Specification of Software 2016/17
Instituto Superior Técnico
1^{st} Project

Due: November 11, 2016, 21:59:59

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 first project of the Specification of Software course consists on the specification and verification of a fragment of a system called *Zöber* that models an application for reservation of private transportation.

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 model. The second aim is to give some experience in usage of automatic verification tools, namely the Rodin Platform.

2 Problem Specification

The problem that should be specified is a simplification of a module system that models the reservation app (henceforth called *Zöber*). Zöber is responsible for managing the clients, drivers, and cars available in the system, as well as the reservation of the rides. The rules for reservation of rides, and the working conditions of the drivers of Zöber are enforced by a given policy (detailed later in this document). Clients, drivers, and cars have unique identifiers within the system and it is the responsibility of *Zöber* to ensure that there are no two clients, nor drivers, nor cars, sharing the same identifier.

This problem can be divided into several different sub-problems/dimensions, dealing with the different levels of abstraction. It is the responsibility of the students to identify at which modelling level it is appropriate to define each property/event.[1]

You should also consider defined the following sets:

- CLIENTS with all the possible clients that can register to Zöber;
- DRIVERS with all the possible drivers that can register to Zöber;
- CARS with all the possible cars that can be added to Zöber;
- NAMES with all the possible names of persons that can register do Zöber;
- RIDES with all the identifiers for rides;
- EMAILS with all possible e-mail addresses;
- LICENSES will all possible driver's licenses;
- ZOBERSERVICE with the names of the services provided by Zöber. At the moment Zöber offers the services ZöberY and ZöberWhite;
- PLAN with the possible client plans, VIP or REGULAR.

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

2.1 Clients

The initial model is concerned with the management of the clients. We say that a *client is registered* in Zöber if he was added to and not yet removed from Zöber.

Not all clients are registered in Zöber. Those that are registered have a name and an associated e-mail. A user e-mail should be unique within Zöber.

At the beginning there are no registered clients in Zöber. One can add new clients to the system (becoming registered clients), and remove clients from the system. Clients may also be upgraded to/downgraded from their VIP status.

Notice that all these operations should preserve the conditions above (and some others specified later in Section 2.6).

2.2 Drivers

This model is supposed to manage the drivers of Zöber. We say that a *driver is in Zöber* if it was added and not yet removed from Zöber.

Each driver in Zöber has 2 attributes: a name, and a valid driver's license.

At the beginning there are no drivers in Zöber. One can add new drivers to Zöber as long as they do not exist yet in the system, and remove drivers that already exist in the system. We can also ban drivers from the system, in which case they will not be able to join again later.

The conditions in which these operations are enabled and the properties they should satisfy are specified in detail in Section 2.6.

[1] See the Section 7 for tips on how to approach this problem.

2.3 Cars

This model manages the cars registered in Zöber. We say that a *car is in Zöber* if it was added and not yet removed from Zöber.

Each car in Zöber has 2 attributes: an owner, and a set of drivers. At the beginning there are no cars in Zöber. One can add cars to Zöber as long as they do not exist yet in the system, and remove cars that exist in the system. One can also add and remove driver's from any given car. The maximum number of drivers per car is 3, and the owner of a car is always a driver of that car. Moreover, each driver cannot be associated with more than 2 cars.

Cars registered in Zöber provide a given level of service, ZöberY or ZöberWhite, that can be upgrade (from ZöberY to ZöberWhite) or downgraded (from ZöberWhite to ZöberY). Cars originally provide ZöberY service.

The conditions in which these operations are enabled and the properties they should satisfy are specified in detail in Section 2.6.

2.4 Rides

This model orchestrates the interactions in Zöber.[2]

Every client may book a ride. A ride is associated with 5 atrributes: a car, a client, a level of service, and the beggining and end of the ride. As a simplification, we assume that the beginning and end of a ride are represented as natural numbers. Consider these to be the number of seconds passed since the beginning of the year. Upon completion of a ride, the client rates that ride from 1 to 5.

It is also possible for a client to cancel an already booked ride.

Cars registered in Zöber provide a given level of service. VIP clients always book ZöberWhite services, whereas regular clients may book ZöberY or ZöberWhite. Also, VIP clients may book any number of rides, whereas regular clients can only book two rides (not completed yet).

2.5 Operations

Our system should provide the following operations:

Input	**Effect**
newClient	
`client, name, email`	Registers a new client with identifier `client`, with the given name, and with the given e-mail
removeClient	
`client`	Removes the registered client `client` from Zöber

[2]See the Section 7 for tips on how to split the development of this model.

newDriver	
`driver, name, license`	Registers a new driver with identifier `driver`, with the given name, and with the given driver's license
removeDriver	
`driver`	Removes the registered driver `driver` from Zöber
addCar	
`car, owner`	Registers a new car `car` in Zöber, owned by `owner`
removeCar	
`car`	Removes `car` from Zöber
addDriverToCar	
`car, driver`	Associates the registered driver `driver` to `car`
removeDriverFromCar	
`car, driver`	Deassociates the registered driver `driver` from `car`
newRide	
`ride, srv, client` `begin, end`	Schedules a new ride `ride` for this client, in the defined period, for some available car of type `srv`
cancelRide	
`ride`	Cancels the previously scheduled `ride`
completeRide	
`ride, grade`	Completes the scheduled `ride` and assigns the grade `grade` to this car
upgradePlan	
`client`	Upgrades a REGULAR client to VIP
downgradePlan	
`client`	Downgrades a VIP client to REGULAR
upgradeService	
`car`	Changes the service offered by `car` to ZöberWhite
downgradeService	
`car`	Changes the service offered by `car` to ZöberY
banDriver	
`driver`	Removes the registered driver `driver` from Zöber, disallowing him from any future registration. If `driver` is the owner of some car, those cars should also be removed from the system

2.6 Restrictions of the Problem

The specification/operations above should satisfy the following constraints. You must identify the most adequate model to define these restrictions:

1. Every client in CLIENT may register in Zöber;
2. All clients registered in Zöber have a name, and an e-mail;
3. The e-mails registered in Zöber are unique;
4. The plan of a client may be REGULAR or VIP;
5. At the beginning there are no clients registered in Zöber;
6. Any client may register himself in Zöber as long as he is not registered yet;
7. The original plan of a client is REGULAR;
8. Only (and every) registered client may be removed from Zöber;
9. Only registered clients may be upgraded to profile type VIP/downgraded to profile type REGULAR;
10. Only REGULAR clients may be upgraded to VIP (and downgraded vice-versa)
11. Every driver in DRIVERS may register in Zöber;
12. All drivers registered in Zöber have a name, and a driver's license;
13. The driver's licenses registered in Zöber are unique;
14. At the beginning there are no drivers registered in Zöber;
15. Any driver may register himself in Zöber as long as he is not registered yet;
16. Only (and every) registered driver may be removed from Zöber;
17. It is possible to ban a driver from Zöber. In this case, it should not be possible to add him again at a later stage;
18. Every car in CARS may be registered in Zöber, as long as it is not registered yet;
19. All cars registered in Zöber have a single owner;
20. A registered car has at least 1 associated driver, and may have at most 3;
21. The owner of a car is one of the drivers of the car;
22. Only registered drivers may be drivers of a car;
23. Each driver cannot be associated with more than 2 cars;
24. Each car provides one of the available services: ZöberY, or ZöberWhite;
25. At the beginning there are no cars registered in Zöber;
26. The initial service provided by every car is ZöberY;
27. Only (and every) registered car may be removed from Zöber;
28. Every (and only) registered driver(s) may be associated with/removed from a car;
29. Each ride has a unique identifier and his composed of a client, a time-frame (begin and end), a type of service, and is associated with a given car;
30. The car associated with a ride, is able to provide the level of service required for that ride.
31. Every ride is well-formed, that is, the end is later than its beginning;
32. No car has overlapping rides;
33. Every completed ride has an associated rating (from 1 to 5);

34. A REGULAR client can have at most 2 booked (non-overlapping) rides at each time; VIP clients may book as many as they want;
35. VIP clients only travel in ZöberWhite cars;
36. Clients with reserved rides cannot be removed from the system;
37. Any non-completed ride can be canceled;
38. If a driver is banned, his rides and the cars he owns are immediately removed from the system;
39. A car cannot be removed from the system if there are pending reservations for this car;
40. The owner of a car cannot be removed from the system if there are pending reservations for one of the cars he owns;

2.7 Extra Machines and Hypothesis

In the previous sections we described the machines and have given the interface for machine *Zöber*. These operations may be *native* from this machine or refined from some other machines. Dependencies among machines and the most adequate machine to define these operations (as well as the restrictions of Section 2.6) should be identified by the students.

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

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 use the Rodin Platform (`http://www.event-b.org/`).

The sytem is proved correct when all the Proof Obligations are marked with a *green check* mark. You may need to use the interactive prover to complete some of these proofs.

4 New Ideas for Zöber

Zöber is a tech start-up on the rise, so we are exploring new ideas all the time. Our newest idea is *ZöberSharing*, an extention of our platform that includes the possibility of ride-sharing between clients. The goal is that two or more clients may share the same ride if they are "somehow on the same path".

You task, should you choose to accept it, is to extend the model developed in the previous sections to include this new functionality. You may assume the following:

1. Our algorithms department is developing an algorithm called `same_route?` that given two rides r_1 and r_2 (of clients c_1 and c_2) says if the two are along the same path;

2. ZöberY cars can transport up to 2 clients at the same time, whereas ZöberWhite cars may transport up to 4 clients at the sametime;
3. ideally ZöberSharing should be provided by our best cars, eg, the top-20% in the rating; although we can also accept a general solution.

Since this will be a prototype, your development should be correct, although no manual proofs will be required.

5 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 **11th of November, 2016, 21:59:59**.

6 Project Evaluation

6.1 Evaluation components

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

1. Correct specification of the requisites of Section 2: 0–12 points.
2. Correct usage of the Interactive Prover to prove 4 of the resulting POs: 0-2 points (0.5 each).
   Notice, it is not enough to open the interactive prover and push the button of one of the automatic provers $p_0, p_1, \ldots$ You should really do some proofs using the interactive prover.
3. Quality and simplicity of the obtained solution: 0–4 points.
4. Solution for the problem presented in Section 4: 0–2 points.
   You will be evaluated both in terms of the inovation and simplicity of the solution, and on your capacity to formalize the given problem.

If any of the above items is only partially developed, the grade will be given accordingly. **Besides the generated Rodin Archive, it is mandatory for every group to include in the final project:**

(a) **the strategy used to develop/refine the system, namely what was addressed in each step of the refinement, and what is the refining sequence;**
(b) **the specification of all the machines performed in 1. referring in each Invariant/Guard/Action which of the requirements is implementing (if any); and**
(c) **the specification of all the machines performed in 4.**
(d) **the reference of the proofs that were done in 2.**

Detailed info about the structure of this report will be provided in the next few days.

6.2 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 ES. Also, students whose grade in the first 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.

6.3 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 ES course of all students involved (including those who facilitated the occurrence).

7 Useful Development Tips

The system to be developed can be divided into several sub-problems/orthogonal dimensions. Considering one dimension at each time corresponds to dealing with the different levels of abstraction.

One first task should be to try to separate the requisites of Section 2.6 into these orthogonal dimensions. Then, and following the strategy that we have done in class, start dealing with one such dimension and keep adding the others via refinement (zoom-in the problem!). As an example of two separate dimensions we have clients and drivers. These two dimensions can be dealt separately ;-).

8 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!