Wrapper Services

Descriptor calculation, feature selection, data filtering (including data transformation), model training and prediction are few of the routines involved in a QSAR session. The connection between these services can either be left up to the client or can be wrapped in a super-service.

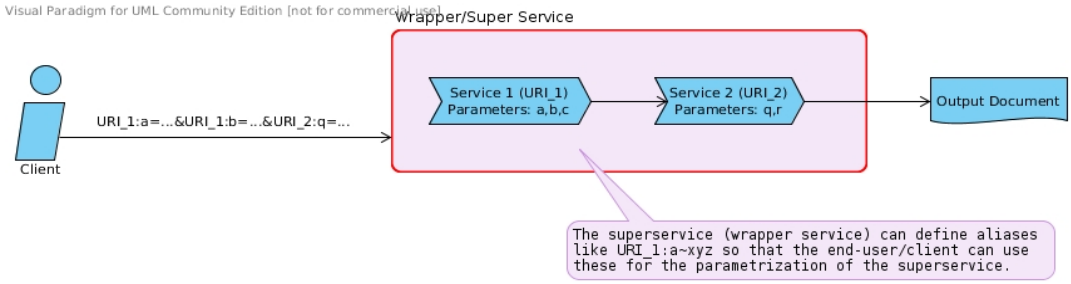

Figure 1: Generic Parametrizable Super-Service

A generic way has to be established to access the parameters of the individual services that compose the overall service. This can be more or less accomplished as shown in Figure 1. If the involved service *S* with URI *U* accepts the parameter *x*, then this should be accessible from the layer of the super-service using the name *U:x* where *U* should be URL-Encoded (for example the client will assign parameters like *http%3A%2F%2Fopentox.ntua.gr%3A3000%2Falgorithm%2Fsvm*:kernel=RBF). Of course, for the sake of convenience, aliases can be adopted by the wrapper service that delegate these lengthy parameter names.

The client can apply an OPTIONS request on the super-service or any of the underlying services and have access to the supported parameters.

Custom Feature Calculation

A descriptor calculation algorithm is not the only process that generates descriptor values! Data transformation or other kind of preprocessing steps might also be necessary prior to the use of a model. In Figure 2 we see a possible layout for a QSAR session including descriptor calculation, feature elimination, some filtering steps such as normalization and scaling and finally data transformation. The final dataset is used for the training of a model which defines (though its RDF representation) a set of independent features. These are actually the features one needs to provide to that model through the input dataset to obtain predictions. The *ot:hasSource* property in that particular case proves to be inadequate since clinging to it complicates things.

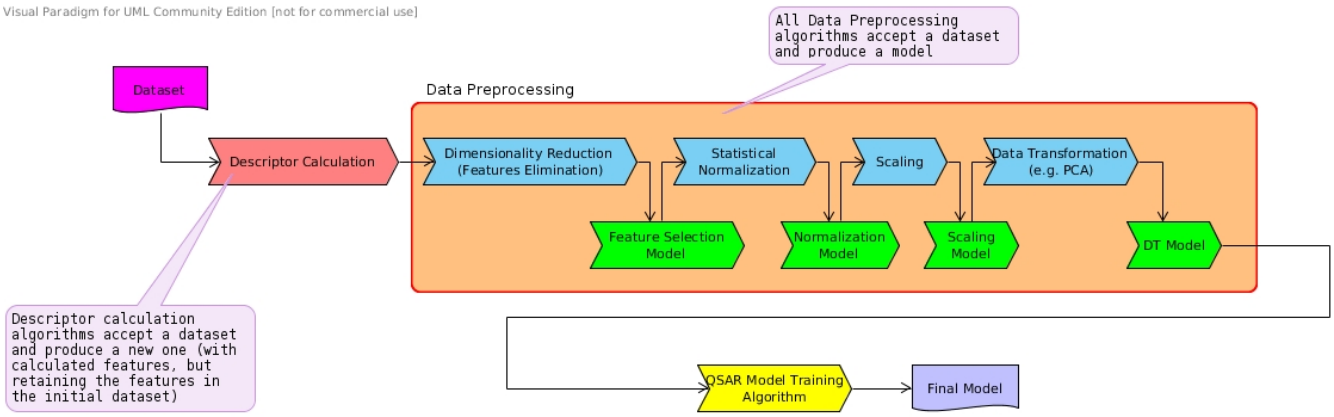

Figure 2: Complete QSAR session

For that purpose a new entity has to be introduced, namely *ot:FeatureCalculationRoutine*, defining for each feature the exact procedure (in terms of successive WS invocations) one should follow to calculate these descriptors for each one of the independent features.

For it to be more comprehensible, let us assume that the initial dataset in Figure 2 employs the features *{f_i} i=1,2,...,K* and the descriptor calculation algorithm outputs a dataset with features *{f_i} i=1,...,K,K+1,...,N*[1]. The feature elimination filtering algorithm selects the features *{f_i_j} j=1,2,...,M<N*. The normalization filter creates the features *{g_j} j=1,2,...,M* and then the scaling algorithm gives the set of features *{h_j} j=1,2,...,M*. Finally the data transformation algorithm creates the features *{Q_l}* l=*1,2,...,P<M* which are the independent features of the trained model. The set of features *Q_l* now has to define unambiguously a way to calculate its values. In a human readable way this is explained by the following sequence of actions:

Input. ***C***::Compound or ***D_0***::Dataset and **M_0**::Model
Expected Output. ***D***::Dataset (with features *{Q_l}* l=*1,2,...,P*)
Step 1. Calculate the features *{f_i_j} j=1,2,...,M* for the compound ***C*** or the dataset ***D_0***. Each one of the features f_i_j links to a descriptor calculation algorithm using the *ot:hasSource* property. This step outputs the auxiliary dataset ***D_1***.
Step 2. Normalize the data in ***D_1*** according to the normalization *model* that was produced by the super-service. This step outputs the auxiliary dataset ***D_2***. The URI of this model should be found in the representation of the produced (super)-model.
Step 3. Scale the data in ***D_2*** according to the normalization *model* that was produced by the super-service. The auxiliary dataset ***D_3*** is produced.
Step 4. Transform the data in ***D_3*** using the PCA model.
Step 5. Append all other features that are needed by the model (are independent features) but were not calculated following this procedure.
Output. Dataset **D**.

This can be represented in a structured machine-readable way using *Feature Calculation Routines*.

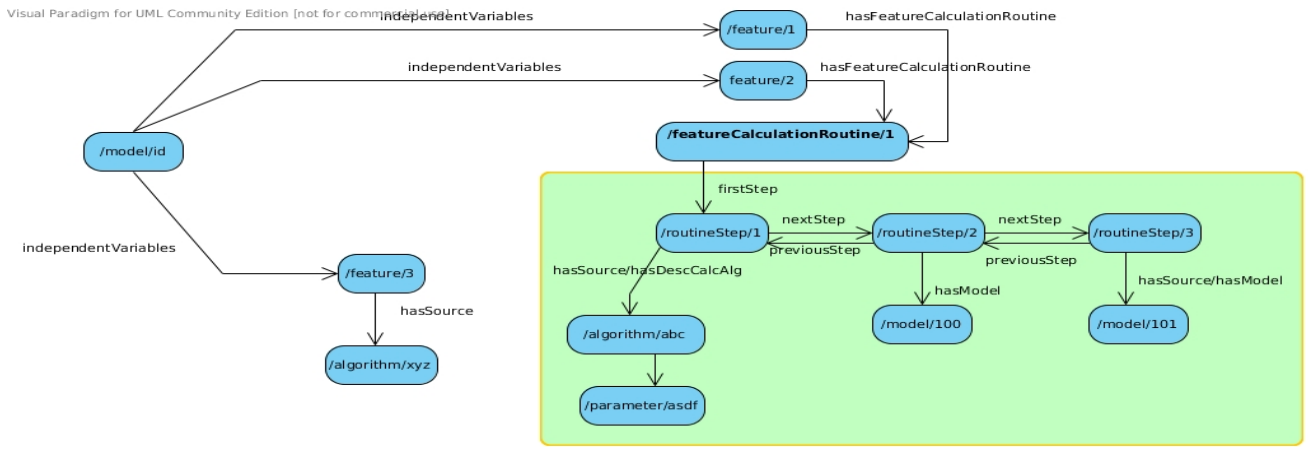

Figure 3: Feature Calculation Routine

1 This means that the initial dataset might have descriptors already calculated. If final model includes features without a source (defined through the *hasSource* property) pointing to a descriptor calculation algorithm then the client is expected to provide these values to the model (if missing from the database). This is the case for example with experimental descriptors.

This way the whole procedure is much more transparent than black-box models that take up the descriptor calculations or any other kind of wrapper services.