Distributed Programming II

A.Y. 2017/18

Assignment n. 3 – part b)

All the material needed for this assignment is included in the *.zip* archive where you have found this file. Please extract the archive to the same working directory where you have already extracted the material for part a) and where you will work. In this way, you should have the xsd and HTML documentation developed in part a) in the `[root]/xsd` and `[root]/doc` folders respectively.

This second part of the assignment consists of three sub-parts:

1. Write a simplified Java implementation (using the JAX-RS framework) of the web service designed in part a). This simplified implementation must implement only functionalities 1, 2, 4 (without removal) and 5 (without removal), as specified in part a). Accordingly, some of the operations that are possible in your design may be disabled in this implementation. If one of these operations is invoked, the service must respond with a 501 (Not Implemented) status code.

The web service has to be packaged into a war archive that can be deployed to Tomcat. This is done by the ant script `[root]/build.xml`, which includes a target named `package-service` that builds and packages the service. The name of the service packaged in this way will be "NfvDeployer", and its base URL will depend on the configuration of Tomcat (it will be http://localhost:8080/NfvDeployer/rest on the Labinf machines). A WADL description will be available at the standard Jersey-provided WADL location (on the Labinf machines, this is http://localhost:8080/NfvDeployer/rest/application.wadl). Of course the service must have no dependency on the actual location of Tomcat.

The service does not have to provide data persistency but has to manage concurrency, i.e. several clients can operate concurrently on the same service without restrictions.

When deployed, NfvDeployer must be initialized by reading the data about the IN (i.e. hosts and relative connections) and the VNF catalog from the NfvReader interface already used for Assignment 1 (defined in package `it.polito.dp2.NFV`), using the same abstract factory already used for Assignment 1. At startup, the service must also deploy the NF-FG named "Nffg0" (which is always among the ones returned by NfvReader).

The service must exploit the Neo4JSimpleXML service (the one already used for Assignment 2) in order to store the NF-FG nodes and links and their allocation relationships, in the same way done for Assignment 2 (if this is not possible for some reason, e.g. service not reachable, a failure has to be reported; for example, if the deployment of an NF-FG is requested and the NF-FG nodes links and allocation relationships cannot be fully stored in the Neo4JSimpleXML, the operation must fail); when reachability information has to be returned (point 1 of the functionality list), this has to be done by exploiting the Neo4JSimpleXML service, as done in Assignment 2 (if the operation that Neo4JSimpleXML is requested to perform fails for any reason, the corresponding operation will fail in NfvDeployer).

NfvDeployer has to read the actual base URL of the Neo4JSimpleXML service as the value of the system property `it.polito.dp2.NFV.lab3.Neo4JSimpleXMLURL`. If the property is not set, the default value to be used is http://localhost:8080/Neo4JSimpleXML/rest.

The service must be developed entirely in package `it.polito.dp2.NFV.sol3.service`, and the corresponding source code must be stored under `[root]/src/it/polito/dp2/NFV/sol3/service`.

Write an ant script that automates the building of your service, including the generation of any

necessary artifacts. The script must have a target called `build-service` to build the service. All the class files must be saved under `[root]/build`. Customization files, if necessary, can be stored under `[root]/custom`, while XML schema files can be stored under `[root]/xsd`. Of course, the schema file developed in part a) can also be used (it is stored in `[root]/xsd`, as specified previously).
The ant script must be named `sol_build.xml` and must be saved directly in folder `[root]`.
Important: your ant script must define `basedir="."` and all paths used by the script must be under `${basedir}` and must be referenced in a relative way starting from `${basedir}`. Any other files that are necessary and that are not included in the other folders should be saved in `[root]/custom`.

The `package-service` target available in `build.xml` calls the `build-service` target of your `sol_build.xml` and then packages the service into the archive `[root]/war/NfvDeployer.war`. The descriptor of the service used for the package is under `WebContent`. The contents of this folder can be customized by you. The deployment of the package to the Tomcat server can be done by calling the `deployWS` target available in the `build.xml` script. Note that this target re-builds the war and also (re-)deploys Neo4JSimpleXML. Before running this target, you need to start Tomcat. This has to be done by calling the `start-tomcat` ant target on `build.xml` (which also sets the expected system properties; do not run Tomcat in another way otherwise the system properties may not be set correctly).
Complete the documentation you already developed for part a) by adding another HTML document that illustrates the most relevant implementation choices you made (including any assumptions you made). Limit the size of this document to 1 page. Store this document in the same `[root]/doc` folder where you already put the other one, and use the name `design.html` for its entry point. Add the contents of all your `[root]/doc` folder to the `WebContent/WEB-INF` folder, so that it can be browsed from Tomcat when the service is deployed.

2. Implement a client for the web service designed in part a) and implemented in part b).1. This client must take the form of a Java library that implements the interface `it.polito.dp2.NFV.lab3.NfvClient`, available in source form in the package of this assignment (its documentation is included in the source file).

The library to be developed must include a factory class named `it.polito.dp2.NFV.sol3.client1.NfvClientFactory`, which extends the abstract factory `it.polito.dp2.NFV.lab3.client1.NfvClientFactory` and, through the method `newNfvClient()`, creates an instance of your concrete class that implements the `NfvClient` interface.

The client must be developed entirely in package `it.polito.dp2.NFV.sol3.client1`, and the source code for the client must be stored under `[root]/src/it/polito/dp2/NFV/sol3/client1`. The actual base URL of the web service to be used by the client must be obtained by reading the `it.polito.dp2.NFV.lab3.URL` system property. If this property is not set, the default URL http://localhost:8080/NfvDeployer/rest/ must be used. Note that your client cannot even assume that port 8080 is used.

In your `sol_build.xml` file, add a new target named `build-client` that automates the building of your client, including the generation of artifacts if necessary. All the class files must be saved under `[root]/build`. You can assume that Tomcat is running with NfvDeployer already deployed when the `build-client` target is called (of course, the target may fail if this is not the case). In your script, you can assume that Tomcat is available on the localhost, with the port number

specified by the ant property PORT (inherited automatically by your script). Do not use hard-coded values for the port. Customization files, if necessary, can be stored under `[root]/custom`.

3. Implement another client for the web service, which provides information about NF-FGs and hosts. The client must take the form of a library similar to the one implemented in Assignment 1. The library must load information about NF-FGs and hosts from the web service at startup. The library must implement all the interfaces and abstract classes defined in package `it.polito.dp2.NFV`. The classes of the library must be entirely in package `it.polito.dp2.NFV.sol3.client2` and their sources must be stored in `[root]/src/it/polito/dp2/NFV/sol3/client2`. The library must include a factory class named `it.polito.dp2.NFV.sol3.client2.NfvReaderFactory`, which extends the abstract factory `it.polito.dp2.NFV.NfvReaderFactory` and, through the method `newNfvReader()`, creates an instance of your concrete class implementing the `NfvReader` interface. The actual base URL of the web service to be used for getting the information must be obtained by reading the `it.polito.dp2.NFV.lab3.URL` system property. If this property is not set, the default URL http://localhost:8080/NfvDeployer/rest/ must be used.

Update your `sol_build.xml` file, so that the target named `build-client` also builds your second client. All the class files must be saved under `[root]/build`. You can make the same assumptions about Tomcat location made for the client developed at point 2, and you can save customization files, if necessary, in the same location.

The client library classes developed in points 2 and 3 must be robust and interoperable, without dependencies on locales. However, these classes are meant for single-thread use only, i.e. the classes will be used by a single thread, which means there cannot be concurrent calls to the methods of these classes.

Correctness verification

Before submitting your solution, you are expected to verify its correctness and adherence to all the specifications given here. In order to be acceptable for examination, your assignment must pass at least all the automatic mandatory tests. Note that these tests check just part of the functional specifications! In particular, they only check that:

- the submitted schema is valid (you can check this requirement by means of the Eclipse XML schema validator);
- the implemented web service and clients behave as expected, in some scenarios: after deploying an NF-FG and adding nodes or links (by means of your client1), the information returned by your service and received by your client2 is consistent with the performed operations.

The same tests that will run on the server can be executed on your computer by issuing the following command

```
ant -Dseed=<seed> run-tests
```

Note that this command also deploys your service and the Neo4JSimpleXML service. However, before running this command you have to start Tomcat and Neo4J on your local machine, as explained for Assignment 2.

Other checks and evaluations on the code will be done at exam time (i.e. passing all tests does not guarantee the maximum of marks). Hence, you are advised to test your program with care (e.g. you can also test your solution by running your service and then test it by using your clients in different scenarios and by means of other general-purpose clients).

Submission format

A single *.zip* file must be submitted, including all the files that have been produced. The *.zip* file to be submitted must be produced by issuing the following command (from the `[root]` directory):

```
ant make-zip
```

In order to make sure the content of the zip file is as expected by the automatic submission system, do not create the *.zip* file in other ways.