers are needed for testing.

6.1 Error handling

There are two kinds of response types, that can be sent from from the server. The first one describes the successful result of the diagram validation. It means, that the diagram follows all the default and custom syntax rules. A correct diagram and the server response is shown in the Figure 6.1.

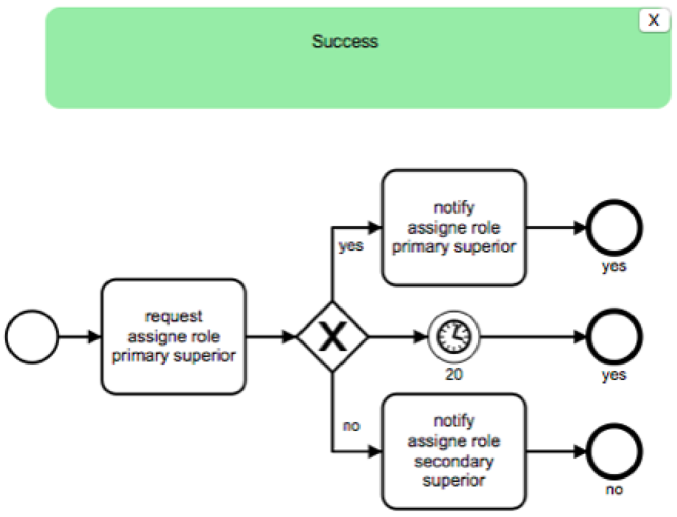
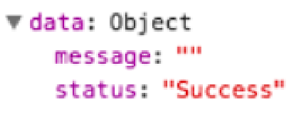


Figure 6.1. Correct diagram

According to the custom rules, the diagram elements are supposed to be in this strictly defined order and the particular elements have descriptions. In this case, the server response in JSON format looks like in the Figure 6.2:

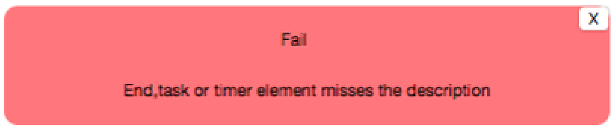


```
▼data: Object
   message: ""
   status: "Success"
```

Figure 6.2. Server response: success

6.1.1 Error example: element misses description

Each element in the workflow has its significant role and most of them need to have description. The description gives the fundamental information about the request. An example of an element, which misses the description, is shown the Figure 6.3.



Fail

End,task or timer element misses the description

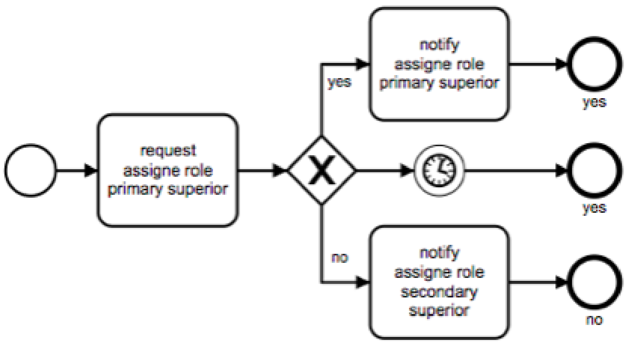
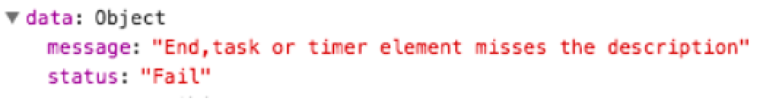


Figure 6.3. Not fully described diagram

In this case, the Timer element misses the description, where the amount of time for waiting has to be defined. The actual server response in shown in the Figure 6.4.



```
▼data: Object
   message: "End,task or timer element misses the description"
   status: "Fail"
```

Figure 6.4. Server response: missing description

6.1.2 Error example: diagram contains isolated elements

According to the custom rules, a diagram can represent only one workflow process. Thereby, the diagram can not contain isolated elements, like it is shown in the Figure 6.5.

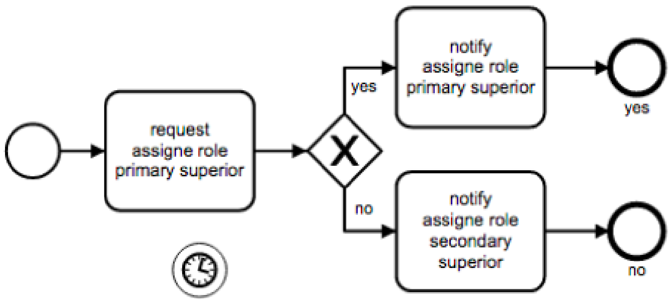
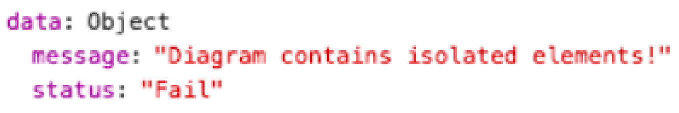


Figure 6.5. Diagram contains isolated elements

Server response:



```
data: Object
  message: "Diagram contains isolated elements!"
  status: "Fail"
```

Figure 6.6. Server response: isolated elements

6.1.3 Error example: diagram is not complete

Each diagram is obligated to have some elements, such as, for example, start and end events. The Figure 6.7 shows an example of diagram, which is not complete.

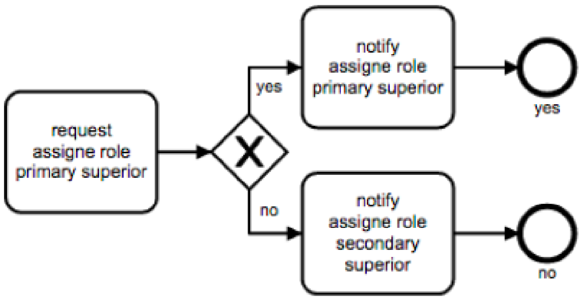
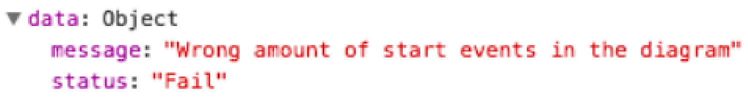


Figure 6.7. Diagram is not complete

Server response:



```
▼data: Object
   message: "Wrong amount of start events in the diagram"
   status: "Fail"
```

Figure 6.8. Server response: incomplete diagram

6.1.4 Error example: wrong order of elements in the diagram

The wrong order of elements is treated by both: XSD validation and Java functions validation. In some cases, it is possible to concretize the exact place of the missing element and in other cases, the general notification is thrown.

Figure 6.9 represents the diagram with the wrong order of elements.

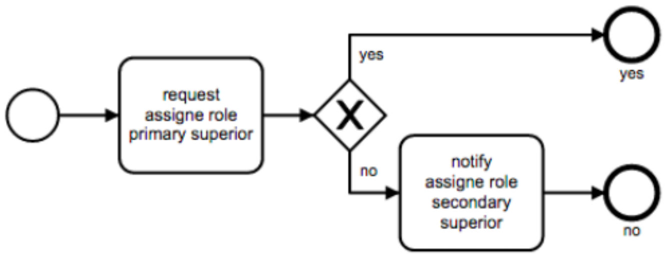
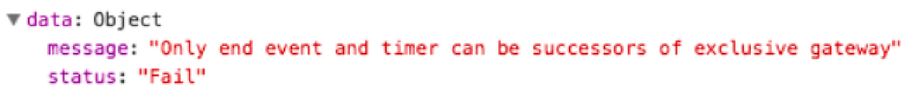


Figure 6.9. Wrong order of elements

Server response:



```
▼data: Object
   message: "Only end event and timer can be successors of exclusive gateway"
   status: "Fail"
```

Figure 6.10. Server response: wrong elements order

6.1.5 Error example: diagram contains loops

The diagram is not allowed to contain loops, like it is shown in the Figure 6.11

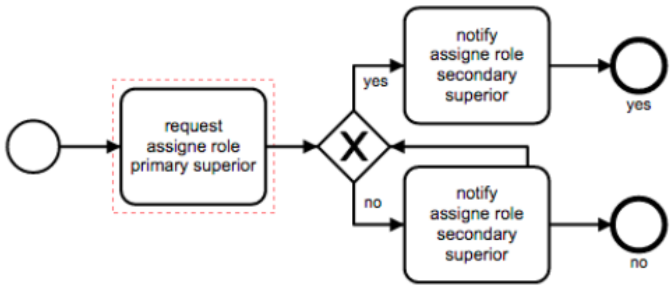
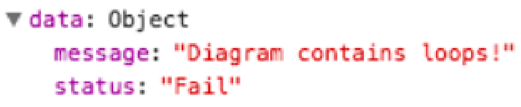


Figure 6.11. Diagram contains loops

Server response:



```
▼data: Object
   message: "Diagram contains loops!"
   status: "Fail"
```

Figure 6.12. Server response: diagram contains loops

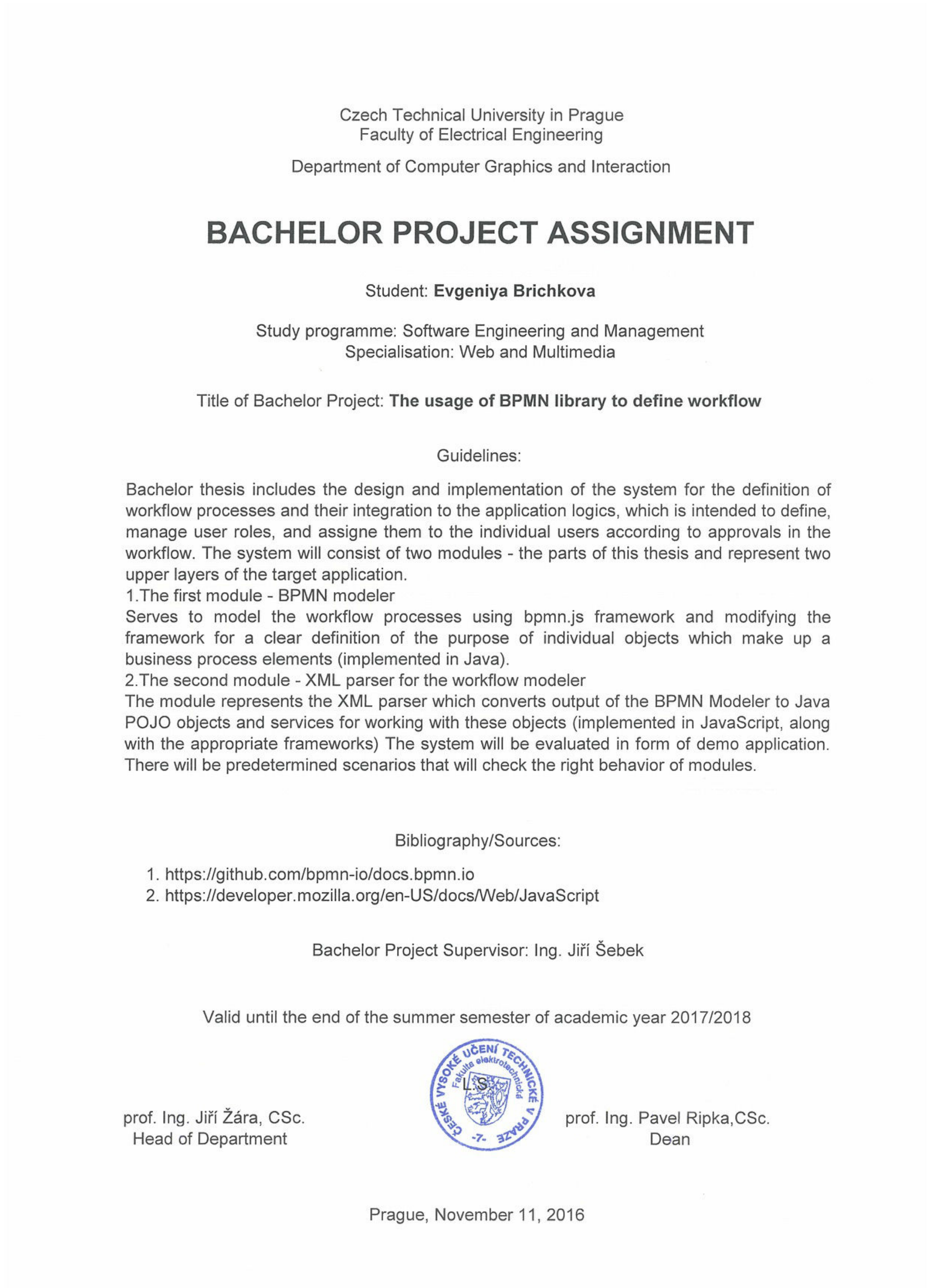
Chapter 7
Installation

- import Maven project to IDE
- deploy the application in Tomcat server
- run the application
- open in browser: http://localhost:port/bpmn-modeler/bpmn/create-diagram with the corresponding port, where Tomcat is running
- done

Chapter 8
Conclusion

The amount of huge companies increases fast and some of their internal management issues can be solved with the help of management tools. The management tools, if they are used appropriately, can be powerful enablers of change and actions in companies. These tools make it possible for people to implement the processes and strategies they desire. One of those management issues is the role approval processes, which is discussed in this thesis. The theoretical part of this thesis is devoted to the investigation of business process management and modeling, as well as the comparison of different modeling techniques and tools. The investigation also included the attempts to find out which tools are appropriate for the purposes of the target system and why BPMN-js library is among them. Since the library is quite new and its development is still in progress, the functionality and programming features of existing modules are explored. The core modules and features of the library are described in the text. Practical part implements the top part of the target application: the two upper layers of it. BPMN modeler belongs to the client side of the target system, while the parser belongs to the server side. On the purpose of this thesis, these two layers are combined into a separate Spring MVC application. Thereby, the modules can work together as a separate application and can be tested independently of the target system. The details of implementation for both client and server sides are described in the Chapter 6. It also includes the brief description of the Spring MVC application configuration. The analysis and design of the application are described in the Chapter 5. Domain models represent the architecture of the application's client side and the way, in which the library was customized. The tests of the application were performed according to the different scenarios, which are described in the Chapter 7. They proved, that both modules work according to the requirements of the target system. Thereby, only the diagrams, which match the custom rules are accepted by the server and ready to be handled by the next layer.

Appendix A
Specification



Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Computer Graphics and Interaction

BACHELOR PROJECT ASSIGNMENT

Student: **Evgeniya Brichkova**

Study programme: Software Engineering and Management
Specialisation: Web and Multimedia

Title of Bachelor Project: **The usage of BPMN library to define workflow**

Guidelines:

Bachelor thesis includes the design and implementation of the system for the definition of workflow processes and their integration to the application logics, which is intended to define, manage user roles, and assigne them to the individual users according to approvals in the workflow. The system will consist of two modules - the parts of this thesis and represent two upper layers of the target application.
1.The first module - BPMN modeler
Serves to model the workflow processes using bpmn.js framework and modifying the framework for a clear definition of the purpose of individual objects which make up a business process elements (implemented in Java).
2.The second module - XML parser for the workflow modeler
The module represents the XML parser which converts output of the BPMN Modeler to Java POJO objects and services for working with these objects (implemented in JavaScript, along with the appropriate frameworks) The system will be evaluated in form of demo application. There will be predetermined scenarios that will check the right behavior of modules.

Bibliography/Sources:

1. https://github.com/bpmn-io/docs.bpmn.io
2. https://developer.mozilla.org/en-US/docs/Web/JavaScript

Bachelor Project Supervisor: Ing. Jiří Šebek

Valid until the end of the summer semester of academic year 2017/2018

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická
L.S.
-7-

prof. Ing. Jiří Žára, CSc.
Head of Department

prof. Ing. Pavel Ripka,CSc.
Dean

Prague, November 11, 2016

Appendix B
Symbols

BPMN	Business Process Model and Notation
XML	Extensible Markup Language
XSD	XML Schema Definition
RAD	Role Activity Diagrams
POJO	Plain Old Java Object
JAXB	Java Architecture for XML Binding
DFD	Data Flow Diagram
DTO	Data Transfer Object
DAO	Data Access Object
NPM	Node Package Manager
AJAX	Asynchronous JavaScript and XML
HTTP	Hypertext Transfer Protocol
XJC	XML to Java Compiler
MVC	Model-View-Controller
js	Java Script
POM	Project Object Model
BO	business object
png	Portable Network Graphics
svg	Scalable Vector Graphics
jpg	Joint Photographic Experts Group
XPDL	XML Process Definition Language
PDF	Portable Document Format

Appendix C
Code examples

Appendix D
References

[1] R. KOSTER, Stefan.*An evaluation method for Business Process Management products.* Enschede, Netherlands, 2009.
https://www.utwente.nl/ewi/trese/graduation_projects/2009/koster.pdf

[2] Executing a Business Model: BUSINESS Modeling LIFE CYCLE. M. BRIDGELAND, DAVID a RON ZAHAVI. *A Practical Guide to Realizing Business Value.* 1. San Francisco, CA, USA: Morgan Kaufmann, 2008, 346-347. ISBN 9780080920955.

[3] ALDIN, Laden a Sergio DE CESARE.*A comparative analysis of business process modeling techniques: role activity diagram (RAD)* [online]. Oxford, UK: UKAIS, 2009, 9-10 [cit. 2017-01-03].
http://aisel.aisnet.org/ukais2009/2

[4] TANGKAWAROW, Irene a J WAWORUNTU.*A Comparative of business process modeling techniques: Activity Diagrams* [online]. IOP Conference Series: Materials Science and Engineering, 2016 , 9-10 [cit. 2017-01-03]. DOI: 10.1088/1757-899X/128/1/012010.
http://iopscience.iop.org/article/10.1088/1757-899X/128/1/012010/pdf

[5] Business Process Model and Notation (BPMN). In:*Quizlet*[online]. 2015 [cit. 2017-01-03].
https://quizlet.com/91709565/learnsmart-chapter-22-flash-cards/

[6] ISO/IEC 19510: Business Process Model and Notation. 2.0.2. Geneva, Switzerland: International Organization for Standardization (ISO), 2013, 570 s.
http://www.omg.org/spec/BPMN/ISO/19510/PDF/

[7] TIBCO Business Studio. In:*TIBCO*[online]. Palo Alto, CA, USA: TIBCO Software, 2014 [cit. 2017-01-03].
http://www.tibco.com/assets/blt21fbce784d794e28/ds-business-studio.pdf

[8] Comparison of BPM tools. In:*Techajo News!*[online]. 2014 [cit. 2017-01-03].
http://www.techajo.com/comparison-bpm-tools-pegasystems-ibm-tibco/

[9] What is Java Web Start and how is it launched? ORACLE.*Java*[online].[cit. 2017-01-03].

https://www.java.com/en/download/faq/java_webstart.xml

[10] Bpmn-js: Getting familiar with bpmn-js, one step at a time. In:*bpmn.io* [online]. camunda Services, 2013 [cit. 2017-01-03].
https://bpmn.io/toolkit/bpmn-js/walkthrough/

[11] Dependency injection. In:*SearchSOA*[online]. TechTarget, ©2001-2017 [cit. 2017-01-03].
http://searchsoa.techtarget.com/definition/dependency-injection

[12] BELL, Donald. The sequence diagram. In:*IBM*[online]. 2004 [cit. 2017-01-03].
http://www.ibm.com/developerworks/rational/library/3101.html

[13] Introduction to JAXB. In:*ORACLE*[online]. [cit. 2017-01-03].
https://docs.oracle.com/javase/tutorial/jaxb/intro/

[14] Java Architecture for XML Binding (JAXB). In:*ORACLE*[online]. [cit. 2017-01-03].
http://www.oracle.com/technetwork/articles/javase/index-140168.html

[15]*RequireJS: A JAVASCRIPT MODULE LOADER*[online]. San Francisco: Andy Chung, ©2011-2015 [cit. 2017-01-03].
http://requirejs.org/

[16] Web MVC framework. Spring.io [online]. [cit. 2017-01-05].
https://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html

Appendix E
Content of the practical part

The content of the practical part:

- materials.zip contains the tex source files, figures and diagrams
- The_usage_of_BPMN_library_to_define_workflow.pdf is the text of the bachelor thesis
- assignment.pdf is the assignment of the bachelor thesis
- modeler.zip contains the implementation of the application