Heliophysics Integrated Observatory

Project No.: 238969
Call: FP7-INFRA-2008-2

Semantic Mapping Service Developers Guide

Version 0.1

Title:	**Semantic Mapping Service – Developers Guide**
Document No.:	HELIO_UNIMAN_S2_005_TN_SMS
Date:	23 March 2012
Editor:	**Anja Le Blanc**, The University of Manchester
Contributors:	
Distribution:	Project

Semantic Mapping Service – Developers Guide
Version 0.1

Revision History

Version	Date	Released by	Detail
0.1	23/03/2012	Anja Le Blanc	Initial Draft

Note: Any notes here.

Semantic Mapping Service – Developers Guide
Version 0.1

1 Introduction

The Semantic Mapping Service (SMS) provides a set of functions which use the HELIO ontology as source of information. It is only implemented as SOAP web service.

1.1 Assumed Knowledge for the Developer

To build the service	
Java (compile service)	http://www.oracle.com/technetwork/java/javase/overview/index.html
Tomcat (web container to host the service)	http://tomcat.apache.org/
Maven (build system)	http://maven.apache.org/ (or use a plug-in to your development environment)
To extend the service	
SOAP + WSDL (web service definition)	http://www.w3.org/TR/wsdl (or use a plug-in to your development environment)
OWL (interaction with the ontology)	http://www.w3.org/TR/owl-ref/ http://owlapi.sourceforge.net/
HermiT (ontology reasoner)	http://hermit-reasoner.com/

2 System Requirements

The SMS requires a web application container such as Tomcat. The web services was developed and tested using Apache Tomcat 6, but the author does not know of any reason why it should not be able to work in a different web container. It was tested in a Windows and Linux OS.

To build the service from its source a Java compiler and the Maven build environment are required; a valid POM file is part of the source.

3 Download, Build, Install

3.1 Downloading Source

The SMS source code is part of the HELIO-vo project in sourceforge. The main page is: http://sourceforge.net/projects/helio-vo/. You can download the complete source code using svn with:

```
    svn co https://helio-vo.svn.sourceforge.net/svnroot/helio-vo helio-vo
```

SMS is located in the helio-sms folder.

3.2 Building SMS

SMS is configured to be built using Maven. It is recommended to make a

```
  maven clean
```

before building and

```
  maven package
```

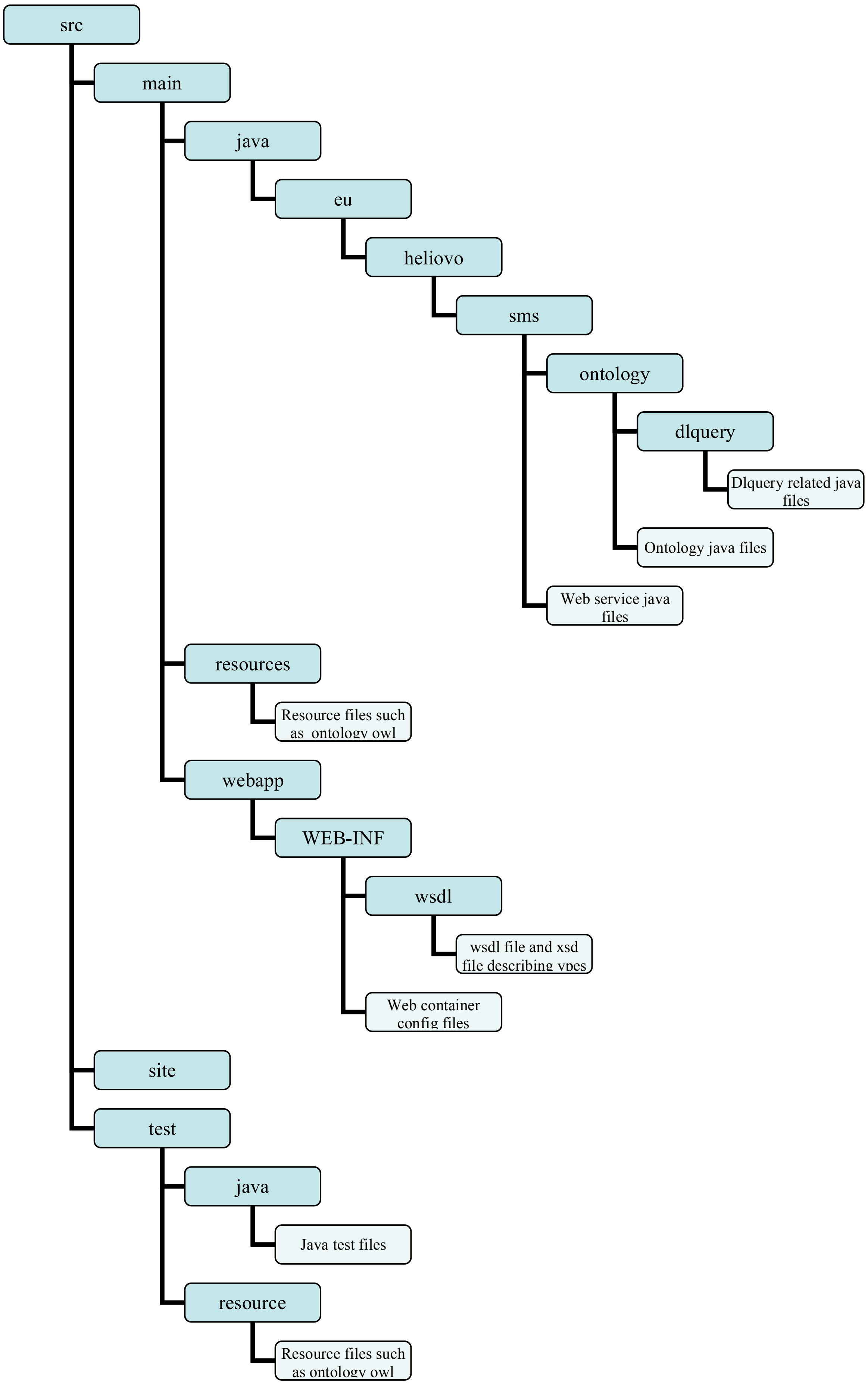
in the helio-sms directory.
This will build a war file (helio-sms-version.war) in the targets directory.

3.3 Installing in Tomcat

Installing SMS in Tomcat is done by copying the sms war file into the `WEBAPPS` directory. The location of the WEBAPPS can be found in the `server.xml` of the Tomcat installation. It is part of the <Host> section. If in the same section the attribute `unpackWARs` is set to 'true' and the attribute `autoDeploy` is set to 'true' the unpack an deploy will happen automatically at restart of Tomcat.

4 Code Structure

All Source code is place in the `src` directory. Building the project will produce a targets directory into which all output files are placed. All java code for the web application is located in a sub directory `main` within `src`. The directory `test` contains files necessary for the unit tests of this application. The service is using an ontology over which it is reasoning to compute the return strings. The ontology owl files are contained in the `resources` folder in the `main` directory. The `WEB_INF` folder contains necessary configuration files for the web applications container (like Tomcat) and the description of the SOAP service in a separate `wsdl` directory.


src
main
java
eu
heliovo
sms
ontology
dlquery
Dlquery related java files
Ontology java files
Web service java files
resources
Resource files such as ontology owl
webapp
WEB-INF
wsdl
wsdl file and xsd file describing types
Web container config files
site
test
java
Java test files
resource
Resource files such as ontology owl

4.1 Dependencies

jxws-api.jar version 2.1
jaxws-rt.jar version 2.1.4
owlapi.jar version 3.1.0
HermiT.jar version 1.3.2

4.2 Logging

SMS is using `log4j` logging mechanism to write its logging information into a file `sms.log` in the tomcat logs directory. The logging configuration file logs4j.properties is placed in the `\WEB-INF\classes` directory. Default logging level is set to INFO.

4.3 Java doc

The Java doc files can be generated from the source. Public functions are annotated with the required tags.

5 Unit Tests

Unit tests are divided into two categories. The first is testing the ontology and the second is testing the web service.

5.1 Ontology tests

Ontology tests are run against the ontology in the main branch of the source code.
There are two tests the first one tests the consistency of the ontology; the second one testing whether there are unsatisfiable classes.

Check consistency of ontology	testOntologyConsistency()	Check whether ontological reasoner returns true for `isConsistent()` call.
Check for unsatifiable classes	testUnsatisfiableClasses()	Check the number of classes which are classified as unsatisfiable by ontological reasoner function call `getUnsatisfiableClasses()` is zero.

5.2 Web service tests

The tests of the web service are run against a stable ontology in the test branch to make sure that results do not change with the further development of the ontology. Tests are done on the functions of the service where the returned values are compared with expected ones.

Get classes from ontology	getOwlClassTest()	Correct input; check against expected output
	getOwlClassTestWrong()	Wrong input; check against expected empty result set
	getOwlClassTestWrong2()	`null` input; check against expected empty result set
Get related classes from	getRelatedclasses()	Correct input; check against

ontology		expected output
	getRelatedWrong()	Wrong input; check against expected empty result set
	getRelatedWrong2()	`null` input; check against expected empty result set
Get equivalent classes from ontology	getEquivalents()	Correct input; check against expected output
	getEquivalentsWrong()	Wrong input; check against expected empty result set
	getEquivalentsWrong2()	`null` input; check against expected empty result set
Get event catalogue names from Ontology	getEventCatalogues()	Correct input; check against expected output
	getEventCataloguesWrong()	Wrong input; check against expected empty result set
	getEventCataloguesWrong2()	`null` input; check against expected empty result set
	getEventCataloguesWrong3()	Input of a known class in ontology which is not a HEC catalogue; check against expected empty result set

6 Maintaining & Extending the Service

Maintaining and extending the service can be split into two sections. The first one is how to maintain and extend the ontology the second is how to do this for the web service.

6.1 Maintaining & Extending the Ontology

To maintain and extend the ontology you should use an ontology editor such as Protegé (http://protege.stanford.edu/). Do not change the names of existing relationships since this might break the existing web service functions. Always apply consistency checks and reason over the ontology before moving modified ontology into the web service.

6.1.1 Add new HEC catalogues

Catalogues are part of the Organisational Ontology. To add a new HEC catalogue you need to add a new class as a child to the class 'Catalog' and annotate this class with the annotation properties:

- HELIOIdentifier with the value of the name of the catalogue as used in the HEC
- HELIOService with the value 'hec'
- dc:description should get the value of a short description of the content of the new catalogue

The new catalogue needs also to be bound to one or more physical phenomena. For this you add each a 'Superclass' with the correct relationship (use only classes which are child classes of 'Phenomenon'), i.e. '`containsDataAboutPhenomenon some CoronalMassEjection`'

6.1.2 Extend the Ontology further

You can introduce new classes and new relationships between classes. If these should be accessible via the web service you need to modify the web service as well. Make sure all additions are properly annotated and do not lead to inconsistencies in the ontology.

6.2 Maintaining & Extending the Web Service

The class '`SMSImpl`' contains the implementation of the web service as defined in the WSDL of the service. The main functionality is implemented in the class '`QueryOntology`' in the ontology directory of the source code. Problems with the code should be fixed in this class.

Another maintaining task would be the upgrading of the ontology libraries used (owlapi, HermiT). For this you modify the '`pom.xml`' in the 'helio-sms' directory by changing the version number to the latest and rebuilt (including a Maven clean). Test thoroughly and also check the log files for warning/error messages.

If you intent to extend the functionality by providing additional web service functions you should follow your own implementation preferences either writing the Java code and generating the WSDL from that or defining the WSDL and generating the Java. Make sure you add also suitable unit test for your new functionality and add Java Doc tags to the source code.