REST SERVICE DESIGN (Assignment n.3 – part a)

	Resources	URLs	XML Repr.	Meaning
*	nffgs	nffgs	nffgs	set of all nffgs
1	{id}	nffgs/{id}	nffg	single nffg
*	policies	nffgs/{id}/policies	policies	policies belonging to nffg {id}

Resource	Method	Query params	Req. Body	Status	Resp. Body	Meaning
nffgs	GET	-	/	200	nffgs	OK
	POST	/	nffg	201	nffg	CREATED
				400	-	BAD request
nffgs/{id}	GET	-	/	200	nffg	OK
				404	-	NOT FOUND
	DELETE	force:xsd:boolean	/	204	-	NO CONTENT
				403	-	FORBIDDEN
				404	-	NOT FOUND
nffgs/{id}/policies	GET	-	/	200	policies	OK
				404	-	NOT FOUND
	DELETE	-	/	204	-	NO CONTENT
				404	-	NOT FOUND

- GET requests on *nffgs* resource allows clients to read information about all the NFFGS loaded on the service
- POST requests on *nffgs* allows clients to add a whole Nffg on the WebService.
  It returns FORBIDDEN if:
  - The name of the Nffg already exists on the Service
  - Two or more Links have same name on the Nffg
  - srcNode and dstNode of a Links does not belong to the Nffg
  - The function Type is not one of the function Types permitted.

  It returns a BAD REQUEST if:
  - The sintax of the nffg request body is not correct

- DELETE requests on *nffgs*/{id} allows clients to delete the {id} Nffg on the Service. Accepts the queryParameter *force*(boolean):
  - If the flag force is TRUE, the DELETE operation will force the removing of all the policies that refer to the nffg to be deleted.
  - If force is set to FALSE the DELETE operation will return FORBIDDEN if exists policies on the service that refer to the nffg to be deleted.

- GET requests on *nffg/{id}* resource allows clients to read information about the {id} NFFG loaded on the service

{id} is a unique name chosen by client.

	Resources	URLs	XML Repr.	Meaning
*	policies	policies	policies	set of all policies
1	{id}	policies/{id}	policy	single policy

Resource	Method	Query params	Req. Body	Status	Resp. Body	Meaning
policies	GET	-	/	200	policies	OK
	POST	/	policy	201	policy	CREATED
				400	-	BAD Request
				403	-	FORBIDDEN
policies/{id}	GET	-	/	200	policy	OK
				404	-	NOT FOUND
	DELETE	-	/	204	-	NO CONTENT
				404	-	NOT FOUND
	PUT		policy	200	policy	OK
				400	-	BAD REQUEST
				403	-	FORBIDDEN
				404	-	NOT FOUND

POST(and PUT) requests on *policies* return a BAD REQUEST if:

- The sintax of the policy request body is not correct:
  - If policyKind is set to TRAVERSAL and the list of traversedFuncType is empty
  - If miss the specification of TRAVERSAL OR REACHABILITY policyKind
  - The nffg reference is missing
  - Other fields are missing

POST(and PUT) requests on *policies* return FORBIDDEN if:

- The policy doesn't satisfy constraints about name.
- The nffg to which policy refer is not existent in the Service
- SrcNode and/or DstNode don't refers to existing nodes on the specified nffg.
- Others specifications are violated in the body request

{id}=unique name chosen by client.

	Resources	URLs	XML Repr.	Meaning
*	verifications	verifications	-	Root point to execute verifications
	volatile	verifications/volatile	-	Service point where to execute verification of NOT LOADED policies

Resource	Method	Query params	Req. Body	Status	Resp. Body	Meaning
verifications	GET	policyNames:stringList	/	200	policies	OK
				400	-	BAD REQUEST
				404	-	NOT FOUND
verifications/ volatile	GET	policyName:xsd:string policyKind:xsd:string policyLogic:xsd:string nffgName:xsd:string srcNode:xsd:string dstNode:xsd:string TraversedFunctions:stringList	/	200	policies	OK
				400	-	BAD REQUEST
				404	-	NOT FOUND

GET use example (request to execute verification of policy0 and policy1):
baseURL/verifications?policyName=Policy0&policyName=Policy1

GET verifications return a BAD REQUEST if:

- The sintax of the request is not correct:
  - Argument name is misspelled(e.g: policyname=”..” or pocyName=”…”);

GET verifications return a NOT FOUND if:

- Even just a policy passed by arguments are not found on the Service; no one policies are verified if this problem occurres.

GET verifications/volatile behaves exactly like the first one but it needs all the parameters that characterize a policy because it is not present on the Service and it must not be loaded too.

REST SERVICE IMPLEMENTATION (Assignment n.3 – part B)

In this WebService Implementation there are three root resource classes (NffgsResource,PoliciesResource and VerificationsResource) that represents the root resources discussed in the former part of this documentation; note that each entity is identified uniquely by the name specified by the client in each request instead of an ID chosen by the Service, so a possible successful response return resource's ID in its header this way : Location: http://localhost:8080/NffgService/rest/nffgs/Nffg100.

The methods associated with the ones on the root classes are implemented respectively on NffgsService, PoliciesService and VerificationsService classes.

NffgServiceFilter.java

Class designed to intercepts each kind of request from client to increment the static counter *requestNumber*.

NffgCache.Java

This is a cache memory implemented to save locally Nffgs already loaded successfully to the NEO4JXML and to retrieve more quickly information about these loaded NFFGs. It's implemented simply as an Nffgs collection Object. It provides the most relevant functions for adding and reading information on the cache memory.

PolicyDB.java

It maintains locally all the policies loaded by the client and provides methods for loading/deleting/uploading policies on the structure.

Neo4JClient.java

Provides the functionalities to interact directly with NEO4JXML as a Client to build Nffg by the form of nodes and relationships or to delete all nodes to reset the NffgSet already loaded.

NffgService.java

- The class provides loadNffg, getNffgs() and getNffg(String) mapped respectively to the POST and the GET on the /nffgs root resource and the GET on the /nffgs/{id} resource, all defined in NffgsResource.java.
- loadNffg(Nffg) performs the loading of a Nffg on the Neo4JXML Server acting as a client and saves it locally on the NffgCache only if the upload is successful (if the operation fails no data are stored in this memory). It uses *isWellFormed(Nffg) and NffgAdmitted(Nffg)* to check if the body of the Nffg received from the client satisfies the constraints to be respected and throw respectively a NotWellFormedException if the body of the request is considered incomplete or a ConstraintsViolatedException if the constraints are violated. NffgServiceException is launched if the operations acted on Neo4jXML are unsuccessful(for example a load of a Nffg).
  The former exceptions are catched in NffgResource and are converted respectively in BadRequest, Forbidden and InternalServerError exceptions.
- NffgService maintain the local cache (about the already loaded Nffgs) to retrieve quickly their information and provides them, if requested, through getNffg(String) and getNffgs() which interacts directly with the NffgCache.
- To perform the creation of relationships and then verificate reachability Policies, NEO4JXML requires its entities ID; to achieve this, NffgService fill(during a load of a Nffg) and use a memory (a Map of Maps indexed by nffgNames and provided by NffgsCache too) that maintains the correspondence between nodes and their NEO4JXML IDs for each Nffg. If during the load of a Nffg some error occurs and the operation fails, the data already stored locally about this mapping are deleted to restore the previous state of the Service while, the Nffg is loaded only In last in the nffgsCache, so only if all the operations go well.

PoliciesService.java

- Actually performs main operations on **PolicyDB.java**
- loadPolicy,getPolicies,getPolicy,updatePolicy,deletePolicy are mapped respectively to POST,GET on /policies and GET,PUT,DELETE on /policies/{id} resources, all defined in PoliciesResource.java.
- As in NffgService.java the methods isWellFormed(Policy) and isForbidden(Policy) are used in storePolicies() and updatePolicies() to check if the body of the Policy received from the client satisfies the constraints to be respected and they throw respectively a NotWellFormedException if the body of the request is considered incomplete and a ConstraintViolatedException if the constraints are violated.

VerificationsResource.java

- It allows the verification of a policy passed as queryParameter through a GET request;
  It is checked if a misspelling occurs on the parameter(BadRequest) and, if not, it performs the execute(policy) method defined in **VerificationService.java**. This throw a NotFoundException if the policy

passed as an argument is not found on the Service otherwise performs the verification and update the policy on Server with the updated verificationTime.

Assumptions

- Each entity is identified uniquely by the name specified by the client in each request instead of an ID chosen by the Service
- The constructors of NffgService, PoliciesService and VerificationsService are dedicated to check the value of the static counter *requestNumber* and to request the initialization of the NffgSet on the NEO4JXML Service if requestNumber is equal to 0 (NffgService just launched and so before the perform of any request arrived).
- Because of the non-persistency constraint, are used only static structures that at each deployment will be re-initialized.
- The verification of a reachability policy is synchronized to prevent any modification on the policy during the execute() operations .
- In general, in this Service the other GET operations are not synchronized despite the POST and PUT operations.
- The structures used to store Nffgs and Policies are concurrent implementations of HashMap to allow concurrent operations on the Service.
- To have more reliability about the requests, it is implemented a Validator in **NffgRequestsValidator.java**
- The Server uses the artifacts generated from the xml schema defined in the part a.
- The 2 clients uses instead artifacts generated from the wadl of the NffgService and They have their own copy to gain their independence.
- During the performing of a loadNffg the clients/requests take a lock of the two object interested in this operations at NffgsResource (nffgsCache/nodeIDsDB)