Deliverable D1.1

Initial requirements, standards and APIs

Grant Agreement	Health-F5-2008-200787
Acronym	OpenTox
Name	An Open Source Predictive Toxicology Framework
Coordinator	Douglas Connect

Contract No.	Health-F5-2008-200787	
Document Type:	Deliverable Report	
WP/Task:	WP1 / D1.1	
Name	Initial requirements, standards and APIs	
Document ID:	OpenTox Deliverable Report WP1-D1.1	
Date:	Feb 28, 2009	
Status:	Final Version	
Organisation:	IST	
Contributors	Christoph Helma	IST
	Nicki Douglas	DC
	Barry Hardy	DC
	David Gallagher	DG
	Nina Jeliazkova	IDEA
	Stefan Kramer	TUM

Distribution:	Public

Purpose of Document:	To document results for this deliverable

Document History:	1 – First draft sent to all partners on 13 Feb 2009
	2 – Revised version 25 Feb 2009
	3 – Final version submitted 28 Feb 2009
	4 – Updated version with updated links 24 Feb 2010

Table of Contents

Table of Figures

1 Summary

A use-case questionnaire was created and sent to potential users of the OpenTox framework. Responses were collected and summarized.

Current standards that are relevant for OpenTox were collected in the developers' area of the OpenTox website (www.opentox.org/dev). This list will serve as a reference for OpenTox developers and will be continuously updated.

To gain a general overview, a list of existing software from OpenTox partners was collected in the developers' area. This list will evolve into an inventory of OpenTox components and will provide high level documentation and dependency tracking.

Extensive discussions about the architecture of the OpenTox framework were carried out on the OpenTox forums and the general agreement was that OpenTox will be a platform-independent collection of components that interact via well defined interfaces. The preferred form of communication between components will be through web services.

A set of minimum required functionalities for OpenTox components of various categories (prediction, descriptor calculation, data access, validation, report generation) was published on the developer web pages. These propositions are the subject of further discussions and revisions to create stable application programming interface definitions.

2 Introduction

OpenTox is an open source project and we are trying to follow the best practices of open source project management. This means that source code, technical discussions and documents are open to the general public and interested parties can participate in development if they have registered (http://opentox.org/join_form) for access to the OpenTox developers' area (http://opentox.org/dev)

Confidential information (e.g. non-anonymized responses to questionnaires, administrative documents) is available in the OpenTox partner area.

3 Evaluation of common use-cases for toxicological end users, data providers, (Q)SAR model developers and algorithm developers

A use-case questionnaire was created and sent to 20 potential users of the OpenTox framework. The list of the questions included as well as results received so far are summarized in Appendix A. Up to now there has been no general converging trend toward a specific type of use case among potential OpenTox users. This may be due to the relatively low number of responses received so far, but it may also indicate that we will need to provide a great flexibility with the OpenTox framework to meet individual requirements. To obtain a larger number of responses we are planning to extend the number of participants and to use web-based questionnaire forms for easier data entry and evaluation (http://www.opentox.org/toxicity-prediction/userinput).

4 Evaluation of current standards that are relevant for the OpenTox framework

Current standards that are relevant for OpenTox have been gathered and uploaded to the Framework Description page in the developers' area for ongoing reference. This list will serve as a reference for OpenTox developers and will be continuously updated.

The most important standards for OpenTox at the current state are ontology-related. The suitability of ToxML and IUCLID5 templates is currently being evaluated by the database Work Package (WP3); see the detailed discussion in the deliverable 3.1 report).

4.1 Standards that are relevant for OpenTox

- Minimum Information Standards for Biological Experiments (http://en.wikipedia.org/wiki/Minimum_Information_Standards)
- Toxicity Data
- Validation
- Algorithm Validation
- (Q)SAR Validation (Model Validation)
- Reports

4.2 Minimum Information Standards for Biological Experiments

(http://en.wikipedia.org/wiki/Minimum_Information_Standards)

Example standards and formats:

- Minimum Information for Biological and Biomedical Investigations (MIBBI) http://mibbi.org/index.php/Main_Page
- Functional Genomics Experiment (FuGE) http://fuge.sourceforge.net/
- MAGE http://www.mged.org/index.html, MIAPE http://www.psidev.info/index.php?q=node/91, ...
- Predictive Model Markup Language (PMML) http://www.dmg.org/pmml-v3-0.html

4.3 Toxicity Data

- DSSTox http://www.epa.gov/ncct/dsstox/
- ToxML http://www.leadscope.com/toxml.php
- PubChem http://pubchem.ncbi.nlm.nih.gov/
- OECD Harmonised Templates http://www.oecd.org/document/13/0,3343,en_2649_34365_36206733_1_1_1_1,00.html
- IUCLID5 templates http://iuclid.echa.europa.eu/index.php?fuseaction=home.format
- Standard for Exchange of Non-clinical Data (SEND) e.g., see http://www.cdisc.org/models/send/v2.3/index.html and http://www.pointcross.com/pharma/sendit.htm

4.4 Validation

4.4.1 Algorithm Validation

- common best practices such as k-fold cross validation, leave-one-out, scrambling

4.4.2 (Q)SAR Validation (Model Validation)

- OECD Principles http://www.oecd.org/dataoecd/33/37/37849783.pdf
- QSAR Model Reporting Format (QMRF) http://qsardb.jrc.it/qmrf/help.html
- QSAR Prediction Reporting Format (QPRF) http://ecb.jrc.it/qsar/qsar-tools/qrf/QPRF_version_1.1.pdf

4.5 Reports

- REACH Guidance on Information Requirements and Chemical Safety Assessment

  http://guidance.echa.europa.eu/docs/guidance_document/information_requirements_en.htm

- Part F - Chemicals Safety Report
  http://guidance.echa.europa.eu/docs/guidance_document/information_requirements_part_f_en.pdf?vers=30_07_08
- Appendix Part F
  http://guidance.echa.europa.eu/docs/guidance_document/information_requirements_appendix_part_f_en.pdf?vers=30_07_08

5 Initial specification of requirements and standards for the OpenTox framework

Existing software from OpenTox partners was collected in the developers' area (http://opentox.org/dev) on the documentation page under Components. This list will evolve into an inventory of OpenTox components and will provide high level documentation and dependency tracking. To date this list includes the following components which are documented in more detail on the website:

5.1.1 Prediction

Table 1 List of Current Prediction Components

Name	Component Description
Fuzzy Means	Fuzzy-means is a fast, one-pass training method for Radial Basis Function (RBF) neural networks and is based on the fuzzy partition of the input space.
Gaussian Processes for Regression	GPR (Gaussian Processes for Regression) is a way of supervised learning. A Gaussian process is a generalization of the Gaussian probability distribution.
iSar	Perl implementation of a lazy SAR algorithm
J48	Implementation of Quinlan's C4.5 algorithm for generating a pruned or unpruned C4.5 decision tree.
kNN	k-nearest neighbor algorithm (kNN), an instance-based, or lazy, learning method.
lazar	lazar command line program C++ implementation of various lazar algorithms
lazar web interface	Web interface for lazar
M5P	Reconstruction of Quinlan's M5 algorithm for inducing trees of regression models.
MaxTox	Comparing the query molecule to each cluster (EP based) and finding an MCS score with respect to molecules of each cluster. Using MCS score(s) in a Machine Learning algorithm, to generate predictive models.
Multiple Linear Regression	Simple and popular statistical technique, using several independent variables to predict the outcome of a dependent variable.
MakeSCR - Self-consistent Regression	Delphi implementation of a self-consistent regression (SCR) algorithm.
Partial-least Squares Regression	Partial-least squares regression (PLS) simultaneously projects the x and y variables onto the same subspace in such a way that there is a good relationship between the predictor and response data. It can thus handle correlated variables, which are noisy and possibly incomplete.
Rumble	RUMBLE (RUle and Margin Based LEarner) is a statistically motivated rule learning system based on the Margin Minus Variance (MMV) approach. It is set up very flexibly as it can make use of different plug-ins (e.g. FTM plugin, PROLOG plugin, Meta plugin) for different kinds of rules.
SMIREP/SMIPPER	SMIREP/SMIPPER is based on combining feature generation and rule learning into one integrated package. The underlying learning algorithm is similar to that of the IREP rule learner employing a reduced error pruning approach.

Support Vector Machines	Support vector machines (SVM) are a set of supervised learning methods used for classification and regression.
Toxmatch	Provides means to compare a chemical or set of chemicals to a toxicity dataset through the use of similarity indices.
Toxtree	Toxtree is a full-featured and flexible user-friendly open source application, which is able to estimate toxic hazard by applying a decision tree approach. Currently it includes five plugins.

5.1.2 Descriptor Calculation

Table 2: List of Current Descriptor Calculation Components

Name	Component Description
AMBIT	• a relational database database schema, allowing the storage and querying of all relevant structure and property information, including data for toxicity endpoints from various sources and formats. Can handle very large number of structures efficiently. • functional modules allowing a variety of evaluations, flexible structure, similarity and other information retrieval. Used in both standalone and web (servlets/taglibs based) applications.
Chemistry Development Kit	The Chemistry Development Kit (CDK) is a Java library for structural chemo- and bioinformatics. A number of descriptor implementations are available.
FreeTreeMiner	The FreeTreeMiner (FTM) software computes all subtrees (substructures) occurring at a given minimum frequency in a set of molecules. The subtrees are built via a depth first search (DFS). Additionally to the minimum frequency support, a maximum frequency constraint can be set.
LibFminer	LibFMiner implements a method for efficiently mining relevant tree-shaped molecular fragments, each representing a geometrical class, with minimum frequency and statistical constraints.
Toxtree	Toxtree is a full-featured and flexible user-friendly open source application, which is able to estimate toxic hazard by applying a decision tree approach.
gSpan'	C implementation of a graph mining algorithm – feature generation: Mining for frequent subgraphs or subpaths/subtrees
JOELib2	Platform independent Java package consisting of an algorithm library designed for prototyping, data mining and graph mining of chemical compounds. JOELib2 is the successor of the OELib library from OpenEye
lazar	lazar command line program C++ implementation of various lazar algorithms • feature generation (paths) • nearest neighbor and kernel classification and regression • local models

	• activity-specific similarities
MakeMNA	MakeMNA is a software product for generating MNA descriptors. These descriptors are based on the molecular structure representation, which includes the hydrogens according to the valences and partial charges of other atoms and does not specify the types of bonds.
MakeQNA	Quantitative Neighbourhoods of Atoms (QNA) descriptors are based on quantities of ionization potential (IP) and electron affinity (EA) of each atom of the molecule.
MaxTox	Comparing the query molecule to each cluster (EP based) and finding an MCS score with respect to molecules of each cluster. Using MCS score(s) in a Machine Learning algorithm, to generate predictive models.
MOPAC	MOPAC (Molecular Orbital PACkage) supports the methods: MNDO, AM1, and PM3, as well as Sparkle/AM1 for the lanthanides. All published NDDO parameter sets are supported.
OpenBabel	OpenBabel is an open source computational chemistry package written in C++.
ToxTree	Toxtree is a fully-featured and flexible user-friendly open source application, which is able to estimate toxic hazard by applying a decision tree approach.
MakeSCR	Delphi implementation of a self-consistent regression (SCR) algorithm.

5.1.3 Feature Selection

Table 3: List of Current Feature Selection Components

Name	Component Description
CFS	CFS is a correlation-based filter method, giving high scores to subsets that include features that are highly correlated to the class attribute, but have a low correlation to each other.
Chi Square	Feature Selection via the chi square (X2) test is a commonly used method. The X2 method evaluates features individually by measuring their chi-squared statistic with respect to the classes.
Fast Correlation-Based Filter	Two-stage algorithm: 1) relevance analysis, 2) redundancy analysis
Information Gain Attribute Evaluation	Information Gain Attribute Evaluation evaluates the worth of an attribute by measuring the information gain with respect to the class.
Principle Component Analysis	Transforms the data to a new coordinate system such that the greatest variance by any projection of the data comes to lie on the first coordinate, the second greatest variance on the second coordinate and so forth. The coordinates are here called principal components.
Wrapper Feature Set Evaluation	Wrapper methods evaluate subsets by running the classifier on the training data, using only the attributes of the subset.

5.1.4 Data Access

Table 4: List of Current Data Access Components

Name	Component Description
AMBIT	relational database schema, allowing user to store and query all relevant structure and property information, including data for toxicity endpoints from various sources and formats. Can handle very large number of structures efficiently.
DSSTox data for lazar	Git repositories for versioned DSSTox sdf files, conversion scripts to generate lazar input files, validation results
Sens-it-iv internal database	Internal database for the Sens-it-iv http://www.sens-it-iv.eu FP6 project
Toxmatch	Provides means to compare a chemical or set of chemicals to a toxicity dataset through the use of similarity indices. Intended use is one to many or many to many quantitative read-across. To help in the systematic formation of groups and read-across.

5.1.5 Report Generation

Table 5: List of Current Report Generation Components

Name	Component Description
AMBIT	• Recording of user actions • Improved data entrance and visualization • Reporting compatible with IUCLID 5
Lazar web interface	Report Generation
CDK Structure Visualizer	Web service for structure visualization and highlighting of substructures.

5.1.6 Validation

Table 6: List of Current Validation Components

Name	Component Description
lazar	**leave-one-out validation** *Input:* chemical structures and activities *Output:* actual vs. predicted values, validation statistics

5.1.7 Integration

Table 7: List of Current Integration Components

Name	Component Description
OpenTox plugin	Ruby on Rails plugin with interfaces to R, OpenBabel, CDK and basic functionality to create predictive toxicology applications.
lazar plugin	Ruby on Rails plugin with interfaces for the lazar command line program • Web based GUI • rake tasks for administration and validation
Lazar web interface	Web interface for lazar

5.2 Architecture

Extensive discussions about the architecture of the OpenTox framework were carried out on the OpenTox forums. The consensus outcome agreement was that OpenTox will be a platform-independent collection of components that interact via well defined interfaces. The preferred form of communication between components will be through web services. An initial description of the framework, that contains also a list of minimum requirements for OpenTox components, has been posted in the developers' area (http://opentox.org/dev).

OpenTox is a framework for the integration of algorithms for predicting chemical toxicity. OpenTox will provide:

- components for specialized tasks (e.g. database lookups, descriptor calculation, classification, regression, report generation) that communicate through well defined language independent interfaces
- example applications that demonstrate the capabilities of OpenTox components for special use cases

The framework supports building multiple applications, as well as providing components for third party applications.

The framework guarantees the portability of components by enforcing language-independent interfaces. Implementation of an integration component in a specific language/platform automatically ports the entire OpenTox framework to that language/platform.

Components are presently classified under the following categories:

- Prediction
- Descriptor calculation
- Feature Selection
- Data access
- Validation
- Report generation
- Integration

5.2.1 System overview

The OpenTox **framework** is composed of

- **Components**. Every component encapsulates a set of functionalities and exposes them via well defined language-independent interfaces (protocols)
- **Data**
- **Repository**

An **application** implements a set of use cases, with the appropriate user interfaces.

The **interactions between components** are determined by their intended use and can differ across different use cases. Use cases represent user stories, or typical uses of the system by various types of users. Each use case consists of a series of steps, applying component functionality on input data.

The interaction between components is implemented as a component. The interaction component offers the following functionalities:

- loads the series of steps, corresponding to the specific use case (from a configuration file on a file system or on a network)
- takes care of loading necessary components
- executes the steps

The framework supports building multiple applications, as well as using the components in third party applications.

The framework guarantees portability of components by enforcing a language-independent architecture of the integration component and externalizing user scenarios in standard configuration files (e.g. xml or txt). The implementation of the integration component in a specific language/platform automatically ports the entire OpenTox framework to that language/platform. It would be desirable to prove the portability of the platform by producing implementations in at least two different languages. We should also aim at providing detailed framework specifications and guidelines, enabling other parties to port and tailor the framework to their specific environment and thus further enrich OpenTox's ecosystem.

5.2.2 Main description

This section provides an overview of the OpenTox framework, listing the elements that constitute the framework and relationships between them.

Table 8 lists OpenTox components, where the column “Component” is the generic component, exposing defined set of functionalities, while the second column lists the specific implementations of the component, available in the framework.

Table 8. OpenTox components

Component	**Instances**
(Q)SAR algorithm	
	(Q)SAR algorithm 1
	(Q)SAR algorithm 2
	…
(Q)SAR model validation	
	Validation algorithm 1
	Validation algorithm 2
	…
(Q)SAR descriptor calculations	
	Descriptor 1
	Descriptor 2
	…
(Q)SAR feature selection	

	Selector 1
	Selector 2
	…
Data access module	
	Data access 1 (e.g. file)
	Data access 2 (e.g. database)
	Data access 3 (e.g. PubChem)
	…
Report generation	
	Report generation format 1
	Report generation format 2
	…
Error handling and reporting	
Ontology/Dictionaries	
	Endpoints
	Descriptors?
	Species?
	Units?
	…?

Table 9 lists the functionalities, exposed by each OpenTox component, where the column "Component" is the name of the generic component, and the second column lists the specific operations, offered by the component.

Table 9. Functionalities (operations), supported by each component

Component	**Operations**
(Q)SAR algorithms	
	Build
	Predict
	…
(Q)SAR model validation	
	Validate
	Get statistics
	…

(Q)SAR descriptor calculations	
	Set parameters
	Calculate
	…
(Q)SAR feature selection	
	Set parameters
	Select
Data access module	
	Set query
	Retrieve data
	…
Report generation	
	Select report type
	Generate report
	…
Error handling and reporting	
	Get error message (user friendly/detailed)
Ontology/Dictionaries	
	Endpoints – get fields, defined for an endpoint
	Descriptors – get implementations of (e.g. LogP) descriptor
	Species – Latin name, common name
	Units – units conversion
	…?

Table 10 defines the steps, which constitute a Use Case. Each step (column 2) is an operation, exposed by a component (column 3).

Table 10: Use cases

Use case	**Steps (operations)**	**Component**
Use case 1 (very simple example)		
	1. Retrieve data	Data access
	2. Calculate descriptors	(Q)SAR Descriptor calculations
	3. Build QSAR model	(Q)SAR Algorithms
	4. Validate the model	(Q)SAR model validation
	5. Generate report	Report generation
	…	
Use case 2		
	Step1	
	Step 2	
	…	

Table 11 describes the applications and specific use cases that they solve.

Table 11: Applications and Use Cases Implemented

Application	**Use cases implemented**
Application 1	
	Use case 1
	Use case 2
	…
Application 2	
	Use case 3
	Use case 4
	…

5.2.3 User Interface

We have asked the question of shall we use a common user interface for each operation?

Advantages:

The above structure results in a layered view (system portability). Higher levels would be allowed to use only functionalities, which are provided by adjacent lower levels. This would help to ensure an implementation-independent protocol stack.

Layer	Level
Application	High
Use case	
Component	
Component operation	Low

Table 12 defines which component is allowed to use functionality by other components.

Table 12. Relationships between components

Using components: The component X …	… is allowed to use any functionality in component Y
(Q)SAR algorithms	(Q)SAR descriptor calculation
(Q)SAR model validation	(Q)SAR algorithms
(Q)SAR descriptor calculation	none
Data access	Ontology/Dictionaries
Report generation	none
Ontology/Dictionaries	None
….	…

5.2.3.1 Data flow view

A data flow view defines how data is processed through the set of operations. It can be specific for each use case and will be defined once detailed use cases are prepared.

5.3 Element Catalog

5.3.1 Elements and their properties

(Q)SAR algorithms. The QSAR algorithms module includes implementation of the relevant algorithms, selected for the OpenTox framework, and provides a unified view of a (Q)SAR algorithm to other modules. The unified view serves as an information hiding and allows algorithms to be easily added/replaced.

Operations – build model, predict chemical compound, get statistics, etc.

Input/Output to be defined

(Q)SAR model validation. The (Q)SAR model validation module includes implementation of the validation elements, selected for the OpenTox framework, and provides a unified view of a validation procedure to other modules. The unified view serves as an information hiding and allows validation algorithms to be easily added/replaced.

Operations – to be defined (e.g. a model as an input, validation statistics as an output)

Input/Output to be defined

Data access module. The data access module hides specifics of data formats and underlying storage mechanisms.

Operations: Retrieve (named) dataset, given some query options. There could be mandatory and optional operations.

Examples:

- Retrieve DSSTOX carcinogenicity dataset version XXX.

- Retrieve all available data for compound with CAS# = YYY-YY-YY

- Retrieve aromatic amines with all data available for endpoint ZZZ.

Input = query, dataset name

Output = set of compounds and related data

(to be refined)

Ontology/dictionary. This module provides a controlled vocabulary necessary for the unified view on the data access and is used by the data access module.

Report generation. This module implements various reporting formats of interest to the end user. Externalizing report generation in a separate module facilitates meeting requirements of different use cases and supporting new types of reports.

Operations: Generate report of type XX, given dataset Y.

(to be refined)

Use cases. A use case is an ordered set of operations from different modules. Use cases are defined by user requirements.

In addition, if there are elements or relations relevant to the view that were omitted from the primary presentation, the catalog is where those are introduced and explained.

5.3.2 Element interfaces

An interface is a boundary across which two independent entities meet and interact or communicate with each other. Documenting an interface consists of naming and identifying it and documenting its syntactic and semantic information. The first two parts constitute an interface's "signature." When an interface's resources are invokable by programs, the signature names the programs and defines their parameters. Parameters are defined by their order, data type, and (sometimes) whether or not their value is changed by the program. A signature is the information that you would find about the program, for instance, in an element's C or C++ header file or in a Java interface.

An interface is documented with an interface specification, which is a statement of element properties the architect chooses to make known. The architect should expose only what is needed to interact with the interface.

A Template for Documenting OpenTox Interfaces

5.3.3 Interface identity

When an element has multiple interfaces, identify the individual interfaces to distinguish them. This usually means naming them. You may also need to provide a version number.

5.3.3.1 Resources provided

The heart of an interface document is the resources that the element provides.

Syntax, semantics (what happens when they are used), and any restrictions on usage are to be included.

5.3.3.2 Resource syntax

Resource name, names and logical data types of arguments (if any), and so forth are described.

5.3.3.3 Resource semantics

This describes the result of invoking the resource. It might include:

- assignment of values to data that the actor invoking the resource can access. It might be as simple as setting the value of a return argument or as far-reaching as updating a central database.
- events that will be signalled or messages that will be sent as a result of using the resource.
- how other resources will behave in the future as the result of using this resource.
- humanly observable results (display)

5.3.3.4 Resource usage restrictions

Under what circumstances may this resource be used? (data initialization, number of actors interacting with the resource, access rights, etc.)

5.3.4 Data type definitions

Data type definitions will need to be defined

5.3.5 Exception definitions

Exceptions that can be raised by the resources on the interface will be described.

5.3.6 Variability provided by the interface

Does the interface allow the element to be configured in some way?

5.3.7 Quality attribute characteristics of the interface

Description of quality attribute characteristics (such as performance or reliability).

5.3.8 Element requirements

Specific, named resources provided by other elements.

5.3.9 Rationale and design issues

Motivation behind the design, constraints and compromises, what alternative designs were considered and rejected (and why), and any insight about how to change the interface in the future.

5.3.10 Usage guide

Protocols used.

5.3.10.1 Element behaviour

Sequence of events; sequence diagram.

5.4 Context diagram

Shows how the system depicted relates to its environment.

Shows which component and connectors interact with external components and connectors, and via which interfaces and protocols.

5.5 Variability guide

Lists decisions which are left unbound:

- the options, among which the choice is to be made (versions, parameterization of components)
- choice of protocols
- Ontology/Dictionaries content to be defined
- Component operations to be defined

5.6 Architecture background

5.6.1 Design rationale

- Encapsulate functionality of components
- Facilitate addition / replacement of compatible components (e.g. QSAR Algorithm N can be easily added to the pool of algorithms, since all Algorithms expose the same interface)
- More to be added

5.6.2 Analysis of results

Module decomposition serves as a basis to achieve the following quality goals:

Table 13: Quality goals from module decomposition

Goal	Achieved by
Ease of change to: (Q)SAR algorithms, validation procedures, data access, report generation	Information hiding
Understand anticipated changes	Evaluation procedure to take advantage of experience of domain experts
Assign work teams so that their interactions were minimized	Modules structured as a hierarchy; each work team assigned to a second-level module and all of its descendants

Uses structure provides a basis to achieve the following quality goals:

Table 14: Quality goals from Uses Structure

Goal	Achieved by
Incrementally build and test modules	Create "is-allowed-to-use" structure for programmers that limits module procedures each can use
Design for platform change	Modules communicate in language and platform independent way
Produce usage guidance of manageable size	Where appropriate, define uses to be a relationship among modules

5.6.2.1 Assumptions

Documentation on assumptions will need to be developed.

5.7 Glossary of terms

Terms used in the views, with a brief description of each, will be provided.

5.8 Other information

Any additional information will be provided under this sub-section.

6 Definition of APIs for the database, algorithm and validation interface

A set of minimum required functionalities for all OpenTox components of various categories (prediction, descriptor calculation, data access, validation, report generation) has been determined and is listed on the Documentation page in the developers' area (http://opentox.org/dev). However, it is possible that there may be additions to this list in future. Where individual use cases need further functionalities, these will be addressed directly by the component developer.

Required functionality for all OpenTox components

Prediction

create model *not applicable in all cases (e.g. expert systems), but required for validation*
Input training structures, training activities
Output prediction model

predict
Input chemical structure, prediction model
Output prediction, confidence, [supporting information]

Descriptor calculation

calculate
Input chemical structure, property
Output descriptor[s]

Data access

create
Input new data

update
Input modified data

query
Input chemical structure, endpoint
Output experimental measurement[s]

delete
Input ID

Validation

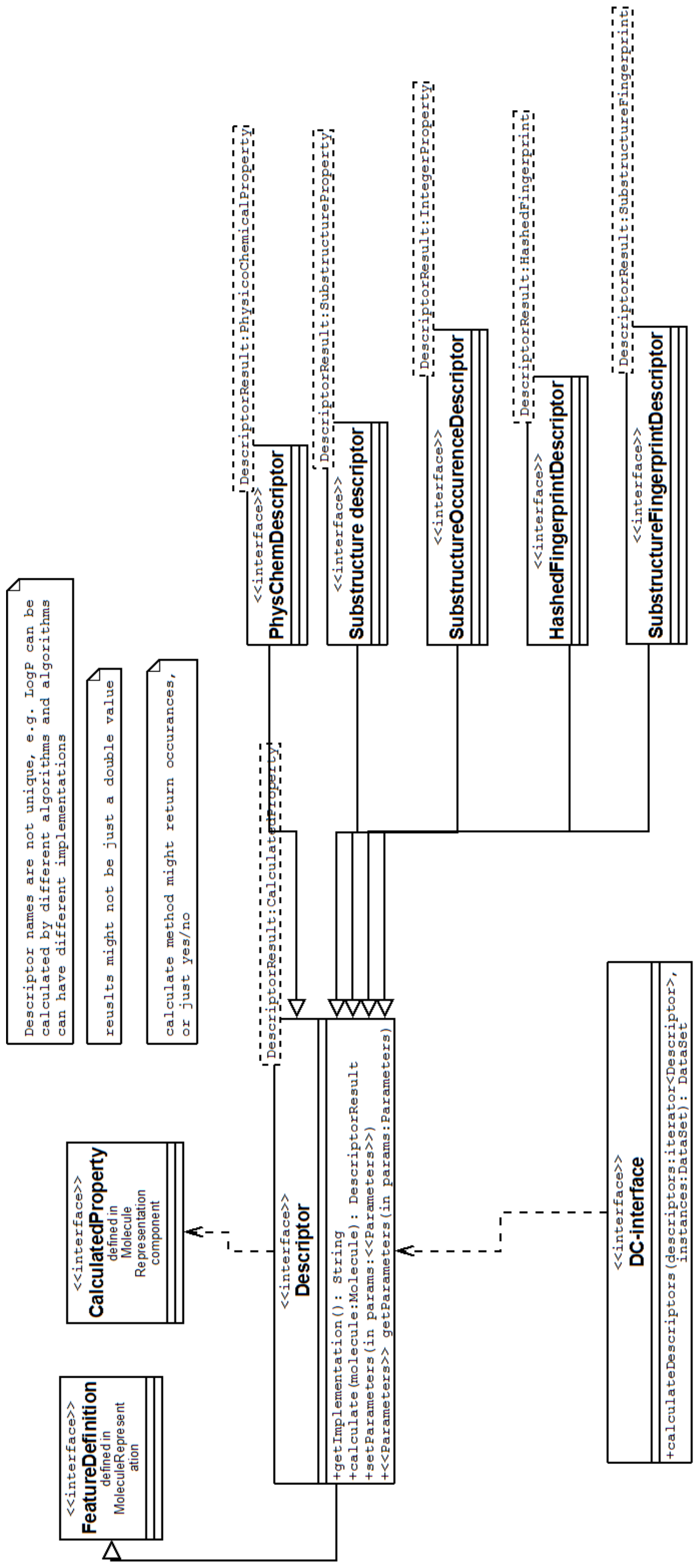
validate
Input prediction_model, validation_method
Output validation statistics, [supporting information]

Report generation

create report
Input data, report type
Output report

Draft class diagram proposals that define interfaces for OpenTox components were created and are presented in the Figures below for Descriptor, Modelling, Similarity, Data Access, Feature Selection and Molecule Representation Components.

Figure 1: Descriptor Calculation Component



Descriptor names are not unique, e.g. LogP can be calculated by different algorithms and algorithms can have different implementations
reuslts might not be just a double value
calculate method might return occurances, or just yes/no
DescriptorResult:PhysicoChemicalProperty
<<interface>> PhysChemDescriptor
DescriptorResult:SubstructureProperty
<<interface>> Substructure descriptor
DescriptorResult:IntegerProperty
<<interface>> SubstructureOccurenceDescriptor
DescriptorResult:HashedFingerprint
<<interface>> HashedFingerprintDescriptor
DescriptorResult:SubstructureFingerprint
<<interface>> SubstructureFingerprintDescriptor
DescriptorResult:CalculatedProperty
<<interface>> Descriptor
+getImplementation(): String
+calculate(molecule:Molecule): DescriptorResult
+setParameters(in params:<<Parameters>>)
+<<Parameters>> getParameters(in params:Parameters)
<<interface>> CalculatedProperty defined in Molecule Representation component
<<interface>> FeatureDefinition defined in MoleculeRepresentation
<<interface>> DC-interface
+calculateDescriptors(descriptors:iterator<Descriptor>, instances:DataSet): DataSet

Figure 2: Modeling Component

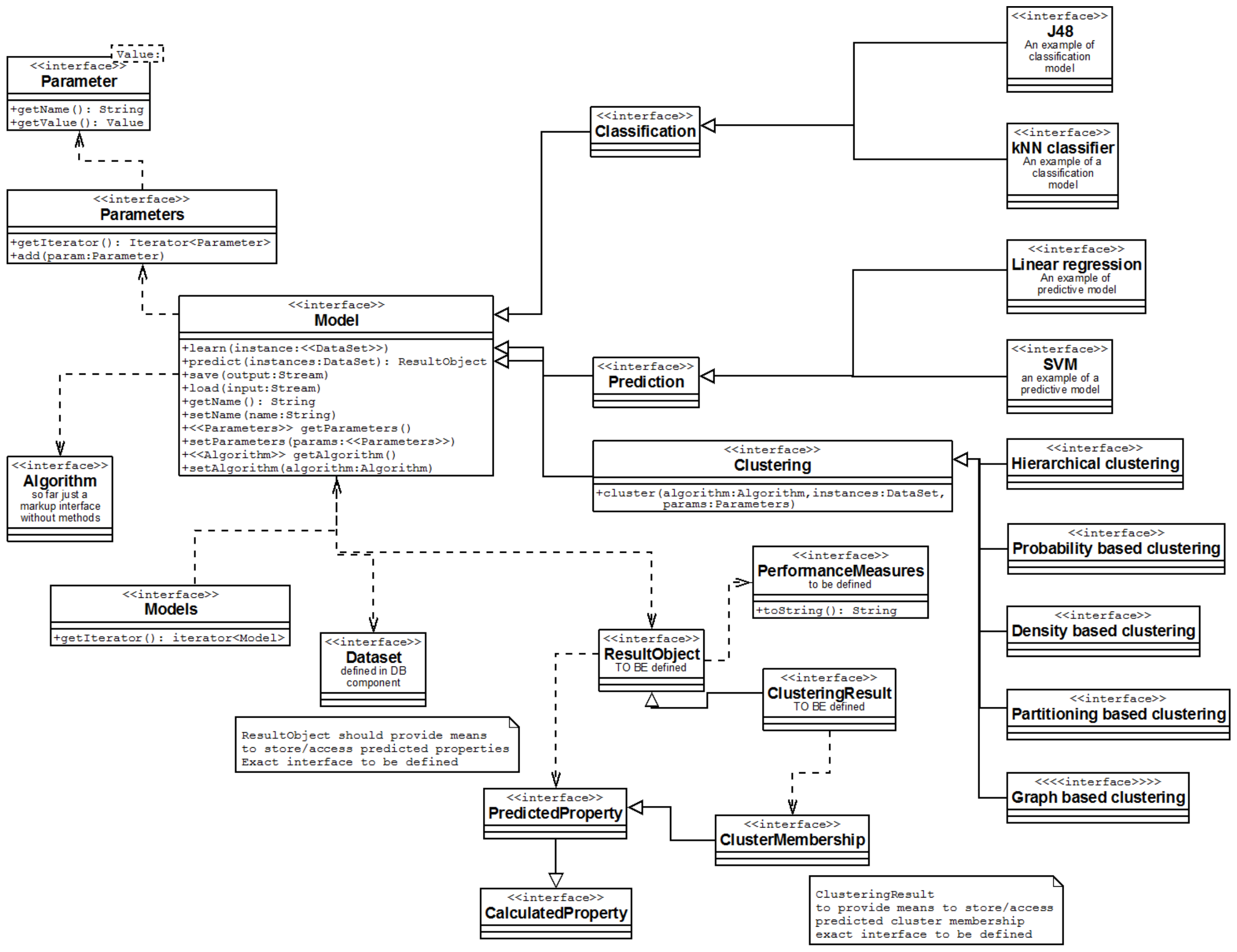
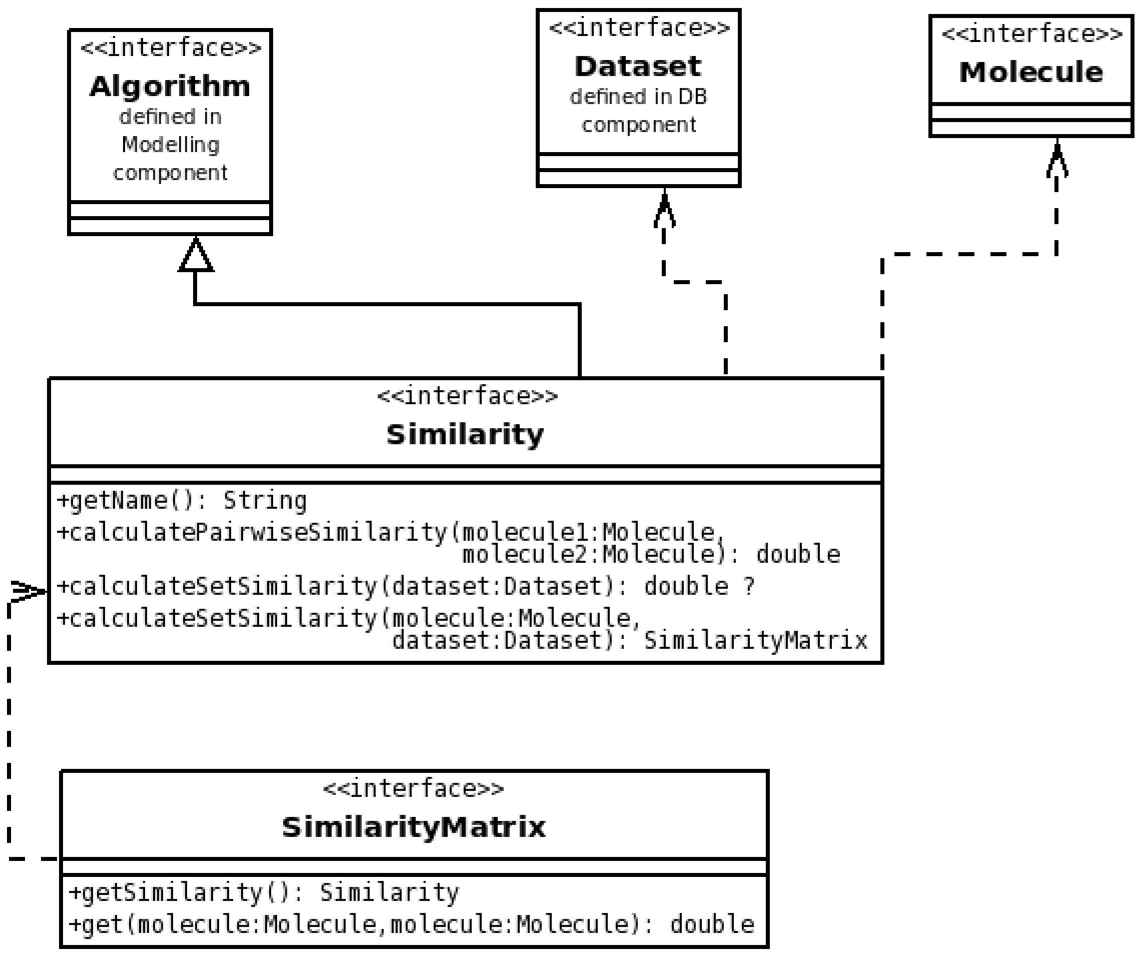
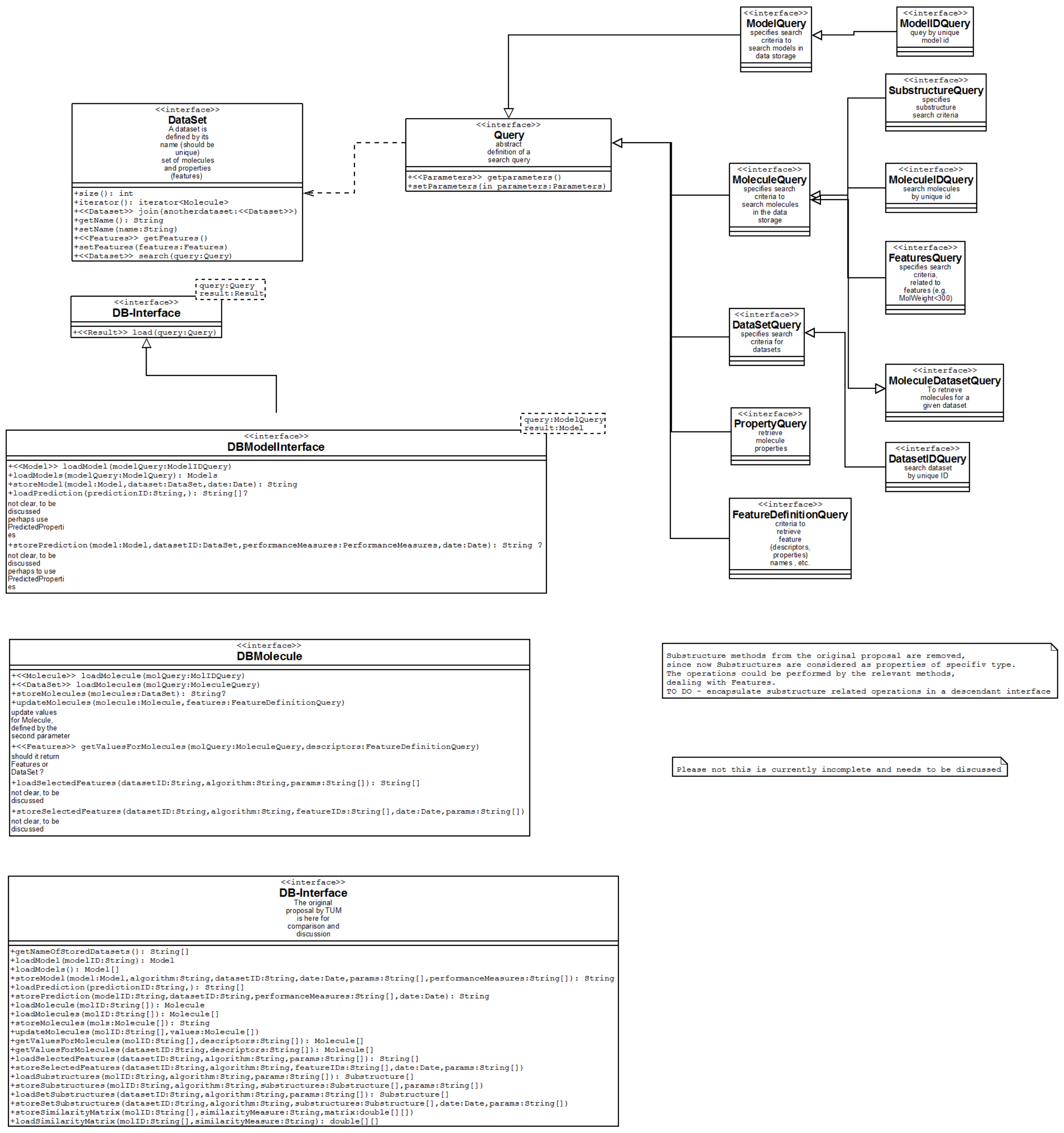


Figure 3: Similarity Component



<<interface>>
Algorithm
defined in Modelling component
<<interface>>
Dataset
defined in DB component
<<interface>>
Molecule
<<interface>>
Similarity
+getName(): String
+calculatePairwiseSimilarity(molecule1:Molecule, molecule2:Molecule): double
+calculateSetSimilarity(dataset:Dataset): double ?
+calculateSetSimilarity(molecule:Molecule, dataset:Dataset): SimilarityMatrix
<<interface>>
SimilarityMatrix
+getSimilarity(): Similarity
+get(molecule:Molecule,molecule:Molecule): double

Figure 4: Data Access Component


<<interface>>
ModelQuery
specifies search criteria to search models in data storage
<<interface>>
ModelIDQuery
quey by unique model id
<<interface>>
SubstructureQuery
specifies substructure search criteria
<<interface>>
Query
abstract definition of a search query
<<interface>>
MoleculeQuery
specifies search criteria to search molecules in the data storage
<<interface>>
MoleculeIDQuery
search molecules by unique id
<<interface>>
FeaturesQuery
specifies search criteria, related to features (e.g. MolWeight<300)
<<interface>>
DataSetQuery
specifies search criteria for datasets
<<interface>>
MoleculeDatasetQuery
To retrieve molecules for a given dataset
<<interface>>
PropertyQuery
retrieve molecule properties
<<interface>>
DatasetIDQuery
search dataset by unique ID
<<interface>>
FeatureDefinitionQuery
criteria to retrieve feature (descriptors, properties) names , etc.
<<interface>>
DataSet
A dataset is defined by its name (should be unique) set of molecules and properties (features)
<<interface>>
DB-Interface
query:Query
result:Result
<<interface>>
DBModelInterface
query:ModelQuery
result:Model
<<interface>>
DBMolecule
<<interface>>
DB-Interface
The original proposal by TUM is here for comparison and discussion

```
+<<Parameters>> getparameters()
+setParameters(in parameters:Parameters)
```

```
+size(): int
+iterator(): iterator<Molecule>
+<<Dataset>> join(anotherdataset:<<Dataset>>)
+getName(): String
+setName(name:String)
+<<Features>> getFeatures()
+setFeatures(features:Features)
+<<Dataset>> search(query:Query)
```

```
+<<Result>> load(query:Query)
```

```
+<<Model>> loadModel(modelQuery:ModelIDQuery)
+loadModels(modelQuery:ModelQuery): Models
+storeModel(model:Model,dataset:DataSet,date:Date): String
+loadPrediction(predictionID:String,): String[]?
```
not clear, to be discussed perhaps use PredictedProperties
```
+storePrediction(model:Model,datasetID:DataSet,performanceMeasures:PerformanceMeasures,date:Date): String ?
```
not clear, to be discussed perhaps to use PredictedProperties

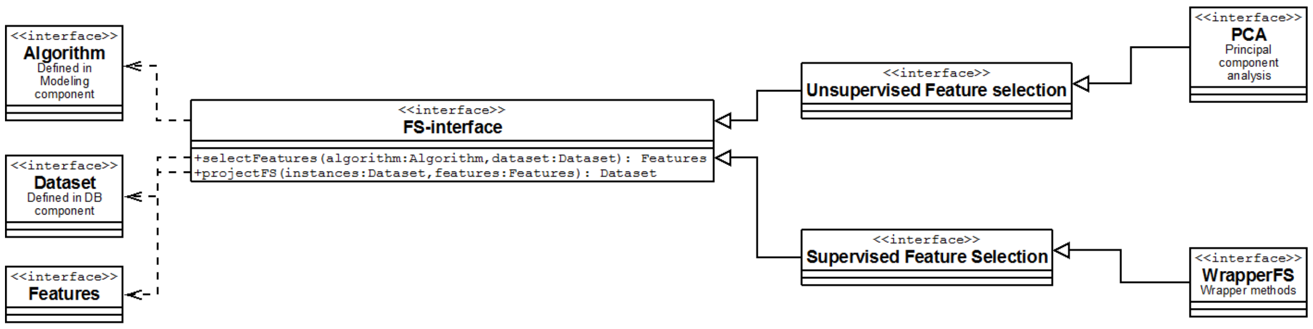
```
+<<Molecule>> loadMolecule(molQuery:MolIDQuery)
+<<DataSet>> loadMolecules(molQuery:MoleculeQuery)
+storeMolecules(molecules:DataSet): String?
+updateMolecules(molecule:Molecule,features:FeatureDefinitionQuery)
```
update values for Molecule, defined by the second parameter
```
+<<Features>> getValuesForMolecules(molQuery:MoleculeQuery,descriptors:FeatureDefinitionQuery)
```
should it return Features or DataSet ?
```
+loadSelectedFeatures(datasetID:String,algorithm:String,params:String[]): String[]
```
not clear, to be discussed
```
+storeSelectedFeatures(datasetID:String,algorithm:String,featureIDs:String[],date:Date,params:String[])
```
not clear, to be discussed

```
+getNameOfStoredDatasets(): String[]
+loadModel(modelID:String): Model
+loadModels(): Model[]
+storeModel(model:Model,algorithm:String,datasetID:String,date:Date,params:String[],performanceMeasures:String[]): String
+loadPrediction(predictionID:String,): String[]
+storePrediction(modelID:String,datasetID:String,performanceMeasures:String[],date:Date): String
+loadMolecule(molID:String[]): Molecule
+loadMolecules(molID:String[]): Molecule[]
+storeMolecules(mols:Molecule[]): String
+updateMolecules(molID:String[],values:Molecule[])
+getValuesForMolecules(molID:String[],descriptors:String[]): Molecule[]
+getValuesForMolecules(datasetID:String,descriptors:String[]): Molecule[]
+loadSelectedFeatures(datasetID:String,algorithm:String,params:String[]): String[]
+storeSelectedFeatures(datasetID:String,algorithm:String,featureIDs:String[],date:Date,params:String[])
+loadSubstructures(molID:String,algorithm:String,params:String[]): Substructure[]
+storeSubstructures(molID:String,algorithm:String,substructures:Substructure[],params:String[])
+loadSetSubstructures(datasetID:String,algorithm:String,params:String[]): Substructure[]
+storeSetSubstructures(datasetID:String,algorithm:String,substructures:Substructure[],date:Date,params:String[])
+storeSimilarityMatrix(molID:String[],similarityMeasure:String,matrix:double[][])
+loadSimilarityMatrix(molID:String[],similarityMeasure:String): double[][]
```

```
Substructure methods from the original proposal are removed,
since now Substructures are considered as properties of specifiv type.
The operations could be performed by the relevant methods,
dealing with Features.
TO DO - encapsulate substructure related operations in a descendant interface
```

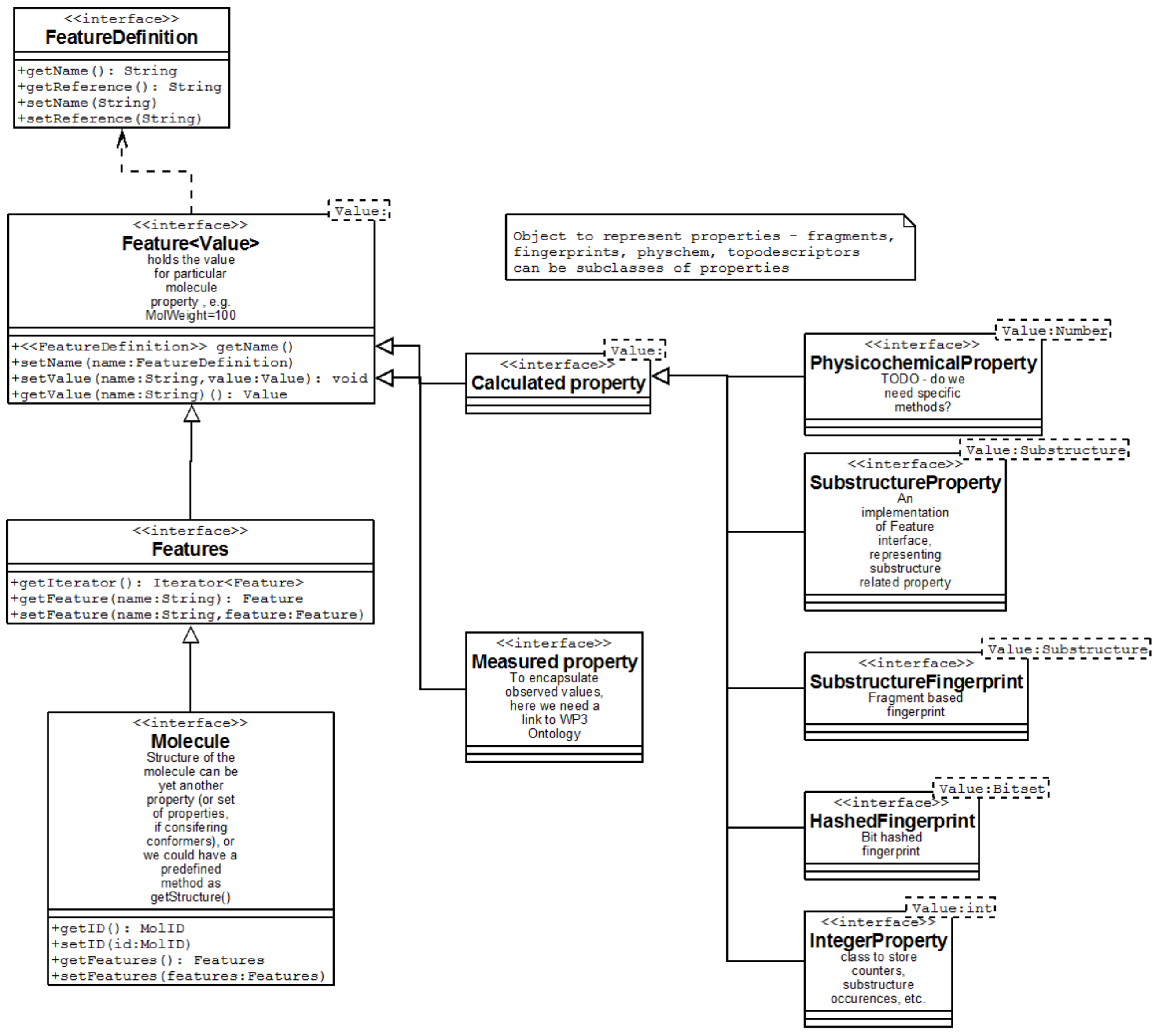
```
Please not this is currently incomplete and needs to be discussed
```

Figure 5: Feature Selection Component



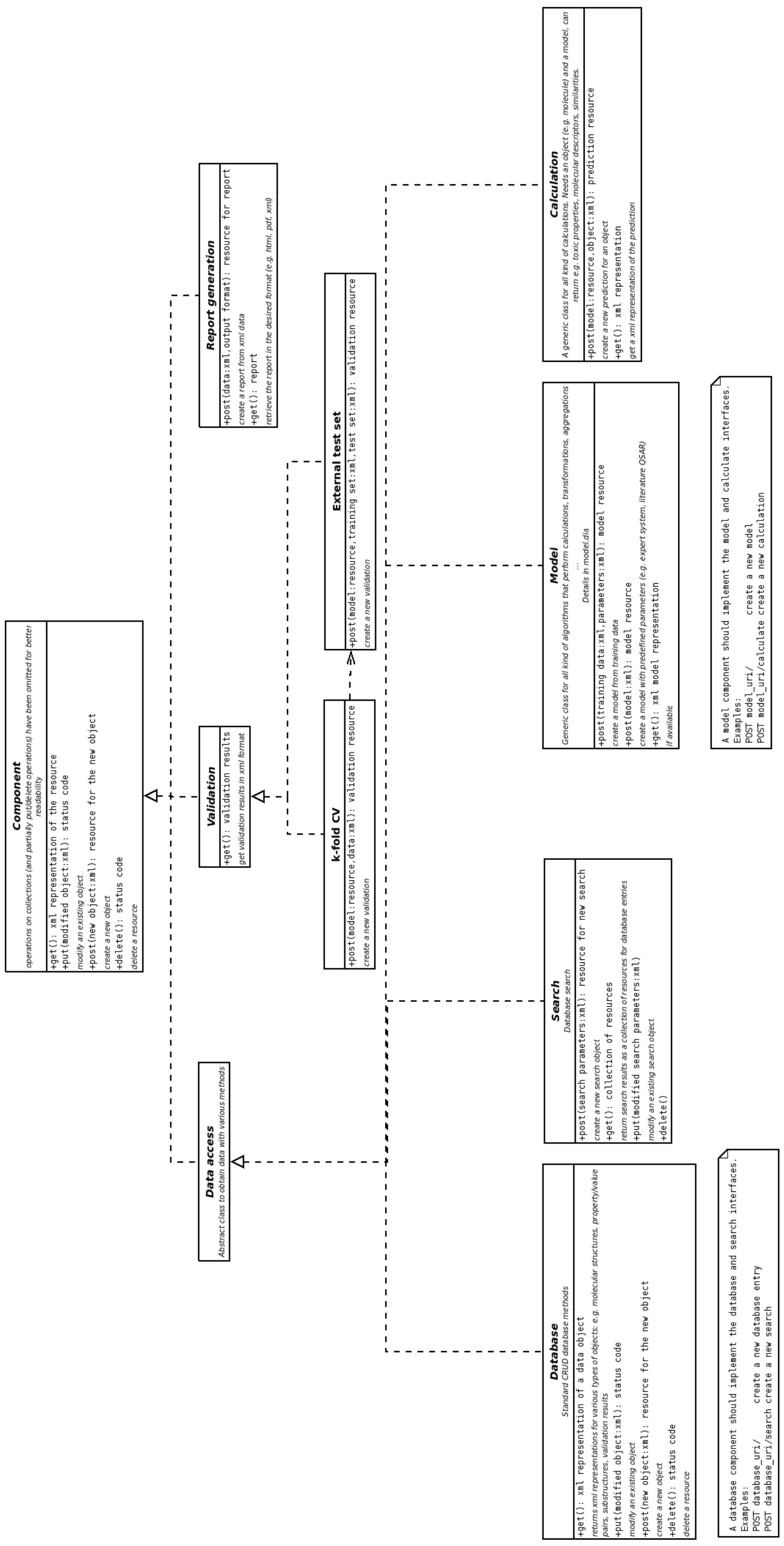
<<interface>>
Algorithm
Defined in Modeling component
<<interface>>
Dataset
Defined in DB component
<<interface>>
Features
<<interface>>
FS-interface
+selectFeatures(algorithm:Algorithm,dataset:Dataset): Features
+projectFS(instances:Dataset,features:Features): Dataset
<<interface>>
Unsupervised Feature selection
<<interface>>
Supervised Feature Selection
<<interface>>
PCA
Principal component analysis
<<interface>>
WrapperFS
Wrapper methods

Figure 6: Molecule Representation



<<interface>>
FeatureDefinition
+getName(): String
+getReference(): String
+setName(String)
+setReference(String)
Value:
<<interface>>
Feature<Value>
holds the value for particular molecule property , e.g. MolWeight=100
+<<FeatureDefinition>> getName()
+setName(name:FeatureDefinition)
+setValue(name:String,value:Value): void
+getValue(name:String)(): Value
Object to represent properties - fragments, fingerprints, physchem, topodescriptors can be subclasses of properties
Value:
<<interface>>
Calculated property
Value:Number
<<interface>>
PhysicochemicalProperty
TODO - do we need specific methods?
Value:Substructure
<<interface>>
SubstructureProperty
An implementation of Feature interface, representing substructure related property
<<interface>>
Features
+getIterator(): Iterator<Feature>
+getFeature(name:String): Feature
+setFeature(name:String,feature:Feature)
<<interface>>
Measured property
To encapsulate observed values, here we need a link to WP3 Ontology
Value:Substructure
<<interface>>
SubstructureFingerprint
Fragment based fingerprint
<<interface>>
Molecule
Structure of the molecule can be yet another property (or set of properties, if consifering conformers), or we could have a predefined method as getStructure()
+getID(): MolID
+setID(id:MolID)
+getFeatures(): Features
+setFeatures(features:Features)
Value:Bitset
<<interface>>
HashedFingerprint
Bit hashed fingerprint
Value:int
<<interface>>
IntegerProperty
class to store counters, substructure occurences, etc.

Figure 7: OpenTox Components and InterActions


Component
operations on collections (and partially put/delete operations) have been omitted for better readability
+get(): xml representation of the resource
+put(modified object:xml): status code
modify an existing object
+post(new object:xml): resource for the new object
create a new object
+delete(): status code
delete a resource
Data access
Abstract class to obtain data with various methods
Validation
+get(): validation results
get validation results in xml format
Report generation
+post(data:xml,output format): resource for report
create a report from xml data
+get(): report
retrieve the report in the desired format (e.g. html, pdf, xml)
k-fold CV
+post(model:resource,data:xml): validation resource
create a new validation
External test set
+post(model:resource,training set:xml,test set:xml): validation resource
create a new validation
Database
Standard CRUD database methods
+get(): xml representation of a data object
returns xml representations for various types of objects: e.g. molecular structures, property/value pairs, substructures, validation results
+put(modified object:xml): status code
modify an existing object
+post(new object:xml): resource for the new object
create a new object
+delete(): status code
delete a resource
Search
Database search
+post(search parameters:xml): resource for new search
create a new search object
+get(): collection of resources
return search results as a collection of resources for database entries
+put(modified search parameters:xml)
modify an existing search object
+delete()
Model
Generic class for all kind of algorithms that perform calculations, transformations, aggregations
...
Details in model.dia
+post(training data:xml,parameters:xml): model resource
create a model from training data
+post(model:xml): model resource
create a model with predefined parameters (e.g. expert system, literature QSAR)
+get(): xml model representation
if available
Calculation
A generic class for all kind of calculations. Needs an object (e.g. molecule) and a model, can return e.g. toxic properties, molecular descriptors, similarities.
+post(model:resource,object:xml): prediction resource
create a new prediction for an object
+get(): xml representation
get a xml representation of the prediction
A database component should implement the database and search interfaces.
Examples:
POST database_uri/ create a new database entry
POST database_uri/search create a new search
A model component should implement the model and calculate interfaces.
Examples:
POST model_uri/ create a new model
POST model_uri/calculate create a new calculation

Furthermore it was proposed to use a Representational State Transfer (REST) architecture for the communication between components. Details of this are shown in Appendix B.

These propositions are currently the subject of further discussions and revisions. At the half year meeting in February it was decided to stabilize the interface definitions first and subsequently to make decisions on the web service architecture.

7 Conclusions

Initial requirements, standards and APIs for the OpenTox framework were defined and published in the developer area of the OpenTox developer website (http://www.opentox.org/dev).

A clear common understanding and definition of the OpenTox Framework has been achieved and initial components documented. Preliminary proposals on interfaces and web services – which are currently the subject of further evaluation and testing – have been documented in detail.

At present there are no major problems within Work Package 1 that inhibit the progress of the project.

8 Appendix A: Use Cases Questionnaire with Summary of Responses

9 Appendix B: Representational State Transfer Architecture

OpenTox: Questions for Use Cases & Number of Responses

1a. What type of institution do you represent?

Industry	7
Government	0
Academia	0

Name of Institution *(optional)*

1b. What is the institution's main business?

Food industry	0
Pharma industry	2
Suppliers of industrial chemicals	1
Other…	4

(type in the gray box)

2. For what purpose do you need to predict/estimate toxicity of chemicals?

(check all that apply)

early candidate screening	5
high throughput screening	1
regulatory submissions,	5
research (toxicological mechanisms...),	5
risk assessment,	3
prioritisation of biological tests	5
Other…	1

(type in the gray box)

3. Who does the prediction measurement/estimation?

trained toxicologist	5
bioinformatician	0
lab technician	1
computational chemist/modeler	3
Other…	

(type in the gray box)

4. How are toxicity data obtained currently?

experimental animal tests	5
QSAR	4
read across	4
Other	3

(type in the gray box)

5. What methods does your institution use?

Experimental testing	5
TopKat	1
Derek	3
ADAPT	1
Codessa	0
Other…	4

(type in the gray box)

6. What level of detail do you need for individual predictions?

just active/inactive predictions	3
detailed information how the prediction was obtained,	5
please explain…	5

(type in the gray box)

7. For which types of compounds would you use a program such as OpenTox?

pharmaceuticals	2
industrial chemicals	4
cosmetics	2
food additives	1
Other…	3

(type in the gray box)

8. What are the most important endpoints?

please describe the purpose e.g. a regulatory endpoint (please specify which one), human adverse effects (which one, do you have human data, what would be suitable animal/in vitro models) for general risk assessment, ecotoxicological effects

6 responses received and are available for view within the partner area of the website.

9. Quantitative predictions?

Yes/No decisions are sufficient	3
Quantitative predictions are needed	6
Comments:	5

(type in the gray box)

10. Types of end-points needed

Single endpoints	4
Activity profiles	4
Comments:	4

(type in the gray box)

11. Do you need to be able to create your own prediction models

Yes 7 No 0

If yes, do you have a preference for certain methods or algorithms? 4

(type in the gray box)

12. Maximum number of compounds processed per day/week/month

per *(type in the gray box)*

3 per month – 1 million per week

Typical number of compounds processed: per *(type in the gray box)*

13. Preferred computer platform(s) for (Q)SAR etc. (if applicable)

Linux	2	
Windows desktop	5	
Macintosh desktop	0	
Windows laptop	3	
Macintosh laptop	0	
Other...	0	*(type in the gray box)*

14. Any restrictions from corporate IT policies

No corporate IT restrictions	2
Must be via client-server on corporate intranet	4
Must be standalone and not send data over the internet	2
Other...	3

(type in the gray box)

15. What level of in-house experience in the use and application of QSAR tools is available?

none	0
limited	0
moderate	5
expert	4

Please explain with examples... 4

(type in the gray box)

16. What level of in-house experience in the development of QSAR models is available?

none	0
limited	2
moderate	1
expert	3

Please explain with examples... 3

(type in the gray box)

17. What do see as the benefits and disadvantages of QSAR methods for toxicity assessment (please list)

Benefits/advantages.... 5

(type in the gray box)

Disadvantages.... 5

(type in the gray box)

18. What you see as the benefits and disadvantages of other non-testing methods for toxicity assessment (please list)

Benefits/advantages.... 2

(type in the gray box)

Disadvantages.... 1

(type in the gray box)

SEVENTH FRAMEWORK PROGRAMME

19. What you see as the benefits and disadvantages of experimental testing methods for toxicity assessment (please list)

Benefits/advantages.... 4

(type in the gray box)

Disadvantages.... 4

(type in the gray box)

20. What features/functionality/culture would be necessary to encourage wider use of QSAR for toxicity assessment

please list and explain... 5

(type in the gray box)

21. Which workflow systems do you currently use (if any)?

None	0
Pipeline Pilot	1
Other	2 *(type in the gray box)*

2. Which workflow systems would you wish to use with OpenTox?

None	☐
Pipeline Pilot	2
Other...	1 *(type in the gray box)*

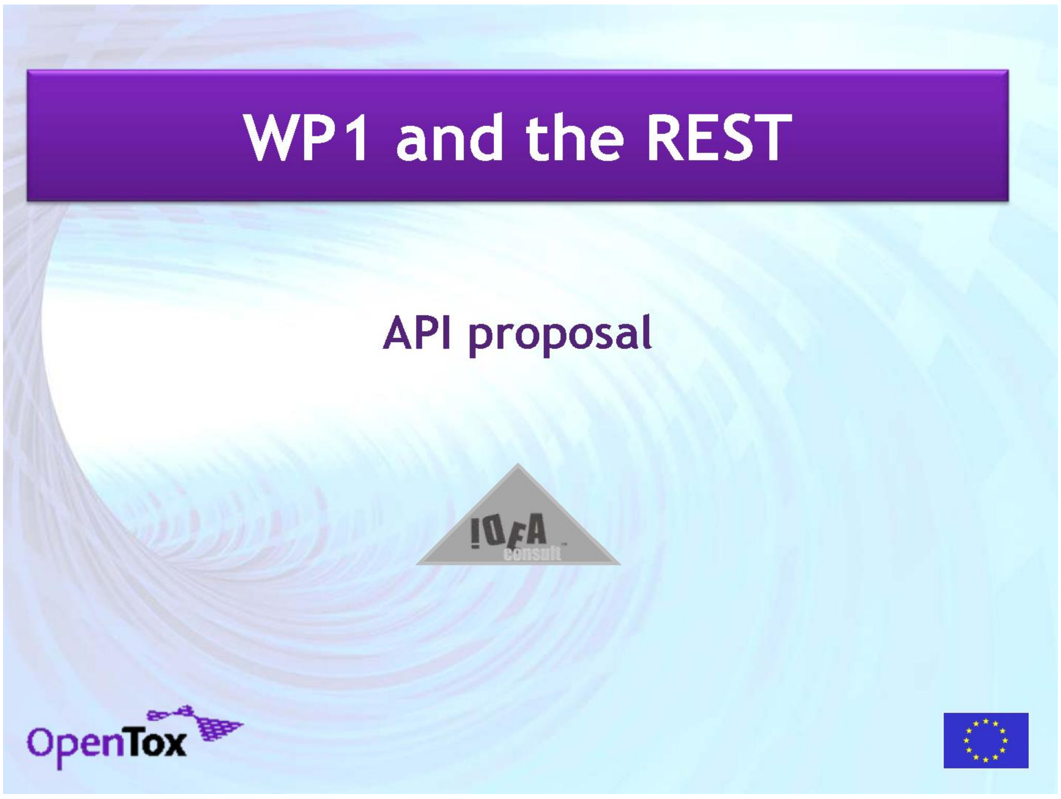
23. Which QSAR models and formats would you want to import into OpenTox?

Models: *(type in the gray box)* Formats: 1

24. What features and capabilities in OpenTox (assuming you could specify them) would make you want to use OpenTox over your existing methods, or in conjunction with them?

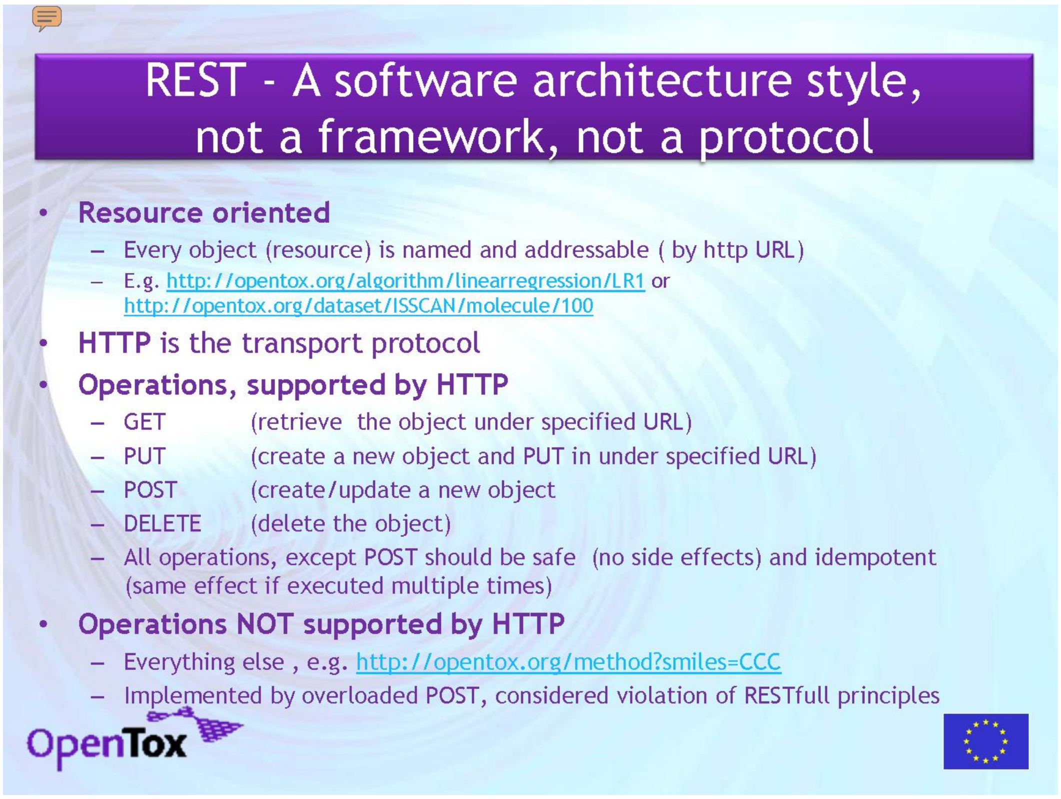
Please specify: 1 *(type in the gray box)*

Representational State Transfer Architecture



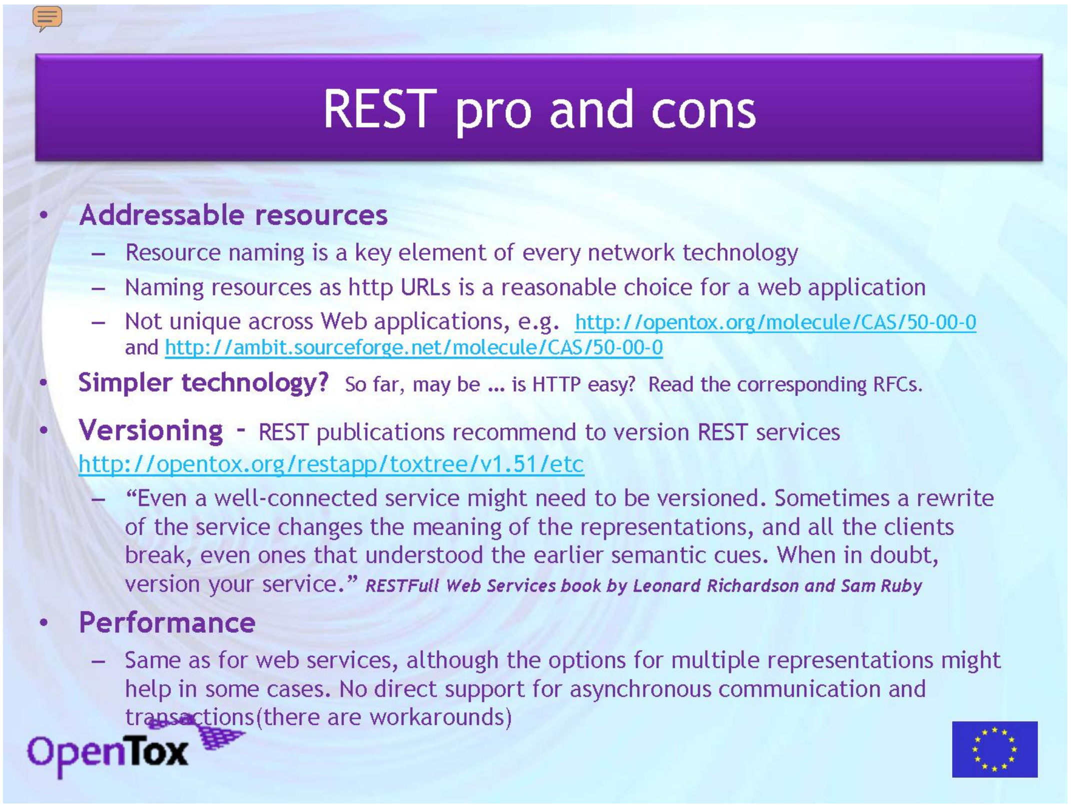
WP1 and the REST

API proposal



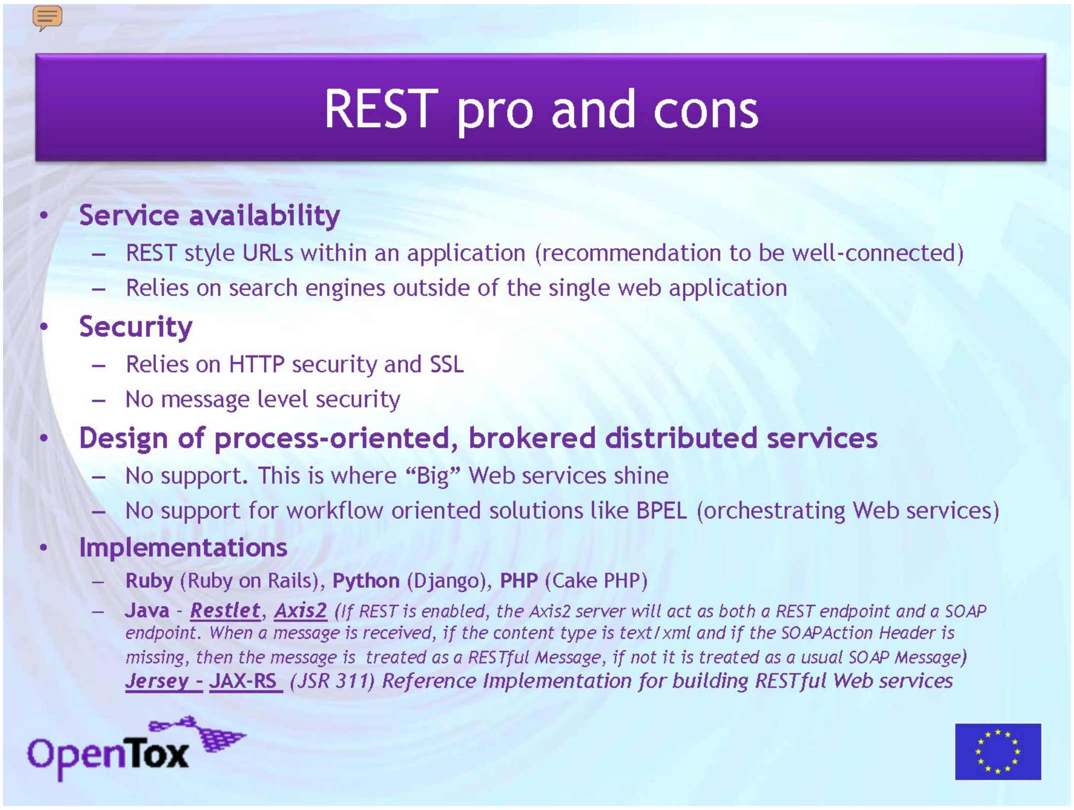
REST - A software architecture style, not a framework, not a protocol

- Resource oriented
  - Every object (resource) is named and addressable (by http URL)
  - E.g. http://opentox.org/algorithm/linearregression/LR1 or http://opentox.org/dataset/ISSCAN/molecule/100
- HTTP is the transport protocol
- Operations, supported by HTTP
  - GET (retrieve the object under specified URL)
  - PUT (create a new object and PUT in under specified URL)
  - POST (create/update a new object
  - DELETE (delete the object)
  - All operations, except POST should be safe (no side effects) and idempotent (same effect if executed multiple times)
- Operations NOT supported by HTTP
  - Everything else , e.g. http://opentox.org/method?smiles=CCC
  - Implemented by overloaded POST, considered violation of RESTfull principles



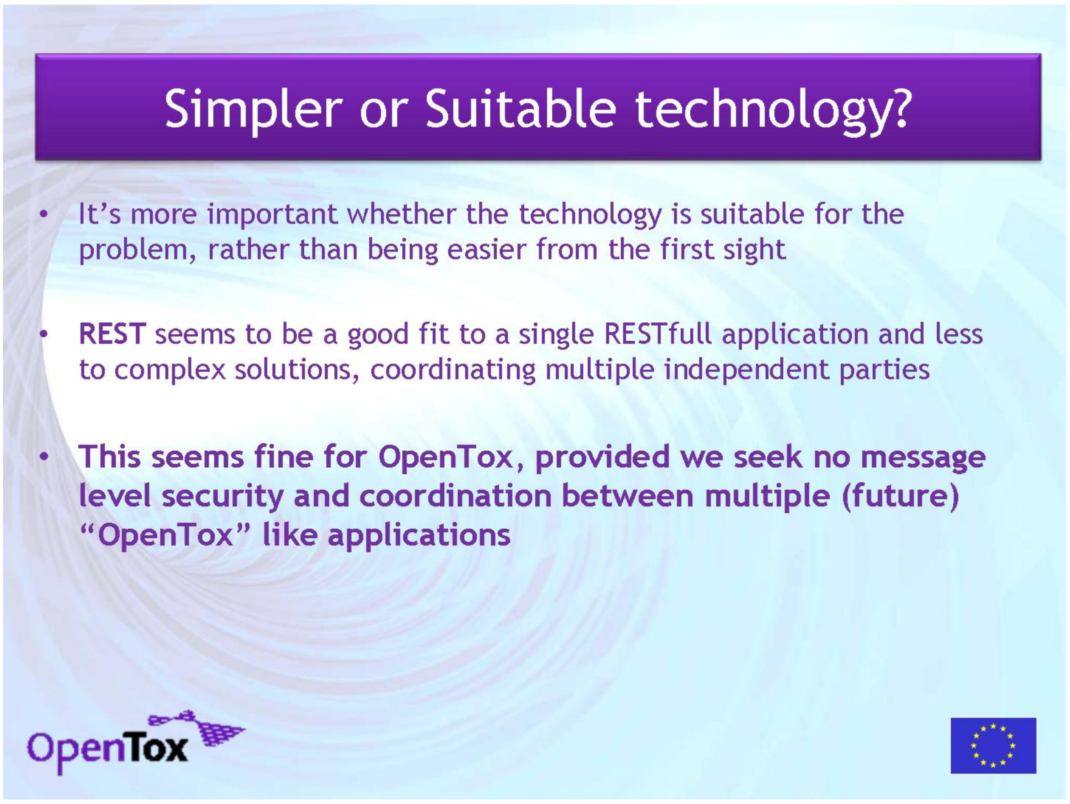
REST pro and cons

- **Addressable resources**
  - Resource naming is a key element of every network technology
  - Naming resources as http URLs is a reasonable choice for a web application
  - Not unique across Web applications, e.g. http://opentox.org/molecule/CAS/50-00-0 and http://ambit.sourceforge.net/molecule/CAS/50-00-0
- **Simpler technology?** So far, may be … is HTTP easy? Read the corresponding RFCs.
- **Versioning** - REST publications recommend to version REST services http://opentox.org/restapp/toxtree/v1.51/etc
  - "Even a well-connected service might need to be versioned. Sometimes a rewrite of the service changes the meaning of the representations, and all the clients break, even ones that understood the earlier semantic cues. When in doubt, version your service." ***RESTFull Web Services book by Leonard Richardson and Sam Ruby***
- **Performance**
  - Same as for web services, although the options for multiple representations might help in some cases. No direct support for asynchronous communication and transactions(there are workarounds)



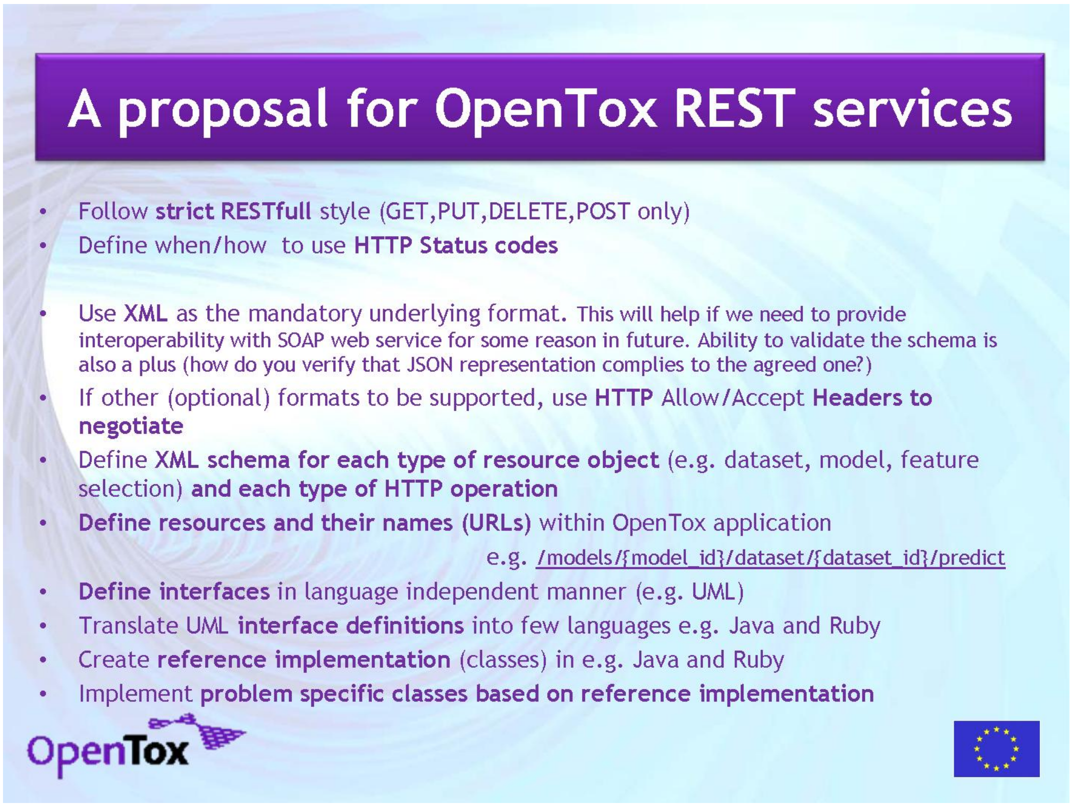
REST pro and cons

- **Service availability**
  - REST style URLs within an application (recommendation to be well-connected)
  - Relies on search engines outside of the single web application
- **Security**
  - Relies on HTTP security and SSL
  - No message level security
- **Design of process-oriented, brokered distributed services**
  - No support. This is where "Big" Web services shine
  - No support for workflow oriented solutions like BPEL (orchestrating Web services)
- **Implementations**
  - **Ruby** (Ruby on Rails), **Python** (Django), **PHP** (Cake PHP)
  - **Java** - ***Restlet***, ***Axis2*** *(If REST is enabled, the Axis2 server will act as both a REST endpoint and a SOAP endpoint. When a message is received, if the content type is text/xml and if the SOAPAction Header is missing, then the message is treated as a RESTful Message, if not it is treated as a usual SOAP Message)* ***Jersey - JAX-RS*** *(JSR 311) Reference Implementation for building RESTful Web services*



Simpler or Suitable technology?

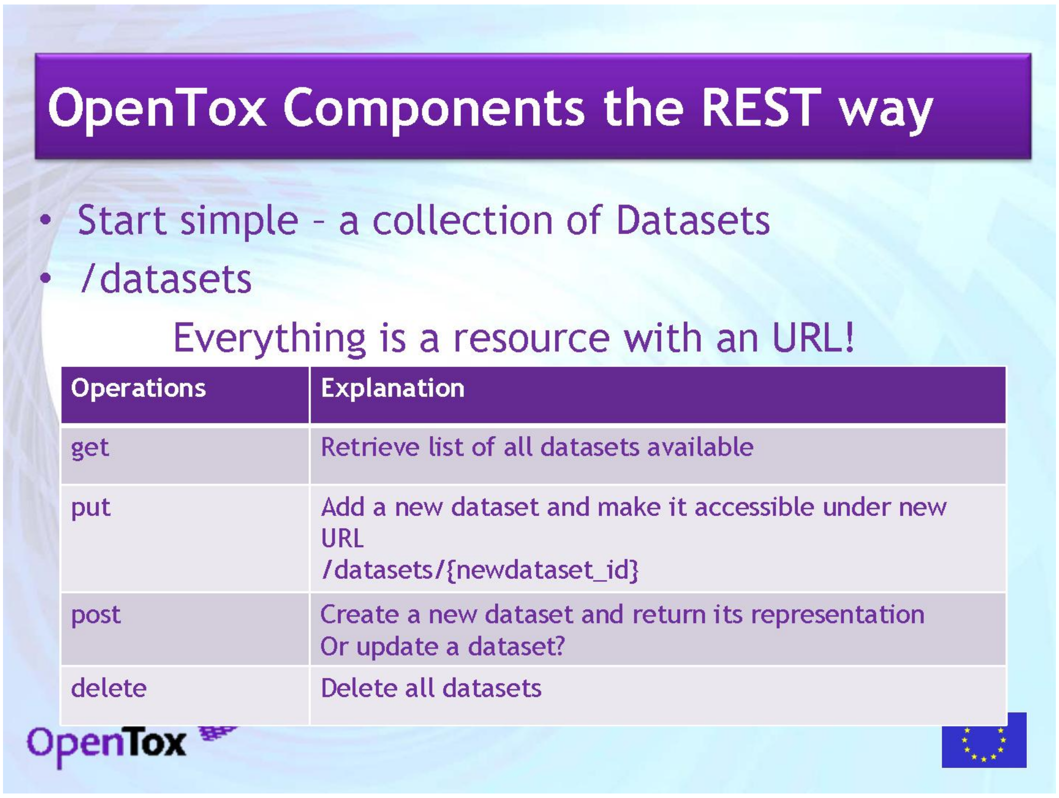
- It's more important whether the technology is suitable for the problem, rather than being easier from the first sight
- **REST** seems to be a good fit to a single RESTfull application and less to complex solutions, coordinating multiple independent parties
- **This seems fine for OpenTox, provided we seek no message level security and coordination between multiple (future) "OpenTox" like applications**



A proposal for OpenTox REST services

- Follow **strict RESTfull** style (GET,PUT,DELETE,POST only)
- Define when/how to use **HTTP Status codes**
- Use **XML** as the mandatory underlying format. This will help if we need to provide interoperability with SOAP web service for some reason in future. Ability to validate the schema is also a plus (how do you verify that JSON representation complies to the agreed one?)
- If other (optional) formats to be supported, use **HTTP** Allow/Accept **Headers to negotiate**
- Define **XML schema for each type of resource object** (e.g. dataset, model, feature selection) **and each type of HTTP operation**
- **Define resources and their names (URLs)** within OpenTox application

  e.g. /models/{model_id}/dataset/{dataset_id}/predict
- **Define interfaces** in language independent manner (e.g. UML)
- Translate UML **interface definitions** into few languages e.g. Java and Ruby
- Create **reference implementation** (classes) in e.g. Java and Ruby
- Implement **problem specific classes based on reference implementation**

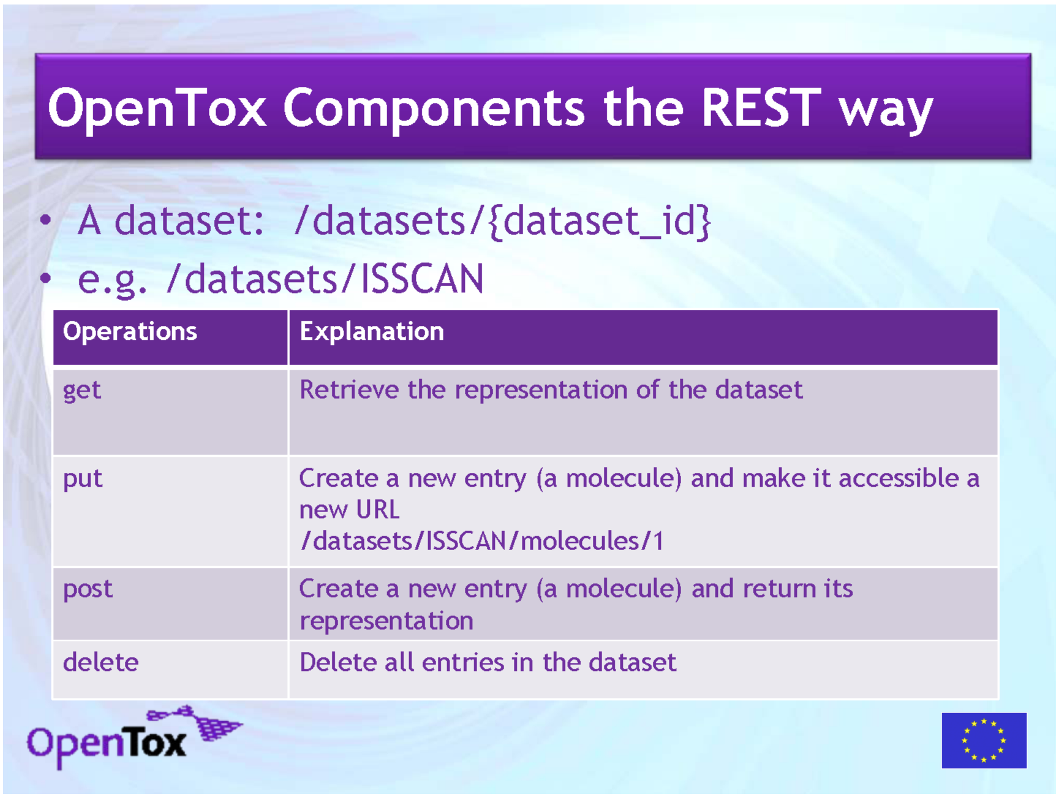


OpenTox Components the REST way

- Start simple – a collection of Datasets
- /datasets

Everything is a resource with an URL!

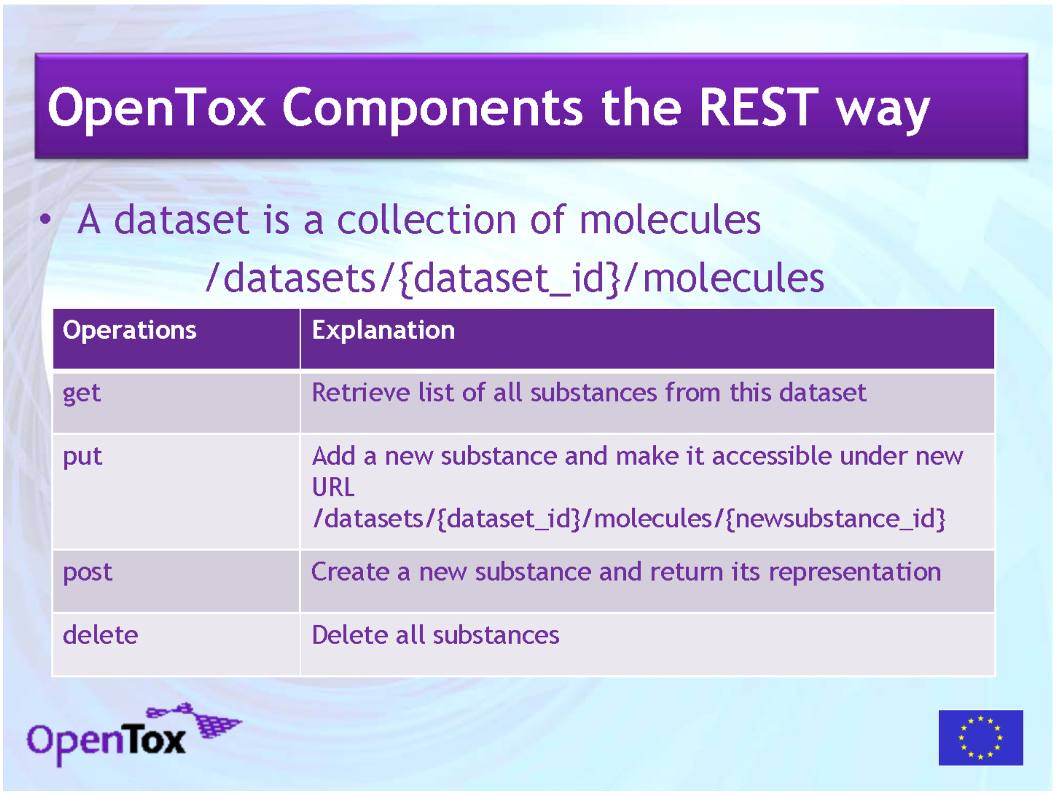
Operations	Explanation
get	Retrieve list of all datasets available
put	Add a new dataset and make it accessible under new URL /datasets/{newdataset_id}
post	Create a new dataset and return its representation Or update a dataset?
delete	Delete all datasets



OpenTox Components the REST way

- A dataset: /datasets/{dataset_id}
- e.g. /datasets/ISSCAN

Operations	Explanation
get	Retrieve the representation of the dataset
put	Create a new entry (a molecule) and make it accessible a new URL /datasets/ISSCAN/molecules/1
post	Create a new entry (a molecule) and return its representation
delete	Delete all entries in the dataset

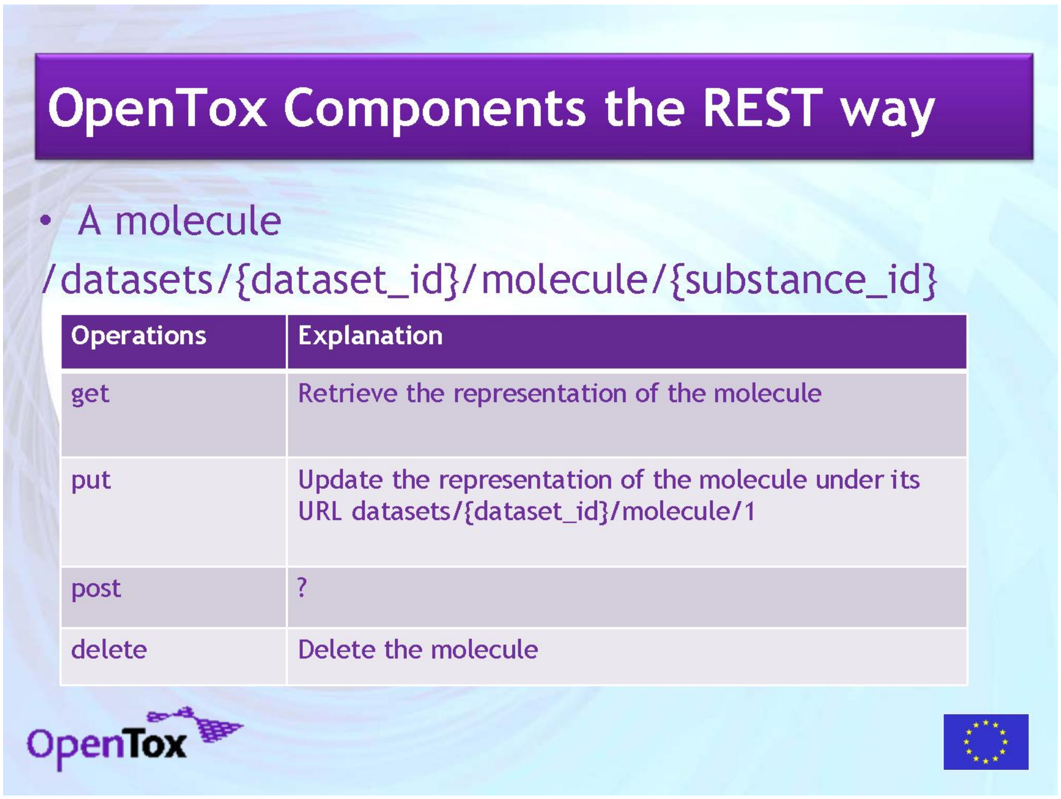


OpenTox Components the REST way

- A dataset is a collection of molecules

/datasets/{dataset_id}/molecules

Operations	Explanation
get	Retrieve list of all substances from this dataset
put	Add a new substance and make it accessible under new URL /datasets/{dataset_id}/molecules/{newsubstance_id}
post	Create a new substance and return its representation
delete	Delete all substances

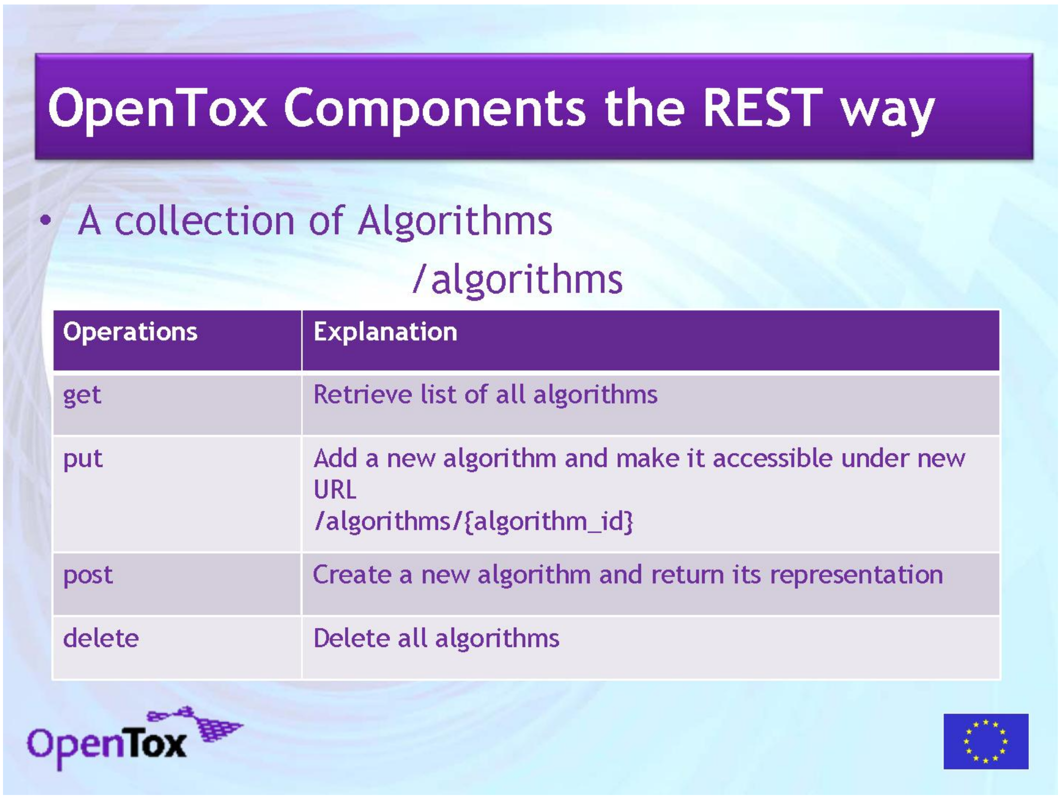


OpenTox Components the REST way

- A molecule

/datasets/{dataset_id}/molecule/{substance_id}

Operations	Explanation
get	Retrieve the representation of the molecule
put	Update the representation of the molecule under its URL datasets/{dataset_id}/molecule/1
post	?
delete	Delete the molecule

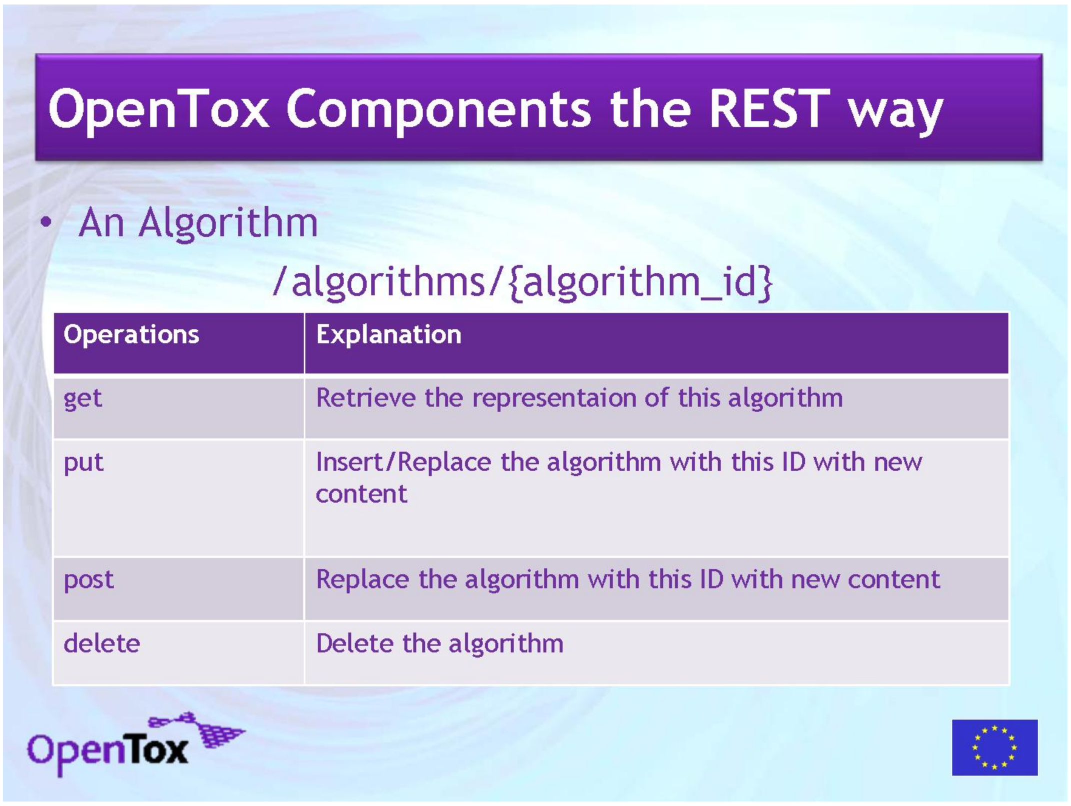


OpenTox Components the REST way

- A collection of Algorithms

/algorithms

Operations	Explanation
get	Retrieve list of all algorithms
put	Add a new algorithm and make it accessible under new URL /algorithms/{algorithm_id}
post	Create a new algorithm and return its representation
delete	Delete all algorithms

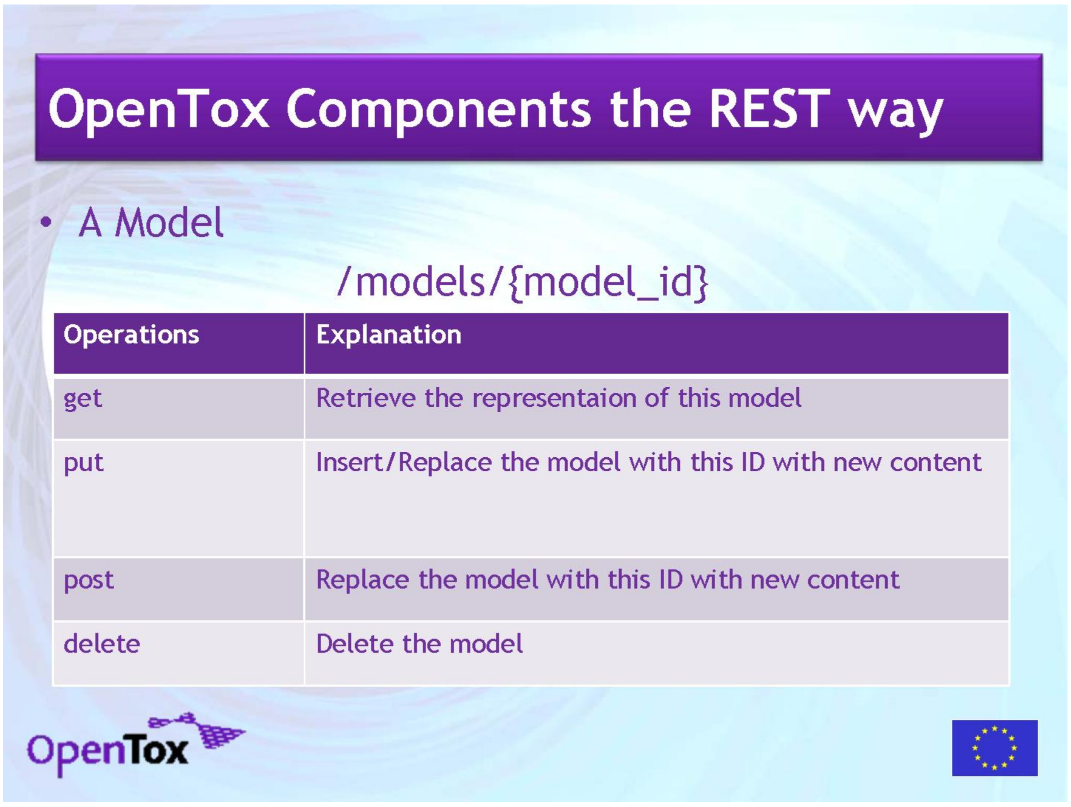


OpenTox Components the REST way

- An Algorithm

/algorithms/{algorithm_id}

Operations	Explanation
get	Retrieve the representaion of this algorithm
put	Insert/Replace the algorithm with this ID with new content
post	Replace the algorithm with this ID with new content
delete	Delete the algorithm

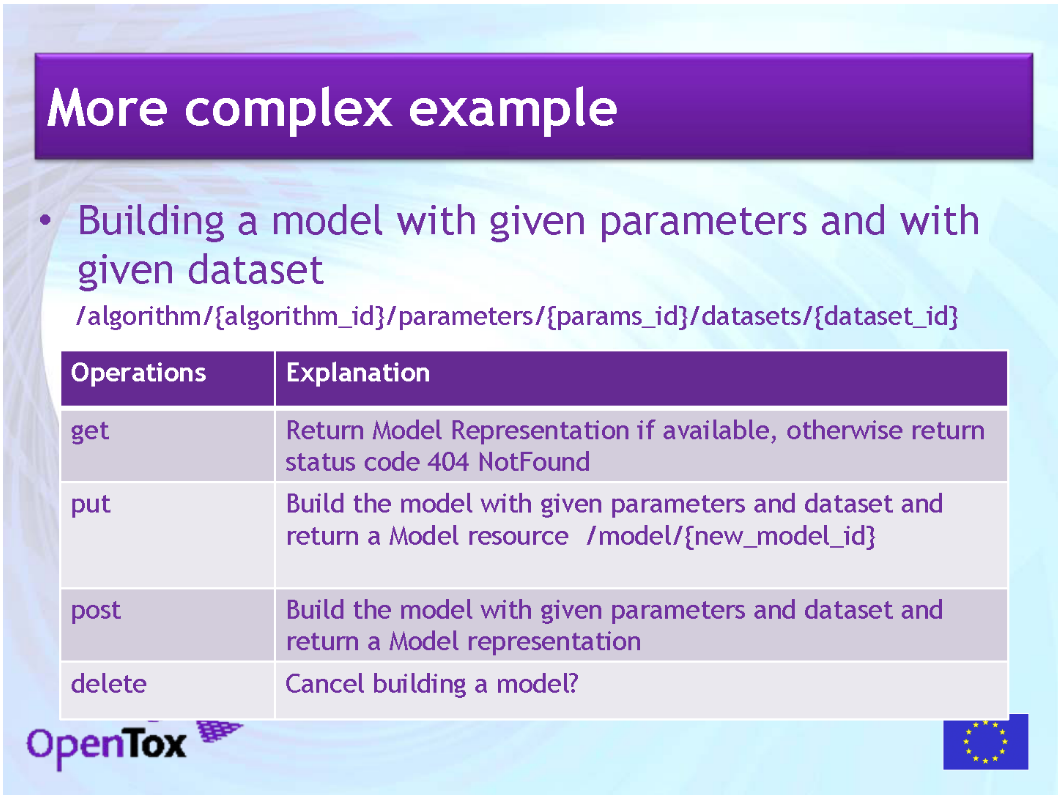


OpenTox Components the REST way

- A Model

/models/{model_id}

Operations	Explanation
get	Retrieve the representaion of this model
put	Insert/Replace the model with this ID with new content
post	Replace the model with this ID with new content
delete	Delete the model

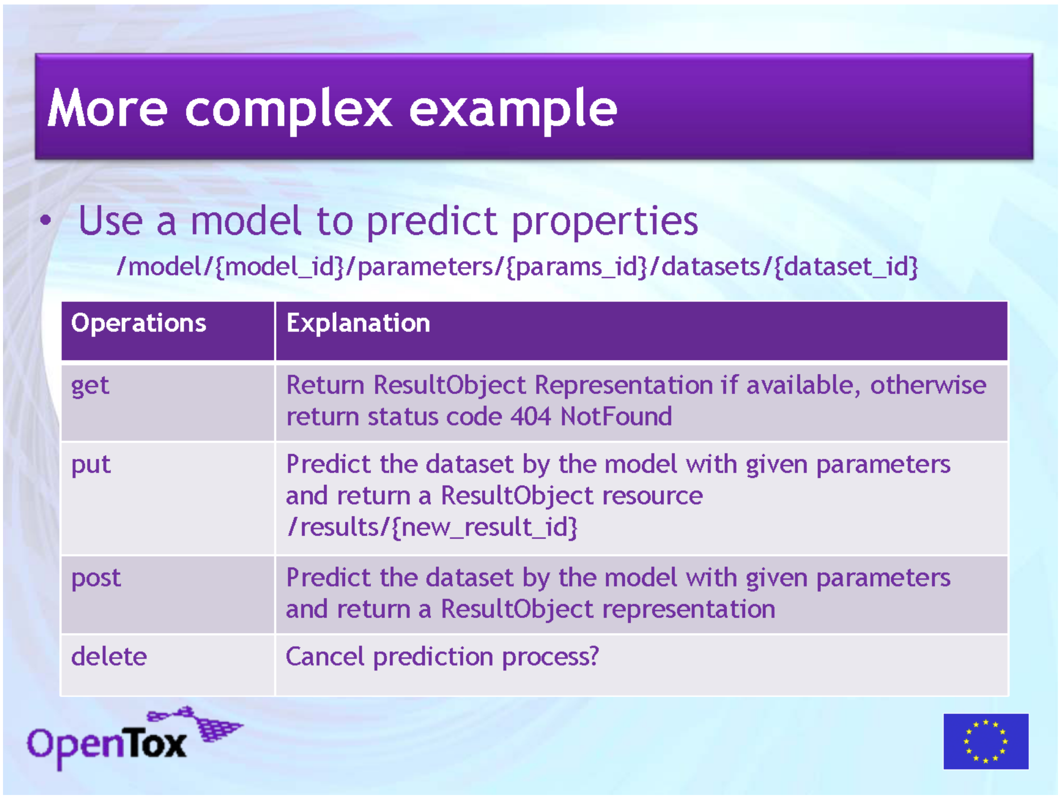


More complex example

- Building a model with given parameters and with given dataset

/algorithm/{algorithm_id}/parameters/{params_id}/datasets/{dataset_id}

Operations	Explanation
get	Return Model Representation if available, otherwise return status code 404 NotFound
put	Build the model with given parameters and dataset and return a Model resource /model/{new_model_id}
post	Build the model with given parameters and dataset and return a Model representation
delete	Cancel building a model?

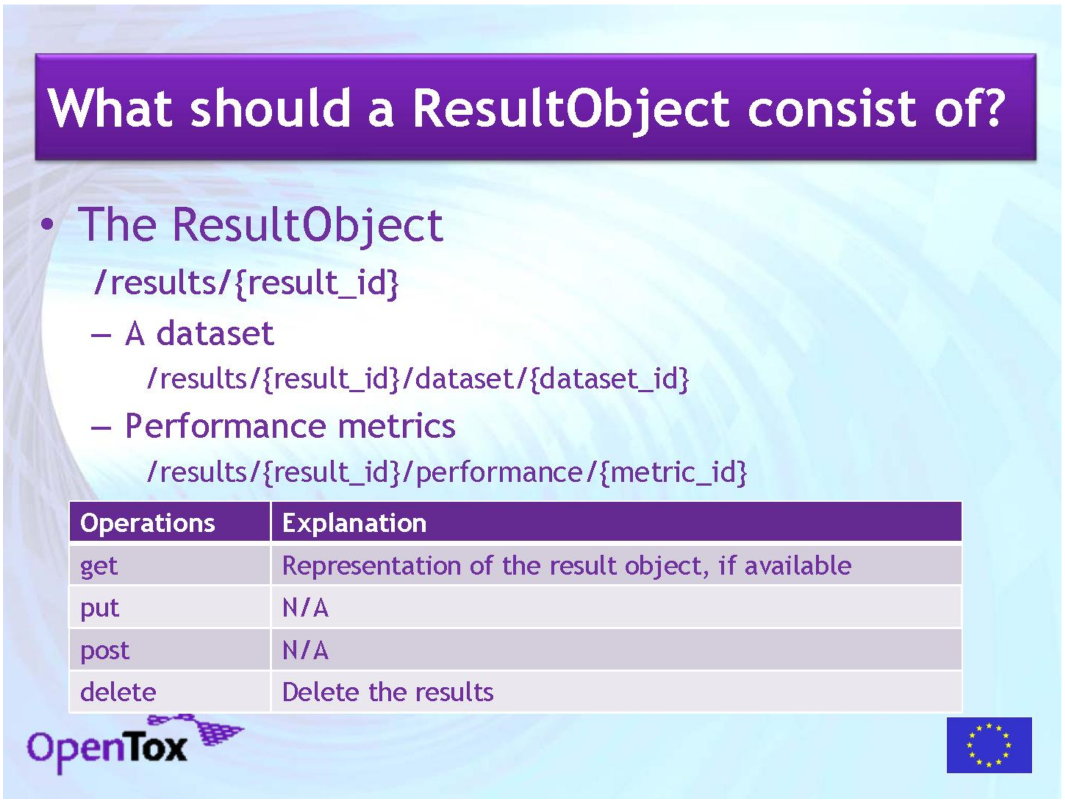


More complex example

- Use a model to predict properties

  /model/{model_id}/parameters/{params_id}/datasets/{dataset_id}

Operations	Explanation
get	Return ResultObject Representation if available, otherwise return status code 404 NotFound
put	Predict the dataset by the model with given parameters and return a ResultObject resource /results/{new_result_id}
post	Predict the dataset by the model with given parameters and return a ResultObject representation
delete	Cancel prediction process?



What should a ResultObject consist of?

- The ResultObject

  /results/{result_id}

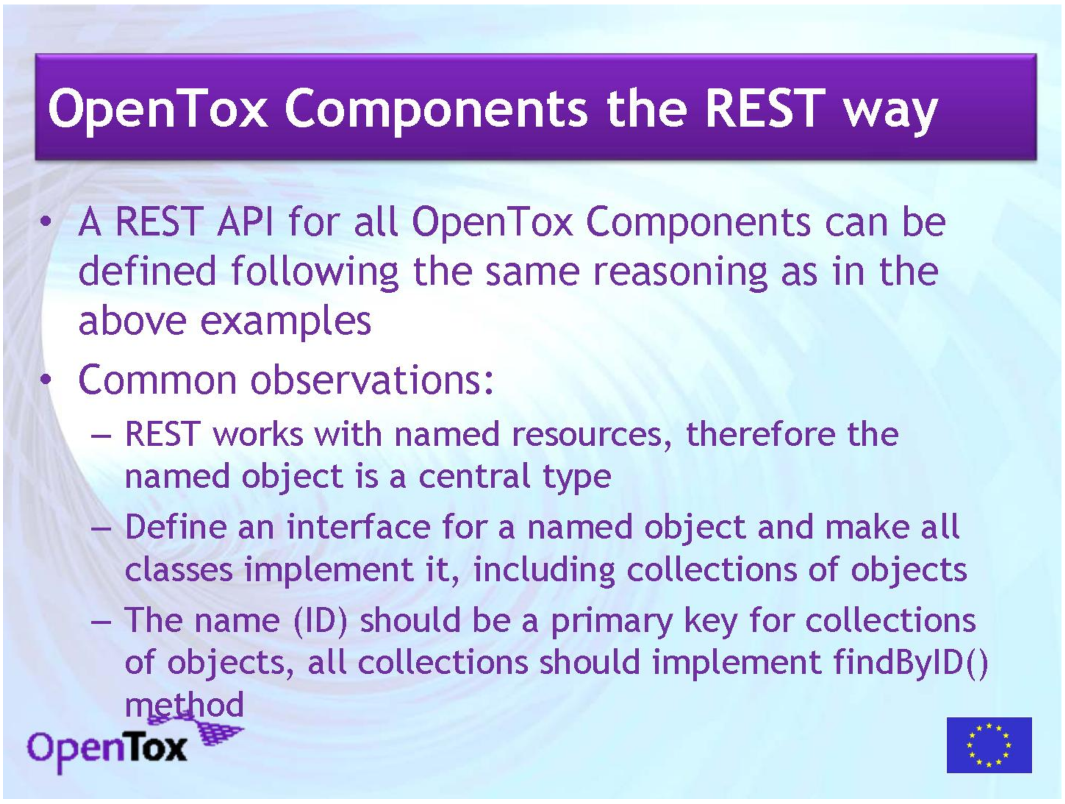
  – A dataset

    /results/{result_id}/dataset/{dataset_id}

  – Performance metrics

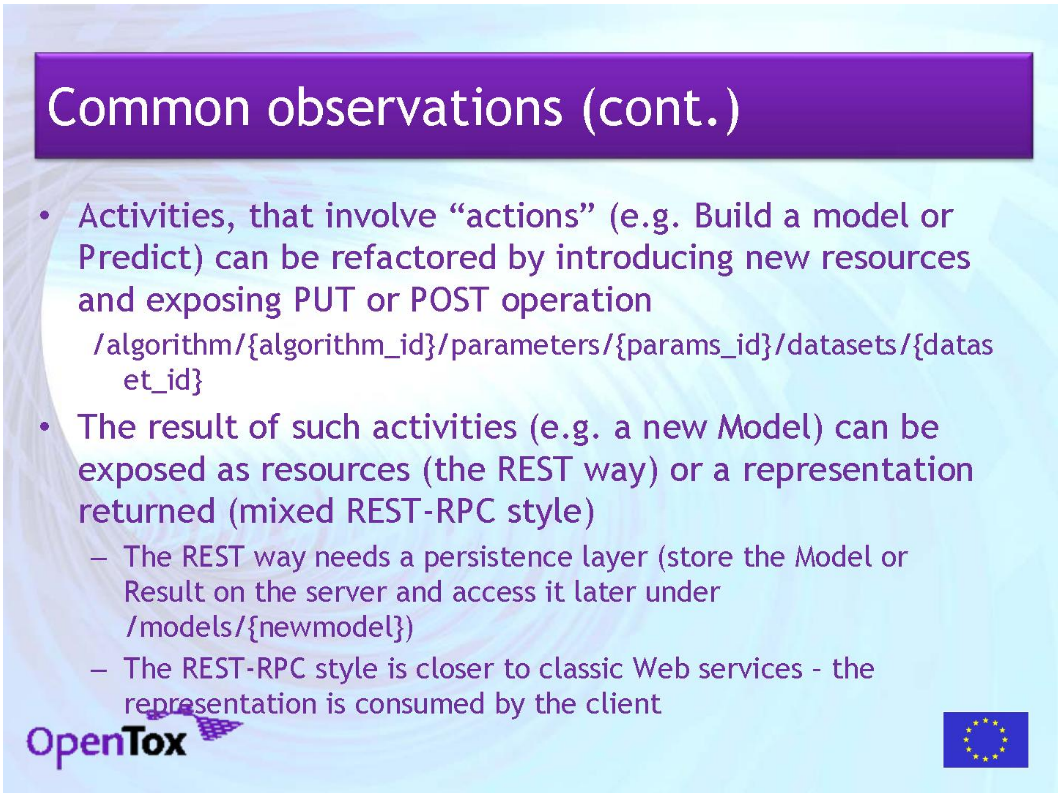
    /results/{result_id}/performance/{metric_id}

Operations	Explanation
get	Representation of the result object, if available
put	N/A
post	N/A
delete	Delete the results



OpenTox Components the REST way

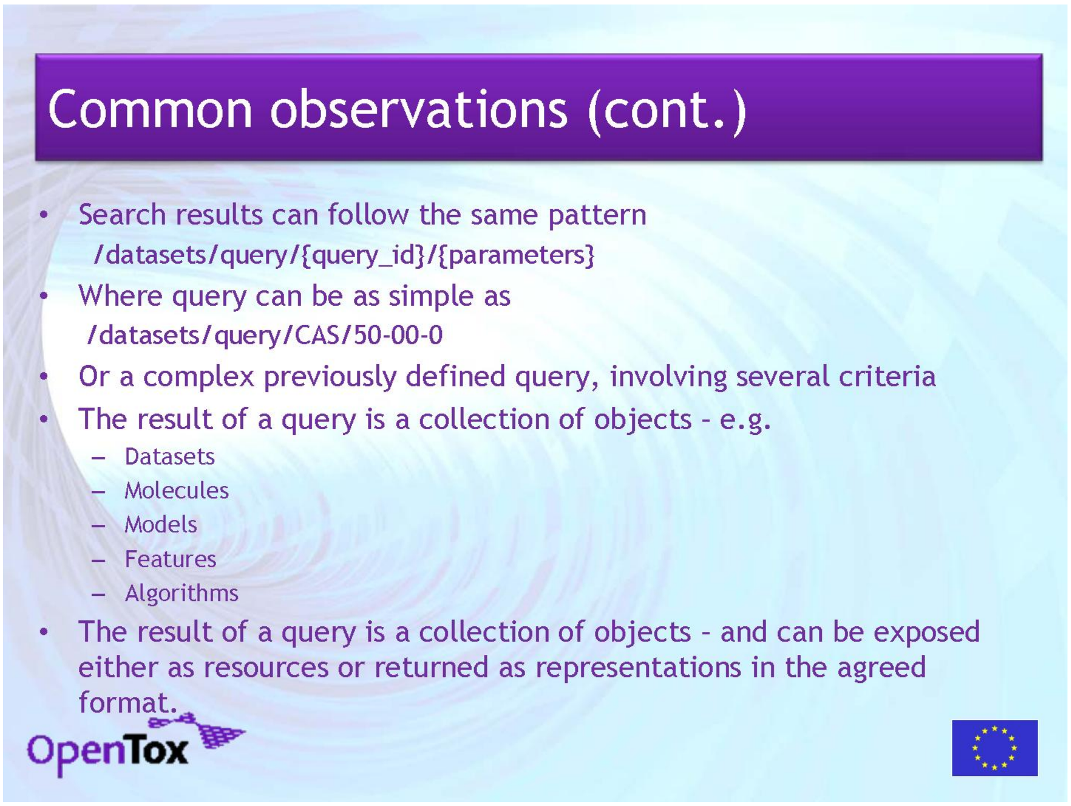
- A REST API for all OpenTox Components can be defined following the same reasoning as in the above examples
- Common observations:
  - REST works with named resources, therefore the named object is a central type
  - Define an interface for a named object and make all classes implement it, including collections of objects
  - The name (ID) should be a primary key for collections of objects, all collections should implement findByID() method



Common observations (cont.)

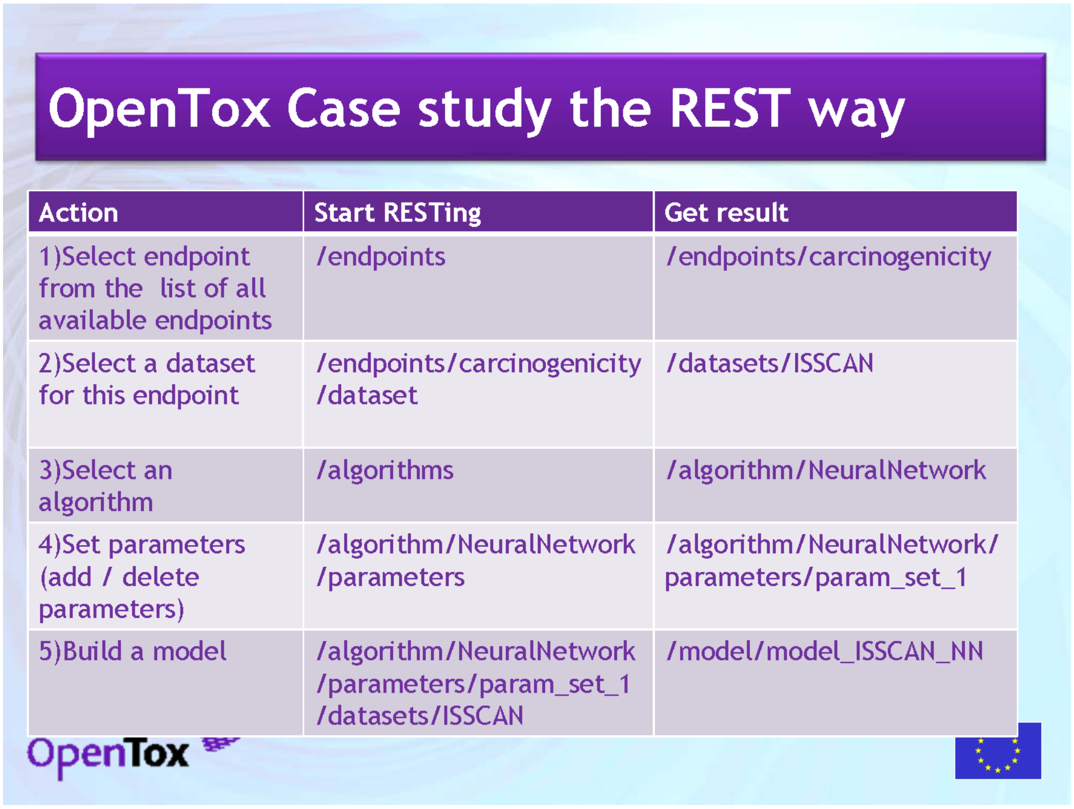
- Activities, that involve “actions” (e.g. Build a model or Predict) can be refactored by introducing new resources and exposing PUT or POST operation

  /algorithm/{algorithm_id}/parameters/{params_id}/datasets/{dataset_id}
- The result of such activities (e.g. a new Model) can be exposed as resources (the REST way) or a representation returned (mixed REST-RPC style)
  - The REST way needs a persistence layer (store the Model or Result on the server and access it later under /models/{newmodel})
  - The REST-RPC style is closer to classic Web services - the representation is consumed by the client



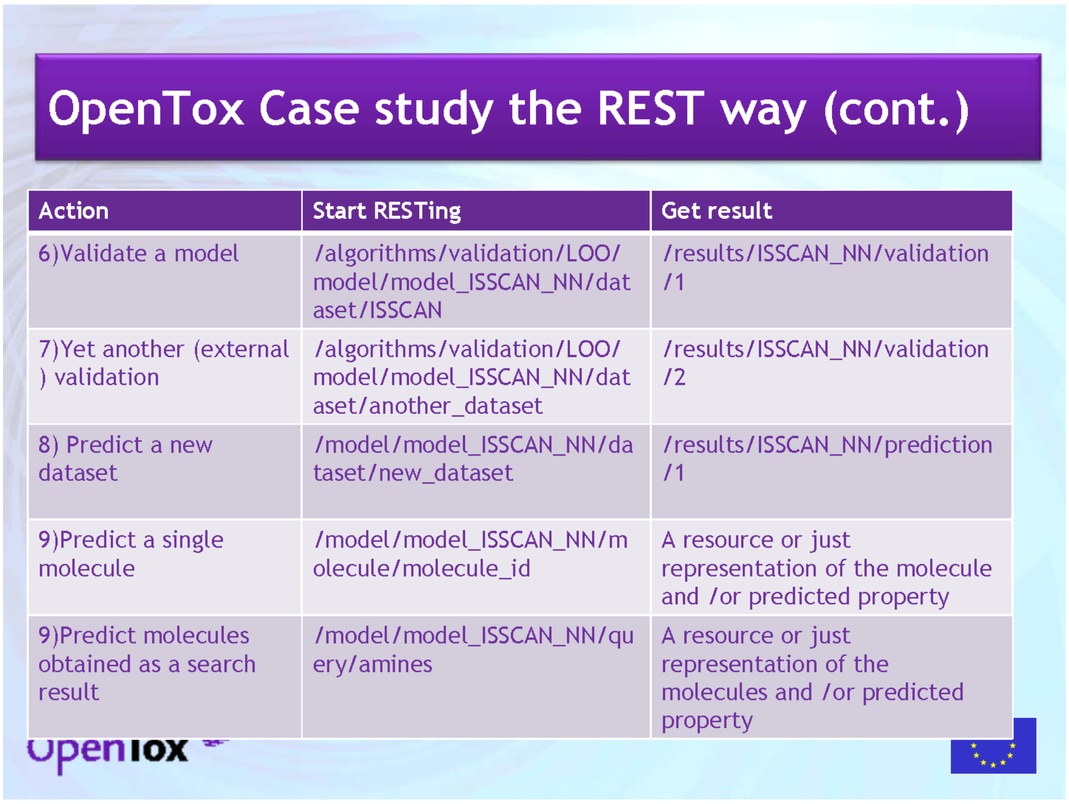
Common observations (cont.)

- Search results can follow the same pattern
  /datasets/query/{query_id}/{parameters}
- Where query can be as simple as
  /datasets/query/CAS/50-00-0
- Or a complex previously defined query, involving several criteria
- The result of a query is a collection of objects - e.g.
  - Datasets
  - Molecules
  - Models
  - Features
  - Algorithms
- The result of a query is a collection of objects - and can be exposed either as resources or returned as representations in the agreed format.



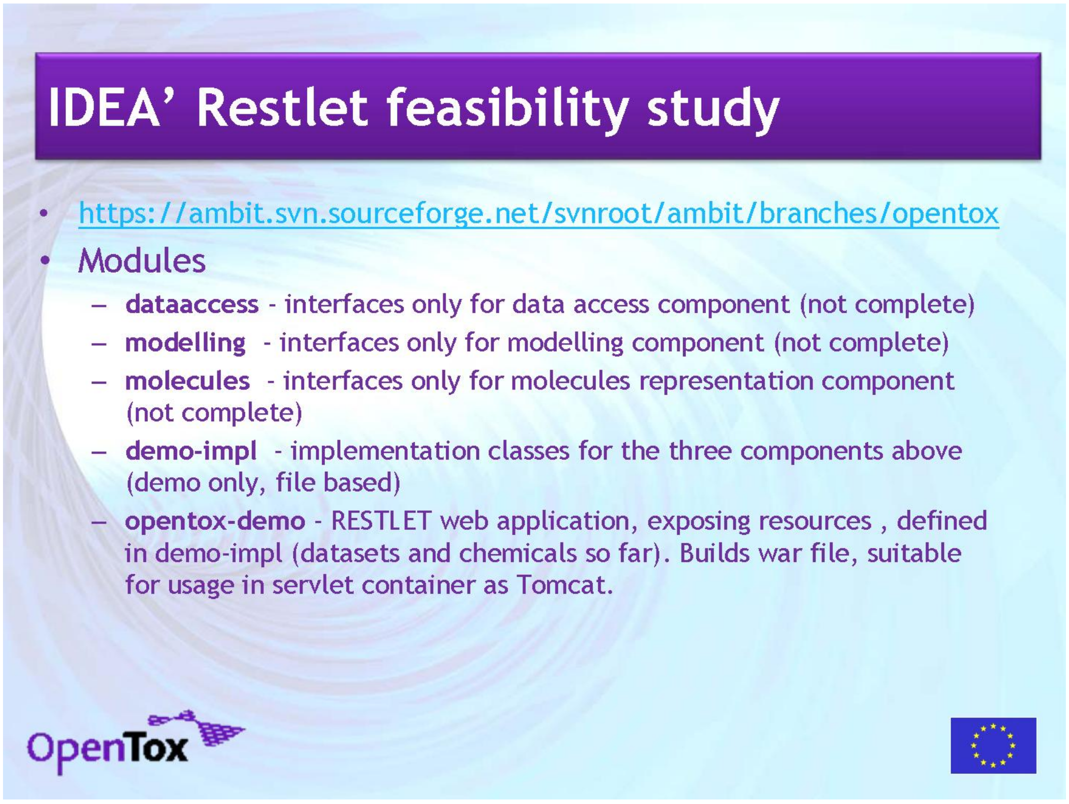
OpenTox Case study the REST way

Action	Start RESTing	Get result
1)Select endpoint from the list of all available endpoints	/endpoints	/endpoints/carcinogenicity
2)Select a dataset for this endpoint	/endpoints/carcinogenicity /dataset	/datasets/ISSCAN
3)Select an algorithm	/algorithms	/algorithm/NeuralNetwork
4)Set parameters (add / delete parameters)	/algorithm/NeuralNetwork /parameters	/algorithm/NeuralNetwork/ parameters/param_set_1
5)Build a model	/algorithm/NeuralNetwork /parameters/param_set_1 /datasets/ISSCAN	/model/model_ISSCAN_NN



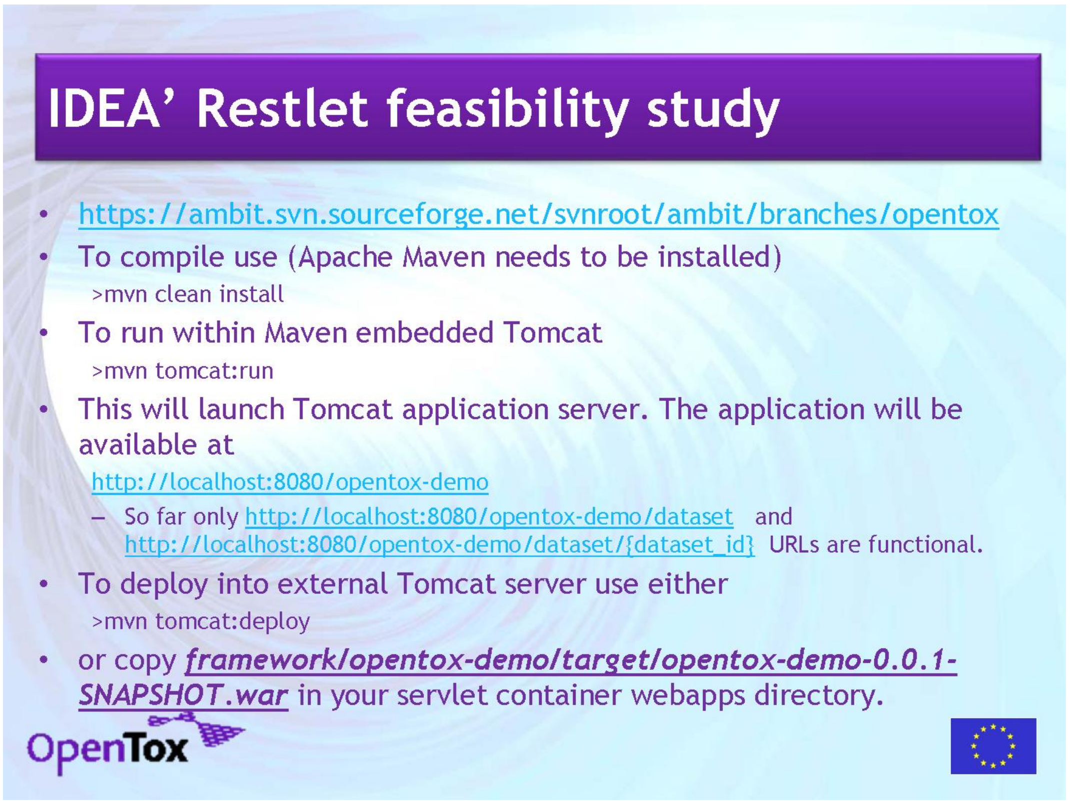
OpenTox Case study the REST way (cont.)

Action	Start RESTing	Get result
6)Validate a model	/algorithms/validation/LOO/ model/model_ISSCAN_NN/dat aset/ISSCAN	/results/ISSCAN_NN/validation /1
7)Yet another (external) validation	/algorithms/validation/LOO/ model/model_ISSCAN_NN/dat aset/another_dataset	/results/ISSCAN_NN/validation /2
8) Predict a new dataset	/model/model_ISSCAN_NN/da taset/new_dataset	/results/ISSCAN_NN/prediction /1
9)Predict a single molecule	/model/model_ISSCAN_NN/m olecule/molecule_id	A resource or just representation of the molecule and /or predicted property
9)Predict molecules obtained as a search result	/model/model_ISSCAN_NN/qu ery/amines	A resource or just representation of the molecules and /or predicted property



IDEA' Restlet feasibility study

- https://ambit.svn.sourceforge.net/svnroot/ambit/branches/opentox
- Modules
  - **dataaccess** - interfaces only for data access component (not complete)
  - **modelling** - interfaces only for modelling component (not complete)
  - **molecules** - interfaces only for molecules representation component (not complete)
  - **demo-impl** - implementation classes for the three components above (demo only, file based)
  - **opentox-demo** - RESTLET web application, exposing resources , defined in demo-impl (datasets and chemicals so far). Builds war file, suitable for usage in servlet container as Tomcat.



IDEA' Restlet feasibility study

- https://ambit.svn.sourceforge.net/svnroot/ambit/branches/opentox
- To compile use (Apache Maven needs to be installed)

```
>mvn clean install
```

- To run within Maven embedded Tomcat

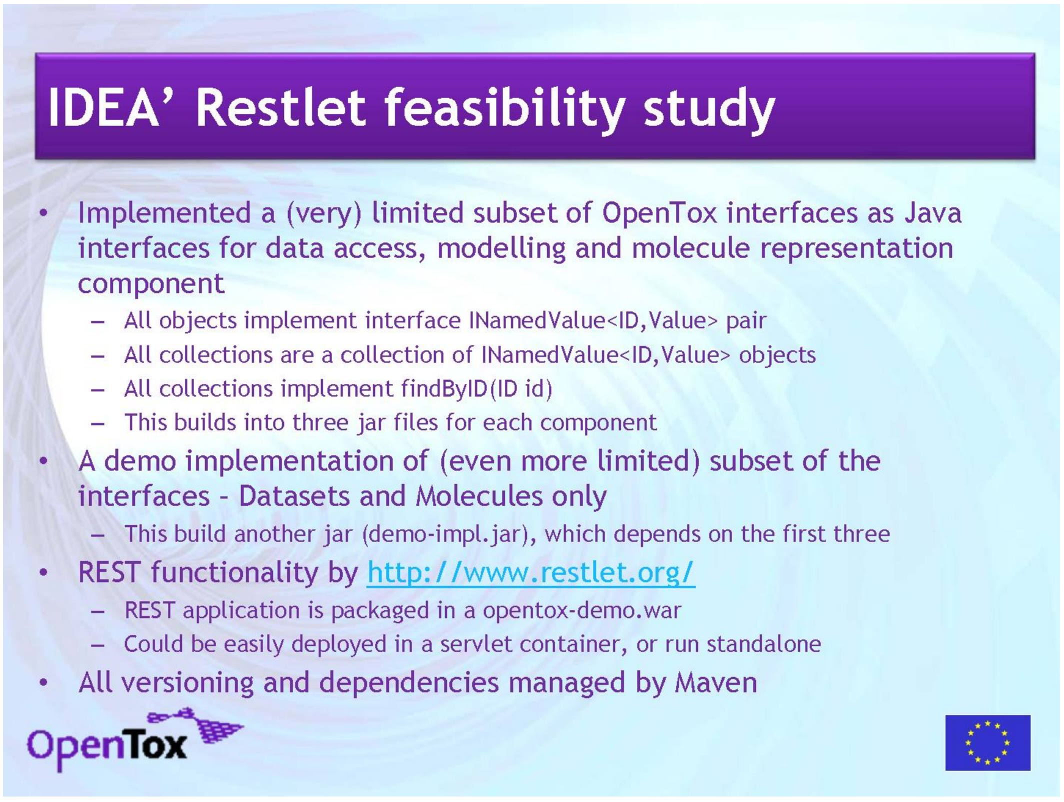
```
>mvn tomcat:run
```

- This will launch Tomcat application server. The application will be available at

  http://localhost:8080/opentox-demo
  - So far only http://localhost:8080/opentox-demo/dataset and http://localhost:8080/opentox-demo/dataset/{dataset_id} URLs are functional.
- To deploy into external Tomcat server use either

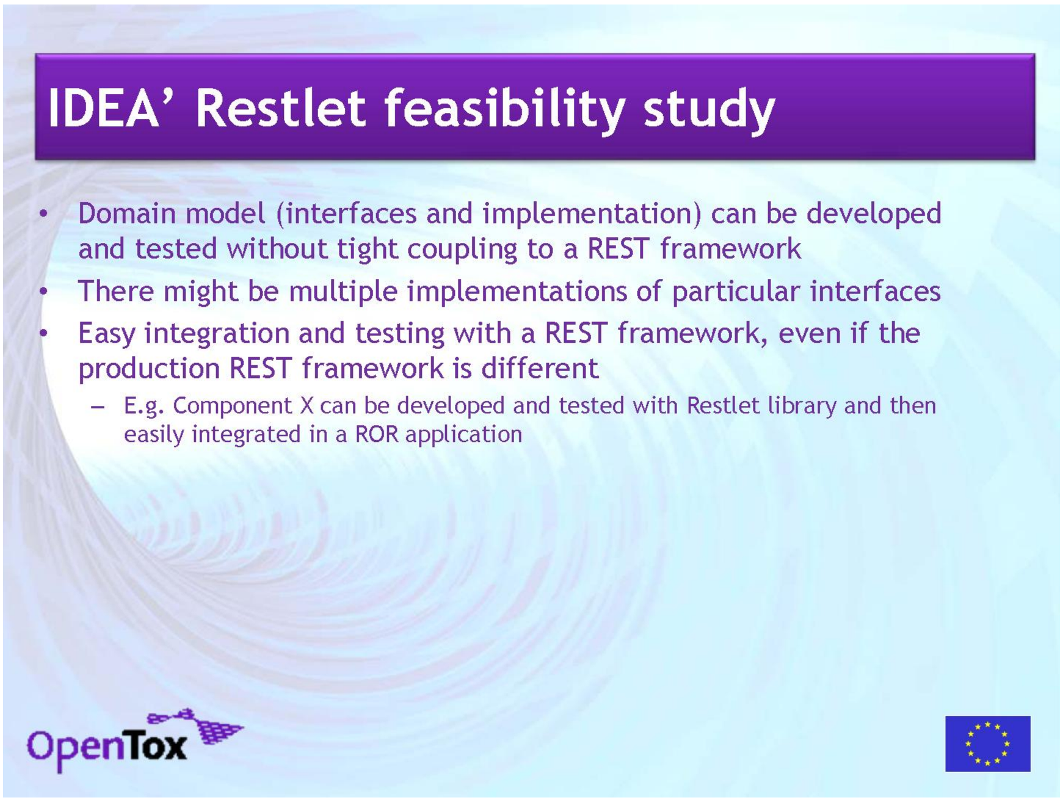
```
>mvn tomcat:deploy
```

- or copy ***framework/opentox-demo/target/opentox-demo-0.0.1-SNAPSHOT.war*** in your servlet container webapps directory.



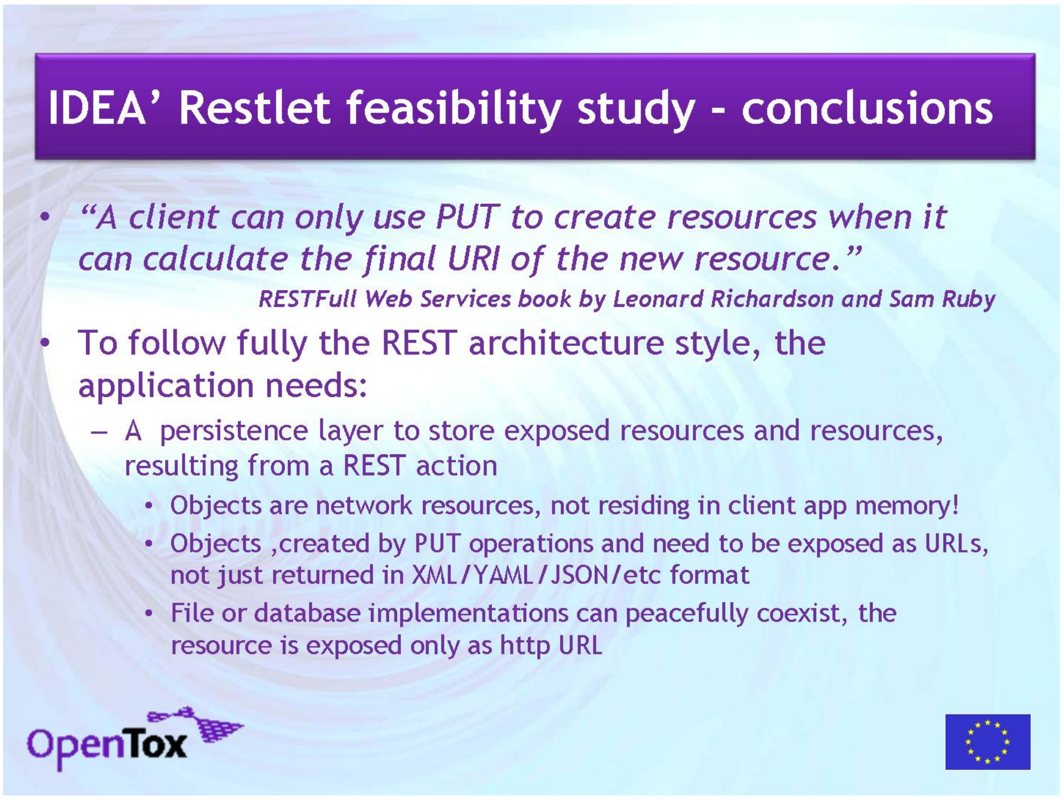
IDEA' Restlet feasibility study

- Implemented a (very) limited subset of OpenTox interfaces as Java interfaces for data access, modelling and molecule representation component
  - All objects implement interface INamedValue<ID,Value> pair
  - All collections are a collection of INamedValue<ID,Value> objects
  - All collections implement findByID(ID id)
  - This builds into three jar files for each component
- A demo implementation of (even more limited) subset of the interfaces - Datasets and Molecules only
  - This build another jar (demo-impl.jar), which depends on the first three
- REST functionality by http://www.restlet.org/
  - REST application is packaged in a opentox-demo.war
  - Could be easily deployed in a servlet container, or run standalone
- All versioning and dependencies managed by Maven



IDEA' Restlet feasibility study

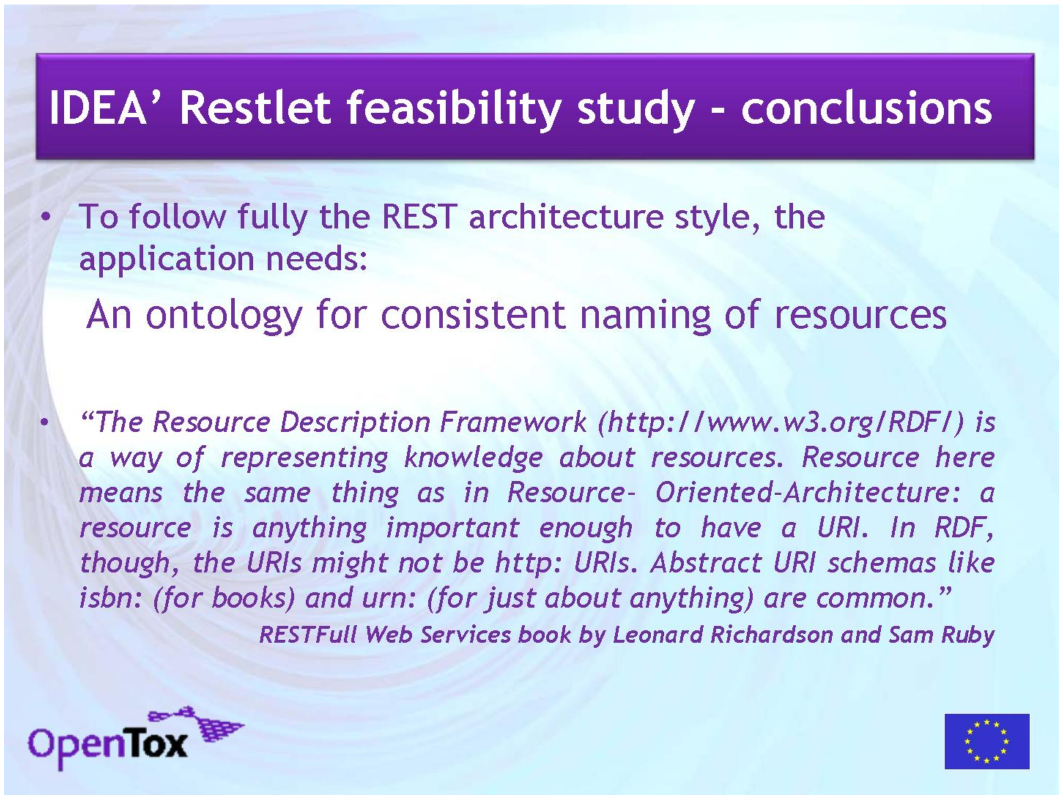
- Domain model (interfaces and implementation) can be developed and tested without tight coupling to a REST framework
- There might be multiple implementations of particular interfaces
- Easy integration and testing with a REST framework, even if the production REST framework is different
  - E.g. Component X can be developed and tested with Restlet library and then easily integrated in a ROR application



IDEA' Restlet feasibility study - conclusions

- *"A client can only use PUT to create resources when it can calculate the final URI of the new resource."*

  RESTFull Web Services book by Leonard Richardson and Sam Ruby
- To follow fully the REST architecture style, the application needs:
  - A persistence layer to store exposed resources and resources, resulting from a REST action
    - Objects are network resources, not residing in client app memory!
    - Objects ,created by PUT operations and need to be exposed as URLs, not just returned in XML/YAML/JSON/etc format
    - File or database implementations can peacefully coexist, the resource is exposed only as http URL



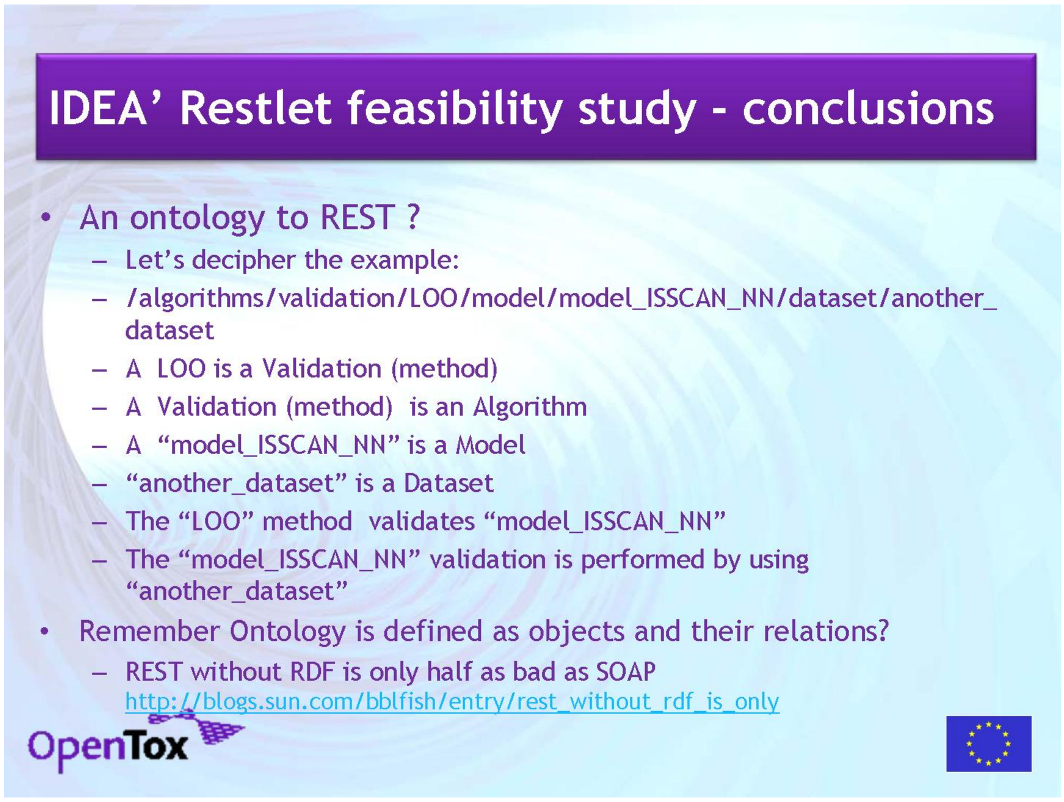
IDEA’ Restlet feasibility study - conclusions

- To follow fully the REST architecture style, the application needs:

  An ontology for consistent naming of resources

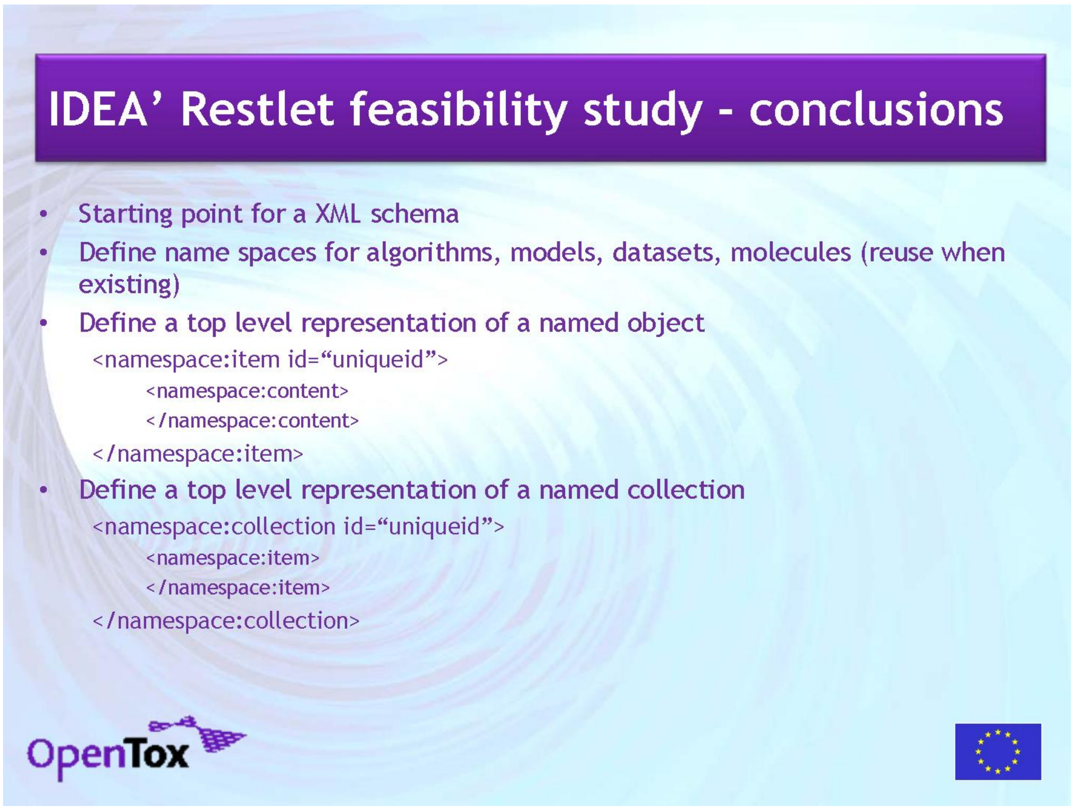
- *“The Resource Description Framework (http://www.w3.org/RDF/) is a way of representing knowledge about resources. Resource here means the same thing as in Resource- Oriented-Architecture: a resource is anything important enough to have a URI. In RDF, though, the URIs might not be http: URIs. Abstract URI schemas like isbn: (for books) and urn: (for just about anything) are common.”*

  RESTFull Web Services book by Leonard Richardson and Sam Ruby



IDEA’ Restlet feasibility study - conclusions

- An ontology to REST ?
  - Let’s decipher the example:
  - /algorithms/validation/LOO/model/model_ISSCAN_NN/dataset/another_dataset
  - A LOO is a Validation (method)
  - A Validation (method) is an Algorithm
  - A “model_ISSCAN_NN” is a Model
  - “another_dataset” is a Dataset
  - The “LOO” method validates “model_ISSCAN_NN”
  - The “model_ISSCAN_NN” validation is performed by using “another_dataset”
- Remember Ontology is defined as objects and their relations?
  - REST without RDF is only half as bad as SOAP
    http://blogs.sun.com/bblfish/entry/rest_without_rdf_is_only



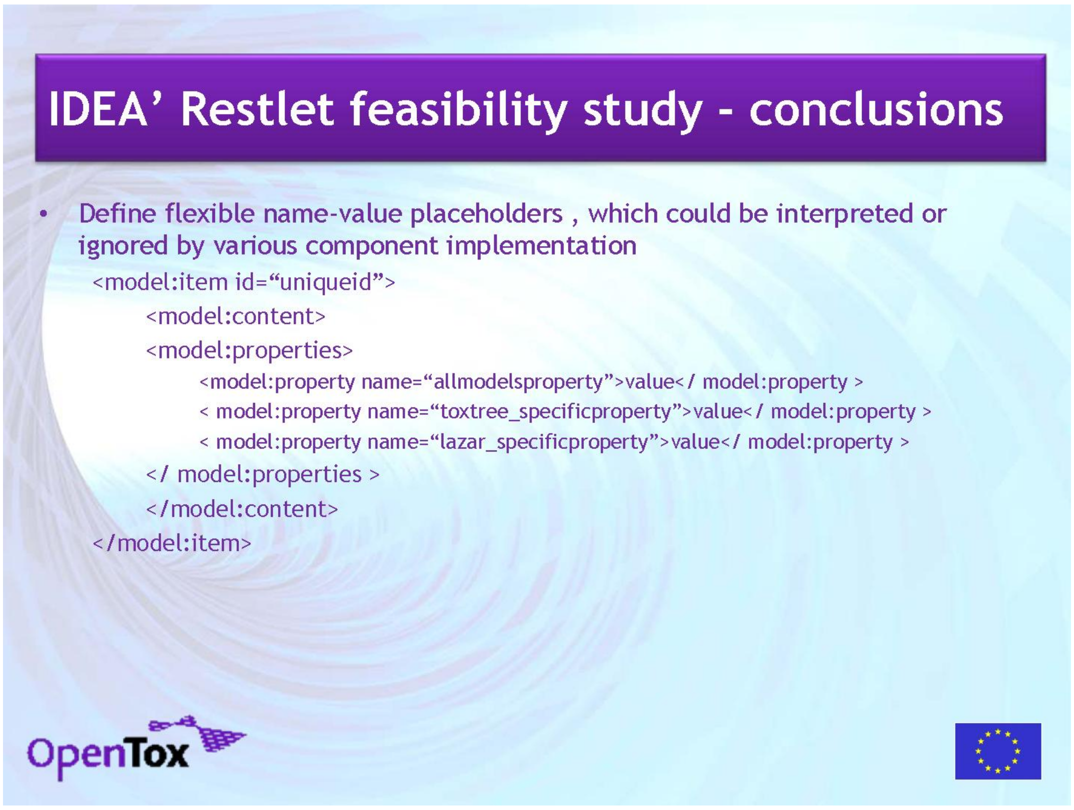
IDEA' Restlet feasibility study - conclusions

- Starting point for a XML schema
- Define name spaces for algorithms, models, datasets, molecules (reuse when existing)
- Define a top level representation of a named object

```
<namespace:item id="uniqueid">
        <namespace:content>
        </namespace:content>
</namespace:item>
```

- Define a top level representation of a named collection

```
<namespace:collection id="uniqueid">
        <namespace:item>
        </namespace:item>
</namespace:collection>
```



IDEA' Restlet feasibility study - conclusions

- Define flexible name-value placeholders , which could be interpreted or ignored by various component implementation

```
<model:item id="uniqueid">
        <model:content>
        <model:properties>
                <model:property name="allmodelsproperty">value</ model:property >
                < model:property name="toxtree_specificproperty">value</ model:property >
                < model:property name="lazar_specificproperty">value</ model:property >
        </ model:properties >
        </model:content>
</model:item>
```