Specification of Software 2015/16
Instituto Superior Técnico
1^{st} Project

Due: November 8, 2015, 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 *GitBob* that models a file-repository where files can be versioned, backed-up, and shared.

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 file-repository (henceforth called *GitBob*). GitBob is responsible for managing the users and files available in the repository, as well as the reading and sharing permissions. *GitBob* also has a versioning and backup component that enforces a given policy (detailed later in this document). Users and files have unique identifiers within the system and it is the responsibility of *GitBob* to ensure that there are no two users nor files with 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]

[1] See the Section 6 for tips on how to approach this problem.

You should also consider defined the following sets:

- USERS with all the possible users that can register to GitBob;
- UTYPES with the possible *profile types* of users; either BASIC or PREMIUM;
- UEMAILS with all the possible e-mail addresses of users that can register to GitBob;
- FILES with all the possible files that can be uploaded to GitBob;
- MODES with the possible *sharing modes* of a file; either REGULAR, SECURE, or READONLY;

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

2.1 Users

The initial model is concerned with the management of the users. We say that a *user is registered* in GitBob if he was added and not yet removed from GitBob.

Not all users are registered in GitBob. Those that are registered have a profile type and an associated e-mail. A user e-mail should be unique within GitBob.

At the beginning there are no registered users in GitBob. One can add new users to the system (becoming registered users), remove users from the system, and change the profile type of registered users upgrading it from BASIC to PREMIUM, or downgrading it from PREMIUM to BASIC.

Notice that all these operations should preserve the conditions above (and some others specified later in Section 2.5).

2.2 Files

This model is supposed to manage the files that are available in GitBob. We say that a *file is in GitBob* if it was added and not yet removed from GitBob.

Each file in GitBob has 3 attributes: a file size (that is a natural number), a file owner (the user that added the file to GitBob), and a file version (the last version that was uploaded to GitBob). The file version is a natural number, greater than 0.

At the beginning there are no files in GitBob. One can add files that do not exist yet in the system, remove files that exist in the system, upload new versions of existing files, and download the most recent version that was uploaded to the system. The conditions in which these operations are enabled and the properties they should satisfy are specified in detail in Section 2.5.

2.3 GitBob

This model is supposed to orchestrate the interactions in GitBob.[2] We say that a *user has a file*, or that a *user has access to a file*, if he is the owner of the file, or if some other user (that has access to this file) shared the file with him. We say that a *file is in backup mode*, or that *backup is enabled*, if backup has been turned on for that file.

Files can be shared among registered users, and one can only share files that one has access to. The owner of a file has always access to a file. A user that has access to a file can revoke the access permission to that file from any other user, except for the file owner.

Files can be shared in any of the 3 sharing modes described before. New versions of files shared as READONLY can only be uploaded to GitBob by the file owner; files can only be shared SECUREly if all the users that have access to the file are PREMIUM. The files are shared by default in REGULAR mode, but the sharing mode can be changed throughout the execution of the system.

Files have versions. Whenever a file is uploaded to GitBob, its version is increased by 1. Whenever one downloads a file from GitBob, the local version of that user becomes the same that is currently in the server. One can only uploaded a file to the server if we have the most up to date version of that file, that is, the local version is the same as the version stored in the server. Uploading and downloading a file from GitBob is only possible to users that have access to the file. Local versions are only updated when the download command is executed, except when a file is removed from GitBob (or a user loses his share privileges) in which case the local versions of the files are immediately removed.

Files can be backed up in GitBob. A file can be backed up in GitBob if some PREMIUM user has access to the file. If a file has its backup mode enabled then all the uploaded versions (after the backup mode was turned on) of this file are recorded and kept in GitBob. If the backup mode is turned off, then all these recorded versions are discarded. Whenever a file is removed from GitBob, its last version is recorded in an archive that can be later accessed by any user that could access the file at the time that the file was removed.

By default, files are shared as REGULAR and its backup mode is disabled.

2.4 Operations

Our system should provide the following operations:

[2] See the Section 6 for tips on how to split the development of this model.

Input	Effect
newUser	
`usr, type, mail`	Creates a new user with identifier `usr`, with the given profile type, and with the given e-mail
removeUser	
`usr`	Removes the registered user `usr` from GitBob
upgradePremium	
`usr`	Upgrades the profile type of `usr` from BASIC to PREMIUM
downgradeBasic	
`usr`	Downgrades the profile type of `usr` from PREMIUM to BASIC
addFile	
`file, size, owner`	Adds a new file `file` to GitBob, with the given `size` and `owner`. The file starts at version 1.
removeFile	
`file, usr`	Removes `file` from GitBob and records its last version in an archive, if `usr` has access to the file; removes the local version of `file` from all users
uploadFile	
`file, usr`	Uploads a new version of `file` to GitBob, if `usr` has access to the file
downloadFile	
`file, usr`	Downloads the current version of `file` from GitBob, if `usr` has access to the file
shareFile	
`file, usr1, usr2`	Gives `usr2` access to `file` from GitBob, if `usr1` has access to the file; also, `usr2` downloads the current version of `file` from GitBob
removeShare	
`file, usr1, usr2`	Revokes access permission of `usr2` to `file`, if `usr1` has access to the file; removes the local version of `file` from `usr2`
changeSharingMode	
`file, usr, mode`	Changes the sharing mode of `file` to `mode`, if `usr` is the owner of the file
turnOnBU	
`file, usr`	Starts logging the versions of `file` starting from the current version in GitBob; `usr` needs to have access to `file`

turnOffBU	
`file, usr`	Stops logging the versions of `file` and discards all the log associated with `file`; `usr` needs to have access to `file`
downloadFromArchive	
`file, usr`	Downloads the version of `file` at the time that it was removed from GitBob; `usr` needs to be a user that had access to `file` at that moment

2.5 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 user in USER can register in GitBob;
2. All users registered in GitBob have a profile type, and an e-mail;
3. The e-mails registered in GitBob are unique;
4. At the beginning there are no users registered in GitBob;
5. Any user may register himself in GitBob as long as it is not registered yet;
6. Only registered users may be removed from GitBob;
7. Only registered users may be upgraded to profile type PREMIUM;
8. Only registered users may be downgraded to profile type BASIC;
9. Only BASIC users may be upgraded to PREMIUM (and downgraded vice-versa)
10. Every file in GitBob occupies a given space, has a (single) owner, and is in a given version;
11. For the sake of simplicity one assumes that every version of a file occupies the same space;
12. GitBob only keeps track of the size, owner, and version of current files;
13. At the beginning there are no files in GitBob;
14. One cannot remove from GitBob a user that is the owner of some file;
15. One cannot add to GitBob already existing files;
16. The owner of a GitBob file should be a registered user;
17. The initial version of a file is number 1;
18. Only existing files may be removed/uploaded/downloaded from GitBob;
19. Upon a successful upload, the version of a file is increased by one;
20. A GitBob file may be shared among several users;
21. A GitBob file can only be shared among registered users;
22. A GitBob file is always shared with its owner;
23. At the beginning there are no shared files in GitBob;
24. One cannot remove from GitBob users that are involved in some sharing of files;
25. Files can only be removed/updated/downloaded by users that have access to that file;

26. A user with access to a file may share it with some other user;
27. One can only share a file with users that do not share yet that same file;
28. A user with access to a file may stop the sharing of that file with any other user, except with its owner;
29. Every active file is shared in one of the possible 3 sharing modes;
30. A file can only be shared securely, if all the users that have access to it are PREMIUM;
31. A user in a SECURE sharing cannot downgrade his profile to BASIC;
32. Files are by default shared in REGULAR mode;
33. READONLY files can only be removed from GitBob by its owner;
34. New versions of READONLY files can only be uploaded by its owner;
35. Only the user that is the owner of a file may change its sharing mode;
36. One can only change the sharing mode of a file to SECURE if all users that have access to the file are PREMIUM;
37. Only active files are versioned in GitBob;
38. The local version of a file (for each user) is never greater than the version of the file stored in the repository;
39. GitBob files may have the backup mode enabled;
40. GitBob only logs versions for the files with the backup mode enabled;
41. A file is only backed up in GitBob if at least one user with PREMIUM profile has access to it;
42. The archive stores the last version of a file prior to its removal from GitBob; only users that had access at that time may retrieve the file from the archive;
43. When a user is removed from GitBob, he no longer has access to his file versions;
44. A user can only upload a file in GitBob if his local version is up-to-date (ie, is the same as in the server);
45. A GitBob user cannot downgrade his profile to BASIC if there is a backup-enabled file for which he is the only PREMIUM user;
46. Whenever a file is removed from GitBob, its last version is removed from the active backups and stored in an archive that can be accessed via de operation `downloadFromArchive`. Simultaneously, all the local version of that file are also removed from GitBob;
47. Every file version uploaded to GitBob should be logged if the backup mode for that file is enabled;
48. The access to a file with the backup mode enabled can only be revoked for a user, if there is at least another user with access to that file that has the profile PREMIUM;
49. Any user that can access a file can change its backup mode;
50. One can only enable backups for files that have their backup disabled (and vice-versa).

2.6 Extra Machines and Hypothesis

In the previous sections we described the machines and have given the interface for machine *GitBob*. 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 2.5) 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 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 **8th of November, 2015, 21:59:59**.

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 usage of the Interactive Prover to prove 6 of the resulting POs: 0-0.5 points each.
3. Quality and simplicity of the obtained solution: 0–5 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 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 reference of the proofs that were done in 2.

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 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.

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

6 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 2.5 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 file-sharing and file-versioning. These two dimensions can be dealt separately ;-).

Also for the purpose of this modelling, the specific content of the files/versions can be ignored.

7 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

28Oct2015—1st Version.
28Oct2015—Fixed minor typos.