Software Development Methods
Lab 11: Object Constraint Language

The Software Development Methods Team

November, 2018

1 Construct a UML Class diagram for the following requirements:

An international airport requires a system to keep track of flight details for customers. For each flight the system needs to store the flight number, destination, departure time, departure gate, airline and flight cost. Some flights are direct flights, i.e. they fly non-stop to the destination and some fly via another airport to their destination. We will call the latter indirect flights. An indirect flight stops at (at least) an airport en route to its destination to refuel. In the case of indirect flights information regarding the transit airport(s) must also be stored. The flight cost is calculated to be the cost charged by the airline per customer plus a percentage of this amount (the profit rate, which is fixed for each airline). In the case of indirect flights an additional levy must be added to this amount per customer in order to cover refueling levies at the transit airport. Furthermore, on some flights additional passengers can board the plane at the transit airport. The system needs to keep track of whether boarding will take place at the transit airport or not. The system also needs to store details of the aircraft used for a flight. The aircraft maker (e.g. Boeing, Airbus), model and capacity (number of passengers that it can carry), must be stored for each aircraft.

- In order to test your model, you will need to install the USE tool. The link for the tool is available in the moodle web page. After downloading the archive containing the USE tool, please unzip it to a folder (e.g. c:). The USE tool is a Java application. To run it, you may simply use the batch file start_use.
- Model the class diagram corresponding to the requirements of this problem with USE. You may build up the specification by using a text editor, such as notepad, and then load it into USE. To learn the syntax of this tool, please have a look at some of the many examples included with the tool and read through the tool's documentation.
  - Start by modeling a single class. Load that class into USE. You can start with an empty class, with no attributes and no methods, and go from there. Is everything working as expected? If so, now add the attributes to that class. Load the new model, and check if it is ok. At this point, all you have is a single class, with the attributes you have just defined. In the class diagram view, in the USE tool, turn on the view for attributes, so that you can see your first class, with the corresponding attributes.
  - Now that you know how to define a class with attributes, repeat this for the remaining classes in your model. For the time being, forget about the operations these classes would need. Check if all is well.

  - Specify the associations among the classes you have just defined. Do not forget to define the roles and cardinalities in the associations. Again, load your model in use and check carefully if all is as expected.

- Flights are uniquely identified by their number and date. Please create an OCL invariant to ensure that, in your model, it is impossible for two different flights to share the same flight number and date. Then, build an object diagram that breaks this rule, and verify, in the USE tool, the automatic detection of this rule violation. Fix the problem with the objects diagram and have USE re-check the diagram. The OCL invariant problem should be solved, by now. In other words, it is no longer being broken.

- Each flight is flown by a particular airplane, with a particular capacity for transporting passengers. Create an OCL invariant which ensures that the airplane's capacity is not exceeded in a given flight. Verify that this rule is observed by instantiating the model incorrectly, as in the previous point.

- In any given flight, the departure airport must not be the same as the arrival airport. Create the corresponding OCL invariant and test it.

- On an indirect flight, the connecting airports must not be the departure or arrival airports.

2 SUDOKU

2.1 Construct a UML Class diagram for the following requirements:

A Sudoku puzzle is a 9x9 matrix of placeholders containing integer values ranging from 1 to 9. A well-designed Sudoku puzzle contains no repetitions in each row, no repetitions in each column, and no repetitions inside each of the 9 3x3 sub-matrices. The following image illustrates a well-designed puzzle.

4	3	1	2	9	5	6	8	7
9	6	8	3	7	1	2	4	5
2	5	7	4	8	6	9	3	1
5	9	6	8	4	7	1	2	3
7	4	2	6	1	3	5	9	8
1	8	3	5	2	9	7	6	4
6	2	4	1	5	8	3	7	9
3	7	5	9	6	4	8	1	2
8	1	9	7	3	2	4	5	6

Figure 1: A valid SUDOKU puzzle

2.2 OCL

Again, the USE tool should be used to encode the necessary invariants in OCL, so that it is not possible to create ill-formed Sudoku puzzles. Consider following a similar approach to the one in the first question.