Distributed Programming II

A.Y. 2017/18

Assignment n. 1 – part a)

The material for this assignment is in the *.zip* archive where you have found this file. Extract the archive to an empty directory that you will use as your working area and that we will call `[root]`.

DP2-NFV is a (simplified) distributed system that provides *Network Function Virtualization (NFV)*. The users of this system can define a desired virtual network service in the form of a Network Function Forwarding Graph (NF-FG), and request that this virtual service is deployed to the physical Infrastructure Network (IN) managed by DP2-NFV. The NF-FG and IN concepts and terminology used in the DP2 assignments is explained in the document file `[root]/intro.pdf`.

The Java interfaces defined in the package `it.polito.dp2.NFV` (available under `[root]/doc` and at https://pad.polito.it:8080/enginframe/dp2/assignments/lab1/doc/index.html) give read-only access to some information about the current status of the DP2-NFV system. **The javadoc of the interfaces documents the kind of information that can be retrieved**. The main interface, from which all the information can be accessed, is `NFVReader`. The methods in this interface can be used to get the information available about known NF-FGs deployed into the system (`getNffg` and `getNffgs`), hosts that make up the IN managed by the system (`getHost` and `getHosts`), performance features of the connections available between hosts in the IN (`getConnectionPerformance`), and catalog of virtual functions available in the system. These methods return sets of interfaces, by which all the available information about the returned NF-FGs, hosts, etc., can be obtained.

The solution to part a) will be submitted along with the solution to part b).

Assignment description

1. Design an *XML* application that can be used to store all the information that can be retrieved by using the *Java* interfaces defined in package `it.polito.dp2.NFV`, starting from interface `NfvReader`. The XML format must be such that all the data that can be retrieved using such interfaces can also be obtained from the *XML* document, and redundancies should be avoided. The *XML* application must be specified by means of a *DTD*, which must be saved in file `[root]/dtd/nfvInfo.dtd`, and by means of an *XML Schema*, which must be saved in file `[root]/xsd/nfvInfo.xsd`. The potential of XML schemas must be exploited, in order to accurately represent constraints on data types, and keep checks as much as possible within the XML processor (rather than in the application).
2. Write a short documentation of your design choices about the DTD and the schema (max 1 page) and save it as an ASCII file in `[root]/doc.txt`.
3. Write a valid *XML* file that references the designed *DTD* locally. The file must be saved as `[root]/dtd/nfvInfo.xml`.
4. Write a valid *XML* file that references the designed *schema* locally. The file must be saved as `[root]/xsd/nfvInfo.xml`

Correctness verification

Before submitting your files, please verify their correctness. The submitted solution must at least satisfy the following requirements, in order to be considered acceptable:

- the file `[root]/dtd/nfvInfo.dtd` must be syntactically correct and the file `[root]/xsd/nfvInfo.xsd` must be a valid XML Schema;
- the file `[root]/dtd/nfvInfo.xml` must be valid and must reference the DTD stored in file `nfvInfo.dtd`, in the same directory as the xml file.
- the file `[root]/xsd/nfvInfo.xml` must be valid with respect to the schema `nfvInfo.xsd` and must reference it, assuming it is stored in the same directory;
- the file `doc.txt` must exist.

The validity of the XML file can be checked by any XML validation program. For example, it can be checked by the Eclipse validate command or by running the `DomParseV` program (distributed in the *XML* examples bundle on the course site). Similarly, it is possible to check the validity of the XML file with respect to the schema and the validity of the schema.