

Feta Engine v1.0 Design and Moby Integration

Pinar Alper

December 14, 2005

1 Introduction

This document aims to:

- List the outcomes of the Registry Hackaton related to *my*Grid and BioMOBY service discovery frameworks convergence.
- Outline the design of Feta Engine back-end v1.0, which will be supporting the required functionality identified during the hackaton.

2 Hackaton Outcomes and Canonical Model

The service description models of *my*Grid and BioMOBY are quite similar. One objective of the registry Hackaton was to closely analyze the service description models of the two projects. During, our work, we have identified the following major differences:

1. BioMOBY describes semantic types parameters through two pieces of mark-up, these are the “moby:Namespace” and “moby:objectType” predicates. *my*Grid, on the other hand, designates the semantic type of a parameter through a single assertion using the “rdf:type” to link it to a class in the *my*Grid annotation vocabulary
2. BioMOBY makes use of certain Dublin Core predicates to specify service provider information
3. BioMOBY describes info regarding constraints on parameter values (such as min, max, ..)

Based on the above observations the following work has been done in the two months after the Hackaton.

1. A **core** canonical model has been developed, that covers the essential aspects of a *Service* from both *my*Grid’s and BioMOBY’s perspective. This core model, an example instance of which can be found at ¹ is represented with a suite of complementary ontologies from *my*Grid ² and BioMOBY ³.

¹<http://phoebus.cs.man.ac.uk:8100/feta-beta/mygrid/ontology/moby-feta/CanonicalDescriptionExample.rdf>

²<http://phoebus.cs.man.ac.uk:8100/feta-beta/mygrid/ontology/moby-feta/MygridServiceOntology.rdfs>

³<http://phoebus.cs.man.ac.uk:8100/feta-beta/mygrid/ontology/moby-feta/MobyServiceOntology.rdfs>

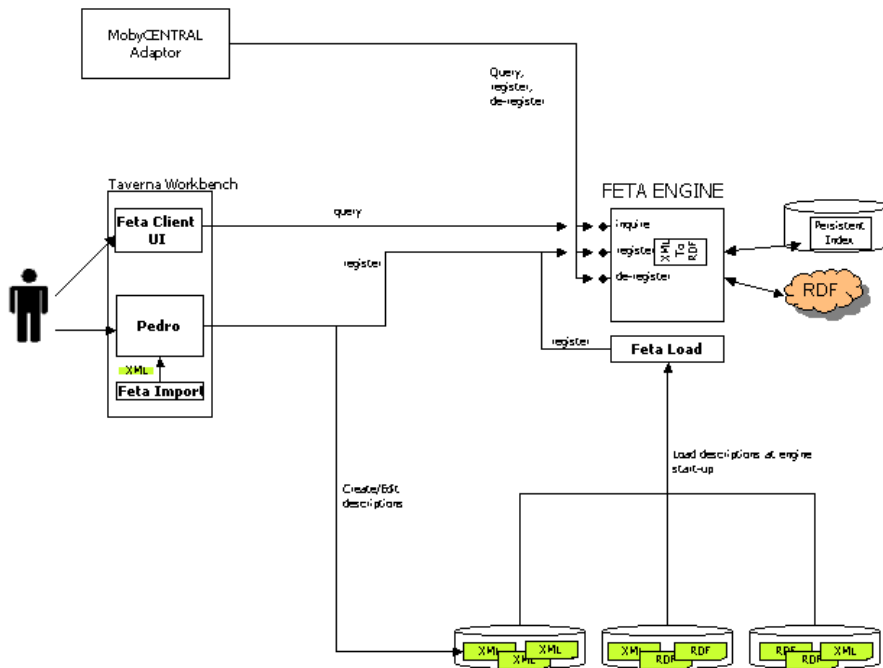


Figure 1: Architecture of Feta Engine 1.0

The canonical model, as it is, does not cover the constraints on parameters and other advanced service information, these aspects are to be considered in the later releases of the canonical model and associated registry.

2. A prototype back-end that supports the registration/de-registration of service descriptions wrt the canonical model has been developed
3. An ontology revision process has been initiated, the first results of which are listed in Section 5.1

The following section gives details on the design of the prototype backend mentioned in item 2, further open issues related to item 1 and item 3 are discussed in the TO-DO section.

3 Feta v.1.0

The overall architecture of the new Feta Engine within the context of *my*Grid components and MobyCENTRAL is given in Figure 1. There is no major architectural change in this framework from the existing Feta Framework. Still, the core functionality is delivered through a web service, which interacts with client side components. The interaction between MobyCENTRAL and Feta Engine is not any different than other clients of the engine.

Aspects of Feta Engine v1.0 that differentiates it from the existing Feta Engine are:

- It's canned queries operate over the canonical service description model.
- It now has a "Publish" interface through which services can be registered and de-registered.
- It supports registration of service descriptions in both RDF and XML formats. We are providing this hybrid approach to be able to relieve the Moby integration efforts from a dependency on availability of RDF generating annotation tools (this are needed within *myGrid*). Currently *myGrid* has an XML based annotation technology, which we hope to replace with an RDF based one. In order for this RDF based development to be a separate thread, we decided to keep the XML support within the Feta Engine until tool support for RDF is ready.

Each component within this architecture is described in the following subsections.

3.1 Feta Engine

The Feta Back-end provides methods that are in two categories, namely inquiry and publishing. The inquiry methods are

- A Canned Query method, that accepts search requests conforming to the *myGrid* service search query schema. The schema defines the following queries (*FindByInput*, *FindByOutput*, *FindByTask*, *FindByMethod*, *FindByApplication*, *FindByResource*, *FindByResourceContent*). The results of the Canned Query interface is the conjuncted list of signature URL's of service descriptions that match all of submitted canned queries.
- A freeform Query method, which accepts SeRQL queries executes them over the store and returns the result to the user. Note that the client is responsible for interpreting the results in the free form query case.

the publish interface methods are:

- Registration method that accepts a signature URL for the service to be registered and returns a result designating the outcome of registration, either a Success or a Failure. As we support a hybrid approach, the given Signature URL could be pointing out to an RDF based description or an XML based description. Feta Engine resolves this SignatureURL into the actual document, converts it into an RDF representation (if it is in XML) and stores it in its in-memory RDF model. The descriptions inside Feta Engine are indexed by their signature URLs, which is a persistent list managed by the engine. If for a reason the Feta Engine web service is re-started, each document listed in the index will be loaded at start-up.
- The de-registration (or delete) method, again, accepts a SignatureURL and removes the description indexed by this URL both from RDF model and from the persistent index.

(As a technical detail, I have used Sesame 2 for the back-end implementation, using its Contexts, and RDF(S) inferencing facilities.)

3.2 Feta Load

Feta Load is a utility component within the engine responsible for initializing the engine publishing all the descriptions harvested from a designated location. Feta Engine can be configured to load from multiple locations which could be web/local directories.

3.3 Feta Client-Side Components

Based on the decision to support a hybrid backend capable of service descriptions in both XML and RDF in Feta in v1.0, there will not be any major changes to the existing client side components that operate over XML based descriptions.

4 MobyCENTRAL - Feta Engine Interaction

As briefly stated in the previous section MobyCENTRAL would act as a client to the Feta Engine. The scenario with which a service provider registers a service to MobyCENTRAL and this registration is propagated to Feta Engine is depicted in Figure 2.

The steps could be summarized as follows

- Service Provider has RDF generated (from somewhere, including via XML interaction with moby-central)
- Service provider sends Signature URL to MobyCENTRAL.
- MobyCENTRAL messaging talks to agent, agent retrieves URL.
- Agent registers RDF document with messaging layer.
- Messaging layer gives the SignatureURL to adaptor.
- Adaptor contact Feta Engine and publishes this service by submitting the Signature URL.
- Adaptor implementation contacts Feta Engine's registration and de-registration methods as the Agent keeps it up-to-date on stale descriptions.

As a result of these interactions Feta will keep a copy of the RDF representation of services registered in MobyCENTRAL and this would enable MobyCENTRAL to farm out queries related to services to Feta.

5 Open Issues and TO DOs

5.1 Domain Ontologies of *my*Grid and BioMOBY

Katy Wolstencroft has been looking into the domain ontologies of two projects to identify overlapping reconcilable parts and differences. Her work has spotted the following which we would like to further discuss with the BioMOBY team.

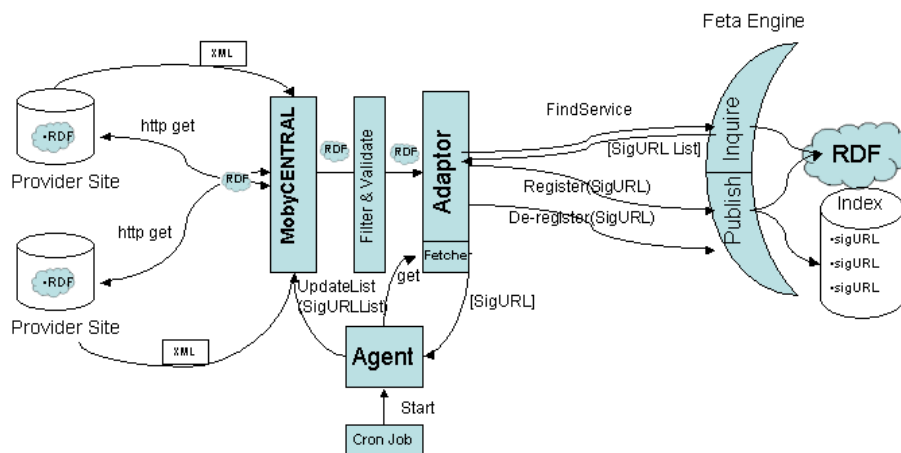


Figure 2: Interaction of MobyCENTRAL with Feta Engine

1. The Moby ObjectTypes ontology provides terms to describe generic Informatics aspects of objects, such as their types (e.g. text-plain) and formats (e.g. b64_ encoded_gif). This level of detail is certainly needed to fully describe services, however, until *my*Grid moves to an expert curator based solution for service annotation, we will not be using this portion of the Moby ObjectType ontology during the *my*Grid service annotation process.
2. The section listing different bioinformatics databases in the Moby Namespace ontology is largely overlapping with *my*Grid's list of bioinformatics resources. An ideal approach would be to reconcile these two.
3. There are certain classes within the Moby Namespace ontology such as "NCBI_gi" and "NCBI_AccVer" which are de-normalized and also contain type information in addition to Namespace info. If these classes are used describe the Namespace of an service parameter, which classes from the ObjectType ontology are used to describe the type, other than the most generic **mobyObject:Object** class -we see this in the example-)?
4. Do you have a mechanism to prevent the registration services parameters of which are described with an in-valid combination of Namespace and ObjectType? When the domain-type of a dataitem is denormalized into a Namespace and an ObjectType, it becomes possible to come up with combinations that are possibly invalid ? This is prevented in the existing de-normalized *my*Grid classes, as currently the annotator is forced to choose from a pre-defined list of combinations.

5.2 Canonical Model Coverage

One thing we have not discussed during the Hackaton is the service grounding information. Both projects have a user-oriented high level view of services and both have buried the grounding information into the middleware and associated

client side components. Therefore the canonical model does not provide grounding information. This brings about certain issues related to recognizability and executability of services discovered through the common registry. The issues and open questions could be listed as follows:

- How will MobyCENTRAL treat services that are discovered through Feta, that are not in its scope? Will it just ignore these results?
- Within *my*Grid the grounding information about services are stored in the Processor-Specifications. This is a Taverna specific piece of XML fragment designating services that Taverna can recognize and invoke (WSDL, Soaplab, Seqhound, BioMoby). This Processor-Spec goes into each service description that is generated by the Annotator from within Taverna. Currently, when a service is discovered through Feta, the client side components first check whether it has a Processor-Spec., if it does they use it to introduce this discovered service to Taverna. If it does not have a Processor-Spec, the client side component follows a best effort approach to re-generate this spec using certain information in the Semantic Description such as Service Type, Service/Operation Name, LocationURL, interfaceWSDL, mobyAuthoritative (if it is a moby service).