BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

POLYMER ELEMENTS FOR A BUSINESS PROCESS MANAGEMENT SYSTEM

POLYMER ELEMENTY PRO SYSTÉM ŘÍZENÍ PODNIKOVÝCH PROCESŮ

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR
AUTOR PRÁCE

Bc. SERHII PAHUTA

SUPERVISOR
VEDOUCÍ PRÁCE

Ing. RADEK BURGET, Ph.D.

BRNO 2017

Master's Thesis Specification/20227/2016/xpahut01

Brno University of Technology - Faculty of Information Technology

Department of Information Systems Academic year 2016/2017

Master's Thesis Specification

For: **Pahuta Serhii, Bc.**
Branch of study: Computer Networks and Communication
Title: **Polymer Elements for a Business Process Management System**
Category: Information Systems

Instructions for project work:

1. Get familiar with the Polymer and Material Design web components and their alternatives.
2. Study the architecture of the jBPM Workbench system and the KIE Execution Server.
3. Design the architecture of a library that integrates a set of Polymer components into the KIE environment.
4. Implement the designed library in JavaScript.
5. Implement an application that demonstrates a real-world usage of the implemented component library. Perform testing of all implemented parts.
6. Summarize the achieved results.

Basic references:

- Juneau, J.: Java EE 7 Recipes, Apress, 2013

Requirements for the semestral defense:
Items 1 to 3

Detailed formal specifications can be found at http://www.fit.vutbr.cz/info/szz/

The Master's Thesis must define its purpose, describe a current state of the art, introduce the theoretical and technical background relevant to the problems solved, and specify what parts have been used from earlier projects or have been taken over from other sources.

Each student will hand-in printed as well as electronic versions of the technical report, an electronic version of the complete program documentation, program source files, and a functional hardware prototype sample if desired. The information in electronic form will be stored on a standard non-rewritable medium (CD-R, DVD-R, etc.) in formats common at the FIT. In order to allow regular handling, the medium will be securely attached to the printed report.

Supervisor: **Burget Radek, Ing., Ph.D.**, DIFS FIT BUT
Beginning of work: November 1, 2016
Date of delivery: May 24, 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

Dušan Kolář
Associate Professor and Head of Department

Abstract

Goal of this thesis is to introduce and analyze Web Components technology and JBoss BPM Suite. Web Components technology gives ability to build lightweight HTML components with predefined functionality. Implementation of this work shows how this approach is applied for creating set of web components, which contains all required custom elements for jBPM user console.

Abstrakt

Cílem této diplomové práce je představit a analyzovat technologie Web Components a JBoss BPM Suite. Technologie Web Components dává možnost vytvořit lehkou HTML komponentu s předefinovanou funkčnosti. Vyslédná implementace ukazuje jak tento přístup se používá pro tvorbu sady web komponent, která obsahuje všechny potřebné elementy pro uživatelské konzoli jBPM.

Keywords

JavaScript, web, Web Components, Polymer, JBoss, BPM, BPMN, reusable HTML elements, Shadow DOM, JavaScript framework, business processes

Klíčová slova

JavaScript, web, Web Components, Polymer, JBoss, BPM, BPMN, znovu použitelné HTML elementy, Shadow DOM, JavaScript knihovna, podnikové procesy

Reference

PAHUTA, Serhii. *Polymer elements for a Business Process Management System*. Brno, 2017. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Radek Burget.

Polymer Elements for a Business Process Management System

Declaration

I hereby declare that I have worked on this master's thesis on my own under the supervision of Ing. Radek Burget, Ph.D. All the relevant information sources, which I used for this project, are mentioned in Literature section.

.....
Serhii Pahuta
2017

Acknowledgements

I would like to thank my supervisor Ing. Radek Burget for his continued support, patience and meaningful remarks and comments. Also I want to thank Ivo Bek, my technical consultant and colleague, for his help in understanding of technologies and architectures used in this thesis.

Contents

Chapter 1

Introduction

Nowadays evolution of web technologies leads to fast growth of functionality in modern web applications. They acquire new powerful features and UI experience without loss of performance. That's why users are starting to prefer on-line web services more than desktop applications.
After releasing new JavaScript standards (EcmaScript 6/7) and application platform NodeJS, JavaScript obtained new meaning as a programming language. Creating full-stack web project requires great flexibility and modularity of language. This is a place where JavaScript developer's community meets the problem.

At this point Web Components try to solve existing issue. This technology brings not only possibility of creating, importing and using reusable custom HTML elements. Also it brings a standard which should be supported by internet browsers.

JBoss BPM (Business Process Management) is an open source project which provides the platform for building automated business processes. This suit contains user console – jBPM Business Central – web application with big amount of instruments for creating, editing and controlling all processes. But for non-technical users such multifunctional application could be very complex and unclear. That's why companies creates own more simple user consoles. Small web application building is not so difficult, but it can take a lot of time.

Objective of this thesis is implementation of Web Components elements with predefined logic and REST query. It should help to create user's web console for jBPM suite quickly and easily.
Web Components as a technology described and analyzed in chapter 2.1, 2.2. Web Components based framework, which is used for this thesis, is described in chapter 2.3.

Whole JBoss BPM system and BPMN standard are introduced in chapter 3. Based on analysis from two previous chapters, architecture of KIE components library was proposed and presented in chapter 4.

In chapter 5 can be found description of implementation and specific for each KIE element technical details. How created components can be integrated into web application shows chapter 5.5.

Chapter 2

Web Components

In scope of this thesis we will implement JavaScript framework which consist of web components collection for jBPM user console. But first we need to get familiar with new technology "Web Components" and look how it works. In chapter 2.1 we briefly look through history of Web Components. Architecture and technology functionality will be described in chapter 2.2 .Web component framework Polymer, which will be used for implementing our library, is described in chapter 2.3.

2.1 History of Web Components

The very first attempt to create a semblance of encapsulating components was made by Microsoft. In 1998 they proposed the technology named "HTML Components" (HTC for short) to the W3C organization [13]. But this attempt failed and Microsoft eventually shut down the project.

Also Mozilla has tried to do own contribution to the web components [14]. They released Xenogamous Binding Language 1.0 in 2001. This technology has gained some popularity, but it has not been widely known. In 2007 Mozilla announces XBL 2.0, but project was failed later.

One of the first successful attempts to make reusable web elements was made by Dojo Toolkit. It's a JavaScript framework designed for complex client-side web development. Among all of Dojo's features (e.g. cross-browser APIs, build tools for optimizing JavaScript and CSS, unit testing, etc.), there's user-interface widgets and layouts. Adding a widget to page is very simple and requires only few lines of code. It's a quite good example of reusable web elements. This framework showed how can be useful and comfortable creating own UI modules [7].

Alex Russel, co-founder of the Dojo Toolkit, saw the potential in these reusable components. Later, together with colleges from Google, he has started work on Web Components. In 2011, Alex had a talk at the conference Fronteers. He presented the lecture "Web Components and Model Driven Views" [15]. In 2012 there was a first draft of Web Components specification. From this moment Google began to make great investment into this technology and started development of library based on Web Components. Thus in 2013 Google has announced open-source JavaScript framework - Polymer. The library is intended to create own HTML elements or even a complex web structures or modules. More detailed information about Polymer see chapter 2.3.

2.2 Main architecture of Web Components

The main advantage of this technology is that it allows you to create self-defined encapsulated elements with the appropriate style and logic of behavior. They can be distributed on the Internet for use by others.

Simple Web Components structure diagram is shown in Figure Figure 2.1

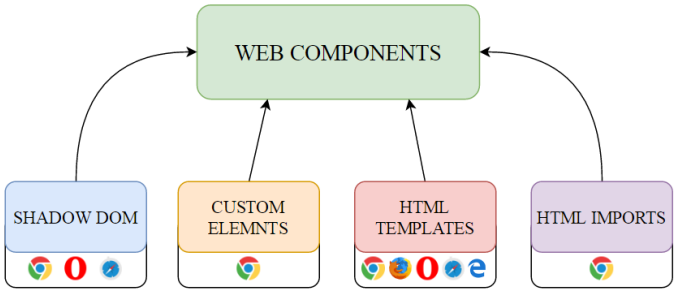


Figure 2.1: Simple structure of Web Components and browser support

Actually Web Components is an umbrella name for a few technologies that allow creating and including reusable components into web application. All of them are well documented and have specifications drafts on W3C website [16].

Next we will analyze how each technology works.

2.2.1 Shadow DOM

Encapsulation is key property of web component development. Without it there's no way to guarantee that CSS rules meant for included component wouldn't cause CSS conflicts. Also it is hard to predict if external frameworks or code snippets doesn't interference with the component's DOM tree.

That is why the existence of Web Components isn't possible without encapsulation of styles, id, names and logic. Shadow DOM is perfectly fitted for solving this problem. There are some standard elements (e. g. `<input>`, `<textarea>`, `<video>`, etc.) which use Shadow DOM for own structure isolation and scoping [4].

"Core" of this technology is possibility to create a scoped DOM tree attached to the element and separated from its actual children at the same time. This isolated subtree has a name – "shadow tree". The element it's a point where shadow tree was attached called "shadow host". Anything added to created shadow tree becomes local to the hosting element. Same rule is applied for `<style>` tag and that's a way how CSS scoping is solved.

Diagram of Shadow DOM scoping is shown in FigureFigure 2.2.

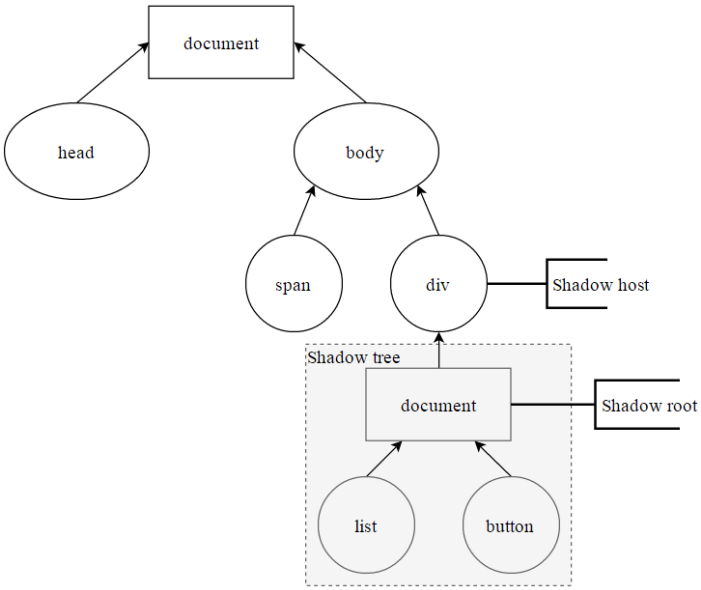


Figure 2.2: Example of Shadow DOM scoping

Besides CSS scoping there are other problems that can be solved by Shadow DOM technology:

- CSS queries simplification. Scoping of DOM elements allows developer use more generic class names, id. In turn it gives a possibility to use simple CSS selectors and don't be afraid of naming conflicts;
- Composition. Creating components with declarative API design;
- Component's DOM isolation. There's no way to get access to component's elements and cause security violation.

Adding Shadow DOM is made by JavaScript command `element.attachShadow(shadowRootInit)` where `element` is a host for shadow tree. `shadowRootInit` is an object which contains additional parameters: `mode` - can be set to `open` or `close`. Option `open` specifies open encapsulation mode. This means there will be access to shadow DOM elements just after it was created. In this case `attachShadow()` function returns an object containing shadow tree

Option `close` specifies close encapsulation mode. Behavior of function will be a quite opposite. Access to shadow DOM elements will be closed and unavailable. Native HTML elements, such as `<video>`, work in a similar way. When `close` option is used `attachShadow()` returns `null` value.

Also in the Shadow DOM specification term "light tree" is used [9] (or "light DOM" in other media resources). This term pointing to tree created by web component's user and appended to custom element. User's structure is not a part of the shadow tree and it is child of a custom HTML element [4]. An example of light tree is shown in the Figure Figure 2.3.

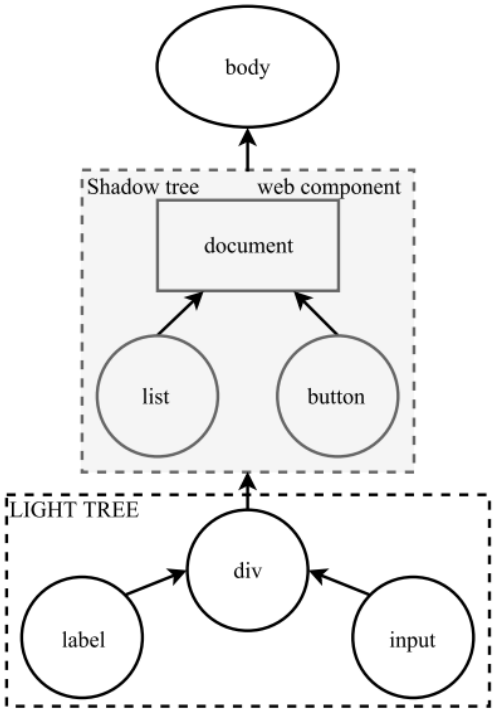


Figure 2.3: Example of light DOM

Also developer can define how to render user's markup inside component. This is done using the tag `<slot>`. He reserves areas in the component structure in which will be added user's elements or whole tree.

There is also exists a concept Flattened DOM tree. This tree is obtained as a result of the browser distributing the user's light DOM into component's shadow DOM and rendering the final product.

For example there is a custom element `<my-element>` and user decides to add some elements to it. Resulting code then looks like this:

```
<my-element>
  <!-- the image and label are my-elements's light DOM -->
  <img src="logo.png" slot="icon">
  <label>Hello, dear customer!</label>
</my-element>
```

Shadow tree of component `<my-element>` will look like this:

```
#shadow-root
  <style>...</style>
  <slot name="icon"></slot>
  <label id="label-wrap">
    <slot>my element's label</slot>
  </label>
```

After browser's render we will get structure named Flattened DOM tree:

```
<my-elemnt>
  #shadow-root
    <style>...</style>
    <slot name="icon">
      <img src="logo.png" slot="icon">
    </slot>
    <slot>
      <label>Hello, dear customer!</label>
    </slot>
</my-elemnt>
```

Nowadays Shadow DOM technology is supported only by 3 popular browsers: Google Chrome, Opera and Safari. For supporting in other browser polyfill library is used (e.g. open-source framework "shadydom")

2.2.2 Custom Elements

Today set of standard HTML tags consist of many necessary elements. Also HTML5 has brought a lot of new useful ones, without which it is hard to imagine modern development of web applications. But this set is not enough to create a big client-side application. To achieve main UI goals developers create huge structures built of large number of native elements. It is very difficult to debug such application.

If Shadow DOM solves the problem with isolation and encapsulation of this structure and its logic, the Custom Elements solves problem with definition of own html element. It is much more handy insert one tag into code than whole bulky tree. It brings a standards-based way to create reusable components using only native web technologies.

Another advantage of Custom Elements is an ability to expand functionality of the native html elements or elements of other users

Custom element creation is made by JavaScript command. `customElements.define(tagName, tagClass)`. A special point here that second argument `tagClass` is a JavaScript class. First parameter `tagName` argument is a string. Whole definition of new element should be done like this:

```
class MyElement extends HTMLElement {...}
window.customElements.define('my-element', MyElement);
```

It is important to notice that `customElemnts.define` should be executed in global scope. That's why `window` object used in example.

To reduce the number of code lines it is possible to pass second argument as an anonymous class:

```
window.customElements.define('my-elemeny', class extends HTMLElement
{...});
```

After this developer can use his tag in html markup like any other native element: `<my-element></ my-element>`. It is also possible to attach event listeners to it, define id, class, properties, etc.

There are some rules and limitations on creating custom elements [5]:

- Name of custom tag must contain a dash symbol. HTML parser should be able to distinguish custom elements from native ones;
- New element can be registered only once. It is not acceptable to register element with the same tag name;
- There is no way to create self-closing element.

As mentioned before it is possible to expand other developer's or native html elements. This is implemented with help of class inheritance:

```
class BetterElement extends MyElement {
  constructor() {
    super(); // this calls extended class constructor
    ...
  }

  myMethod() {
```

```
    //extend method functionality here
    //if parent class functionality is required
    //call super.myMethod()
  }

  mySecondMethod() {
    ...
  }
}

customElements.define('better-element', BetterElement);
```

Original element will stay the same without any changes. Expanding of functionality is made in new class.

For now Custom Elements v1 is supported only by Google Chrome browser. For other browsers polyfill library can be used [5].

2.2.3 HTML Templates

The main task of this technology is to make browser to ignore the contents of the template. This allows to store structure of the component directly on the web page at the same time avoiding the rendering. Thus the developer can control the moment when media resources should be loaded (e.g. contents of tags `<img>`, `<video>`, etc.) or scripts should be executed.

This ability is important for Web Components. Because when you import components into web page, the contents of custom element will be "invisible" to the browser. This is good optimization, because without templates performance of browser would go down with each additional component.

To create HTML template developer needs to use `<template>` tag. An example of usage is shown below:

```
<template id="hello-template">
  <img src="" alt="Hello, WORLD!">
  <span class="desc"></span>
</template>
```

For displaying template as an element on the page, it is needed to copy its content into another element. This is done in the following way:

```
var temp = document.querySelector('#hello-template');
```

```
temp.content.querySelector('img').src = 'icon.png';

var clone = document.importNode(temp.content, true);
document.body.appendChild(clone);
```

HTML Templates are supported by all modern popular browsers: Google Chrome, Mozzila Firefox, Opera, Safari, Microsoft Edge [3].

2.2.4 HTML Imports

Import of web components is the last thing missing for increasing the flexibility and modularity in web development. HTML Imports technology solves this problem. Among standard HTML tags already exist those that can do an import of styles (`<link rel = "stylesheet">`), scripts (`<script src>`), media information (`<img>`, `<audio>`, `<video>`, etc.)

Import HTML is not something new. Before it there are several methods already exist:

- `<iframe>`. Heavy weight and very difficult to control;
- AJAX with `responseType=”document”`. Because of executing in JavaScript code this method is implicit.
- Some community methods. For example content of HTML page is embedded into string which is received by executing some script. Then string is parsed and all HTML structure is appended to existing HTML element.

HTML Imports brings standardized way how to do it. Also it allows import HTML pages with styles and scripts included. Need to mention that this type of import has `text/html` mimetype.

For Web Components this technology has meaningful purpose. Web developer can store his entire component’s markup, styles, JavaScript logic in one `.html` file. Then component’s consumer just imports it into own web page only with one line of code.

For importing `<link rel=”import”>` tag’s used. Example of usage:

```
<link rel=”import” href=”path/to/page.html”>
```

One more additional feature if HTML Imports that it allows to pass to href attribute URL to another domain. But such import requires enabled CORS (Cross Origin Resource Sharing) on the side from where import is happening. HTML tag in this case will be like this:

```
<link rel=”import” href=”http://other-domain.com/example.html”>
```

Similar to HTML Templates, content of HTML page is not rendered at just after importing. Browser is only parsing it. But if HTML page for import contains scripts they will be executed. In

this case to avoid rendering block of main page `<link rel="import">` can be used with `async` attribute.

All scripts and other included things in imported page are parsed only once [17]. It's a good feature for dependency management. In this way HTML Imports secures that all resources are loaded and no conflicts caused by double parsing same include or import.

HTML Imports is supported only by Google Chrome. But for other browsers polyfill framework can be used.

2.3 Polymer framework

At Google I/O 2013, Google presented a new web user interface (UI) framework called Polymer. After a huge contribution to the development of Web Components specification, Google realized that there is a need in the library which offers all the features of this technology. Because today Web Components are not standardized yet, Polymer makes it possible to create and "touch" reusable elements. This framework has a cross-browser ability because of using polyfill functions. The library also has some additional features: data binding, predefined Google elements, binding properties and attributes.

Polymer's architecture has the following layers:

- Foundation layer. Provides basic technologies for use (Shadow DOM, HTML Imports, etc.);
- Core layer. Provides functions and technologies which are not in future Web Components standard;
- UI elements layer. Provides a set of components and elements .built on Core layer.

Among all standard Web Components technologies in Foundation layer there are also:

- DOM Mutation Oberservers and `Object.observe()` which probably will appear in EcmaScript 7 standard. They allow observing changes in DOM elements and JavaScript objects.
- Model-Driven Views (MDV). It gives an ability to make data bindings in HTML.Does data-binding directly in HTML (this technology not standardized).
- Web Animations. API unifying several of the web's animation approaches.

Core and UI elements layers purpose is provide a huge number of fully functional UI widgets. All of them are belong to some category:

- App elements - basic for web application elements. For example tool-bar menus or scroll boxes;
- Iron elements - set of visual and non-visual utility elements. Includes elements for working with layout, user input, selection, and scaffolding apps;

- Paper elements - visual elements that implement Google's Material Design. Basic UI is here: buttons, checkboxes, sliders, tabs, etc.;
- Google Web Components – elements with predefined logic for working with Google's Web Services and other Google APIs;
- Gold Elements - these elements are built for e-commerce purposes. For example input fields for credit cards and CVV numbers;
- Neon elements - provides all necessary functionality for creating web animations and special effects;
- Platinum elements - consists of elements for implementing push notification technology, working with Bluetooth devices, subscribing and messaging services;
- Molecules - there are elements that wrap another necessary for web developing JavaScript (third-party) libraries and frameworks.

Polymer is designed in way to support interoperability. It means that web component produced by other framework, e.g. X-Tag by Mozilla, will work correctly in Polymer's code [12].

New element can be registered by calling Polymer function. Registration is made in separate HTML file, where all styles, DOM structure and logic should be stored. For example: if developer wants to create new component `<my-element>` he needs to create my-elem.html file and put there following code:

```
<link rel="import"
href="https://polygit2.appspot.com/components/polymer/polymer.html">

<script>
  Polymer({
    is: "my-element",

    ready: function() {
      this.textContent = "Hello, my dear customer!"
    }
  });
</script>
```

After defining Polymer component developer needs to import it to main HTML page:

```
<!DOCTYPE html>
  <head>
    <script
src="https://polygit2.appspot.com/components/webcomponentsjs/webcomp
onents-lite.js"></script>
    <link rel="import" href="my-element.html">
  </head>
  <body>
    <my-element></my-element>
```

```
  </body>
</html>
```

As we can see on first line used HTML import to load Polymer library. Framework itself is a web component too. After that Polymer function is called. It takes object as a single argument. This object contains set of parameters describing how component should be created. In example above object's property is defining name of custom tag and property ready defining a callback function which will be executed when element is included to main HTML page [2].

There are some alternative web components frameworks like X-Tag, Bosonic, SkateJS. But for this master's thesis Polymer was chosen because of its high performance.

Chapter 3

BPMN standard and jBPM architecture

Since this master's thesis about creating set of web components for KIE user console we need to know bunch of BPM principles and internal structure of workflow engine.

What is BPMN standard and its some specifications are introduced in chapter 3.1. JBoss BPM system/suit architecture and how it works is described in chapter 3.2. Short information about jBPM Business Central user console is in chapter 3.3.Which REST API's requests will be used by web components is described in chapter 3.4.

3.1 Standard BPMN 2.0

BPMN (Business Process Model and Notation) is a standard which was developed by Object Management Group in 2005.Later in 2011 BPMN v2 was released. Actually before second version release it was named as Business Process Management Initiative [1]. But name was changed because of new introduced features: notational and diagramming elements.

This standard describes a bunch of rules and elements required to create a business process. It is designed in such way that any user would be able correctly create or read already existing business process [6]. It means that people without technical education can use BPMN standard. At the same time designed business procedure will be correctly implemented by developers. Such approach guarantees one-sided understanding of technology by both parties (designing and implementing) and does not require any special skills.

Created according to the standard business process can look like a notation or diagram of actions. As a result, business procedure can be easily read by a person as well as by program that executes the process.

For graphical representation certain set of figures and icons is used. Software that implements BPMN must strictly use graphic elements specified in the standard.

Main target of this thesis is to create a library which allows developing a web application that will be able to manage the existing business processes. So here will not be given all elements and events that are included into standard BPMN. We will look through basic elements only.

All elements are divided into 5 main categories:

1. Flow Objects
   - Events
   - Activities
   - Gateways

2. Data
   - Data Objects
   - Data Inputs
   - Data Outputs
   - Data Stores
3. Connecting Objects
   - Sequence Flows
   - Message Flows
   - Associations
   - Data Associations
4. Swimlanes
   - Pools
   - Lanes
5. Artifacts
   - Group
   - Text Annotation

Flow Objects are used for defining business process definition. Connecting objects are describing how connect Flow Object with each other. Purpose of Swimlanes is grouping elements. Artifacts are used to provide additional information about the Process. There are only two standardized Artifacts, but modelers or modeling tools are free to add as many Artifacts as necessary [10].

List of some basic elements is presented in Table Table 3.1

Table 3.1: Basic BPMN elements

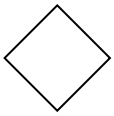
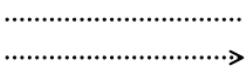
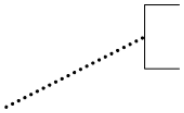
Element name	Graphical representation	Description
Event		"Event" is an action that can be happen during executing of process. Events have a triggers and results. There are three types of Events: Start, Intermediate, End.
Activity		An Activity is a generic term for work that company performs in a Process. An Activity can be atomic or compound. Types of Activities: Sub-Process and Task
Gateway		Gateway controls process flow. It can be used for branching, forking or merging.
Sequence Flow		Shows order of Activites in process
Message Flow		Message Flow shows information flow between two Participants(two separate Pools)

Table 3.2 (cont.): Basic BPMN elements

Association		An Association links information between Artifacts and other graphical elements. An arrowhead indicates a direction of flow
Pool		A Pool represents Participant. It also used as a "swimlane" and a container for partitioning a set of Activities from other Pools.
Lane		Lane can be sub-partition of a Process or Pool. Generally they need for organizing and categorize Activities
Data Object		Data Objects show what Activities has on input and what they have on output.
Message		A Message is used to determine the contents of a communication between two Participants
Group		Group element is used for grouping elements are belongs to same type or category.
Text Annotation		Text Annotation shows additional text information about process or other elements in this process

Each element has expanded graphical representation that corresponds to element's function. For example Gateway element for forking and merging has own icons (look at Figure Figure 3.1).

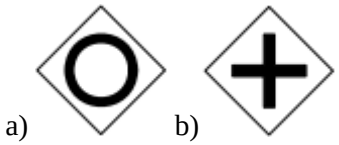


Figure 3.1: a) Merging gateway; b)Forking gateway

Example of simple business process is shown on Figure Figure 3.2

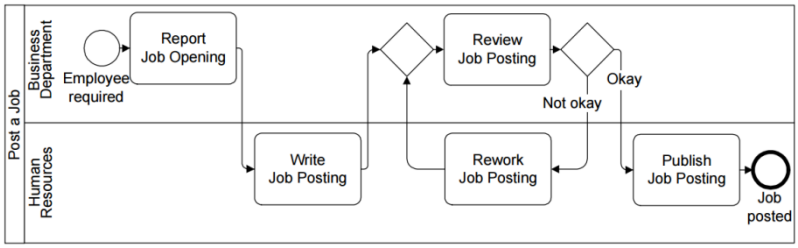


Figure 3.2: Simple BPMN model

3.2 JBoss BPM Suite

Every BPM systems consist of software parts each of which controls business through the whole BPM cycle. Main purpose of such system is making executables definitions from graphical representation of process. It means that everything is happening on action diagram should be done properly by BPM software. If a group of tasks can be concurrent, they turn them into parallel tasks. If output from previous action should be saved into report, system will store this into file on a disk. If some task requires a human activity, BPM software asks suitable person for it (by email or other communication channels). All this actions is hard to control and that's why BPM system usually has a set of modules and layers. Each of them controls own part of business process.

JBoss BPM (jBPM) has the same structure and performs same control over processes. Internal architecture of jBPM system is shown on Figure Figure 3.3

Each component in this chart stands for one particular function inside the BPMS architecture. In this chapter we shortly analyze some of them and how they work.

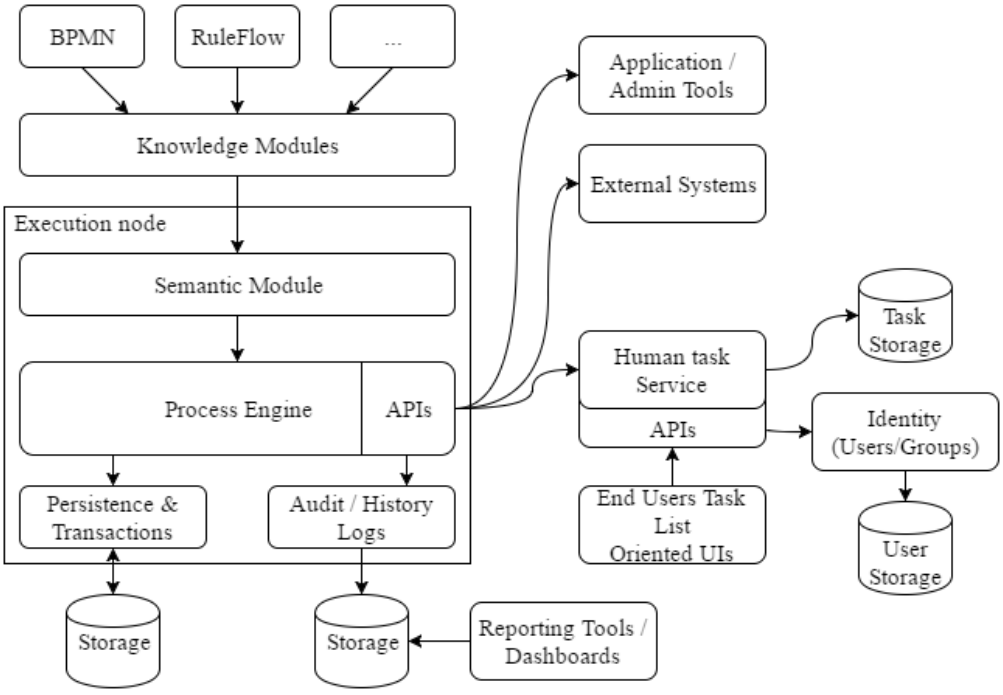


Figure 3.3: BPMs internal structure

From jBPM version 6, jBPM and another parent project called Drools (jBPM implements workflow engine and Drool implements rule engine) have got umbrella name KIE (Knowledge Is Everything) [11]. So in next chapters KIE and jBPM acronyms will be used since they point to same software.

3.2.1 Execution Node

In BPM system exists two terms related to business procedure: process definition and process instance. Process definition is a scaffold of process, bunch of rules and task defined with BPMN standard. Figure 3.2 from previous chapter in BPM scope is a process definition. Created in BPM process by its definition is called process instance.

The execution node is responsible for receiving and parsing process definitions, executing it (or create a process instances in other words), tracking each step of process and to be aware of results from these steps. Parsing is performed by semantic module. It consists of a series of parsers to understand BPMN definition and produce internal structure which will be executed by processes engine.

In turn process engine creates process instances and makes all this tracking [11].
Persistence and Transactions used to provide strategy for content which processes need to be able to execute them from different threads or servers.

3.2.2 Reporting tools / Dashboards

This module is designed for displaying information from given interval of time of executions in particular environment. All this statistics are presented in a way that assists users can track environment's efficiency.

3.2.3 Human Task Service

Process definitions sometimes can contain task that need a human input. In such case process instance should be stopped (if there is no other rules for this situation) and wait until user finish this task. Human Task Service component receives such tasks and starts an interaction with right user. This component guarantees that all tasks are delivered and updates its status.

Searching for proper user is done in following way:

- Identity component is called to get access to the company's users;
- Store the internal state of tasks;
- Update user's UIs to show information related to tasks (all forms and initial information which allow to finish a task).

3.3 jBPM Business Central user console

For managing business processes JBoss BPM Suite provides user console – Business Central. This web application has a lot of instruments for performing all existing operations over the process. One of those instruments is "Process Modeling". It is a simple flow chart drawer (Figure Figure 3.4), where user can draw order of action and tasks, define input and output of process, enable reporting and logging, etc. All of the BPMN elements are there. It makes defining of the process very easy and fast.

Problem can appear when non-technical business user get access to this user console. For using such powerful tool special skills and knowledge is required. A lot of companies who uses jBPM create own web apps with simple UI and logic for their employees. This master's thesis is about simplification of development such applications. Business Central is a great and powerful instrument, but it also can be unclear and difficult to some of business users.

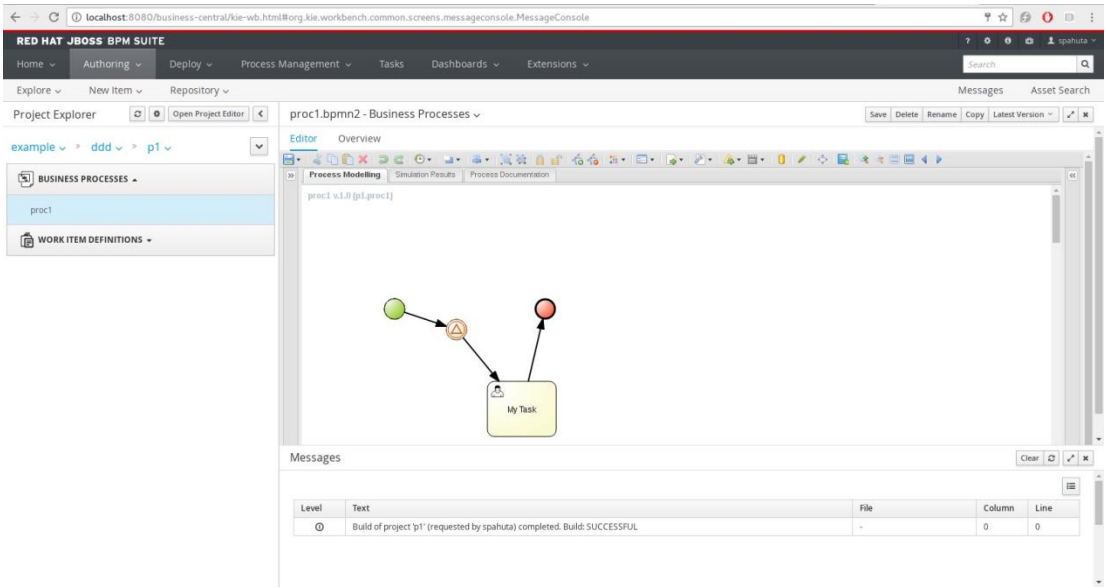


Figure 3.4: Process modeling in Business Central user console

3.4 REST interaction with KIE server

JBoss BPM has a few ways how to create process definitions and drives its instances. One of them is jBPM Business Central mentioned in previous part. Another method is an KIE API. Developer can get access to process directly from Java code with help of API's commands. But for this project REST API was chosen. It defines a lot of HTTP queries for all required operations over the business process: creation of process definitions, executing them, managing process instances, task controlling, etc. For purposes of our framework not all of them will be used.

For each HTTP request base URL (KIE endpoint) defined as

```
http://localhost:8080/kie-server/services/rest/
```

Here `localhost:8080` is used only as example. In real life there should be name of domain. In a list below some of HTTP request (only tail URLs used) to KIE server are introduced and explained:

Process Instances:

Service: `server/containers/{id}/processes`

Endpoints:

- `/instances (DELETE)` - deletes process instances in container with given {id};
- `/instances/{pInstanceId} (DELETE, GET)` – gets information about process instance with given `{pInstanceId}` or deletes it;
- `/instances/{pInstanceId}/variable/{varName} (GET, PUT)` – gets value of given {varName} variable or write a new value;
- `/instances/{pInstanceId}/variables (GET, POST)` – gets information about all variables of given process instance;

Process Definitions:

Service: `server/containers/{id}/processes/definitions`

Endpoints:

- `/{pId} (GET)` – gets information about given `{pId}` process definition;
- `/{pId}/subprocesses (GET)` – gets list of all subprocesses of a given `{pId}` process definition;
- `/{pId}/tasks/users (GET)` – gets list of users who can be responsible for tasks of this process definition;
- `/{pId}/tasks/users/{taskName}/inputs (GET)` – gets information about what data can be as input for particular task;
- `/{pId}/tasks/users/{taskName}/outputs (GET)` – gest information about what data can be as output for particular task;
- `/{pId}/variables (GET)` – gets information about all variables of process definition;

Tasks:

Service: `server/containers/{id}/tasks`

Endpoints:

- `/{tInstanceId} (GET, POST)` – gets or changes task by `{tInstanceId};`
- `/attachments/{attachmentId} (DELETE, GET)` – deletes or get attachment of particular task;
- `/attachments/{attachmentId}/content (GET)` – gets content of a given attachment in a task;
- `/name (PUT)` – changes name of task;

- `/states/ (PUT) –` change status of task. Can be completed, activated, failed, resumed, etc.

Chapter 4

Proposed architecture of the library

Due to this thesis we should implement library, which consists of web components with predefined RESTful API for communication with KIE system. Because of reasons discussed in previous chapter, library will use only small part of jBPM's REST API. Also we already defined some of HTTP requests that will be required.

Architecture of library is separated in parts. Each part defines scope of functionality it uses. It's a main idea of this library and also it fits into Polymer ideology very well. Abstract structure of library is shown on Figure Figure 4.1.

In chapter 4.1 we will look through items and resources needed for managing process instances, and try to decide what UI solutions we can apply for that. The same thing will happen in chapter 4.2, but there we take a look at tasks. Mainly we focus on human tasks, because they required input from user and at this point we need validation of user's data. It will protect executing processes againts human error.

Chapter 4.3 is about listing BPM items: process definitions, instances, tasks. Also we will analyze how filtration on these items will be done and what Polymer tools can help with that.

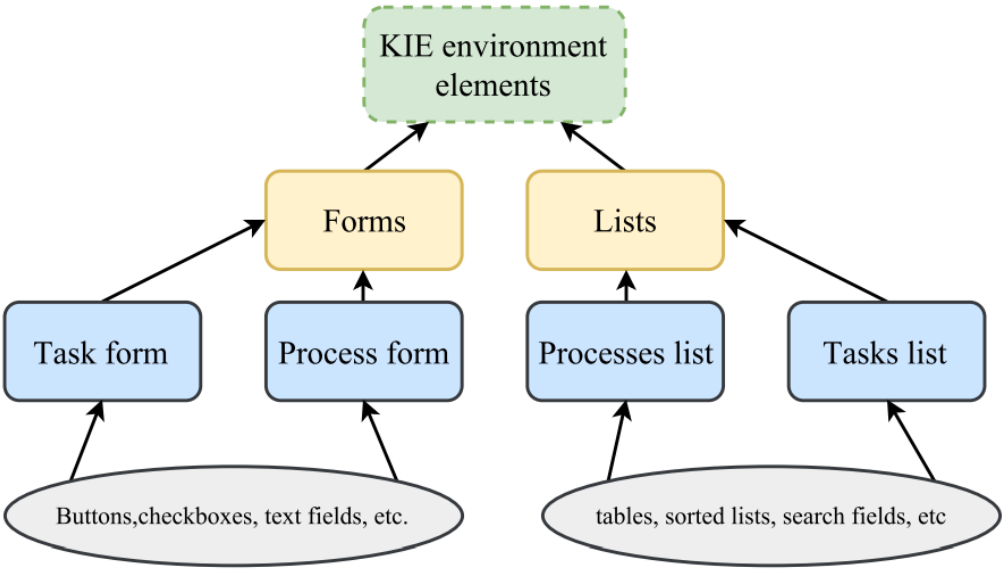


Figure 4.1: Proposed abstract architecture

4.1 User forms for processes

Both process definitions and process instances sometimes require input data. In case of definitions it means only specifying type of data. If we are talking about process instances it means actual information. Also definitions and instances have properties and or settings. Business Central console give a several ways how to monitor or change them. Our library should allow same ability to set up properties by proper UI elements. Some Polymer components and UI elements required for building this layer are described in Table Table 4.1.

Table 4.1: List of required UI elements (process form)

Element name	Usage purpose	Description
iron-ajax	Calling jBPM endpoints	Non-UI element. Contains all functions for making ajax call and response parsing
iron-form	Group of input elements which should be validated	Validate and submit any custom elements as well as any native HTML elements.
paper-dialog	Warn users that process can't be started or aborted.	Dialog window with option to choose animation of appearing and disappearing
paper-listbox	Listing of human tasks of current process. Listing of users responsible for current process task	Listbox with a different highlightings on selected and unselected items
paper-checkbox	Swithing on and off process attributes or properties (e.g. logging)	Same as `<input type="checkbox">`, but with ability of styling
paper-button	Starting process, saving configuration, etc.	Same as `<button>`, but with styling
paper-spinner	Represents the waiting for precess starting or aborting	Shows a progress of work as spinning ring

4.2 User forms for tasks

Tasks will take the most of attention in library implementation. And there is a reason why: task is a highly demanding on incoming data. It's a place where user should do something to complete task. It means that UI solutions for this type of BPMN elements should be well ergonomic. And at the same time Polymer components need to accept only validate data. Without input data control executing

process can be aborted in middle of its runtime. Some of main UI elements and Polymer components needed for these purposes are listed in Table Table 4.2.

Table 4.2: List of required UI elements (task form)

Element name	Usage purpose	Description
iron-textarea	User's reports or annotations for a task.	Text field which grows with amount of content
iron-form	Group of input elements which should be validated	Validate and submit any custom elements as well as any native HTML elements.
iron-input	Custom validation can be used for specific input	Provides a data-binding and custom validation
paper-dialog	User notification about bad input or wrong actions	Dialog window with option to choose animation of appearing and disappearing
paper-checkbox	User's input	Same as `<input type="checkbox">`, but with ability of styling
iron-localstorage	Uploading/downloading reports, certificates or other required for task documents	Element implements access to Web Storage API
paper-radio-button	Checking group of task properties	Styled radio button
paper-slider	User's input	Allows select a value from a range of values by moving the slider thumb. Intensity level can be changed for better UI experience

4.3 BPM's items listing

One of most important tools in BPM user's console is a list. It allows user to find particular element or sort them by defined property. This tool also can be very handy for monitoring some category of processes. Or user can list all tasks that belong to him.

This instrument seems simple on a paper, but there a lot of complex logic behind the scenes. And this is a place where Polymer components will be very useful. Some of UI and Polymer elements that will be used for library implementation are introduced in Table Table 4.3.

Table 4.3: List of required UI elements (BPM's elements listing)

Element name	Usage purpose	Description
iron-collapse	Wrapper box for detailed info about process or task. Can be opened directly in a list	Block of content that can be opened or collapsed
table	Global container for listed items	Native HTML table
iron-input	Search and filtration purposes	Provides a data-binding and custom validation
paper-icon-button	Can be used for "search button" or other button in a search toolbar	Button with an image in a center and special effect when button is pressed
paper-menu	General list for BPM items	Very similar to paper-listbox, but has more styling methods.
paper-toolbar	Searching or/and filtration toolbar	Horizontal bar containing items that can be used for label, navigation, search and actions.
paper-progress	Progress of searching or filtration of big amount of items	Stylized progress bar

Chapter 5

Implementation

This chapter contains a description and some technical details of the implementation, which was designed according to previously proposed architecture.

The main task of this thesis is development of such Polymer elements, which can be used separately from each other, or in combination. This allows developer to use web components in more various ways in own web solution.

For example, KIE elements consumer can use the UI components `<kie-tasks-list>` and `<kie-processes-list>` separately to display a list of tasks and processes. But also he can use them together on same page, forcing them to synchronize with each other (e.g. the process started - a new task appeared).

Chapter 5.1 will focus on the implementation features of the element for communicating with the KIE server. How elements for process and task forms are constructed is described in chapter 5.2. The technical details of two largest listing components are given in chapters 5.3 and 5.4.

5.1 Communication with KIE execution server

Various KIE web components require different data for their work. This means that each element should call an endpoint corresponding to its function or needs. To avoid overloading with code for composing HTTP requests, the UI component named `<kie-server-execute>` was developed.

The core of this component is Polymer element `<iron-ajax>` [18]. It affords an abstract level above the standard browser API function - `XMLHttpRequest`. In the tag attributes, developer can specify URL, HTTP method (`GET`, `PUT`, `POST` or `DELETE`), content of body request, HTTP headers, etc.

Implementation of `<kie-server-execute>` uses `<iron-ajax>` with these attributes:

```
<iron-ajax
  verbose
  id="ajax"
  url="[[url]]"
  method="[[method]]"
  body="[[body]]"
  content-type="[[contentType]]"
  params="[[params]]"
  with-credentials="true"
  on-response="_handleResponse"
  on-error="_handleError">
</iron-ajax>
```

Expressions like `[[url]]`, `[[method]]` or `[[body]]` are called “dom-binding”. This technology is provided and maintained by the Polymer framework [19]. Variables within these expressions are properties of host element (i.e. `<kie-server-execute>`). Full list of implemented element’s properties and methods can be found in Table Table 5.1.

All the properties specified in the tag attributes get their values in element’s method - `execute()`. The code of composing request headers is given below:

```
_makeHeaders() {
  var obj = {};
  obj.Accept = this.acceptType;
  obj.Authorization = "Basic " + btoa(this.username +
":"+this.password);
  obj["X-Requested-With"] = "XMLHttpRequest";
  this.$.ajax.headers = obj;
  if (this.acceptType.indexOf("json") > -1) {
    this.$.ajax.handleAs = "json";
  } else if (this.acceptType.indexOf("xml") > -1) {
    this.$.ajax.handleAs = "text";
  }
}
```

The request is sent by calling function `this.$.ajax.generateRequest()`, where `ajax` is id of `<iron-ajax>` element in the HTML markup.

As a result, the user of **`<kie-server-execute>`** component does not have to worry about writing code to compose the headers or the body of the request. He only needs to specify the endpoint, request method, user credentials and the server address, if it differs from the default one.

The code of using `<kie-server-execute>` is given below:

```
<kie-server-execute
  id="getProcess"
  op="queries/processes/definitions"
  username="[[user.password]]"
  password="[[user.userId]]"
  server-url="http://localhost:8080/kie-execution-
server/services/rest/server/"

  on-kieresponse="_setUpData">
</kie-server-execute>
```

Polymer provides its own way of attaching event handlers to elements. In code fragment above it is done by defining tag’s attribute `on-response`. The value of this attribute is the callback function that will be triggered when the "response" event occurs [20]. Also it automatically binds tag’s attributes to corresponding element properties. For this property and attribute should have same names. If property has two words in name, it should be defined in “camelCase” style and

corresponding attribute in “dash-case” style. In code fragment above value in `server-url` attribute will be stored to `serverUrl` property.

Table 5.1: List of `<kie-server-execute>` methods and properties

Name of property/method	Type	Description
`serverUrl`	String	Used for specifying URL of KIE server’s enpoints service. Has a default value set to `localhost:8080`
`url`	String	Stores full URL to exact enpoint. Value is computed of `serverUrl` and `op.`
`op`	String	Keeps URL of endpoint. Can be specified by component’s consumer
`method`	String	Keeps name of method of request. Can be specified by component’s consumer. By default has a value is “GET”
`body`	String	This property is used for storing body of request.
`contentType`	String	Specifies type of content of request. Value is consumed by `<iron-ajax>`
`params`	Object	Optional property. If there are some extra properties for HTTP request they can be specified in this property.
`username`	String	Need to be specified for making authentication
`password`	String	Together with `username` is used for making authorization request header
`acceptType`	String	Defines accepted types of response form KIE server
`error`	Object	Stores an object with details about error if it occurs.
`response`	Object	Property keeps an object of HTTP response from KIE server. This property used by `<iron-ajax>` and never by component’s consumer
`execute()`	Function	Method starts a headers build and when they are ready makes HTTP request.

5.2 Forms

Process definitions and human tasks can have their own forms. Form is a set of UI fields in which user can write the data required to start a business process or complete a task. The problem is that forms metadata, received from KIE server, is different for processes and tasks. That’s why three web components were implemented for working with forms:

- <kie-form>: designed to combine the form fields into one UI element, to process input user data, to submit the form and send data provided by user.
- <kie-form-start-process>: non-UI element. The main goal of this component is to gather information about the form's fields and input data for these fields. Then, this information is passed to the <kie-form> element to display the form on the page.
- <kie-form-complete-task>: non-UI element. Just like <kie-form>-start-process this element collects information about form fields. The difference lies in using another endpoints and extra processing of task-specific data. After all operations, information about tasks is passed to <kie-form>.

The <kie-form-start-process> and <kie-form-complete-task> components collect information with different structure and significance. But after processing, they pass unified data to <kie-form>.

5.2.1 Fetching form's fields

In the implementation of this thesis, two endpoints are used to get basic data about the form fields:

- for fetching fields for the process start form:

```
server/container/{container-id}/forms/processes/{process-id}
```

- for fetching fields for the complete task form:

```
server/container/{container-id}/forms/tasks/{task-id}
```

In both cases, the components use <kie-server-execute> element to call the endpoint. For <kie-form-start-process> data, returned by KIE server, is enough to build a full-fledged form. In case of <kie-form-complete-task>, among other stuff, you need to request information about the input data and binding type of field. The complete task forms can contain fields with input binding and/or output binding [21]. The input binding field is initialized with the values of process variables to which field is binded. The second type of field writes the received input to the binded process variables. Because of this reason, <kie-form-complete-task> uses two additional endpoints:

- containers/{container-id}/tasks/{task-id}/contents/input
- containers/{container-id}/processes/definitions/{process-id}/tasks/users/{task-name}/outputs

When all information is collected, the element creates an object with a suitable structure for the <kie-form>. Once this object falls into the component's property, the dom-binding system registers it and <kie-form> builds a full-fledged form.

A list of properties and methods for the `<kie-form-complete-task>` and `<kie-form-start-process>` components is provided in Table Table 5.2 and Table Table 5.3, respectively.

Table 5.2: List of `<kie-form-complete-task>` methods and properties

Name of property/method	Type	Description
`serverUrl`	String	Used for specifying URL of KIE server's enpoints service. Has a default value set to `localhost:8080`
`container`	String	Component's consumer can specify ID of container where required project with tasks is running.
`taskId`	String	This property is required. It is used for composing HTTP request to the right endpoint.
`user`	Object	Contains user credentials (i.e. username and password) required for authetication and authorization to KIE server.
`buttonName`	String	The name of „Submit" button can be specified in this property. Default value is „Complete Task"
`hidden`	Boolean	If this property is set to `true`, the whole form will not be displayed until its value changed to false. By default `hidden` property is `false.`
`noInit`	Boolean	Setting this property to `true` means that component starts to fetch all reqired data (i.e. fields, input/ouput bindings) as soon as it is registered by Polymer.
`fetchTaskForms()`	Function	Method starts a process of fetching and gathering all required data about task's form and fields.

Table 5.3: List of `<kie-form-start-process>` methods and properties

Name of property/method	Type	Description
`serverUrl`	String	Used for specifying URL of KIE server's enpoints service. Has a default value set to `localhost:8080`
`container`	String	Component's consumer can specify ID of container where required processes are running.
`processId`	String	This property is required. It is used for composing HTTP request to the right endpoint.
`user`	Object	Contains user credentials (i.e. username and password) required for authetication and authorization to KIE server.

Table 5.4 (cont.): List of <kie-form-start-process> methods and properties

Name of property/method	Type	Description
buttonName	String	The name of „Submit“ button can be specified in this property. Default value is „Start Process“
hidden	Boolean	If this property is set to true, the whole form will not be displayed until its value changed to false. By default hidden property is false.
noInit	Boolean	Setting this property to true means that component starts to fetch all reqired data (i.e. fields, names and type of variables) as soon as it is registered by Polymer.
fetchProcessForms()	Function	Method starts a process of fetching and gathering all required data about process definition‘s form and fields.
startProcess()	Function	If called starts the <kie-form>‘s method submitForm().

5.2.2 Submitting form

After the <kie-form> has received all the necessary information, it begins to process it and build the form. For construction several Polymer elements with Material design are used:

- <paper-input>. One-line input element. It can be used for various kinds of input: numbers, text, date, etc. Has the ability to auto-validate. [22]
- <paper-textareat>. Multi-line input element. Has all the same properties as <paper-input>.
- <paper-checkbox>
- <paper-button>. Used for submitting the form. By changing the attributes, it can switch its design depending on the purpose. [23]
- <paper-toast>. “Pop-up” element for notifications.

Form is assembled in <template is=”dom-repeat”> and <template is=”dom-if”> elements. The first tag repeats the pattern defined inside it. The second tag stamps its content into the DOM only when the condition becomes truthy. In <kie-form> it looks like this:

```
<div id="form">
  <template is="dom-repeat" items="{{fields}}" as="field">

    <template is="dom-if" if="{{_isTextBox(field)}}">
```

```
      <paper-input auto-validate value="{{field.innerData}}"
type="{{_getType(field)}}" label="{{field.label}}"
placeholder="{{field.placeHolder}}" required="{{field.required}}"
readonly="{{_readOnlyCheck(field)}}"></paper-input>
    </template>

    <template is="dom-if" if="{{_isTextArea(field)}}">
      <paper-textarea auto-validate value="{{field.innerData}}"
label="{{field.label}}" placeholder="{{field.placeHolder}}"
required="{{field.required}}" readonly="{{_readOnlyCheck(field)}}"
rows="{{field.rows}}"></paper-input>
    </template>

    <template is="dom-if" if="{{_isCheckBox(field)}}">
      <paper-checkbox style="margin-top: 5px"
checked="{{field.innerData}}" required="{{field.required}}"
readonly="{{_readOnlyCheck(field)}}">{{field.label}}</paper-checkbox>
    </template>

  </template>
</div>
```

On the Figure Figure 5.1 you can find example of rendered form.

The form submitting for processes and for tasks has different consequences. For processes, this is start of a new process instance. For human tasks submitting forms usually cause completing (i.e. changing status) of this task. But component consumer can clarify what he wants to do by submitting form. He can do it by passing full URL to `<kie-form>` `url` property. List of element properties and methods you can find in Table Table 5.5.

Table 5.5: List of `<kie-form>` properties and methods

Name of property/method	Type	Description
`url`	String	Recievied its value from component's consumer. Keeps URL adress to endpoint for submitting form.
`formData`	Object	In this property form data shoud be defined. `<kie-form>` uses it for building form and composing body of HTTP request.
`buttonName`	String	Name of „Submit“ button. By default property is set to „Submit“
`hideButton`	Boolean	If set to `true` hides button for submitting form.
`method`	String	Property value specifies which HTTP request method will be used. Default value is `POST.`
`notificationMessage`	String	Private property which can be set by element's methods. Used as a message container for `<paper-dialog>`

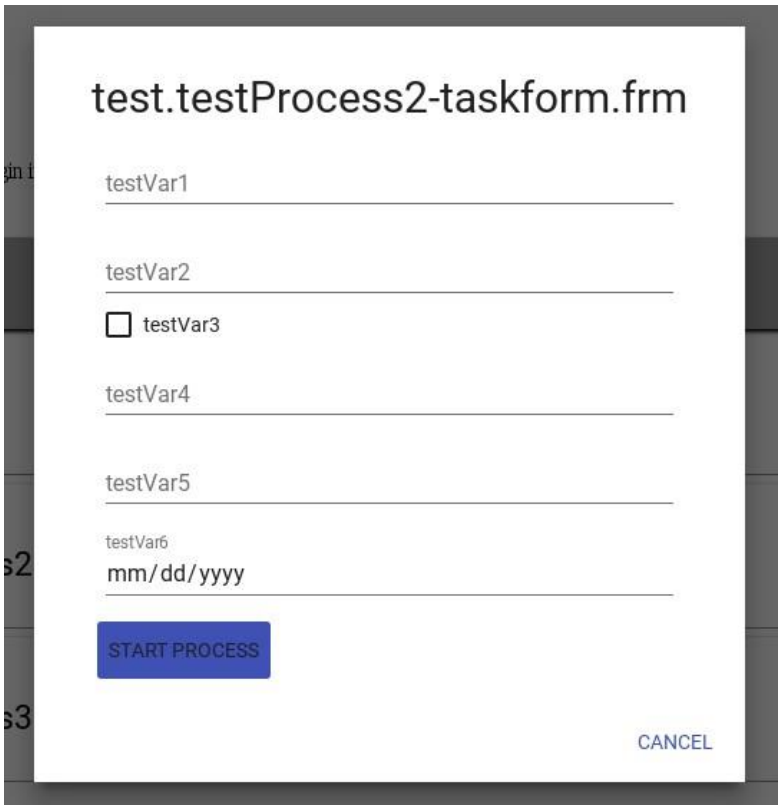


Figure 5.1: Example of rendered form

5.3 Processes listing

Processes listing was implemented in `<kie-processes-list>` component. This is a complex custom element, which includes native HTML elements, Polymer Material design elements, Polymer behavior elements, and some of the KIE elements.

As input parameters are `serverUrl`, user, and `listOf` properties. This component is made in such way that a developer can choose what exactly he wants to listing. For this goal `listOf` property was designed. It can have the string value “definitions” or “instances”. Also this component has an ability to filter the content, depending on the user's search query. Full list of element’s properties and methods is represented in Table **Ошибка! Источник ссылки не найден.**.

`<kie-processes-list>` uses five different endpoints to fetch data:

- `queries/processes/definitions`
  KIE server responds with array of objects, which contains basic information about process definitions. Used for getting information about all processes and initializing list.
- `queries/processes/instances`
  Same as endpoint above, but it is used for process instances.
- `containers/{container-id}/processes/definitions/{process-id}`

KIE server responds with details about specified by id process definition. This endpoint is need for getting information about process variables.

- `queries/processes/instances/{process-instance-id}/variables/instances`
  Like as endpoint for process definition component receives list of process instance variables and their values.
- `containers/{container-id}/processes/instances/{process-instance-id}`
  This endpoint is called with HTTP method 'DELETE'. It allows to abort process instance. No specific respond from KIE server.

All of these endpoints are called by `<kie-server-execute>` element.

The list itself is implemented with help of Polymers `<template is="dom-repeat">` and `<template is="dom-if">`. Depends on value of `listOf` property different DOM structures are generated. It is done in such way because of distinction of process data, which should be shown to user. For example of process definitions and process instances lists see Figure Figure 5.2:.

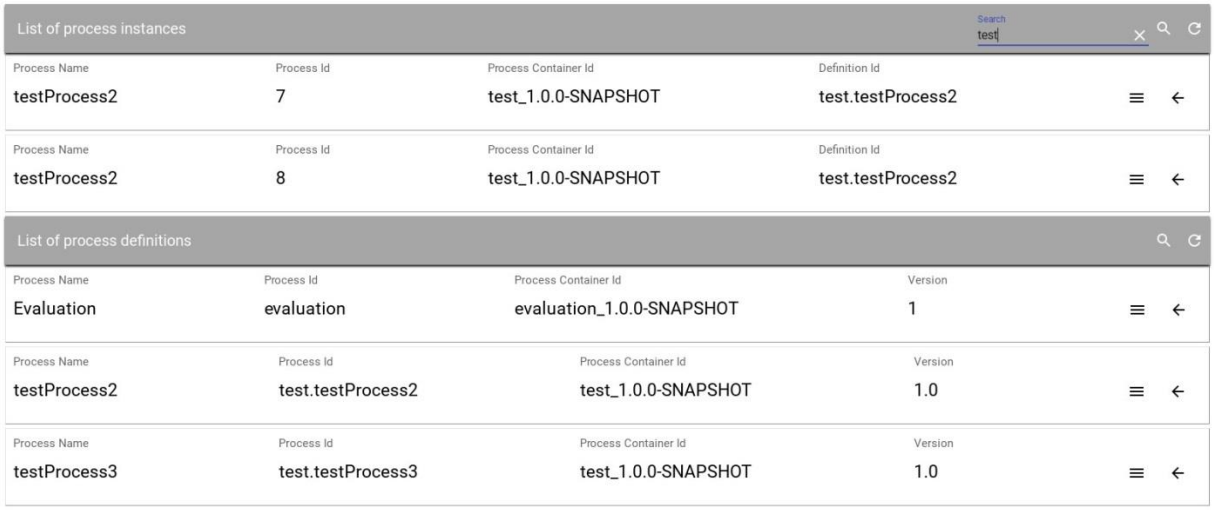


Figure 5.2: Example of lists

Custom element `<kie-processes-list>` also uses `<kie-form-start-process>`. Its public method `startProcess()` is called when user clicks on list item. When all form data is ready `<paper-dialog>` pops up and asks for user's input. It's a good example of combining web components and making them work as on solid mechanism.

Table 5.6: List of <kie-processes-list> methods and properties

Name pf property/method	Type	Description
listOf	String	Component's consumer can specify which type of list he wants by storing value "definitions" or "instances" into this property
searchStr	String	Used for implementations of dynamic user search. Keeps current user input.
getProcessUrl	String	It's a string with URL of endpoint which is used for getting general data about processes
getProcessVarsUrl	String	Same as getProcessUrl keeps URL address to endpoint for getting process variables
user	Object	Contains user credentials (i.e. username and password) required for authetication and authorization to KIE server.
processData	Array	Stores array of processes, which is sent as respond by KIE server.
procVars	Object	Used by element for storing all variables of specified process.
refreshList()	Function	This method can be called by component's consumer tp force list of tasks update itself. refreshList() calls another private method – this._getProcessesData() and clear the list for correct re-rendering
serverUrl	String	Specified by component's consumer KIE server url.

5.4 Human tasks listing

Tasks list can be created by component <kie-task-list>. Its logic exactly the same as <kie-processes-list>, but has simpler HTML markup and layout. Also only one endpoint is used here:

```
queries/tasks/instances
```

By calling this service KIE server responds with array of tasks and details about them.

Element's API pretty similar to <kie-processes-list> and differs only in names of properties and methods. List of properties and method are represented in Table Table 5.7.

Table 5.7: List of <kie-tasks-list> methods and properties

Name of property/method	Type	Description
user	Object	Keeps user credentials (i.e. username and password) required for authetication and authorization to KIE server.
taskData	Array	Stores array of tasks, which is sent as respond by KIE server.
searchStr	String	Used for implementations of dynamic user search. Keeps current user input.
refreshList()	Function	This method can be called by component's consumer tp force list of tasks update itself.
serverUrl	String	Specified by component's consumer KIE server url.

5.5 Demo application for testing components

To test all the implemented components, a small demo application was developed. Demo application consists of six parts in each of which you can test all the KIE elements. The last part demonstrates how the created elements can interact and update the data of each other. Synchronization between lists occurs as follows: when the user submits the form, <kie-form> dispatches custom event named "submitted". It is bubbling from the <kie-form> up through all parents DOM nodes. To make a "submitted" event pass through shadow DOM boundaries, the composed flag is set to true [26].

```
this.dispatchEvent(new CustomEvent('submitted', {
  detail: {"formName":this.formData.name},
  bubbles: true,
  composed:true
}));
```

Component's consumer can attach event listener to <kie-form>`s parent, for example to <kie-processes-list>. Inside listener function developer should call method of <kie-tasks-list> element - refreshList(). It forces list of tasks to update itself. For example in demo application it is done like this:

```
    var kieServer = document.getElementById("getProcess");
```

```
var proc = document.getElementById('pInstances');
var def = document.getElementById('pDefinitions');
var tasks = document.getElementById('tTask');
def.addEventListener('submitted',function() {
proc.refreshList();
tasks.refreshList();
});
tasks.addEventListener('submitted',function() {
    proc.refreshList();
});
proc.addEventListener('aborted',function() {
    setTimeout(function() {tasks.refreshList();},500);
});
```

Where object `def` keeps `<kie-processes-list>` for definitions, `proc` – for instances, `tasks` – stores `<kie-tasks-list>`. It is necessary to notice when user makes process instance abort, `<kie-processes-list>` fires `"aborted"` event.

All parts of demo are steps for testing KIE elements. Each step asks user to do "monkey testing" [24]. This method was chosen because of significance role of UI. User must make sure that all forms are shown correctly and correspond to forms existing in jBPM business central web app.

Performed tests in different browser showed that KIE elements have best performance in Google Chrome and Opera browsers. It can be explained by their native support for Web Components technology [25]. In other browsers KIE elements work and displayed correctly only in case if web components polyfill was used. Detailed information about browser support you can find in Table .Table 5.8

Table 5.8: Support of Web Components by different browsers

Web Standart	**Chrome**	**Opera**	**IE 11+/Edg**	**Firefox**	**Safari 9+**
Template	Native	Native	Partial	Native	Native
HTML Imports	Native	Native	Pollyfill required	Pollyfill required	Partial
Custom Elements	Native	Native	Pollyfill required	Pollyfill required	Partial
Shadow DOM	Native	Native	Pollyfill required	Pollyfill required	Partial

Chapter 6

Conclusion

In scope of this master's thesis, the research of new technology Web Components was done. Despite the relatively young age of all parts of this technology, it is gaining popularity. Such a conclusion can be made from Table Table 5.8. Among all the libraries aimed on developing a web components, Polymer was chosen for thesis implementation. This decision was made for the reason that this framework is actively supported by developers and its community. It has "close to native" API and it has powerful helper tools. To create the KIE elements, the latest version of Polymer 2.0.0 was used. Its advantage is the using features of the new JavaScript standards: EcmaScript 5 and 6. On the other hand, the use of new technologies have some drawbacks. Support for modern standards is not yet quite popular, and not all the most well-known browsers implement Web Components standards. Therefore, the KIE elements user will be forced to use the polyfill frameworks for correct work in the browser without full support. This can slow down the loading of pages, the runtime of the web component or lead to code conflicts.

However, the usage of these standards in the implementation of this thesis is justified for several reasons. One of them is a low obsolescence of such components. Developer, who will use these components, doesn't need to keep his web application in applicable for KIE elements state. Also, such web components are easier to maintain. For upgrading to new versions of used technologies (i.e Polymer), a minimum of work should be performed.

As a result of work on this thesis, I developed a set of Web components with help of Polymer framework. All of them was designed in way to have ability work with each other or separately. The created KIE element API and UI, created in scope of this thesis, is enough for performing basic operations over business processes.

Each of elements was published to GitHub. This allows easily to install elements with help of front-end package manager such as Bower [26]. The last one also was used in implementation to keep all dependencies for components in one place.

6.1 Future steps

The general mission of future development is adding new functionality from jBMP business central web app. The problem here that endpoints, provided in documentation, don't cover all functionality requirement. There is no "easy" way to get metadata about all process instances. Used in this master's thesis endpoint gives information only about running instances and nothing about aborted or fulfilled

process instances. To all other there's no way to tell KIE server to save form data without submitting it. These problems should be investigated and API of each KIE element should be improved.

Also one of the future steps is an integration of Red Hat's Patternfly project [27]. It's a set of UI solutions that are made in company's style. The problem which can be met here is absence of standard way to load script into Polymer element. It can be done by simple including script, but this way doesn't guarantee that no code conflicts occur while runtime.

After that created KIE elements will be published to web components database [28] and Bower packages.

Bibliography

[1] Business Process Model and Notation. *Wikipedia* [online]. Wikipedia, 2016. Available at the URL: https://en.wikipedia.org/wiki/Business_Process_Model_and_Notation

[2] Quick tour of Polymer. *Www.polymer-project.org* [online]. Polymer. Available at the URL: https://www.polymer-project.org/1.0/docs/devguide/quick-tour

[3] HTML's New Template Tag. *Html5rocks.com* [online]. Eric Bidelman, 2013. Available at the URL: https://www.html5rocks.com/en/tutorials/webcomponents/template/

[4] Shadow DOM v1: Self-Contained Web Components. *Developers.google.com* [online]. Eric Bidelman, 2016. Available at the URL: https://developers.google.com/web/fundamentals/getting-started/primers/shadowdom

[5] Custom Elements v1: Reusable Web Components. *Developers.google.com* [online]. Eric Bidelman, 2016. Available at the URL: https://developers.google.com/web/fundamentals/getting-started/primers/customelements

[6] ALLWEYER, Thomas. *BPMN 2.0: introduction to the standard for business process modeling*. 11, 1. Aufl. Norderstedt: Books on Demand, 2010. ISBN 38-391-4985-1.

[7] Web Components: A discussion surrounding Web Components and their role in the future of web development. *Http://kaytcat.github.io/* [online]. Kaitlin Rathwell, 2015. Available at the URL: http://kaytcat.github.io/web-components/

[8] Web Components - building blocks of the future web. *Infinum.co* [online]. infinum.co, 2014. Available at the URL: https://infinum.co/the-capsized-eight/web-components-building-blocks-of-the-future-web

[9] Shadow DOM: W3C Working Draft. *Www.w3.org* [online]. W3C, 2016. Available at the URL: https://www.w3.org/TR/shadow-dom/

[10] *Business Process Model and Notation (BPMN)*. 2. 2011.

[11] SALATINO, Mauricio, Esteban ALIVERTI a Mariano NICOLAS DE MAIO. *JBPM 6 Developer Guide*. 3. Birmingham B3 2PB, UK: Packt Publishing, 2014. ISBN 978-1-78328-661-4.

[12] OVERSON, Jarrod a Jason STRIMPEL. *Developing web components: [UI from Jquery to Polymer]*. ISBN 14-919-4902-3.

[13] HTML Components: Componentizing Web Applications. W3C [online]. Available at the URL: https://www.w3.org/TR/NOTE-HTMLComponents

[14] XBL 2.0: W3C Working Draft 2 May 2012. W3C [online]. Available at the URL: https://dev.w3.org/2006/xbl2/

[15] Web Components and Model Driven Views by Alex Russell. Fronteers [online]. Available at the URL: https://fronteers.nl/congres/2011/sessions/web-components-and-model-driven-views-alex-russell

[16] Web Components Current Status. W3C [online]. Available at the URL: https://www.w3.org/standards/techs/components#w3c_all

[17] HTML Imports: #include for the web. HTML5 ROCKS [online]. Available at the URL: https://www.html5rocks.com/en/tutorials/webcomponents/imports/

[18] Iron-ajax: Easily make ajax requests. WebComponents [online]. Available at the URL: https://www.webcomponents.org/element/PolymerElements/iron-ajax/elements/iron-ajax

[19] Data binding. Polymer Project [online]. Available at the URL: https://www.polymer-project.org/2.0/docs/devguide/data-binding

[20] Handle and fire events: Polymer Project [online]. Available at the URL: https://www.polymer-project.org/2.0/docs/devguide/events

[21] Field types. JBPM Documentation [online]. Available at the URL: http://docs.jboss.org/jbpm/release/7.0.0.CR3/jbpm-docs/html_single/#_sect_formmodeler_fieldtypes

[22] Paper-input: A Material Design text field. WebComponents [online]. Available at the URL: https://www.webcomponents.org/element/PolymerElements/paper-input/elements/paper-input

[23] Paper-button: A button à la Material Design. WebComponents [online]. Available at the URL: https://www.webcomponents.org/element/PolymerElements/paper-button/elements/paper-button

[24] Monkey testing. Wikipedia [online]. 2017. Available at the URL: https://en.wikipedia.org/wiki/Monkey_testing

[25] Browser support overview: Platform features. Polymer Project [online]. Available at the URL: https://www.polymer-project.org/2.0/docs/browsers

[26] About Bower. Bower [online]. Available at the URL: https://bower.io/docs/about/

[27] Pattern Library: Get Started. Patternfly [online]. Available at the URL: https://www.patternfly.org/get-started/

[28] About webcomponents.org. WebComponents [online]. Available at the URL: https://www.webcomponents.org/about

Appendices

List of Appendices

Appendix A

CD Content

/demo Demo application for testing

/DP_pdf PDF file with this master's thesis

/DP_word Microsoft Word file with text of this master's thesis

/kie-elements Source codes of implemented KIE web components

/README Manual how to setup and run environment for demo application