

Quality of Software 2014/15

Instituto Superior Técnico

2nd Project

Due: December 5, 2014

1 Introduction

The development of large information systems is a complex process and demands several layers of abstraction. One first step is the specification of the problem in a rigorous way. After a rigorous specification of the problem, one may perform formal analysis and prove correctness or, in case the specification is not correct, to unveil faults that would be hard to detect by other means.

The second project of the Quality of Software course consists on the specification and verification of a fragment of a system called *Rest* that models an application that is a portal of restaurants.

The main objective of this project is to encourage the practice of specification of problems in a rigorous way, and then to perform proofs of correctness over the chosen implementations. The second aim is to give some experience in usage of automatic verification tools, namely the AtelierB.

2 Problem Specification

The problem that should be specified is a simplification of a module system that models an application that manages a portal of restaurants. The system is responsible for managing the restaurants and dishes available in the portal, as well as managing the orders placed by the clients. All these entities have a unique identifier that is assigned by the portal *Rest*, and it is the responsibility of the system to ensure that there are no two restaurants, dishes, nor orders sharing the same identifier.

This problem is divided into 4 different sub-problems, each corresponding to the specification of a different machine. **Dependencies and relations** among machines should be identified by the students.

You should also consider defined the following sets:

- *CLIENTS* with all the client-identifiers of the portal;

- *DISHES* with all the possible dish-identifiers of the portal;
- *RESTAURANTS* with all the possible restaurant-identifiers of the portal;
- *TRAYS* with all the possible order-identifiers of the portal.

We will now briefly describe each of these entities and provide later a list of requisites that is more extensive and detailed than the following description.

2.1 Dish

The dishes prepared by each restaurant are represented by the Dish entity. A dish has a description (from *DESCRIPTION*) and a name (from *NAMES*). The dish can be a vegetarian dish or not. New dishes can be added to the system and existing dishes can be removed as long as they are not in the menu of a restaurant.

Notice that, contrary to the first project, the same dish can be a part of several restaurant's menus. In the beginning there are no dishes in the system.

2.2 Restaurant

Each restaurant has a name (from *RESTNAMES*) and an address (from *RESTADDRESSES*). Each restaurant is responsible for managing the information about its dishes and has to satisfy the following conditions: a restaurant can be vegetarian and in this case it can only offer vegetarian dishes. Each restaurant has to offer at least 5 dishes and can have at most 15 dishes. A restaurant can not offer two dishes with the same name. This means that the name of each dish is unique inside each restaurant. Each restaurant sets the price of its dishes. However, there is a maximum price (equal for all restaurants and defined by the system) that can not be exceeded. The restaurants can offer free dishes (which normally should only happen in special cases).

The `addDishToRestaurant` method is responsible for adding a dish with the given price to the restaurant. When the dish already exists, then this method updates the price of the dish.

2.3 ShoppingTray

Each shopping tray is associated with a client (from *CLIENTS*) and may contain any number of (different) dishes. A tray can be in status OPEN, CLOSED, or PAID. The contents of the shopping tray can only be changed while the shopping tray is in status OPEN, and a tray can be cancelled until it becomes PAID. When cancelled, a tray is removed from the system. The price of a tray that is not yet paid is the sum of the minimum price that is charged for each dish in the tray. Whenever the client checks-out, the tray becomes CLOSED. The price of a tray is fixed when the client pays for the tray (and the tray becomes PAID).


The initial state of a shopping tray is OPEN.

2.4 Rest

This module orchestrates the entire system and is responsible for the dishes, restaurants, and shopping trays. Among others, it adds new dishes, restaurants, and orders to the portal, and is responsible to ensure that there are no two restaurants, dishes, nor orders sharing the same identifier.

Machine *Rest* should provide the following operations:

Input	Effect	Output
newDish ✓		
name, desc, bool	Creates a new dish with name name and description desc . The dish is vegetarian if and only if bool=TRUE	
removeDish ✓		
dd	Removes an existent dish dd and all its information from the system, as long as the dish is not in a restaurant's menu	
getDishName ✓		
dd	Returns the name of an existent dish dd	name
getDishDescription ✓		
dd	Returns the description of an existent dish dd	desc
isVegetarianDish ✓		
dd	Returns TRUE if dish dd is vegetarian, and FALSE otherwise	bb
newRestaurant ✓		
name, addr, ldish, lprice	Creates a new restaurant with name name , address addr , and list of dishes and prices ldish and lprice	
addDishToRestaurant ✓		
rr,dd,pp	Adds a dish dd with price pp to restaurant rr . If the dish is already present in the restaurant's menu, then it just updates its price	
removeDishFromRestaurant ✓		
rr,dd	Removes the dish dd from the restaurant rr , if it exists in the restaurant	
changePrice ✓		
rr,dd,pp	Changes the price of an existent dish dd to pp in restaurant rr , if the dish exists in the restaurant	

Input	Effect	Output
getDishPrice ✓		
<code>rr, dd</code>	Returns the price <code>pp</code> of the dish <code>dd</code> in restaurant <code>rr</code> , if it exists in the restaurant	<code>pp</code>
setVegetarianRestaurant ✓		
<code>rr, bb</code>	Changes the type of restaurant <code>rr</code> to vegetarian if <code>bb=TRUE</code> and to non-vegetarian otherwise	<code>res</code>
isVegetarianRestaurant ✓		
<code>rr</code>	Returns TRUE if restaurant <code>rr</code> is vegetarian and FALSE otherwise	<code>res</code>
getRestaurantName ✓		
<code>rr</code>	Returns the name of the restaurant <code>rr</code>	<code>name</code>
getRestaurantAddress ✓		
<code>rr</code>	Returns the address of the restaurant <code>rr</code>	<code>addr</code>
getRestaurantsDish ✓		
<code>dd</code>	Returns the list of the names of the restaurants that serve dish <code>dd</code> , if they exist	<code>lrr</code>
getDishRestaurant ✓		
<code>rr</code>	Returns the list of the dishes served in restaurant <code>rr</code>	<code>ldd</code>
newTray ✓		
<code>cc</code>	Creates a new shopping tray for client <code>cc</code>	
addDishToTray ✓		
<code>tr, dd</code>	Adds a new dish <code>dd</code> to the shopping tray <code>tr</code>	
removeDishFromTray ✓		
<code>tr, dd</code>	Removes dish <code>dd</code> from shopping tray <code>tr</code> if it exists in the tray	
checkoutTray ✓		
<code>tr</code>	Begins the checkout process for shopping tray <code>tr</code>	
payTray 		
<code>tr</code>	Pays tray <code>tr</code> computing its final cost using the minimum price available for each dish in the tray	

Input	Effect	Output
cancelTray ✓		
tr	Cancels the shopping tray tr unless it was already paid	
computePriceOfTray ✓		
tr	Computes the current cost of shopping tray tr	pp

2.5 Restrictions of the Problem

The specification should satisfy the following constraints. You must identify the most adequate machine to define these restrictions:

1. Each dish has a description and a name;
2. Some dishes are vegetarian;
3. All dishes are different and one cannot reuse dish-identifiers;
4. All dishes have different names;
5. System only records the information (name and description) of current dishes;
6. One cannot remove from the portal dishes that are in a restaurant's menu, neither dishes that are in some non-PAID tray;
7. Every restaurant has a name and an address;
8. Each restaurant has a menu that offers a minimum of 5 and a maximum of 15 dishes;
9. Each restaurant sets the price of each of its dishes;
10. The maximum cost of a dish is limited for every restaurant and bounded by maxprice; the minimum cost of a dish is 0.
11. There are vegetarian restaurants;
12. Vegetarian restaurants only offer vegetarian dishes;
13. A dish may be offered by more than one restaurant (and eventually at different prices);
14. A restaurant cannot offer dishes that are not defined in the portal;
15. The dishes in the menu of a restaurant have all different names;
16. Whenever adding a dish that already exists in the menu of the restaurant, its price should be updated to the new price;
17. One can only change the price of dishes that are present in the menu of some restaurant;
18. One can only remove from a restaurant the dishes that are present in the menu of the restaurant;
19. Each tray is property of a single client;
20. A tray may contain any number of (different) dishes;
21. A tray is either in state OPEN, CLOSED, or PAID;
22. The price of a a tray is only set when a tray is paid;

23. The price of a paid tray is always the same and determined when the tray is paid;
24. Initially a tray is in state `OPEN`;
25. One cannot add to a tray dishes that are already there;
26. One can only add dishes to a tray in state `OPEN`;
27. Every dish on a non-PAID tray has to be in some restaurant's menu;
28. One cannot remove from a tray dishes that are not in the tray;
29. One can only remove dishes from a tray in state `OPEN`;
30. Checking-out an `OPEN` tray changes its state to `CLOSED`;
31. Paying a `CLOSED` tray changes its state to `PAID`;
32. The price of a tray is computed using the minimum price available for each dish on the tray;
33. When a tray is cancelled all its information is removed from the system;
34. One can only cancel non-PAID trays;
35. A tray may only contain dishes that are (or were) defined in the portal.

2.6 Extra Machines and Hypothesis

In the previous sections we described the machines and have given the interface for machine *Rest*. These operations may be *native* from this machine or promoted or redefined from one of the other machines. Dependencies among machines and the most adequate machine to define these operations (as well as the restrictions of Section 2.5) should be identified by the students.

You may define other machines not stated here if you feel the need to. You may also need to find reasonable preconditions for the operations.

When *including* a machine, you should also identify which operations should be exported for the users and which should be redefined. In the case you need to redefine operations (and for sure you will!) you should apply a proper renaming.

You cannot use constructor USES!

3 Verification of Correctness

Once the problem is specified, each group should prove the consistency of the abstract machines that were obtained. For that, each group should type-check their solution using the tool AtelierB (<http://www.atelierb.eu/en/>). Then, each group should manually prove the correctness of operations `addDishToRestaurant` of machine *Rest*, and `removeDish` of machine *Rest*.

4 Delivery of the Project

The delivery protocol, as well as the documentation to be delivered, will be announced on the course's website.

The project is due on the **5th of December, 2014, 16:59:59**.

5 Project Evaluation

5.1 Evaluation components

In the evaluation of this project we will consider the following components:

1. Correct specification of the requisites: 0–12 points.
2. Correct implementation of the specification using one of the tools (Type-Check for Correction): 0-1 point.
3. Manual proof of correctness of operation `addDishToRestaurant` of *Rest*: 0–1.5 points.
4. Manual proof of correctness of operation `removeDish` of *Rest*: 0–2.5 points.
5. Quality and simplicity of the obtained solution: 0–3 points.

If any of the above items is only partially developed, the grade will be given accordingly.

It is mandatory for every group to include in the final project a) the diagram of relations between the different machines; b) the specification of all the machines performed in (1); and c) the manual proofs performed in (3) and (4).

5.2 Other Forms of Evaluation

It may be possible *a posteriori* to ask the students to present individually their work or to perform the specification of a problem similar to the one of the project. This decision is solely taken by the professors of QS. Also, students whose grade in the second test is lower than this project grade by more than 5 may be subject to an oral examination.

In both cases, the final grade for the project will be individual and the one obtained in these evaluations.

5.3 Fraud Detection and Plagiarism

The submission of the project presupposes the **commitment of honour** that the project was solely executed by the members of the group that are referenced in the files/documents submitted for evaluation. Failure to stand up to this commitment, i.e., the appropriation of work done by other groups, either voluntarily or involuntarily, will have as consequence the immediate failure of this year's QS course of all students involved (including those who facilitated the occurrence).

6 Final Remarks

All information regarding this project is available on the course's website, under the section *Project*. Supporting material such as links, manuals, and FAQs may be found under the same section.

In cases of doubt about the requirements, or where the specification of the problem is possibly incomplete, please contact the Professors of the course.

GOOD LUCK!

Log of Changes

11Nov2014—1st Version.

25Nov2014—Removed the output of operation **setVegetarianRestaurant**.