DP2 2016/2017 Assignment 3 Web Service

Giacomo Ranieri

Contents

1 Resources Structure **2**

2 Resource Mapping **2**

3 Resource Operation **3**

3.1 nffgs 3

3.2 nffgs/{name} 3

3.3 nffgs/{name}/policies 4

3.4 /policies 5

3.5 /policies/{name} 5

3.6 /policies/{name}/verificationResult 6

3.7 /policy/verificationResult 6

4 Design and Implementation Choice **7**

Abstract

The web service for assignment 3 provide the possibility to store a set of nffgs and policies, and verify policies against nffgs.

1 Resources Structure

The resource structure is represented in this diagram:

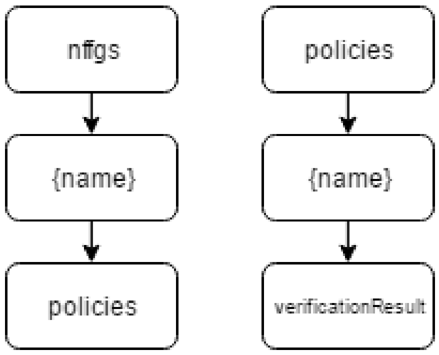

Figure 1: Resources structure diagram

Resource	**Allowed Method**
nffgs	GET / DELETE / POST
nffgs/{name}	GET / DELETE
nffgs/{name}/policies	GET / DELETE / POST
policies	GET / DELETE
policy/verificationResult	POST
policies/{name}	GET / DELETE / PUT
policies/{name}/verificationResult	POST

2 Resource Mapping

The resource are mapped, starting from the base path *http://localhost:8080/NffgService/rest* following URL:

/nffgs	The set of nffgs
/nffgs/{name}	The nffg identified by name
/nffgs/{name}/policies	The set of policy of the nffg identified by name
/policies	The set of policies

/policies/{name}	The policy identified by name
/policies/{name}/verificationResult	The verification result of the policy identified by name
/policy/verificationResult	The verification result of a policy not stored in the service

3 Resource Operation

The operations available for these resources are listed below:

3.1 nffgs

URL: http://localhost:8080/NffgService/rest/nffgs
These operations let you manage the nffgs collection. You can add a new NFFG, retrieve or delete the entire collection.

Method	Input	Code	Meaning	Output
POST	nffg			
		200	OK	nffg
		400	Bad Request	–
		500	Internal Server Error	
GET				
		200	OK	nffgs
		500	Internal Server Error	
DELETE				
		200	OK	nffgs
		500	Internal Server Error	

3.2 nffgs/{name}

URL: http://localhost:8080/NffgService/rest/nffgs/{name}
These operations let you manage a single nffg already in the service selected by means of its name, allowing to retrieve or delete it.

Method	**Path Param.**	**Code**	**Meaning**	**Output**
GET	name			
		200	OK	nffg
		404	Not found	
		500	Internal Server Error	
DELETE	name			
		200	OK	nffg
		404	Not found	
		500	Internal Server Error	

3.3 nffgs/{name}/policies

URL: http://localhost:8080/NffgService/rest/nffgs/{name}/policies
These operations let you manage the set of policies bounded to a particular nffg, selected by means of its name. You can add a new policy for this nffg, retrieve or delete all the policies bounded to the nffg. The POST operation let you add a new policy bounded to the nffg, is the only way to add a policy.

Method	**Path Param.**	**Input**	**Code**	**Meaning**	**Output**
POST	name	policy			
			200	OK	policy
			400	Bad Request	
			404	Not found	
			500	Internal Server Error	
GET	name				
			200	OK	policies
			404	Not found	
			500	Internal Server Error	
DELETE	name				
			200	OK	policies
			404	Not found	
			500	Internal Server Error	

3.4 /policies

URL: http://localhost:8080/NffgService/rest/policies
These operations let you manage the collection of policies stored in the service. You can retrieve or delete all the policies.

Method	**Code**	**Meaning**	**Output**
GET			
	200	OK	policies
	500	Internal Server Error	
DELETE			
	200	OK	policies
	500	Internal Server Error	

3.5 /policies/{name}

URL: http://localhost:8080/NffgService/rest/policies/{name}
These operations let you manage the a single policy stored in the service. You can update , retrieve or delete the policy selected by means of it's name. The put operation allow only the update of already existing policies.

Method	**Path Param.**	**Input**	**Code**	**Meaning**	**Output**
PUT	name	policy			
			200	OK	policy
			400	Bad Request	
			403	Forbidden	
			404	Not found	
			500	Internal Server Error	
GET	name				
			200	OK	policy
			404	Not found	
			500	Internal Server Error	
DELETE	name				
			200	OK	policy
			404	Not found	
			500	Internal Server Error	

3.6 /policies/{name}/verificationResult

URL: http://localhost:8080/NffgService/rest/policies/{name}/verificationResult
These operations let you verify one or more policy stored inside the service and retrieve their verificationResult.
To verify more policy the parameter ”names” must be used, listing the name of the other policy to verify.

Method	Path Param.	Query Param.	Code	Meaning	Output
POST	name:nameType	names:nameListType			
			200	OK	results
			400	Bad Request	
			404	Not found	
			500	Internal Server Error	
GET	name:nameType				
			200	OK	verificationResult
			404	Not found	
			500	Internal Server Error	

3.7 /policy/verificationResult

URL: http://localhost:8080/NffgService/rest/policy/verificationResult
This operation let you verify one or more policy not stored inside the service and obtain their verificationResult.
Using POST operation the list of policies to be verified is passed to the service, and a list of VerificationResult is returned.

Method	Input	Code	Meaning	Output
POST	policies			
		200	Success	results
		400	Bad Request	
		500	Internal Server Error	

4 Design and Implementation Choice

During the design and implementation of the service, I've made a few choice. As the service is only supposed to handle ReachabilityPolicy I've decide to use only a generic policy type in the interface, that inherit from the RechabilityPolicy type created in the first.
The new EnhancedPolicyType has also an optional attribute identifying the nffg which is bounded to, and is exploited when a policy is returned from the service. Is important also to notice that is not allowed to change the nffg to which a policy is bounded, you have to delete and re-add the policy.
This thought has also been used for the verification result management.
To store nffgs and policies I've used a ConcurrentHashMap with the name of the element as key, as used into the service description. This concurrent map let me operate with less forced synchronization. A map is also used to map Neo4J element, loaded during a policy insertion, to be used during policy verification. Two singleton service allows to operate on resources, one for service data, the other for Neo4J data. Java synchronization mechanism are used most during the management of policies, but also during the insertion of a new forwarding graph. As this is a long operation, due to the necessity to add all the nodes into Neo4J, is synchronized even if it would cause some performance issue, to avoid conflict during the long process, like the insertion of the same policy. A lot of synchronization is avoided as the delete of a graph is not implemented.
Policies are a more complicated type of data, they need synchronization during add, delete and update phases (also verification is treated as an update operation). Having decided to return the objects that involve each operation, is necessary to retrieve them, so during this operation, from retrieval to operation on the object the method are synchronized.
The data exchanged between the service and the clients, is of course in xml format and is managed in Java using jaxb framework and xjc generated classes. This class are common to both client and service.