Semantic Web Lab

# Integration of I 4.0 standards

Mentor: Irlan Grangel

[This document describes the technical system specification for implementing Industry 4.0 Web Application Software]

Students:
Alina Arunova, Maxim Maltsev, Pylyp Matiash, Sattar Rahimbeyli

Bonn University 2016

# Table of Contents

# 1. Generalities

## 1.1. Overview

The aim of the project was to create a web-based tool that utilized RDF vocabularies for integrating instances of AutomationML and OPCUA standards into a single one. The particular implementation incudes validation of each document, converting both into RDF and consequant matching of two documents and with unified standard file generation.

## 1.2. Reference

The following are the reference document names:

- Industry 4.0 Tool Requirement Presentation
- Industry 4.0 Tool Requirement Document
- Industry 4.0 Tool Architectural presentation
- Industry 4.0 Tool Use Case Presentation
- Industry 4.0 Tool Documentation Presentation

These documents are available on GitHub. Below is the link:

https://github.com/IntegrationI40StandardsSemLab/Integration-I4.0/tree/master/Presentations

https://github.com/IntegrationI40StandardsSemLab/Integration-I4.0/tree/master/Docs

## 1.3. Definitions, Acronyms, Abbreviations

- RDF: Resource Description framework is a family of World Wide Web Consortium (W3C) specifications used as a general method for conceptual description or modeling of information that is implemented in web resources, using a variety of syntax notations and data

serialization formats.

- I 4.0 or Industry 4.0: is a combination of production methods with state-of-the-art information and communication technology. In the world of Industry 4.0, people, machines, equipment, logistics systems and products communicate and cooperate with each other directly.
- AML (Automation Mark-up Language): is a neutral data format based on XML for the storage and exchange of plant engineering information, which is provided as open standard. Goal of AutomationML is to interconnect the heterogeneous tool landscape of modern engineering tools in their different disciplines, e.g. mechanical plant engineering, electrical design, HMI development, PLC, robot control.
- OPCUA: is an industrial M2M communication protocol for interoperability developed by the OPC Foundation. It is the successor to Open Platform Communications (OPC). Although developed by the same organization, OPC UA differs significantly from its predecessor. The Foundation's goal for this project was to provide a path forward from the original OPC communications model (namely, the Microsoft Windows only process exchange COM/DCOM) to a cross-platform service-oriented architecture (SOA) for process control, while enhancing security and providing an information model. After more than three years of specification work and another year for a prototype implementation, the first version of the Unified Architecture was released in 2006.

## 2. About Industry 4.0 Tool

The goal of Industry 4.0 is to consolidate the standards. The importance of the project is resulted from the fact that globalization forces large companies to go international, and followed by the ability to handle with different industrial standards. In order to avoid wasting time onto manual object notation transcription, Industrial 4.0 tool was made up.

The aim of the research group is to create a web-based tool that uses RDF(S) vocabularies for integrating instances of such standards as AutomationML and OPCUA into one piece.

Industry 4.0:

- validate each document against xsd schema;
- match each entity of the document to a respectful entity of another document if possible according to ontology;
- consolidate the knowledge into the new unified standard.

# 3. Project Description

The project is aim to handle and match different industrial standards.

## 3.1. Use Case Diagram:

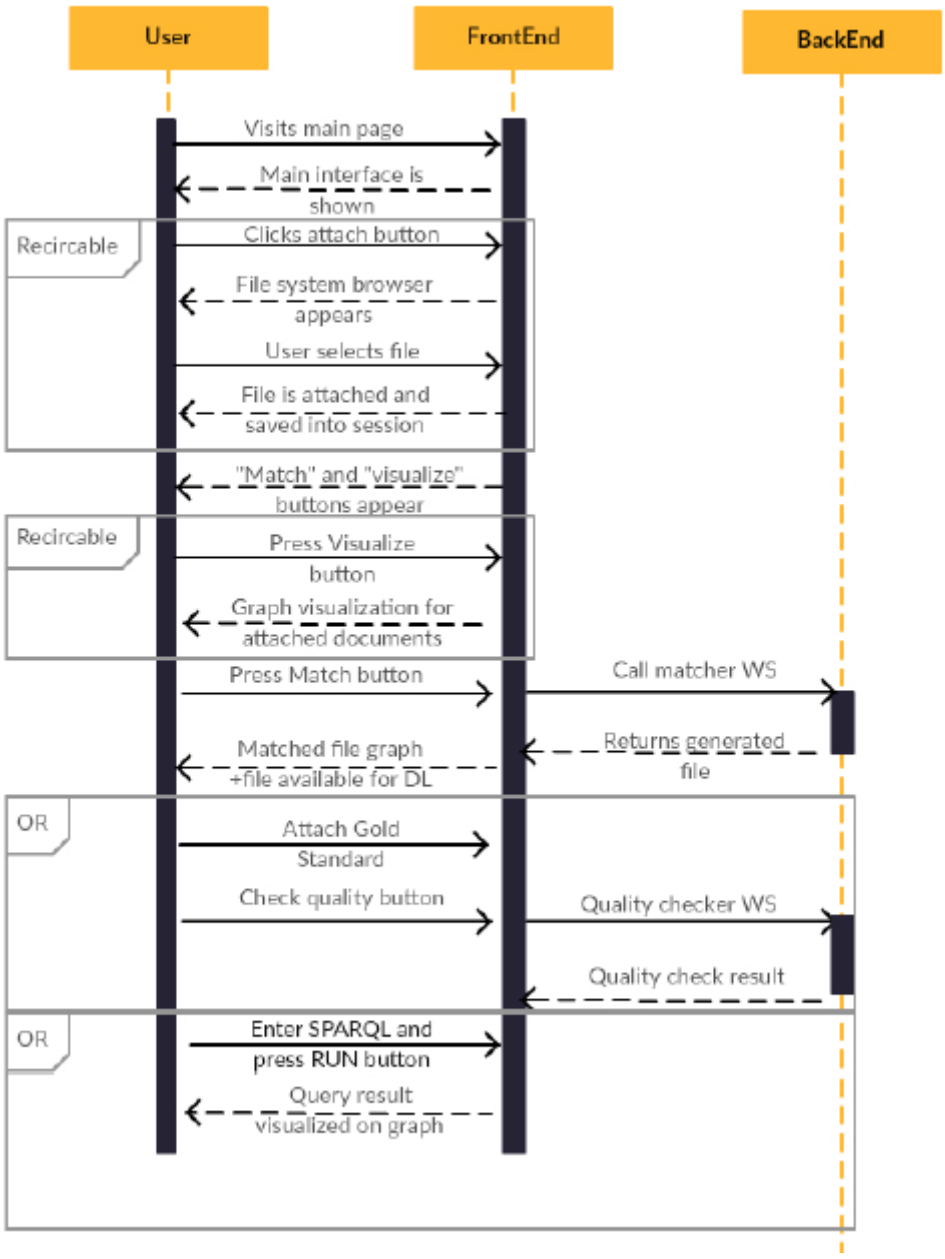


Image 1. User – application interaction diagram

**User:** An element representing the role of a person, object or device that interacts with the system.

**User – Application Interaction Diagram:** A behaviour diagram that visually describes the functional requirements of a proposed system and shows the relationships between Actors and Use Cases. Actors, Use Cases and Associations (connectors) mainly form a User – Application Interaction diagram.

## 3.2. Scenario:

### STEP 0: Selecting a dataset

User selects two files of .aml/.opcua type, one by one (Image 2).

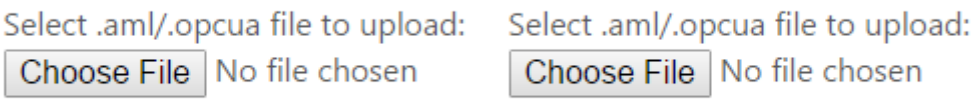


Image 2. Upload form.

### STEP 1: Interacting with the original dataset

User clicks on the visualization button and interacts with the graph (Image 3, Image 4).


Image 3. Form for visualizing and downloading the file.

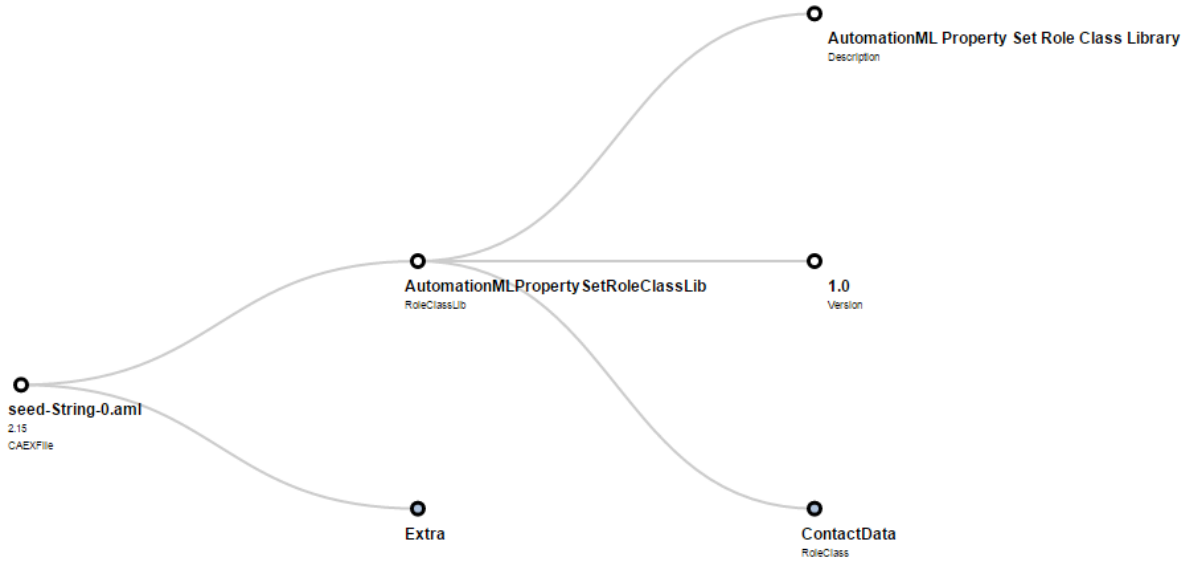


Image 4. Visualized data.

User chooses the way of semantic fuzzy matching of the files and submits files to the server (Image 5).

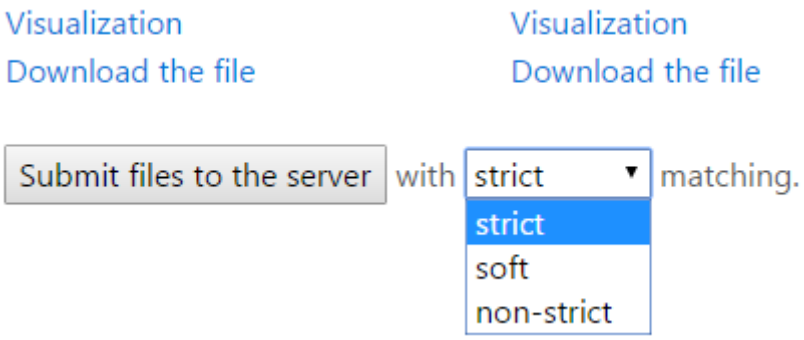


Image 5. Matching selection and submit.

## STEP 2: Interacting with the produced data

User downloads the integrated file choosing one out of 4 possible file types (Image 6).

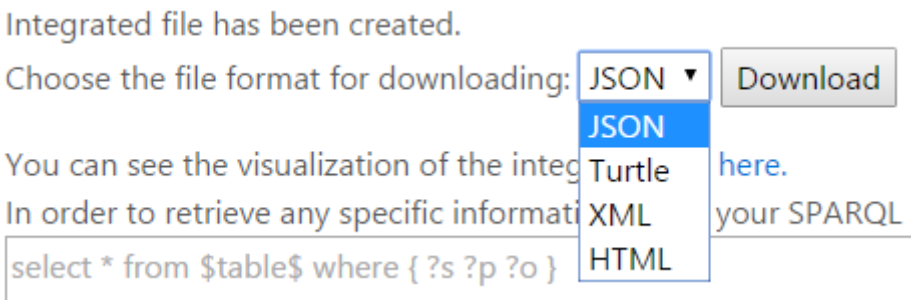


Image 6. Integrated file downloading.

User interacts with the visualization, clicking on the link (Image 4, Image 7) and retrieves some data from the produced integrated file using SPARQL (Image 7).

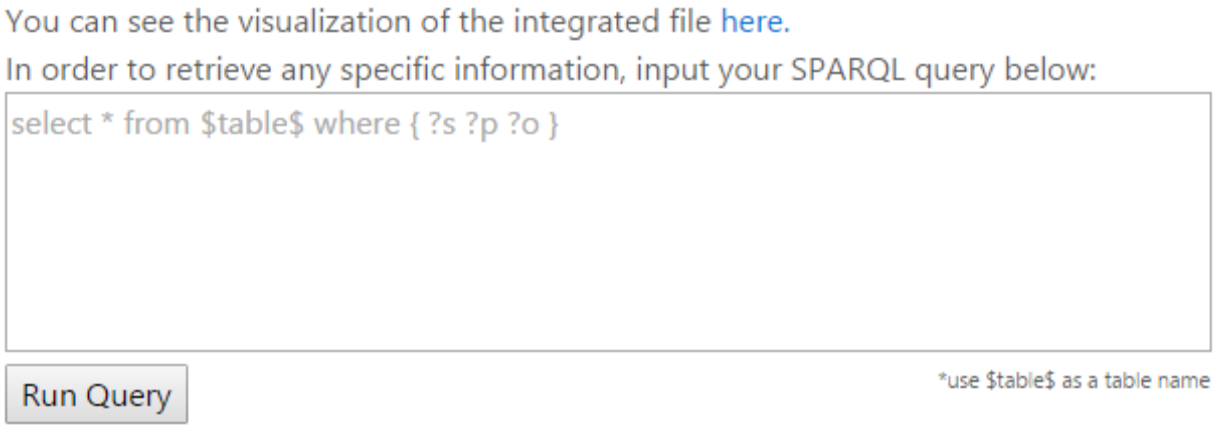


Image 7. Visualization and data retrieving.

User runs the query and visualize the result or download it in the JSON format (Image 4, Image 8).

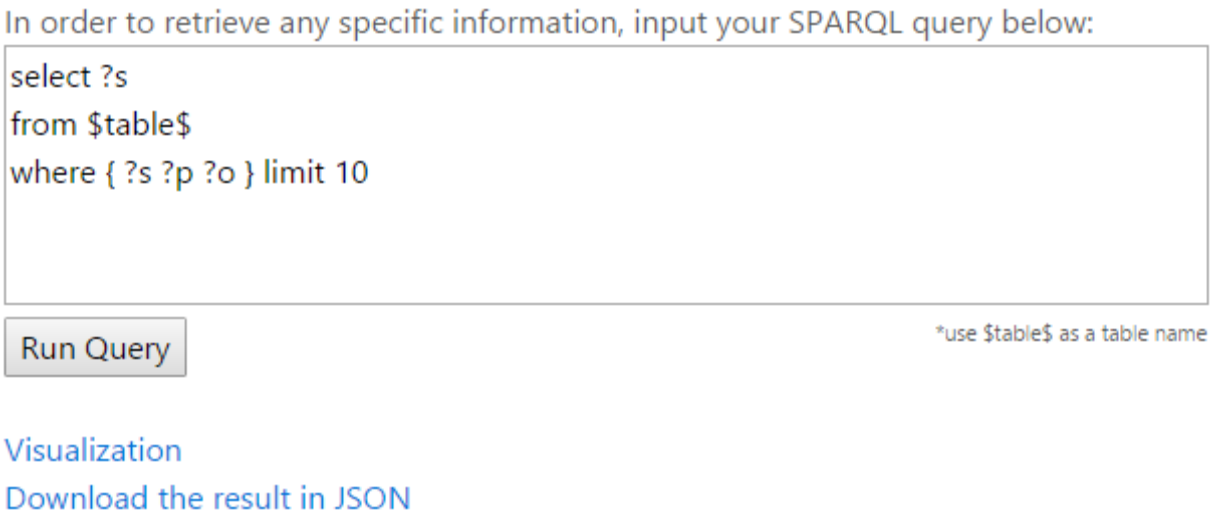


Image 8. Results of retrieving.

# 4. Requirements

## 4.1. Functional Requirements:

| REQ_ID | REQ_NAME | DESCRIPTION | PRIORITY |
|---|---|---|---|
| 1 | **Provide automatic document validation** | Checking input files for structural and syntax errors by means of xsd validation | 1 |
| 2 | **Visualizing input (tree mode)** | Visualizing input document in a tree view | 1 |
| 3 | **Direct topology mapping and producing integration file** | Map topologies (AutomationML and OPC UA) and provide an integration file with common instances | 1 |
| 4 | **Demo testing of the output against Gold Standard** | Manual development of a Gold Input and Output files and demo page for comparing teacher value with achieved one. Precision ratio (matching quality). | 2 |
| 5 | **SPARQL query for the output file** | Query the resulting file and highlighting it on the tree | 2 |

## 4.2. Non - Functional requirements:

| REQ_ID | REQ_NAME | DESCRIPTION | PRIORITY |
|---|---|---|---|
| 6 | **Security and fault tolerance** | Penetration prevention, spoofing/sniffing protection, data privacy and protection | 1 |
| 7 | **User-friendly interface** | The ability of the system to be visualized in different devices such as smart-phones, tablets and computers | 3 |
| 8 | **Semantic fuzzy matching** | To perform sufficient and determined level of matching with high level of quality enable: strict (statement match), soft (subject predicate match), non-strict (subject match) matching | 3 |
| 9 | **Time tolerant processing** | Time optimization of performance | 2 |
| 10 | **Scalability** | The complexity of matching algorithm should not be exponential in order for the rising amount of incoming data to perform | 2 |

| | | | |
|---|---|---|---|
| | | rapid and time-efficient processing | |

# 5. Technical Software Requirement

## 5.1. Software Aspects

The following are the software requirement for Industry 4.0 tool:

- Git
- Tomcat
- Virtuoso
- Ubuntu
- Java
- Maven
- AngularJS
- JS

# 6. Operational Specification

## 6.1. Installing Git

The git can be installed by the following command in Ubuntu:

*sudo apt-get install git-core*

Git is mainly used to download the I 4.0 software from the GitHub repository.

## 6.2. I4matcher

Clone git repository of Industry 4.0 tool:

*git clone https://github.com/IntegrationI40StandardsSemLab/Integration-I4.0.git*

## 6.3. Installing Virtuoso

- Virtuoso-opensource can be downloaded from the below link:

  https://sourceforge.net/projects/virtuoso/files/latest/download?source=files

- Navigate to the downloaded folder.
- Extract the package.
- Make sure the following packages and recommended versions are installed on your system:

  *sudo apt-get install autoconf automake libtool flex bison gperf gawk m4 make OpenSSL*

- Run the following comand in Ubuntu:

  *sudo apt-get install libssl-dev*

- Go the extracted package location. It is better to run autogen.sh by typing ./autogen.sh, which checks for the presence and right version of some of the required components, and if it reports any missing package then, install that package.

- Execute ‘./configure’ command to configure the package for your system.
- Execute ‘sudo make’ command to compile the package
- Type `sudo make install’ to install the programs and any data files and documentation.
- Start the Virtuoso Server by the following step:

*sudo /usr/local/virtuoso-opensource/bin/virtuoso-t -f -c /usr/local/virtuoso-opensource/var/lib/virtuoso/db/virtuoso.ini*

For more information, please visit:

http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VOSMake

## 6.4. Installing Java

The link below is where you can use steps for installing java:

http://www.oracle.com/technetwork/java/javase/downloads/index.html

The following commands are used to install Java in Ubuntu:

*sudo apt-get install oracle-java8-installer*

*sudo update-alternatives --config java*

## 6.5. Installing Tomcat

- Create Tomcat user:

*groupadd tomcat*

*sudo useradd -s /bin/false -g tomcat -d /opt/tomcat tomcat*

- copy the link from

  http://tomcat.apache.org/download-80.cgi

  (Binary Distribution → Core → tar.gz) and download it:

*wget http://...*

- create a new repository and unzip there the file:

*sudo mkdir /opt/tomcat*

*sudo tar xvf apache-tomcat-8*tar.gz -C /opt/tomcat --strip-components=1*

- change to the Tomcat installation path and give the tomcat user write and read access to the conf directory:

*cd /opt/tomcat*

*sudo chgrp -R tomcat conf*

*sudo chmod g+rwx conf*

*sudo chmod g+r conf/**

- make the tomcat user the owner of the work, temp, and logs directories:

*sudo chown -R tomcat work/ temp/ logs/*

- create and open the Upstart script:

*sudo nano /etc/init/tomcat.conf*

- paste there the following script (change the link at env JAVA_HOME to the one where you store your JAVA):

*description "Tomcat Server"*

*start on runlevel [2345]*

*stop on runlevel [!2345]*

*respawn*

*respawn limit 10 5*

*setuid tomcat*

*setgid tomcat*

*env JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64/jre*

*env CATALINA_HOME=/opt/tomcat*

*# Modify these options as needed*

*env JAVA_OPTS="-Djava.awt.headless=true -Djava.security.egd=file:/dev/./urandom"*

*env CATALINA_OPTS="-Xms512M -Xmx1024M -server -XX:+UseParallelGC"*

*exec $CATALINA_HOME/bin/catalina.sh run*

*# cleanup temp directory after stop*

*post-stop script*

*rm -rf $CATALINA_HOME/temp/**

*end script*

- save, exit the script and reload the Upstart configuration:

*sudo initctl reload-configuration*

- start Tomcat:

*sudo initctl start tomcat*

- Now you can see whether it works or not by going to the http://localhost:8080

There should be a working web page. For more information please visit: https://www.digitalocean.com/community/tutorials/how-to-install-apache-tomcat-8-on-ubuntu-14-04

## 6.6. Installing Maven

The following commands are used to install Maven in Ubuntu:

*Sudo apt-get install maven*

## 6.7. Building and deployment

Firstly, navigate to project i4matcher dir and build WAR with

*mvn clean dependency:copy-dependencies package*

Then

*sudo mv target/i4matcher.war /opt/tomcat/webapps*

# 7. List of AutomationML and OPC UA files

- AutomationML files

  Type: aml

  *Source:* *https://github.com/i40-Tools/HeterogeneityExampleData*

- OPC UA file

  Type: opc ua

  *Source:* *http://lib.tkk.fi/Dipl/2010/urn100411.pdf*

  Can be copied from pdf file.

# 8. Implementation

## 8.1. Frontend

### 8.1.1. File Upload Handling

In order to create a smooth and fast file uploader, our team used JQuery, AJAX and HTML DOM approaches. We introduced a function *handleFileSelect* that analyzes an input in order to proceed only valid data. Than it controls that exactly two files were uploaded. After that, the data is stored in the user's browser cash, encoded into the base64 format, which allows to send the information to the visualization tool or to download it directly from the page. Modern front-end methods that we use, allow doing everything without reloading the page.

### 8.1.2. Data Visualization

Visualization of the incoming data is based on a d3.js library that works on a json-like data. Therefore, in order to draw aml/opcua files, we had to create a tool that converts any xml-like data into an appropriate format. After that, the basic interface was improved, providing more user-friendly feedback for the user when he visualizes the data. Thus, a collapsible, colorful tree-like data visualization was implemented.

### 8.1.3. SPARQL Query

A tool that uses a SQPRQL in order to retrieve any specific data from the produced integrated file is based on a Virtuoso that is implemented at the back end side of the application. Its main goal is to provide the user an ability to get rid of useless information in the produced file and extract only what is necessary for the further work.

### 8.1.4. Web-Design

Our team used Twitter Bootstrap as a core of an interface for the application. Built at Twitter, Bootstrap utilizes LESS CSS, is compiled via Node, and is managed through GitHub. It is a cross-platform solution with responsive CSS. Above that, some user features were implemented into the web part of the project, which helped to achieve a more user-friendly and understandable design.

### 8.1.5. Gold Standard

Gold Standard for the matching part is a Demo in which the User can experience and feel the output of the project. Once the appropriate button is hit, two concrete files are attached at the UI part from resource, routed to Matcher Controller and consequently the matched file is compared with predeveloped hand written matched object instance.

## 8.2. Backend

### 8.2.1. Transformation

The means for transformation from source industry specific XML (automationML) to RDF were chosen as following: Saxon XSLT transforming tool with xsl templates inherited from Krextor© project. Generic Javax TransformerFactory implements Saxon transformation routine and outputs .turtle file, which gets mapped onto Jena model; for transformation from OPCUA to RDF were chosen Sun's XSOM library and for processing XML Schemas, Apache Xerces for processing XML data and Apache Jena for managing data.

### 8.2.2. Matching

Matcher uses two rdf files, files after transformation from any format to rdf. Apache Jena is used for matching process. Firstly, we get statements for each file correspondingly by the means of reading turtle files. Then a new Jena model is created. There are 3 available options of matching depending on a user choice which is implemented through RESTful parameter:

- Strict – match of the whole statement (subject, predicate, object);
- Soft – match subject and predicate (combine objects that have the same subject and predicate);
- Non-strict – match subject (combine predicates with objects that have the same subject).

Duplicates are avoided.

### 8.2.3. RESTful matcher controller

RESTful spring based mvc controller that listens to HTTP POST on the /upload destination with following /{value} parameter that specifies matching rigorousness. The method implements validation based on each of the files' data types against respectful XSD schemas. The schemas for validation are taken from resources project folder, handled by javax SchemaFactory and processed with javax Validator. Afterwards, controller calls corresponding methods for semantifying two files, passing them to Jena model and providing the main matching. Finally Controller writes the resulting matched object to Virtuoso as a graph.

### 8.2.4. SPARQL Query Controller

Query controller listens to HTTP GET on the /get destination and accepts "query" parameter which should containg URLencoded SPARQL query text.

This controller creates an instance of JenaProvider VirtualQueryExecution and fetches the results into result set that is later output in JSON format.

# 9. Test cases

| № | Functionality | Test case description | Expect results | Passed/ not passed |
|---|---|---|---|---|
| 1 | **Uploading to the web page** | Two files of type .aml/.opcua are chosen in the upload form. | Files uploaded. Visualization and downloading provided. | **Passed** |
| | | A file of type .xml is chosen in the upload form. | Files not uploaded. Error appeared. | **Not Passed** |
| 2 | **Unit Testing (frontend)** | Testing tree drawing functionality on valid JSON file. | Tree is drawn and user can interact with it. | **Passed** |
| | | Invalid JSON file is uploaded. | Tree is not drawn. Error appeared. | **Not Passed** |
| 3 | **Unit Testing (backend)** | Transformation OPCUA to RDF | Output is turtle file | **Passed** |
| | | Transformation AML to RDF | Output is turtle file | **Passed** |
| | | Transformation XSD schema to OWL ontology | Output is OWL ontology file | **Passed** |
| 4 | **Gold standard** | Matching two AutomationML files | Output is integrated file | **Passed** |
| 5 | **Stress Handling** | DDOS | | **passed** |

# 10. Complexity measurements of front-end scripts

The Cyclomatic complexity is a metric developed by Thomas J. McCabe in 1976. It measures the complexity of a program by counting the number of linearly independent paths through a program's source code. The lower the cyclomatic complexity the better. McCabe suggested that programmers should split a software module into smaller parts whenever the cyclomatic complexity of the module exceeded 10. Below we provide the Cyclomatic complexity and some other important information about the front-end scripts.

Another measurement of complexity is Halstead metrics, according to Halstead: a computer program is an implementation of an algorithm considered to be a collection of tokens which can be classified as either operators or operands.

| Function | Lines | Program Level | Cyclomatic Complexity | Halstead Volume | Halstead Potential |
|---|---|---|---|---|---|
| submitFormFunc | 60 | 0.0114 | 5 | 415 | 4.75 |
| downloadInt | 37 | 0.00741 | 6 | 1080 | 8.00 |
| getQuery | 12 | 0.0314 | 1 | 255 | 8.00 |
| handlefileSelect | 81 | 0.0102 | 4 | 787 | 8.00 |

Table 1: File name: my_finctions.js

## External References

1. Examples of AutomationML data

   https://github.com/i40-Tools/HeterogeneityExampleData

2. Krextor project

   https://github.com/EIS-Bonn/krextor

3. OPCUA article and OPCUA file

   http://lib.tkk.fi/Dipl/2010/urn100411.pdf

4. AutomationML Whitepaper

   https://www.automationml.org/o.red.c/dateien.html

5. Transformation XML Schemas (XSD) and XML Data to RDF/OWL

   http://www.srdc.com.tr/projects/salus/blog/?p=189