NffgService Design document

Aim of this document is to explain what have been the design choices of the server.

Wherever it has been possible, the rule followed is “information hiding”. Thanks to this I have tried to minimize the coupling between objects, while allowing flexibility.

For example, the main class of the server (NffgVerifierService) that is the REST front end to the clients, delegates the operation to an util class, NffgVerifierUtil. This class knows what are the operations to perform over the data, but it does not know from where the data comes, in fact it relies over a factory class to obtain the two source of data, the policies and the nffgs. In this way, for example, if we want to put the policies on a database, we just need to write a new class that implements the interface “PoliyDataSource” and tell the factory to reply with that class, all the other code can stay as already written.

Same principles apply for the nffgs, if we change the Neo4J server or we implement locally the logic to check the path, we just need a new class implementing “NffgDataSource”.

I defined three server exceptions, in order to take in account extraordinary cases. These exceptions have the same name of the ones used in the client (AlreadyLoadedException, UnknownNameException, ServiceException) only because I found that they were describing pretty well all the possible cases of non ordinary execution. These exceptions, in the REST front end, are converted in standard HTTP error messages and codes. Using the standard java ws library, I decided to use WebApplicationException and their extensions, as, for example, InternalServerErrorException.

The classes used to connect to the neo4j service are generated at build time with an ant task from the wadl, while the classes used for marshalling and unmarshalling request and response to the clients are generated from the local schema file “NffgVerifier.xsd”. As I decided to avoid splitting the schema in multiple files, I used an xjb file to annotate nearly each class I created with “XmlRootElement”, in order to allow the use of each of them as base request and response. Without that annotation, I should have had modified the response of the rest methods, because the standard message writer provider is not able to marshal a new messages using as root element a classes without “XmlRootElement”.

As final point, concurrency issues are dealt making sure that the data providers of policies and nffgs in the server are thread safe, using a ConcurrentHashMap as local storage.