Contract No.	Health-F5-2008-200787
Document Type:	Software Documentation for Developers
WP/Task:	WP4/??
Document ID:	OpenTox-Documentation
Version:	0.1
Date:	September 25th, 2010
Status:	Draft Status

Organisation:	TUM
Authors:	Tobias Girschick (TUM) Fabian Buchwald (TUM)

Distribution:	Partnership and Development area of website

Purpose of Document:	Better documentation and easier implementation of new OT web services for internal and external developers

Document History:	1 – Produced by Tobias Girschick [TG] (TUM)
	2 – Corrections by Fabian Buchwald (TUM) 3 – TG finished first draft version

1 Overview

This document is intended for developers as a guideline - not a step-by-step programming tutorial - to set up an OpenTox [HAR10] descriptor calculation web service similar to the Free Tree Miner (FTM) web service provided through OpenTox by the Technical University of Munich. Basic understanding of REST web services [REST] and of the Java programming language is a prerequisite to understand and use this document. Documentation on usage of the FTM service can be found in TUM's OpenTox wiki at: http://opentox.informatik.tu-muenchen.de/trac/TUMOpenTox/wiki/FTM. A useful tool to test REST-based web services is the open source command line tool curl [CURL].

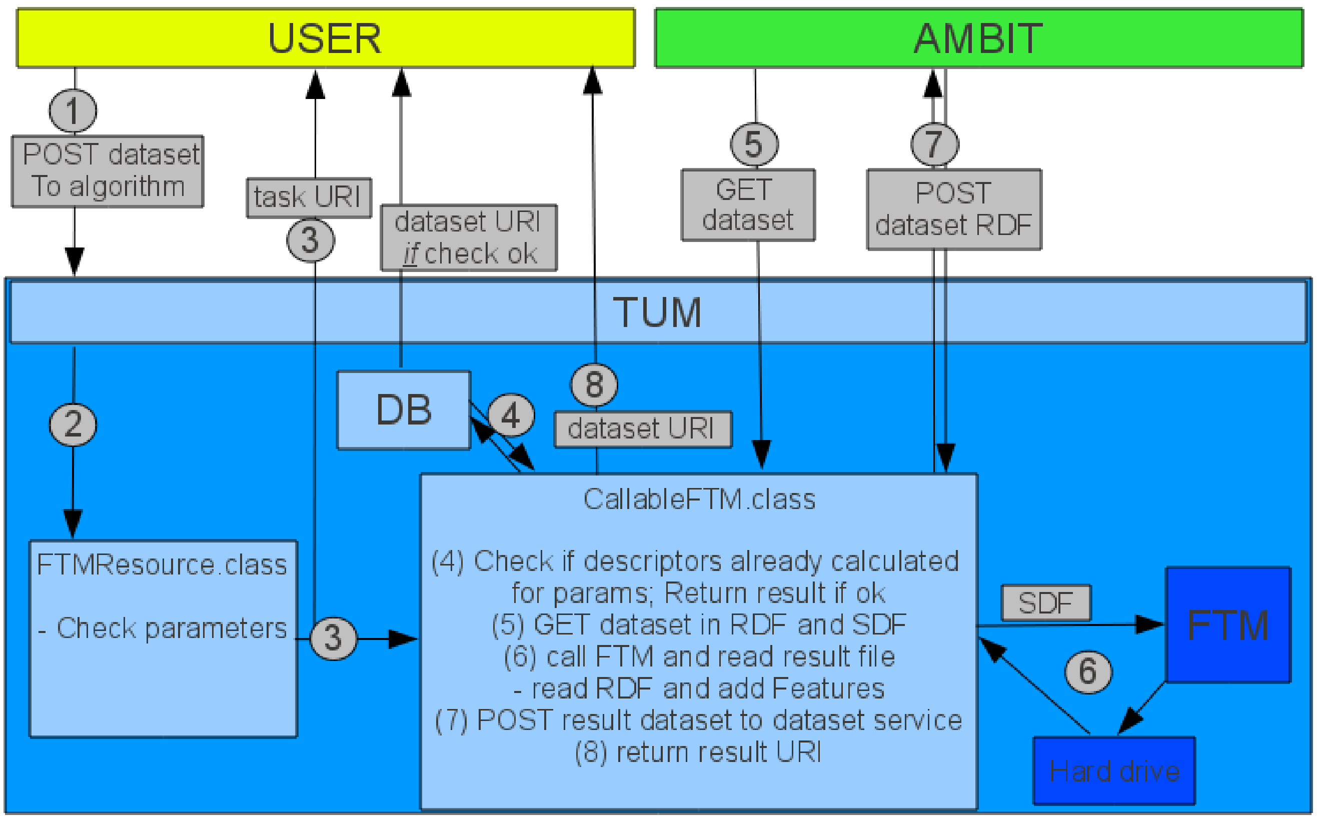
2 Step-by-step

This documentation will take a look at what is happening behind the scenes of a POST call to the FTM OpenTox descriptor calculation web service in a step-by-step manner. An example curl call to issue such a POST is the following:

```
curl -X POST -d "dataset_uri=http://ambit.uni-plovdiv.bg:8080/ambit2/dataset/6" -d
"minSup=0.95" http://opentox.informatik.tu-muenchen.de:8080/OpenTox-
dev/algorithm/FTM
```

The FTM [RUE04] OpenTox web service is a descriptor calculation service that mines an input dataset for frequently occurring sub-trees of the molecular graph structure. Consequently, the produced descriptor set is dataset dependent. Therefore, the dataset URI of the input dataset is considered a parameter of the algorithm. The descriptor for one compound is a bit-string of sub-tree occurrences. To be able to retrieve information on the “instance” of the FTM algorithm that has been used to calculate a specific descriptor set, the exact parameter configurations are stored in a MySQL database and a unique URI is assigned to the parameter setting, e.g., http://opentox.informatik.tu-muenchen.de:8080/OpenTox-dev/algorithm/FTM/3 for the third parameter configuration (including the input dataset URI and further parameters (defaults)) that has been used. Storing the parameter settings also prevents recalculations, as the result dataset URI is also stored.

Graphical overview


USER
AMBIT
1
POST dataset To algorithm
task URI
3
dataset URI if check ok
5
GET dataset
7
POST dataset RDF
TUM
2
DB
4
8
dataset URI
CallableFTM.class
(4) Check if descriptors already calculated for params; Return result if ok
(5) GET dataset in RDF and SDF
(6) call FTM and read result file
- read RDF and add Features
(7) POST result dataset to dataset service
(8) return result URI
FTMResource.class
- Check parameters
3
SDF
FTM
6
Hard drive

Steps:

(1) The user issues the POST method of the FTM algorithm, e.g., with a curl call like the following:

```
curl -X POST -d "dataset_uri=http://ambit.uni-plovdiv.bg:8080/ambit2/dataset/6" -d
"minSup=0.95" http://opentox.informatik.tu-muenchen.de:8080/OpenTox-
dev/algorithm/FTM
```

(2) The POST method is picked up by the web service application Java class and directed to the FTMResource class. There the parameters are checked for validity.

(3) If the parameters are valid a asynchronous task is created which is handled by the CallableFTM class. The user is returned a task URI. If not, an error is returned.

(4) Before doing any calculations the asynchronous task checks if the descriptors were already calculated for the given parameter setting and returns the result dataset URI if so. [This should be moved to the FTMResource class as no asynchronous calculations are necessary here.]

(5) Otherwise, the asynchronous job issues a GET call to the dataset URI which will be located, e.g., at the AMBIT server, to retrieve an SDF and an RDF [RDF] representation of the dataset.

(6) The dataset SDF file is then used as input for the FTM software with the according parameters. The FTM software writes its output to the harddrive. The asynchronous Java class then reads the results from the harddrive. Furthermore the task reads the dataset RDF. The calculated descriptors are then added to the RDF tree as features and feature values (see section 3).
(7) The updated (new features and values added) dataset is then POSTed to the dataset service and a task URI or a dataset URI is retrieved.
(8) This URI is then returned to the user, if he GETs the status of the task and the task is done with calculations.

3 Adding information to the RDF

The TUM web services use the JENA Java package [JENA] to handle RDF. This section provides some brief explanation of what is going on in the server side code. For initial information on JENA please refer to the JENA API introduction at: http://jena.sourceforge.net/tutorial/RDF_API/index.html.
The complete TUM OpenTox Java classes are available on the web at: http://opentox.informatik.tu-muenchen.de/trac/TUMOpenTox/browser/trunk/src/opentox. Most of the code below is taken from the class opentox/algorithm/descriptorcalculation/CallableFTM.java

Reading the RDF

```
/*
 * GET dataset in RDF and read RDF
 * The Util.getStreamfromURL issues a GET to the dataset URI with media  type RDF-XML
 */
InputStream rdf = Util.getStreamfromURL(uri.toString(), "application/rdf+xml");
OntModel oM = ModelFactory.createOntologyModel(OntModelSpec.OWL_DL_MEM, null);
		Map<String, String> prefixesMap = new HashMap<String, String>();
		prefixesMap.put("ot", NS);
		prefixesMap.put("owl", OWL.NS);
		prefixesMap.put("dc", DC.NS);
		prefixesMap.put("ota", OTA);
		prefixesMap.put("bo", BO);
		jenaModel.setNsPrefixes(prefixesMap);
oM.read(rdf, null);
```

The method OntModel.read() provided by JENA parses the RDF. The OntModel object can then be used for adding nodes. To access the dataset node and the information below we create an OntResource:

```
OntClass aClass2 = OT.OTClass.Dataset.getOntClass(oM);
if (aClass2 != null) {
		ExtendedIterator<? extends OntResource>  a = aClass2.listInstances();
		if (a!=null)
			while (a.hasNext()) {
				OntResource dataset = a.next();
[...]
```

Create a Feature

For every sub-tree a feature is created in the RDF dataset.

```
// Create feature (must be the same for all molecules!! only featureValues can change)
Individual feature = null;
feature = oM.createIndividual(OT.OTClass.Feature.getOntClass(oM));
feature.addProperty(DC.title, "TUM_FTM_" + tmp[0]);
feature.addProperty(OT.units, "count");
feature.addProperty(OWL.sameAs, "http://www.blueobelisk.org/ontologies/chemoinformatics-
algorithms/#subtree");

feature.addProperty(OT.isA, OT.OTA.concat("PatternMining"));
String encoded = URLEncoder.encode(tmp[0], "UTF-8");
feature.addProperty(OT.hasSource, hasSource_string+"/"+encoded);
```

After creating the features the FeatureValues have to be added. To ensure a mapping to the compounds the compounds have to be read from the dataEntries in a similar way to the dataset node above (http://opentox.informatik.tu-muenchen.de/trac/TUMOpenTox/browser/trunk/src/opentox/algorithm/descriptorcalculation/CallableFTM.java#L300). Finally, the updated information has to be added to the dataset node.

4 References

[CURL]	http://curl.haxx.se/
[HAR10]	Hardy, B., Douglas, N., Helma, C., Rautenberg, M., Jeliazkova, N., Jeliazkov, V.,Nikolova, I., Benigni, R., Tcheremenskaia, O., Kramer, S., Girschick, T., Buchwald,F., Wicker, J., Karwath, A., Gütlein, M., Maunz, A., Sarimveis, H., Melagraki, G.,Afantitis, A., Sopasakis, P., Gallagher, D., Poroikov, V., Filimonov, D., Zakharov, A., Lagunin, A., Gloriozova, T., Novikov, S., Skvortsova, N., Druzhilovsky, D., Chawla, S., Ghosh, I., Ray, S., Patel, H., Escher, S.: Collaborative Development of Predictive Toxicology Applications, accepted. Journal of Cheminformatics (2010)
[JENA]	http://jena.sourceforge.net/
[RDF]	http://www.w3.org/RDF/
[REST]	Fielding, R.: Architectural Styles and the Design of Network-based Software Architectures. PhD thesis, University of California, Irvine (2000)
[RUE04]	Rückert, U and Kramer, S., Frequent Free Tree Discovery in Graph Data, in: *SAC '04: Proceedings of the 2004 ACM Symposium on Applied Computing*, pp. 564-570 (New York, NY, USA: ACM Press, 2004).