Distributed Programming II

A.Y. 2018/19

Assignment n. 3 – part a)

All the material needed for this assignment is included in the *.zip* archive where you have found this file. Please extract the archive to an empty directory (that will be called `[root]`) where you will work.

This assignment is about the design of a RESTful API for the DP2-RNS system considered in the previous assignments. The web service has to be designed according to the following specifications.

1. There are two types of clients of the system: administrators and vehicles.
2. A vehicle that wants to enter the system has to request permission to the system. In the permission request the vehicle must specify the entrance (IN or INOUT) gate where it is, and the destination place it wants to go to, in addition to the information about the vehicle itself (plate id and vehicle type). The system may grant permission or not. If the destination is not reachable, the system does not grant permission, but there may be other reasons (e.g. congestion or other system policies or vehicle already in the system) for not granting permission. If the permission is granted, the system computes a suggested path to the destination, adds the vehicle to the set of vehicles that are tracked in the system, with its current position set at the entrance gate and its initial state set to IN_TRANSIT, and communicates the suggested path to the vehicle. If instead the permission is not granted, the system communicates the reason for not granting the access to the vehicle, and the vehicle is not added to the set of vehicles that are tracked in the system.
3. Whenever a tracked vehicle moves from a place to another one, it informs the system about the new place where it is, and the system records the new current position of the vehicle. If the new place is not on the path suggested by the system for that vehicle, the system first checks if the new place is reachable from the previous current position of the vehicle. If so, it computes a new path from the new current position of the vehicle to the destination, and communicates this new path to the vehicle. If the path cannot be computed (e.g. because the destination is not reachable from the new current place), the vehicle remains without a suggested path. Finally, if the new place is not reachable from the previous current position of the vehicle, the request from the vehicle is considered wrong by the system, and nothing changes.
4. When a tracked vehicle is in an OUT or INOUT gate, it can decide to exit from the system through that gate. When doing so, it contacts the system and communicates to the system that it has left the system. Upon receiving this information, the system removes the vehicle from the set of vehicles that are tracked in the system and forgets about it.
5. A tracked vehicle can request the system to change its state at any time. The request is always accepted by the system.
6. An administrator must be able to read all the information available in DP2-RNS about places and tracked vehicles in the system. The information to be made available for reading is the same that was defined for Assignment 1, plus information about the suggested path for each tracked vehicle in the system. The administrator should be able to easily get the information about the vehicles that are in a given place.
7. At any time, an administrator can request to remove a vehicle from the set of vehicles that are tracked in the system and the request is always accepted by the system.
8. The service must support both XML and JSON media types for all the operations.

9. For simplicity, the service is made available without authentication.

The output of this design phase has to be documented in the following set of documents: an XML schema, which describes the data types used by the service, an HTML document, which is the documentation of the RESTful API for the users, and another HTML document, which is the explanation of the design decisions made and of how they implement the guidelines and best practices taught in the course. The schema has to be stored in the file `[root]/xsd/RnsSystem.xsd`, the HTML user documentation of the API has to be stored entirely into the folder `[root]/doc/user`, and made accessible starting from the file `[root]/doc/user/index.html`, and the HTML design documentation must be stored entirely into the folder `[root]/doc/design`, and made accessible starting from the file `[root]/doc/design/index.html`. If deemed useful, the schema can be split into multiple files. In this case, `RnsSystem.xsd` must be the main file which includes the other local files, and all files must be stored into `[root]/xsd`.

The HTML user documentation of the service must include at least the following items:

1. Description of the conceptual structure of resources (identify the resources, with their hierarchical structure and meaning; use plural names for collections and singular names for resources that are not collections; specify the HTTP methods allowed for each resource).
2. Mapping of the resources to URLs (use URLs that are relative to the base URL of the service; for resources that belong to collections, use the notation {id}, where id is the name of the field that identifies the resource in the collection)
3. Description of each possible operation. For each resource, and for each method allowed on the resource, specify at least the allowed query string parameters and request body, the possible status codes, and, for each possible status code, the response body if any. For each allowed query string parameter, specify name and XML schema type of the value (either a standard XML schema simple type or one defined by you in the schema document). For request body and response body, specify the XML element, which must be defined at the global level in the schema document. Schema elements and types in this description must be referenced by name. In practice, the schema document must include a set of global named types and a set of global elements. As far as possible, the definitions already developed in the schema for Assignment 1 should be reused. For each possible operation, a short description should also be added when not obvious, and supported HTTP headers should also be specified.

The web service must be designed so as to be compliant with the REST principles and conventions, robust, easy to be used by clients, and enabling efficient execution of all the operations for which it has been designed. The HTML documentation should be self-contained (it should include all that is necessary for a user to know to be able to consume the service, apart from what can be deduced from the service itself and from the HTTP standard).

When designing the service, consider that it should be able to scale up to world size.

Correctness verification

In order to be acceptable for examination, the produced XML schema document must be valid. This can be checked by means of the Eclipse schema validator.

Submission format

The produced documents will be submitted along with Part b) of Assignment 3, for which instructions will be given later on.