Luca Vacchetta
231819

REPORT

Neo4jManager

1. XML Schema

I define the xml schema for graph in this way:
The root element is graphs, which contains one or more elements called graph, each graph has a unique id (optional attribute) and contains an undefined number of other object called node. Each node has:

- Id: it unique identifies this node inside a specific graph, it is an optional attribute because this id, in general, is chosen by the server.
- Name: it unique identifies this node inside a specific graph, and it is used for reference this node.
- Functional type: it states what is the role of this node inside the graph, and in particular it has as value one of those defined into the functional types enumeration, so are admitted:
  - ACL_FIREWALL
  - END_HOST
  - ANTISPAM
  - CACHE
  - DPI
  - MAILCLIENT
  - MAILSERVER
  - NAT
  - VPN_ACCESS
  - VPN_EXIT
  - WEBCLIENT
  - WEBSERVER
- List of neighbours: each of these neighbours have:
  - Id: which unique identifies the neighbour belonging to a specific node, also this id is optional.
  - Name: this field reference to another node, so this is the name of another node present in this graph.

I also define three schema for query response, so:

1. **Paths always contains:**
   - Source: it states the name of the source node.
   - Destination: it states the name of the destination node.
   - Direction: it states in which direction this search is done.
   - And one of these two fields:
     - Message: it is present, if there aren't path between source and destination nodes.
     - Path: it list all available paths, if there is at least one present.

2. **Reachability always contains:**
   - Source: it states the name of the source node.
   - Destination: it states the name of the destination node.
   - Direction: it states in which direction this search is done.
   - Result: it is a Boolean value which states if there is reachability or less between source and destination nodes.

3. **Policy contains:**
   - Source: it states the name of the source node.
   - Destination: it states the name of the destination node.

- Middlebox: it states the name of an intermediate node between source and destination nodes, furthermore this attribute is optional because is useful only for traversal and isolation policies.
- Type: it states which kind of policy has been verified, so this field can have one of these values:
    - Reachability.
    - Traversal.
    - Isolation.
- Result: it is a Boolean value which states the result of this verification.

2. Code Organization

For this project I split the code in five packages:

1) **it.polito.nffg.neo4j.manager:**

In this package there are all the interactions with the Neo4j Database, in fact we found the java interface which contain these methods:

```
public void createGraphs(Graphs graphs);
public Graph createGraph(Graph graph);
```

These two methods respectively create more than one graph and single graph, both throw MyNotFoundException if one neighbour doesn't exist.

```
public Node createNode(Node node, long graphId);
```

This method create a single node and it throws MyNotFoundException if it doesn't found the graph with id graphId and DuplicateNodeException if there is already inside the graph another node with the same name.

```
public Neighbour createNeighbour(Neighbour neighbour, long graphId, long nodeId);
```

This method add a neighbour to a node with id nodeId, and it throws MyNotFoundException if it doesn't found the graph (with id graphId) or the node (with id nodeId).

```
public Graphs getGraphs();
```

Return a list of Graph i.e. all graphs which have been stored into neo4j Database.

```
public Graph getGraph(long id);
```

Return a Graph with Id id and it throws MyNotFoundException if it doesn't found this graph.

```
public Set<Node> getNodes(long graphId);
```

Return a set which contain all nodes of the graph with id graphId and it throws MyNotFoundException if it doesn't found this graph.

```
public Node getNode(long graphId, long nodeId);
```

Return a node with id nodeId belonging to a graph with id graphId and it throws MyNotFoundException if it doesn’t found the graph or the node.

```
public Set<Neighbour> getNeighbours(long graphId, long nodeId);
```

Return a set of all neighbours of node with id nodeId belonging to a graph with id graphId, and it throws MyNotFoundException if it doesn’t found the graph or the node.

```
public Neighbour getNeighbour(long graphId, long nodeId, long neighbourId);
```

Return a specific neighbour with id neighbourId of a node with id nodeId belonging to graph with id graphId and it throws MyNotFoundException if it doesn’t found the graph and/or the node and/or the neighbour.

```
public void deleteGraph(long id);
```

This method deletes a graph with Id id if it is present inside the database, otherwise it throws MyNotFoundException.

```
public void deleteNode(long graphId, long nodeId);
```

This method deletes a node with id nodeId belonging to a graph with id graphId if it is present otherwise it throws MyNotFoundException.

```
public void deleteNeighbour(long graphId, long nodeId, long neighbourId);
```

This method deletes a neighbour with id neighbourId belonging to a specific node with id nodeId inside the graph with id graphId, it throws MyNotFoundException if it doesn’t found this neighbour.

```
public Node updateNode(it.polito.nffg.neo4j.jaxb.Node node, long graphId, long nodeId);
```

This method update a specific node with id nodeId belonging to a graph with id graphId, i.e. it allows to rename this node, change its functional type and add some neighbours. It throws MyNotFoundException if it doesn’t found this node and MyInvalidObjectException if the node passed as argument doesn’t respect the node constraints.

```
public Graph updateGraph(Graph graph, long graphId);
```

This method update a specific graph with id graphId belonging, i.e. it allows to add/rename some nodes, change their functional types and add some neighbours. It throws: MyNotFoundException if it doesn’t found this graph, MyInvalidObjectException if the new graph doesn’t respect the node constraints and DuplicateNodeException if in graph passed as argument there are one or more duplicate node (i.e. with the same name).

```
public Node updateNeighbour(Neighbour neighbour, long graphId, long nodeId, long neighbourId);
```

This method update a specific neighbour with id neighbourId belonging to a specific node with id nodeId inside the graph with id graphId, i.e. it allows to change the end node of this relationship. It throws MyNotFoundException if it doesn’t found this neighbour and MyInvalidObjectException if the neighbour passed as argument doesn’t respect the node constraints.

```
public Reachability checkReachability(long graphId, String srcName, String dstName, String direction);
```

This method check if two nodes whose names are srcName and dstName belonging to a graph with id graphId are connected in a direction specify as fourth argument (possible directions are: OUTGOING, BOTH). It throws MyNotFoundException if it doesn't found at least one of these nodes.

```
public Paths findAllPathsBetweenTwoNodes(long graphId, String srcName, String dstName,
        String direction);
```

This method return all possible path, without loops, between two nodes whose names are srcName and dstName belonging to a graph with id graphId in a direction specify as fourth argument (possible directions are: OUTGOING, BOTH). It throws MyNotFoundException if it doesn't found at least one of these nodes.

All the concurrency issues are solved because the implementation of these methods use transaction, which are thread safe.

2) **it.polito.nffg.neo4j.exceptions:**

In this package there are all Exceptions which I define inside this project, in details there are:

- DuplicateNodeException:
    - It is raised when the software detect a duplicate node.
- MyInvalidDirectionException:
    - It is raised when someone pass as direction a wrong Direction (are only admit BOTH and OUTGOING).
- MyInvalidIdException:
    - It is raised when someone pass as Id not a number.
- MyInvalidObjectException:
    - It is raised when an Object (Graph, Node, Neighbour) doesn't respect its constraints.
- MyNotFoundException:
    - It is raised when some Graph/Node/Neighbour is not found.

3) **it.polito.nffg.neo4j.jaxb:**

In this package there all annotated classes generated by XJC starting from xml_components.xsd (xml schema).

4) **it.polito.nffg.neo4j.service:**

In this package there is a Service class, that contains all methods which are useful in more than one package.

5) **it.polito.nffg.neo4j.resources:**

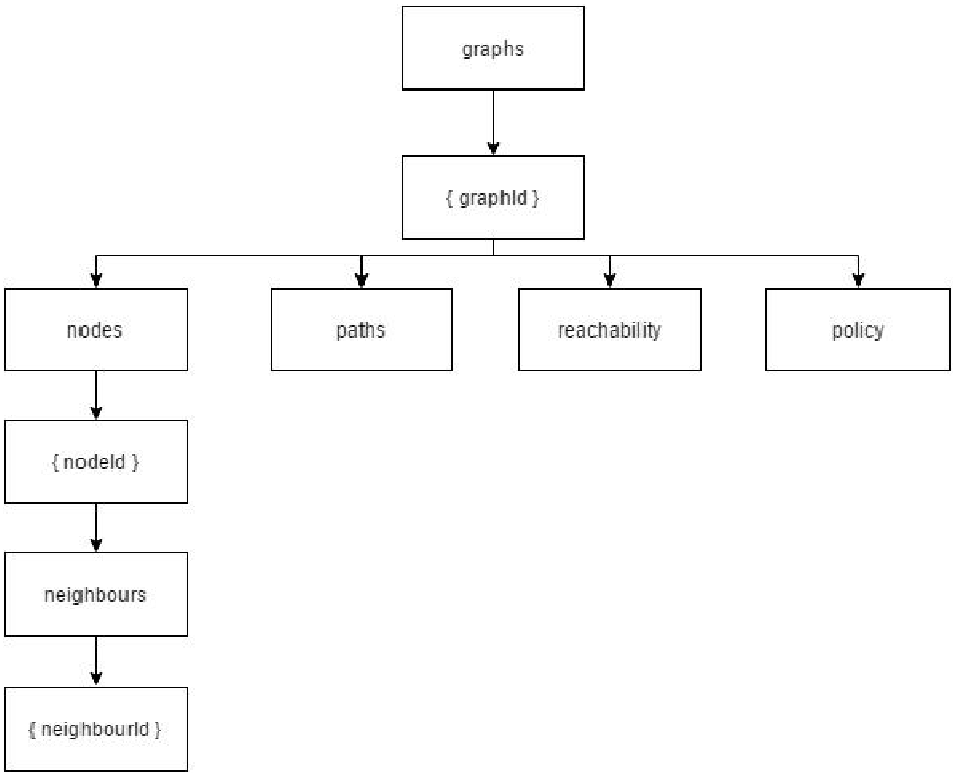
In this package there is GraphResource class which contains the implementation of the REST interface.

Conclusion:

- ✓ I split the code in this way in order to have a better separation of tasks, so the code is easier to maintain because is faster to find the piece of code to modify.

3. REST Interface

The REST interface respect this logical schema:



In details these are the operations allowed:

1) On the graphs resource:

HTTP Method	URL	Description
POST	/graphs	Creates a new graph
GET	/graphs	Retrieves all graphs
GET	/graphs /{graphId}	Retrieves a graph
PUT	/graphs /{graphId}	Updates a graph
DELETE	/graphs /{graphId}	Deletes a graph
GET	/graphs /{graphId} /policy ?source=*nodeA* &destination=*nodeB* &type=*policy* *&middlebox=nodeC*	Performs a verification according to three mandatory query parameters: source, the name of the source node, destination, the name of the destination node and type, the policy to be verified between the given source and destination. A fourth query parameter named middlebox is required in case isolation or traversal was chosen as type.
GET	/graphs /{graphId} /paths ?src=*nodeA* &dst=*nodeB* &dir=*direction*	Performs a verification according to three mandatory query parameters: src, the name of the source node, dst, the name of the destination node and dir, the direction of these paths. So this method return all paths between nodeA and nodeB without loops.

GET	/graphs /{graphId} /reachability ?src=*nodeA* &dst=*nodeB* &dir=*direction*	Performs a verification according to three mandatory query parameters: src, the name of the source node, dst, the name of the destination node and dir, the direction of these paths. So this method return a Reachability object which indicates the reachability between nodeA and nodeB.

Operation	Status Codes	Reason	Response
POST **/graphs**	201	Graph created successfully	Graph object
	400	Invalid graph	
	500	Internal Server Error	
GET **/graphs**	200	OK	All graph objects
	500	Internal Server Error	
GET **/graphs** **/{graphId}**	200	OK	Graph object
	403	Invalid graph id	
	404	Graph not found	
	500	Internal Server Error	
PUT **/graphs** **/{graphId}**	200	OK	Updated graph object
	400	Invalid graph object	
	403	Invalid graph id	
	404	Graph not found	
	500	Internal Server Error	
DELETE **/graphs** **/{graphId}**	204	OK	Empty
	403	Invalid graph id	
	500	Internal Server Error	
GET **/graphs** **/{graphId}** **/policy** **?source=nodeA** **&destionation=nodeB** **&type=policy** **&middlebox=nodeC**	200	OK	Policy object
	403	Invalid graph id	
	404	Graph not found	
	500	Internal Server Error	
GET **/graphs** **/{graphId}** **/paths** **?src=nodeA** **&dst=nodeB** **&dir=direction**	200	OK	Paths object
	400	Invalid graph id and/or direction	
	404	Graph not found	
	500	Internal Server Error	
GET **/graphs**	200	OK	Reachability object

/{graphId} /reachability ?src=nodeA &dst=nodeB &dir=direction			
	400	Invalid graph id and/or direction	
	404	Graph not found	
	500	Internal Server Error	

<span style="color:red">**2) On the nodes resource:**</span>

HTTP Method	URL	Description
POST	/graphs /{graphId} /nodes	Creates a new node in graph with id graphId.
GET	/graphs /{graphId} /nodes	Retrieves all nodes of graph with id graphId.
GET	/graphs /{graphId} /nodes /{nodeId}	Retrieves node with id nodeId of graph with id graphId.
PUT	/graphs /{graphId} /nodes /{nodeId}	Updates node with id nodeId of graph with id graphId.
DELETE	/graphs /{graphId} /nodes /{nodeId}	Deletes node with id nodeId of graph with id graphId.

Operation	Status Codes	Reason	Response
POST **/graphs /{graphId} /nodes**	201	Node created successfully	Node object
	400	Invalid node	
	403	Invalid graph id	
	404	Graph not found	
	500	Internal Server Error	
GET **/graphs /{graphId} /nodes**	200	OK	All node objects
	403	Invalid graph id	
	404	Graph not found	
	500	Internal Server Error	
GET **/graphs /{graphId} /nodes /{nodeId}**	200	OK	Node object
	403	Invalid graph and/or	

		node id	
	404	Graph and/or node not found	
	500	Internal Server Error	
PUT **/graphs /{graphId} /nodes /{nodeId}**	200	OK	Node object
	400	Invalid node	
	403	Invalid graph and/or node id	
	404	Graph and/or node not found	
	500	Internal Server Error	
DELETE **/graphs /{graphId} /nodes /{nodeId}**	204	OK	Empty
	403	Invalid graph and/or node id	
	404	Graph and/or node not found	
	500	Internal Server Error	

3) On the neighbours resource:

HTTP Method	URL	Description
POST	/graphs /{graphId} /nodes /{nodeId} /neighbours	Creates a new neighbour of node with id nodeId in graph with id graphId.
GET	/graphs /{graphId} /nodes /{nodeId} /neighbours	Retrieves all neighbours of node with id nodeId in graph with id graphId.
GET	/graphs /{graphId} /nodes /{nodeId} /neighbours /{neighbourId}	Retrieves neighbour with id neighbourId of node with id nodeId in graph with id graphId.
PUT	/graphs /{graphId} /nodes /{nodeId} /neighbours /{neighbourId}	Updates neighbour with id neighbourId of node with id nodeId in graph with id graphId.
DELETE	/graphs /{graphId} /nodes /{nodeId} /neighbours /{neighbourId}	Deletes neighbour with id neighbourId of node with id nodeId in graph with id graphId.

Operation	Status Codes	Reason	Response
POST **/graphs/{graphId}/nodes/{nodeId}/neighbours**	201	Neighbour created successfully	Neighbour object
	400	Invalid neighbour	
	403	Invalid graph and/or node id	
	404	Graph and/or node not found	
	500	Internal Server Error	
GET **/graphs/{graphId}/nodes/{nodeId}/neighbours**	200	OK	All neighbours objects
	403	Invalid graph and/or node id	
	404	Graph and/or node not found	
	500	Internal Server Error	
GET **/graphs/{graphId}/nodes/{nodeId}/neighbours/{neighbourId}**	200	OK	Neighbour object
	403	Invalid graph and/or node and/or neighbour id	
	404	Graph and/or node and/or neighbour not found	
	500	Internal Server Error	
PUT **/graphs/{graphId}/nodes/{nodeId}/neighbours/{neighbourId}**	200	OK	Node object
	400	Invalid neighbour object	
	403	Invalid graph and/or node and/or neighbour id	
	404	Graph and/or node and/or neighbour not found	
	500	Internal Server Error	
DELETE **/graphs/{graphId}/nodes/{nodeId}/neighbours/{neighbourId}**	204	OK	Empty

	403	Invalid graph and/or node and/or neighbour id
	404	Graph and/or node and/or neighbour not found
	500	Internal Server Error