SemanticProxy

Technical documentation

For version: v0
Last change: 2 September 2014
This document is intended as guidance for developers and frequently subject to change.

Table of Contents

General Description

The RichWPS SemanticProxy (SP) is a directory service for use within the RichWPS orchestration environment. Its task is to provide information about OGC Web Processing Service instances in a specific network. It therefore SP stores and maintains information about WPS and their respective processes as a Resource Description Framework (RDF) graph. The data can be manipulated in CRUD fashion (Create, Read, Update, and Delete) and is exposed to clients via a Representational State Transfer (REST) interface based on the Hypertext Transport Protocol (HTTP). WPS instances and their respective processes are the objects in this paradigm. Expected client is the RichWPS ModelBuilder.

The software is written in Java 7 using the NetBeans 7.3 IDE with Maven as well as applying the JDK 7. As administrator the SP can be configured via an XML configuration file beforehand, and sample/default data can be loaded into the DB right at system start.

Applied Techniques

RDF and REST are two man techniques/technologies applied by the SP, a basic introduction can be found below

RDF

The Resource Description Framework (RDF) is used to describe WPS services. RDF is extendable and allows a smooth transition into the application of semantically enabled operations like semantic searches. Applying the linked data principle, the service can also be integrated into existing RDF environments.

From: Unpublished Paper

RDF/XML is used as the SPs data exchange format and currently the only notation supported. For RDF related tasks, the SP relies on OpenRDF Sesame RDF grounds on resources which can be described with statements that consist of a subject, predicate and object. Subjects, predicates and objects can be resources themselves, and therefore be described by other statements. That way the relation between resources in RDF can be represented as a graph. An object can also be a literal. A vocabulary is needed in order to have a base for statements. The vocabulary itself consists of statements. The W3C already offers vocabulary for certain domains. In further iterations, it is considered to employ existing vocabulary like OWL-S, but for now there is no scope for this. A vocabulary is created on whose base the data is structured. Because RDF is not quite compatible with the concept of a directory service links to resources that belong to other descriptions are also thoroughly regulated.

A special property of RDF is that statements can refer to each other beyond system borders. This is possible because each resource is assigned a globally unique ID in form of a URI. The provision of information via this URI on a web server is a fundamental principle of linked data.

REST

REST is a paradigm to organize and access data in a structured way. It also defines a Web Service interface often based on HTTP. The SP applies the java-based SPARC Web Server for exposing its services. REST organizes server-side data as (virtual) objects or resources that can be accessed via a

URI. Often and also here peer resources are also published as a list resource. HTTP verbs are used to manipulate these objects in a CRUD fashion.

- A POST to a list resource (obviously containing the right data) creates a new resource.
- A PUT to a specific resource updates the resource.
- A Delete to a specific resource deletes the resource.
- A GET to a specific resource or list resource retrieves a representation of this resource, for now RDF/XML is the only supported format for this.

Further information about the exchanged massages will be described in a later chapter.

Architecture (Layering)

The SP consists of the following layers.

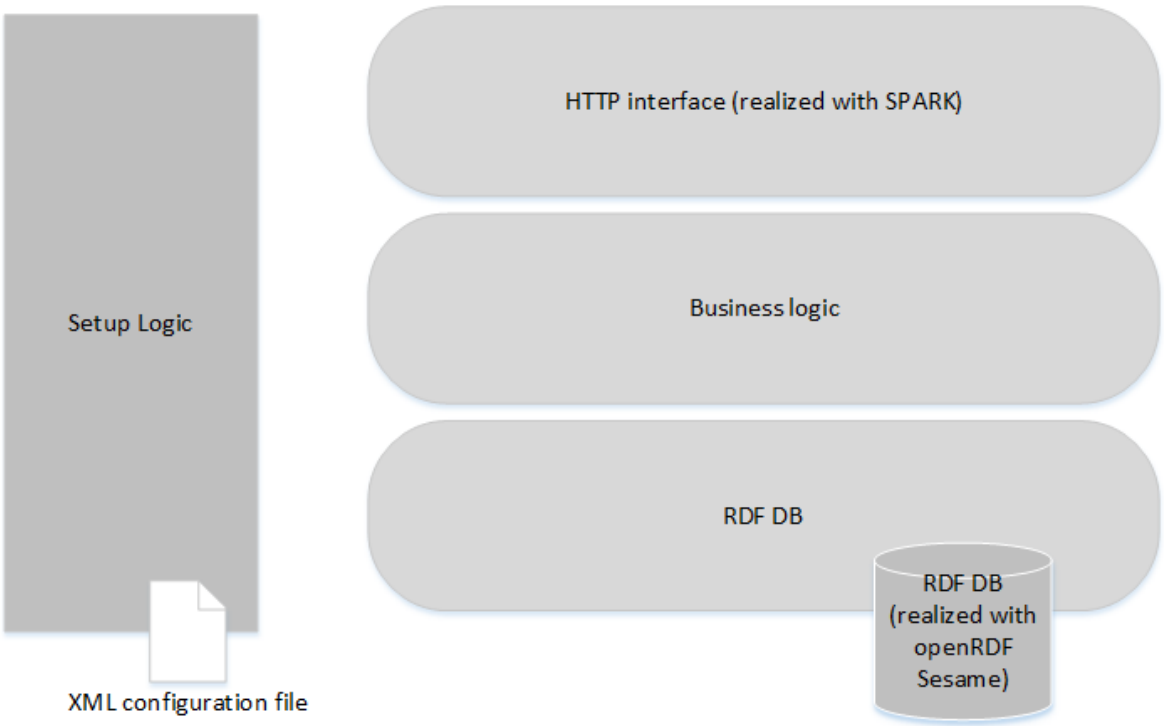

Figure 1 Architecture, based on layers

The setup logic is not actually a layer; it is used as an organizing component to setup the application. Therefore, it relies on a configuration file. The respective layers serve the following purposes.

HTTP interface	Realizes the web access for clients.
Business logic	Contains the CRUD logic for data manipulation, contains also the functions for search and validation.
RDF DB	Layer for customized data access.

The code is mostly organized with static methods, since there is no data to store at this point. HTTP Requests obviously arrive at the HTTP interface, which deals with the HTTP processing, meaning the

desired operation and resource are extracted from the request and forwarded to the business for further processing. Finally the DB is used to fulfil the task and an answer message, obviously depending on the success of the operation is forward upwards the call stack.

Project Setup

Here follows a small orientation of the project. The favored IDE is NetBeans.

Packaging

The project is organized and built with Maven and contains the following packages. These are mostly according to the architectural layers.

de.hsos.richwps.sp	Contains setup and configuration functionality.
de.hsos.richwps.sp.web	Contains the code for the web interface.
de.hsos.richwps.sp.restlogic	Contains the CRUD logic for data manipulation, contains also the functions for search and validation.
de.hsos.richwps.rdfdb	Package for customized data access.
de.hsos.richwps.sp.types	Contains custom data types.

Build process

As a Maven-organized project the build process is prescribed in the Project Object Model (POM) file and follows the Maven conventions in terms of directory structure etc. However for application of the software outside of the IDE (which is the productive case) the dependencies are relocated from the Maven repository into a local /lib/- directory while the original SP code is composed into a JAR-Archive. As a matter of fact it is not possible to encapsulate all dependencies in single JAR because this will interfere with the Service Provider Interface (SPI) used by Sesame.

Directory structure

SemanticProxy	Folder	Main directory
SemanticProxy/lock	Folder	Contains lock information for the Sesame DB
SemanticProxy/RDF	Folder	Contains sample file for RDF descriptions of WPS and processes
SemanticProxy/src	Folder	Contains source code
SemanticProxy/target	Folder	Contains class-files, JARs etc.
SemanticProxy/config.xml	File	The configuration file
SemanticProxy/log4j.properties	File	Determines settings for logging with log4j
SemanticProxy/memorystore.data	File	Contains the RDF DB
SemanticProxy/nbactions.xml	File	NetBeans owned file
SemanticProxy/nb-configuration.xml		NetBeans owned file
SemanticProxy/pom.xml	File	Maven POM file
SemanticProxy/README.txt	File	Contains basic use and installation instructions
SemanticProxy/SemanticProxy.log	File	Contains logged information

General program routines

There are some routines that are recurrent and therefore expressive for the general working of the SP. As mentioned before there is mostly rather sequential programming than OOP.

Setup

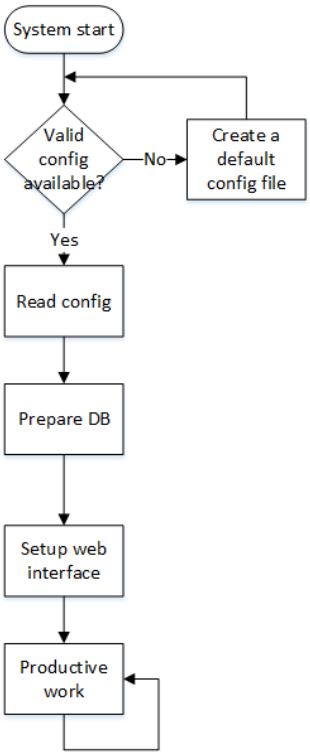

Figure 2 Startup routine

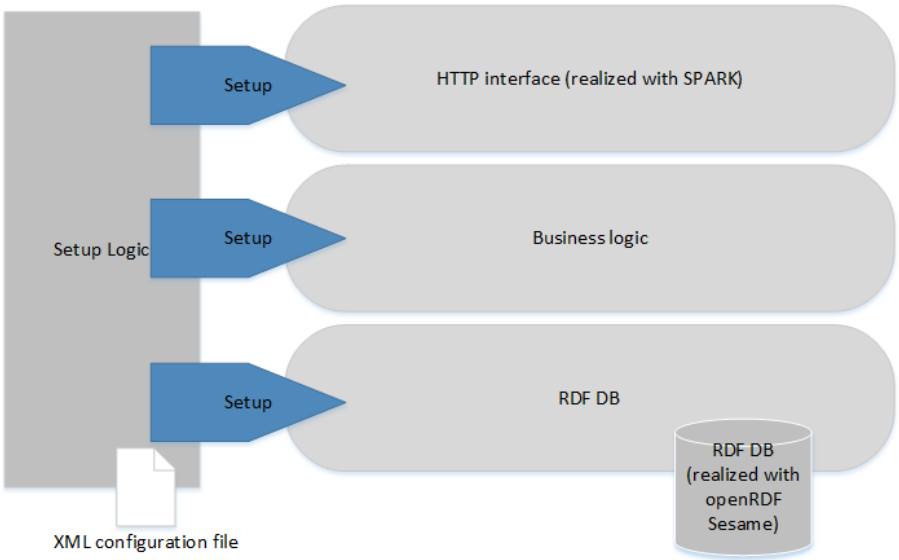

Figure 3 System configuration at startup

This routine sets up the whole application from the bottom layer to the top. At system start the configuration file is loaded, if no config file can be found a default file is created and used. The next

step depends on the configuration, if defined the DB is deleted recreated and loaded with sample data as specified in the configuration. A fresh Database gets an RDF resource inserted that represents the network the SP operates for. The new resource serves as root node for following RDF resources. When the DB is prepared the web access is set up. This happens separately for CRUD and Search front ends.

Create process

A process can is created with the following procedure. At first a HTTP POST Request to the process list resource is received at the web interface. Here the message body is extracted and the RDF description is forwarded to the matching business logic method. Then the RDF data is validated. The validation process is rather extensive the data is checked for adequate vocabulary and integrity also it is checked whether the DB already contains one of the resources described in the data. Finally it is determined whether the specified parent WPS is present.

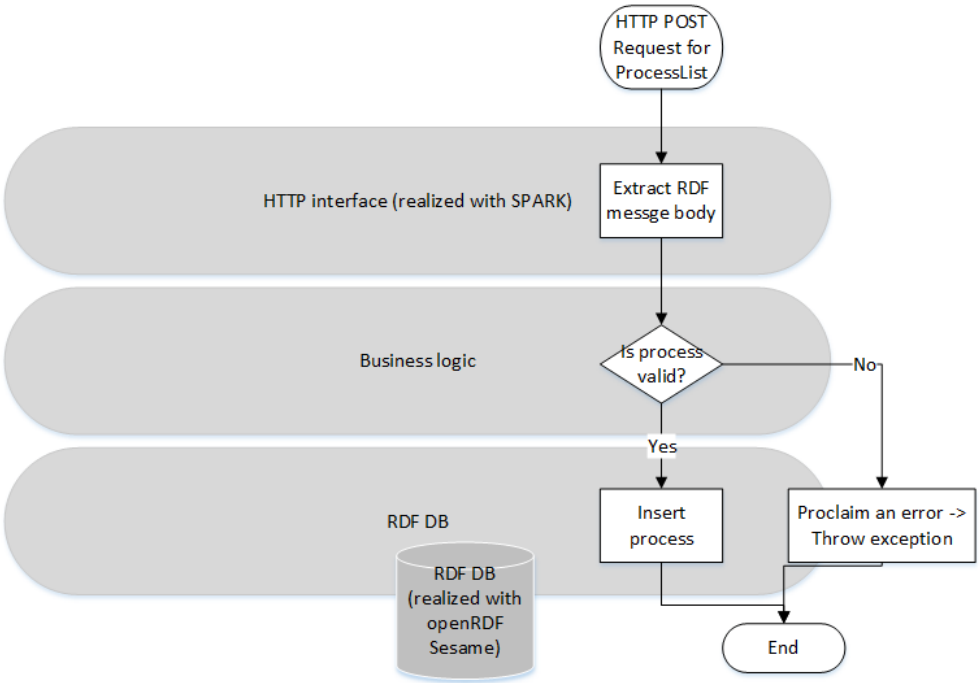

Figure 4 Create process procedure

When inserting, two RDF statements are created that explicit mark the process as a child of the WPS and another one vice-versa that specifies the WPS as a parent of the process. This needs to be done in order to create a cohesive graph.

Data documentation

The data managed by the SP is organized in RDF statements (See chapter RDF) that form a graph. The statements are the only data type managed by the DB, they can be handled individually and independently from each other. However as a directory service that is intended to provide the technical specification of WPS processes for the ModelBuilder the data has to be complete and structured as well.

Data structure

Figure 5 shows the general data structure. The root node represents the sub network; it is associated with an owner and a domain name. On the second layer are the WPS services and on the third layer are the processes. The resources are connected like this in order to be able to browse the graph later on with a client.

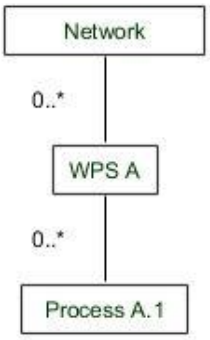

Figure 5 General data structure / RDF graph

The network node is inserted into the DB at system start. After that WPSs and processes can be added, updated and removed.

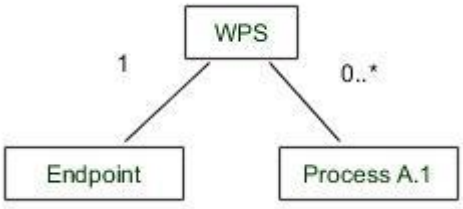

Figure 6 WPS RDF graph

Figure 6 shows the RDF graph for a WPS. Next to the processes an endpoint (URL) is provided that is to be used to access the WPS service.

Whereas the WPS graph is quite simple, the graph for a process is way more complicated as depicted in Figure 7. The structure was designed according to the WPS DescribeProcessResponse message.

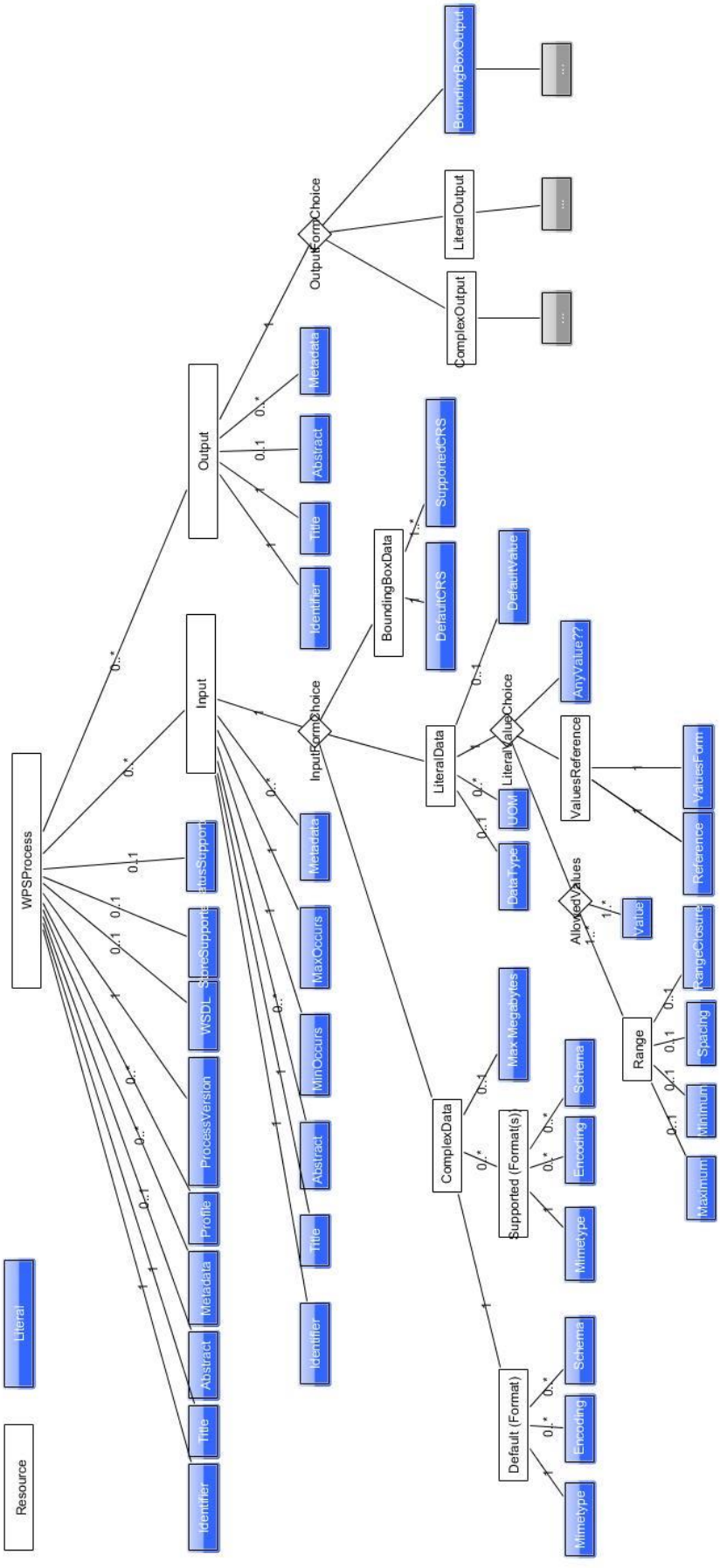

Figure 7 Process RDF graph

Vocabulary

The RDF vocabulary forms the base on which the whole service operates.

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
        xmlns:rdf="http://öwww.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
        xmlns:rwps="http://localhost:4567/semanticproxy/resources/vocab#">

<rdf:Description rdf:about="http://localhost:4567/semanticproxy/resources/vocab#domain">
        <rdfs:label>Has domain</rdfs:label>
</rdf:Description>

<rdf:Description rdf:about="http://localhost:4567/semanticproxy/resources/vocab#owner">
        <rdfs:label>Is owned by</rdfs:label>
</rdf:Description>

<rdf:Description rdf:about="http://localhost:4567/semanticproxy/resources/vocab#wps">
        <rdfs:label>Has as WPS</rdfs:label>
</rdf:Description>

<rdf:Description rdf:about="http://localhost:4567/semanticproxy/resources/vocab#endpoint">
        <rdfs:label>Has endpoint</rdfs:label>
</rdf:Description>

<rdf:Description rdf:about="http://localhost:4567/semanticproxy/resources/vocab#process">
        <rdfs:label>Has as process</rdfs:label>
</rdf:Description>

<rdf:Description rdf:about="http://localhost:4567/semanticproxy/resources/vocab#identifier">
        <rdfs:label>Has identifier</rdfs:label>
</rdf:Description>

<rdf:Description rdf:about="http://localhost:4567/semanticproxy/resources/vocab#title">
        <rdfs:label>Has title</rdfs:label>
</rdf:Description>

<rdf:Description rdf:about="http://localhost:4567/semanticproxy/resources/vocab#abstract">
        <rdfs:label>Has abstract</rdfs:label>
</rdf:Description>

<rdf:Description rdf:about="http://localhost:4567/semanticproxy/resources/vocab#metadata">
        <rdfs:label>Has metadata</rdfs:label>
</rdf:Description>

<rdf:Description rdf:about="http://localhost:4567/semanticproxy/resources/vocab#profile">
        <rdfs:label>Has profile</rdfs:label>
</rdf:Description>

<rdf:Description rdf:about="http://localhost:4567/semanticproxy/resources/vocab#wsdl">
        <rdfs:label>Is described with WSDL</rdfs:label>
</rdf:Description>

<rdf:Description
rdf:about="http://localhost:4567/semanticproxy/resources/vocab#processversion">
        <rdfs:label>Has process version</rdfs:label>
</rdf:Description>

<rdf:Description
rdf:about="http://localhost:4567/semanticproxy/resources/vocab#storesupported">
        <rdfs:label>Supports store</rdfs:label>
</rdf:Description>

<rdf:Description
rdf:about="http://localhost:4567/semanticproxy/resources/vocab#statussupported">
        <rdfs:label>Supports status</rdfs:label>
</rdf:Description>

<rdf:Description rdf:about="http://localhost:4567/semanticproxy/resources/vocab#input">
        <rdfs:label>Has input</rdfs:label>
</rdf:Description>

<rdf:Description rdf:about="http://localhost:4567/semanticproxy/resources/vocab#output">
        <rdfs:label>Has output</rdfs:label>
```

```
</rdf:Description>

<rdf:Description rdf:about="http://localhost:4567/semanticproxy/resources/vocab#minoccurs">
        <rdfs:label>Occurs at minimum</rdfs:label>
</rdf:Description>

<rdf:Description rdf:about="http://localhost:4567/semanticproxy/resources/vocab#domain">
        <rdfs:label>Has domain</rdfs:label>
</rdf:Description>

<rdf:Description
rdf:about="http://localhost:4567/semanticproxy/resources/vocab#inputformchoice">
        <rdfs:label>InputFormChoice</rdfs:label>
</rdf:Description>

<rdf:Description
rdf:about="http://localhost:4567/semanticproxy/resources/vocab#outputformchoice">
        <rdfs:label>OutputFormChoice</rdfs:label>
</rdf:Description>

<rdf:Description rdf:about="http://localhost:4567/semanticproxy/resources/vocab#networkclass">
        <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
        <rdfs:label>Network</rdfs:label>
</rdf:Description>

<rdf:Description rdf:about="http://localhost:4567/semanticproxy/resources/vocab#wpsclass">
        <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
        <rdfs:label>Web Processing Service</rdfs:label>
</rdf:Description>

<rdf:Description rdf:about="http://localhost:4567/semanticproxy/resources/vocab#processclass">
        <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
        <rdfs:label>WPS process</rdfs:label>
</rdf:Description>

<rdf:Description
rdf:about="http://localhost:4567/semanticproxy/resources/vocab#datainputclass">
        <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
        <rdfs:label>Data input</rdfs:label>
</rdf:Description>

<rdf:Description
rdf:about="http://localhost:4567/semanticproxy/resources/vocab#processoutputclass">
        <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
        <rdfs:label>Process output</rdfs:label>
</rdf:Description>

<rdf:Description
rdf:about="http://localhost:4567/semanticproxy/resources/vocab#complexdataclass">
        <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
        <rdfs:label>Complex data</rdfs:label>
</rdf:Description>

<rdf:Description
rdf:about="http://localhost:4567/semanticproxy/resources/vocab#literaldataclass">
        <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
        <rdfs:label>Literal data</rdfs:label>
</rdf:Description>

<rdf:Description
rdf:about="http://localhost:4567/semanticproxy/resources/vocab#boundingboxdataclass">
        <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
        <rdfs:label>BoundingBox data</rdfs:label>
</rdf:Description>

</rdf:RDF>
```

Interface

The interface is REST based and offers the following endpoints:

http://[host]:[port]/semanticproxy	User info about SP
http://[host]:[port]/semanticproxy/resources	Network resource/root node for further browsing
http://[host]:[port]/semanticproxy/resources/vocab	RDF vocabulary
http://[host]:[port]/semanticproxy/resources/wpss	List resource WPSs
http://[host]:[port]/semanticproxy/resources/processes	List resource processes
http://[host]:[port]/semanticproxy/resources/[process/processA]	An individual resource
http://localhost:4567/semanticproxy/search?keyword=[select]	Search endpoint

The SP implements the linked data principles what means that each resource is accessible by its name/URL.

Read resource

To read a resource use g GET request to the resource URI, if present, the SP will send back a representation of that resource. The desired representation can be specified in the header of the HTTP request but the service is not obligated to answer with that representation.

Create WPS

A WPS can be created using a POST request sent to the WPS list. The body of the message has to contain the RDF description of the WPS.

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
        xmlns:rwps="http://localhost:4567/semanticproxy/resources/vocab#">

<rdf:Description rdf:about="http://localhost:4567/semanticproxy/resources/wps/MacrophyteWPS">
        <rdf:type
rdf:resource="http://localhost:4567/semanticproxy/resources/vocab#wpsclass"/>
        <rwps:endpoint>http://www.lkn.de/geo/wps/macrotphytewps</rwps:endpoint>
</rdf:Description>

</rdf:RDF>
```

Create process

A Process can be created using a POST request sent to the process list. The body of the message has to contain the RDF description of the process.

```
<rdf:RDF
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
        xmlns:rwps="http://localhost:4567/semanticproxy/resources/vocab#">

        <!-- process -->

        <rdf:Description
rdf:about="http://localhost:4567/semanticproxy/resources/process/compare">
                <rdf:type
rdf:resource="http://localhost:4567/semanticproxy/resources/vocab#processclass"/>
                <rwps:identifier>baw.modelvalidation.compare</rwps:identifier>
```

```
                <rwps:title>Compare</rwps:title>
                <rwps:abstract>Compares model and field data</rwps:abstract>
                <rwps:processversion>1.0</rwps:processversion>
                <rwps:input
rdf:resource="http://localhost:4567/semanticproxy/resources/input/in_fielddata"/>
                <rwps:input
rdf:resource="http://localhost:4567/semanticproxy/resources/input/in_modeldata"/>
                <rwps:output
rdf:resource="http://localhost:4567/semanticproxy/resources/output/out_differencedata"/>
                <rwps:wps
rdf:resource="http://localhost:4567/semanticproxy/resources/wps/ModelValidationWPS"/>
        </rdf:Description>

        <!-- complex input -->

        <rdf:Description
rdf:about="http://localhost:4567/semanticproxy/resources/input/in_fielddata">
                <rdf:type
rdf:resource="http://localhost:4567/semanticproxy/resources/vocab#datainputclass"/>
                <rwps:identifier>in.fielddata</rwps:identifier>
                <rwps:title>Field data</rwps:title>
                <rwps:abstract>Ingoing field data</rwps:abstract>
                <rwps:minoccurs>1</rwps:minoccurs>
                <rwps:maxoccurs>1</rwps:maxoccurs>
                <rwps:metadata>http://www.metadata.de/metadata</rwps:metadata>
                <rwps:inputformchoice
rdf:resource="http://localhost:4567/semanticproxy/resources/complexdata/complex_fielddata"/>
        </rdf:Description>

        <rdf:Description
rdf:about="http://localhost:4567/semanticproxy/resources/complexdata/complex_fielddata">
                <rdf:type
rdf:resource="http://localhost:4567/semanticproxy/resources/vocab#complexdataclass"/>
        </rdf:Description>

        <!-- complex input -->

        <rdf:Description
rdf:about="http://localhost:4567/semanticproxy/resources/input/in_modeldata">
                <rdf:type
rdf:resource="http://localhost:4567/semanticproxy/resources/vocab#datainputclass"/>
                <rwps:identifier>in.modeldata</rwps:identifier>
                <rwps:title>Model data</rwps:title>
                <rwps:abstract>Ingoing model data</rwps:abstract>
                <rwps:minoccurs>1</rwps:minoccurs>
                <rwps:maxoccurs>1</rwps:maxoccurs>
                <rwps:metadata>http://www.metadata.de/metadata</rwps:metadata>
                <rwps:inputformchoice
rdf:resource="http://localhost:4567/semanticproxy/resources/complexdata/complex_modeldata"/>
        </rdf:Description>

        <rdf:Description
rdf:about="http://localhost:4567/semanticproxy/resources/complexdata/complex_modeldata">
                <rdf:type
rdf:resource="http://localhost:4567/semanticproxy/resources/vocab#complexdataclass"/>
        </rdf:Description>

        <rdf:Description
rdf:about="http://localhost:4567/semanticproxy/resources/output/out_differencedata">
                <rdf:type
rdf:resource="http://localhost:4567/semanticproxy/resources/vocab#processoutputclass"/>
                <rwps:identifier>out.differencedata</rwps:identifier>
                <rwps:title>Difference data</rwps:title>
                <rwps:abstract>Difference of the two input data sets</rwps:abstract>
                <rwps:metadata>http://www.metadata.de/metadata</rwps:metadata>
                <rwps:outputformchoice
rdf:resource="http://localhost:4567/semanticproxy/resources/complexdata/complex_differencedata
"/>
        </rdf:Description>

        <rdf:Description
rdf:about="http://localhost:4567/semanticproxy/resources/complexdata/complex_differencedata">
                <rdf:type
rdf:resource="http://localhost:4567/semanticproxy/resources/vocab#complexdataclass"/>
        </rdf:Description>

</rdf:RDF>
```

Delete WPS

Send a DELETE request to a WPS resource. If the WPS has any processes these will be deleted as well.

Delete process

Send a DELETE request to a process resource.

Update WPS

Send a PUT request to a WPS. The body needs to contain a new description, process statements are not allowed since the old ones will remain in the DB. The WPS is updated by deleting the old statements and inserting the new statements.

Update process

Send a PUT request to a process. The body needs to contain a new description of the process, its inputs outputs etc. also a statement specifying the parent WPS is required (see body of create process message). The process is updated by deleting the old statements and inserting the new statements.

Miscellaneous

- Installation instruction can be found in README.txt
- Junit testing is applied for DB operations
- A client is developed