



Software Requirements Specification

for project

Semantic Pipelines Editor

by

Artem Kelpé, Yan Doroshenko

Czech Technical University in Prague

November 27, 2016

Contents

1	Introduction	3
1.1	Purpose	3
1.2	Intended Audience	3
1.3	References	3
2	Overall Description	3
2.1	Product Scope	3
2.2	Product Functions	3
2.3	User Classes and Characteristics	3
2.4	Operating Environment	4
2.5	Design and Implementation Constraints	4
2.6	Third Party Dependencies	5
3	External Interface Requirements	5
3.1	User Interfaces	5
3.2	Software Interfaces	6
4	System Features	6
4.1	See pipelines	6
4.2	Generate form (execute pipeline)	6
4.3	Create pipelines	6
4.4	Save/load pipelines	6
4.5	Alter pipelines	6
5	Other Requirements	7
	Appendix A: Glossary	7
	Appendix B: Illustrations	8

1 Introduction

1.1 Purpose

This document describes the Semantic Pipelines Editor software, a specialized graph editor for the Semantic Pipelines framework, as of version 1.0.

1.2 Intended Audience

This document is intended for developers, interested in Semantic Pipelines Editor, Semantic Pipelines infrastructure and its integration structure.

1.3 References

JSON-LD Primer

RDF Primer

P. Křemen, Z. Kouba, Ontology-Driven Information System Design

ESWC 2016, JOPA: Efficient Ontology-based Information System Design

2 Overall Description

2.1 Product Scope

Semantic Pipelines Editor is an application, whose purpose is to be integrated with Semantic Pipelines framework and to provide a graphical editor in form of an oriented graph for it with a possibility to execute pipelines from the editor itself. It is also capable of module saving and loading, which is provided by persisting data into the ontology storage. The main goal is to provide a free and open source software edition tool for the above-stated framework and to enable developers to interact and manage pipelines in a self-explanatory and convenient way without having to deal with proprietary software or restrictive licenses.

2.2 Product Functions

Product functions can be divided into two groups:

- Graph CRUD operations
- Pipeline execution

Functions are described in more detail in chapter System Features.

2.3 User Classes and Characteristics

There are two user roles defines:

- **Unauthorized**
Any user that did not log in with has permissions to create new pipelines, edit and execute them, but has no read/write access to the ontology storage and therefore can not save or load pipelines.
- **Authorized**
Logged in users can see, modify and create pipelines as well as save and load them.

Authorization capabilities are enabled by Spring Security on the backend. From the UI standpoint the difference is the following:

- There are no Save/Load features shown to an unauthorized user
- The link for unauthorization is shown to an authorized user

2.4 Operating Environment

Operating environment consists of several parts:

- **Application Server**
The software is meant to be run inside of a container like an application server. Originally Semantic Pipelines Editor was designed to run inside of Tomcat 8 application server, but should work in any other.
- **Database Server**
The application is designed to work with RDF4J ontology storage.
- **Semantic Forms**
Functionality is partially dependent on the Semantic Forms to be available as a web service.

2.5 Design and Implementation Constraints

There are several constraints that rigidly specify the product's relation to the infrastructure and possible ways of application.

- **Semantic Forms Integration**
Semantic Pipelines Editor is relying on Semantic Forms for the form generation so the WS communication is done in a very specific way which requires bigger changes for the form generation strategy to be replaced.
- **Ontology Storage**
Data is stored in the ontology storage (RDF4J), which makes for the specific way of entity objects design.
Another constraint regarding ontology is a very limited set of frameworks and libraries available, which also influences the implementation.

- **Java/Scala Interoperability**
All the business logic of the application is implemented in Scala programming language. This is done in order to minimize the unnecessary (from the business standpoint) implementation details for the logic layer to be clearer and more refined for the developer. However, Scala's interoperability with Java is limited in some specific areas.
- **JavaScript-based client Semantic Pipelines Editor** has a thin client and most of user interaction is made with JavaScript. Therefore the application is meant to be accessed with a web browser that is can run JavaScript code.

2.6 Third Party Dependencies

- **JDK Oracle JDK or OpenJDK** is required for running the backend
- **Scala SDK** Scala SDK is required for running the backend
- **Spring** Several components of the Spring Framework are used for IoC
- **JOPA Persistence layer** is done with JOPA
- **Apache Maven** Apache Maven build tool is used for building the project and sources generation
- **JUnit Testing** is partially made with JUnit
- **ServletAPI** ServletAPI is required by the web application
- **Logback** Logback is used for logging
- **JSON-Core** JSON-Core is required for JSON operations in Java and Scala
- **Jaxb-JSONLD-Jackson** Jaxb-JSON-LD-Jackson provides JSON-LD support
- **Mockito** REST WS testing is done with Mockito
- **Sigma.js** Sigma.js is providing frontend graph representation
- **ReactJS** ReactJS is required for UI implementation

3 External Interface Requirements

3.1 User Interfaces

Interaction with the user is provided by a web interface, built around the JavaScript library for graph representation Sigma.js. The main window consists of two parts: left panel and a working surface for editing graphs (Figure

1). Left panel contains elements for authorization, saving and loading graph and list of available types of nodes that can be added to the graph. At the working surface user can see the current graph (loaded or created), edit it (move nodes, draw edges etc.). Node double click shows the form generated from this node and its dependencies in the popup (Figure 2).

3.2 Software Interfaces

Application consists of frontend and backend parts with backend accessing the ontology storage as well as a Semantic Pipelines web service. Communication between backend and frontend is implemented with JSON-LD format messages sent through REST API. All communication is done through unsecured HTTP protocol. Authentication encryption is provided by integrated Spring Security tools.

4 System Features

As mentioned before, there are 5 main functions of Semantic Pipelines Editor software.

4.1 See pipelines

Pipelines are represented in form of an oriented graph. Nodes represent modules and edges are dependencies between them, which makes for intuitive and functional user interface.

4.2 Generate form (execute pipeline)

Semantic Pipelines Editor enables semantic forms to be generated from the pipeline. Module dependencies are executed recursively.

4.3 Create pipelines

There exists a possibility for creating new pipelines from scratch by adding individual modules and dependencies between them.

4.4 Save/load pipelines

The software allows saving pipelines into the ontology storage and loading them.

4.5 Alter pipelines

Pipelines can be modified by adding or deleting modules, changing their properties and dependencies between them.

5 Other Requirements

- The application is licensed under the GNU GPLv3 license

Appendix A: Glossary

API - Application Programming Interface

JDK - Java Development Kit

JOPA - Java OWL Persistence API

JSON - JavaScript Object Notation

JSON-LD - JavaScript Object Notation for Linked Data

OWL - Web Ontology Language

RDF - Resource Description Framework

RDF4J - RDF for Java programming language

REST - Representational State Transfer

SDK - Software Development Kit

UI - User Interface

WS - Web service

Appendix B: Illustrations

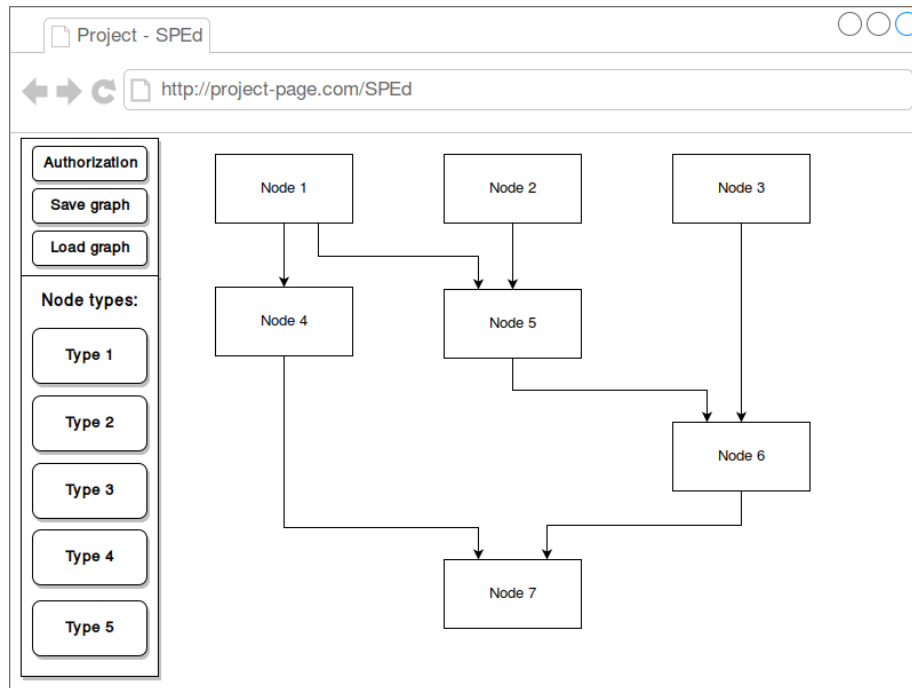


Figure 1: Main window

The image shows a web browser window titled "Project - SPed" with the address bar displaying "http://project-page.com/SPed". The main interface has a sidebar on the left with buttons for "Authorization", "Save graph", and "Load graph". Below these is a section titled "Node types:" containing five buttons labeled "Type 1", "Type 2", "Type 3", "Type 4", and "Type 5". The main content area features three tabs: "Form 1", "Form 2", and "Form 3". A modal form is currently open, containing the following elements: two text input fields labeled "Textfield 1" and "Textfield 2"; a dropdown menu labeled "Option 1" with a downward arrow; two radio button options labeled "Setting 1" and "Setting 2"; and a "Submit" button. In the background, a diagram is partially visible, showing a box labeled "Node 7" with arrows pointing to it from above.

Figure 2: Generated form