Project – Software Validation

This project is to be done in teams of FOUR or FIVE students. You are required to sign up to one of the project groups in myCourses by Sunday, **October 18th, end of day**. Keep in mind that two students who were on the same team for the assignment are not allowed to be on the same team for the project. In addition, you are also required to set up a GitHub repository by October 18th, which your team will use to work on the project. Last but not least, you are required to give the TA (username: mschoettle) access to your GitHub repository from October 18th at the latest until the final grades have been published. While October 18th is the due date for these tasks, it is strongly suggested to form teams as early as possible. Do not wait until the last minute, because this will allow you to start working on your project earlier. We highly recommend starting with the project as early as possible as it requires a decent period to get set up and going.

Your team is required to hand in a ***single zip file*** via myCourses containing your reports as well as your implementation as described below. If you realize that you need to make changes to your submission, do not resubmit only the file(s) that have changed, but rather resubmit another complete zip file. If you are using an application other than MSWord for your report, convert your report first to either a PDF file or a DOC(X) file.

Each team member must make contributions to the project. A team member who does not contribute to the project receives a mark of 0 for the project. A team member may optionally email a confidential statement of work to the instructor before the due date of the project. A statement of work first lists in point form the parts of the project to which the team member contributed. In addition, the statement of work also describes whether the work load was distributed fairly evenly among the team members. A statement of work may be used to adjust the mark of a team member who is not contributing sufficiently to the project. In case of unequal contributions, information from the GitHub repository of your project may also be used to adjust the mark of a team member. It is not necessary to send a statement of work, if a team distributed the work for the project fairly evenly and each team member contributed sufficiently.

Description

The TouchCORE tool is a touch-enabled software engineering tool for model-based software development. For this project, you are asked to test functionality related to the modeling of message views in TouchCORE. A message view is similar to a UML sequence diagram. Your testing efforts of message views must involve ***Unit Testing***, ***Integration Testing***, and ***Acceptance Testing***. A significant part of your project is to determine ***how many*** test cases are needed with the help of some of the testing techniques covered in the course and then ***justifying*** your choice of testing techniques. You are required to test the following tasks:

- the ability to add a message that creates a new instance (and consequently a new lifeline),
- the ability to add a message with a new lifeline as target,

- the ability to add a message with an existing lifeline as target,
- the ability to add a reply message, and
- the ability to delete one or several messages.

You are not required to test the various parameters a message may have, the various options to which the returned value of a message may be assigned, and the return value of a reply message. If needed, your test cases may always use the same parameter, return value, etc. The project is about testing the creation and deletion of messages and not about testing the details of the messages.

The source code of the TouchCORE tool will be available from https://github.com/mschoettle/ecse429-fall15-project. The source code will also be made available as a zip file in myCourses.

If you are doing an excellent job, you may be asked to have your code included in the TouchCORE tool. However, this will only be done after the final grades have been determined for the course and your decision to either allow or deny the request does not have any influence on the final grade. Your code will not be used without your permission. Similarly, you are not allowed to use the source code of TouchCORE without permission except for this course project.

The tutorial on Friday **October 09th, 11:35-13:25**, includes a demo of the tool, an overview of relevant parts of the TouchCORE metamodel, a description of the overall structure of the tool, a description on how to set up the environment, and several hints on how to go about testing the tool. The concepts of a message view are shown in Figure 1 and relevant excerpts of the metamodel are shown in Figure 2 to Figure 4.

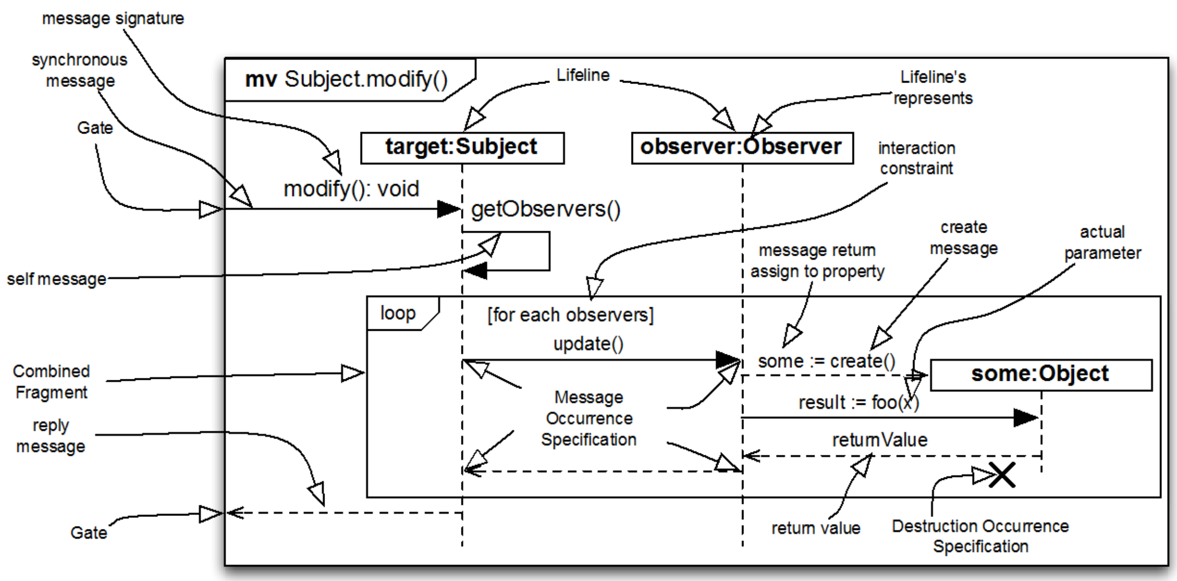


Figure 1. Message View Concepts

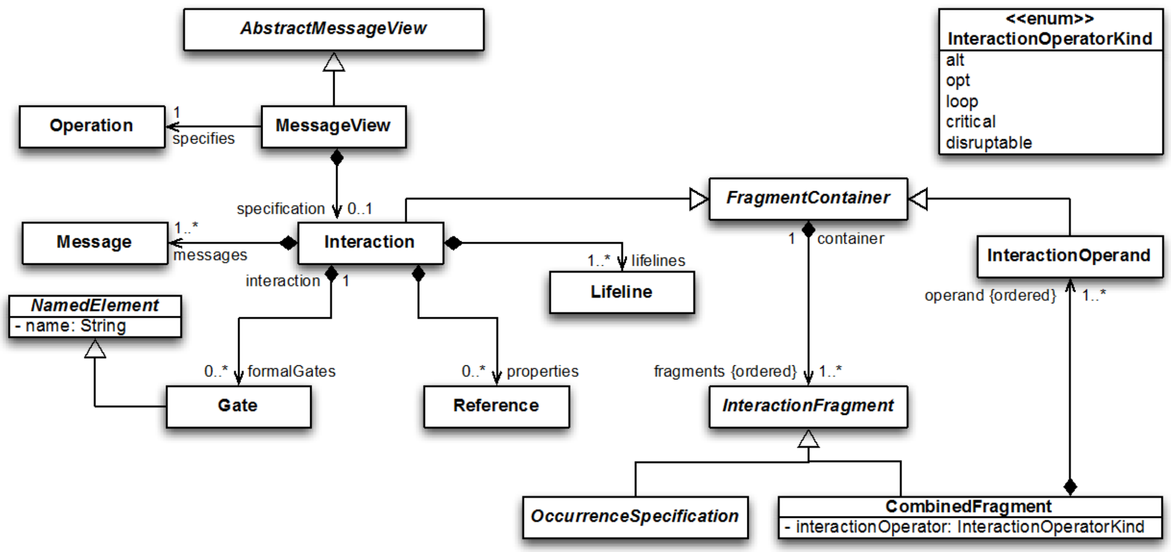


Figure 2. Excerpt of TouchCORE Metamodel (Interaction)

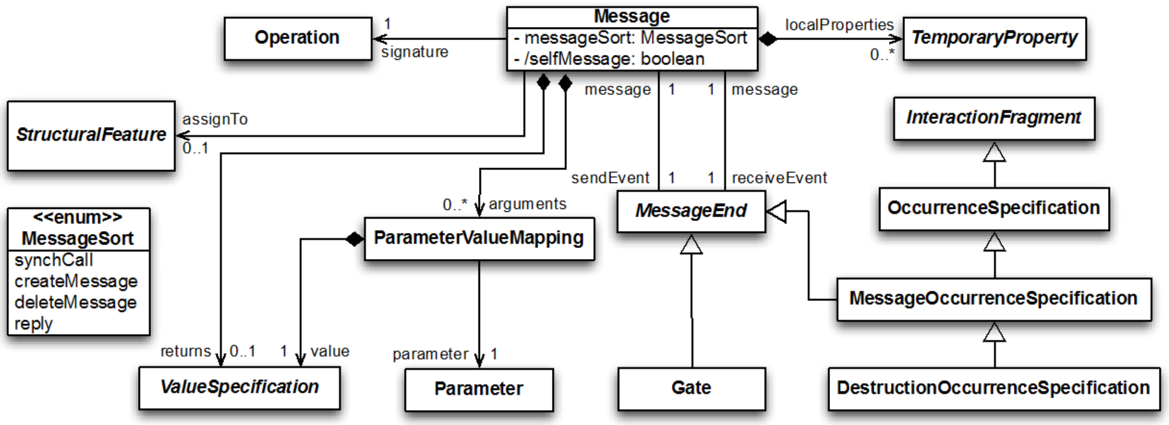


Figure 3. Excerpt of TouchCORE Metamodel (Message)

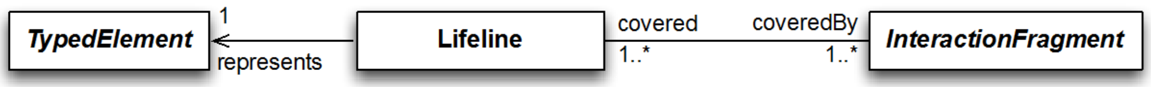
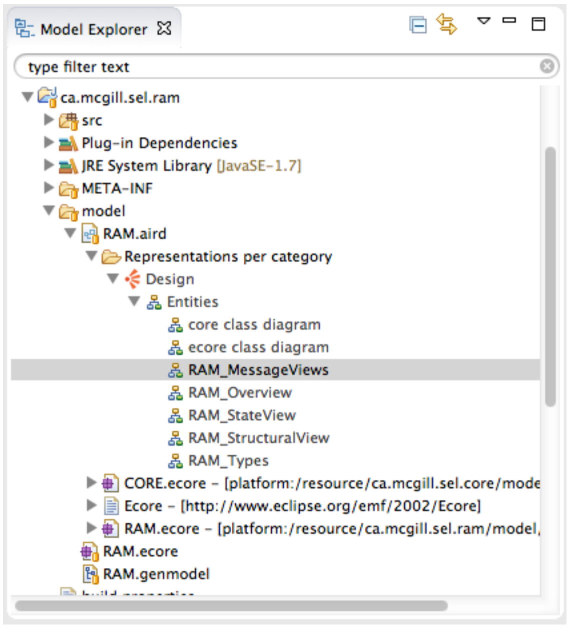


Figure 4. Excerpt of TouchCORE Metamodel (Lifeline)

The full metamodel is available at `ca.mcgill.sel.ram/model/RAM.aird`. It can be viewed by opening the `Modeling` perspective in the Eclipse Modeling Tools (version Mars), double-clicking above mentioned file and expanding it as follows: `RAM.aird > Representations per category > Design > Entities`. Then, select the part of the metamodel that you are interested in (starting with `RAM_`) (see Figure 5).



```
ca.mcgill.sel.ram
  src
  Plug-in Dependencies
  JRE System Library [JavaSE-1.7]
  META-INF
  model
    RAM.aird
      Representations per category
        Design
          Entities
            core class diagram
            ecore class diagram
            RAM_MessageViews
            RAM_Overview
            RAM_StateView
            RAM_StructuralView
            RAM_Types
      CORE.ecore - [platform:/resource/ca.mcgill.sel.core/mode
      Ecore - [http://www.eclipse.org/emf/2002/Ecore]
      RAM.ecore - [platform:/resource/ca.mcgill.sel.ram/model,
    RAM.ecore
    RAM.genmodel
```

Figure 5. Structure of RAM.aird File

Part 1 – Acceptance Testing

For acceptance testing, you have to apply a ***black-box testing*** technique of your choice (but one that was discussed in the course) to manually demonstrate that the TouchCORE tool is capable of performing the tasks listed above.

You are required to hand in a ***report*** detailing your acceptance testing approach (i.e., how you determined which test cases are needed). You have to justify why you have chosen the technique you have chosen and discuss the coverage you have achieved with your chosen technique. You have to include all test cases with sufficient information to run the test cases, all expected and actual results, and clear verdicts of whether a test case passed or failed. You are not required to automate the acceptance tests, but you rather will have to manually exercise the tasks with the TouchCORE tool according to your test cases.

Part 2 – Integration Testing Strategy

You are required to perform unit tests and integration tests for the following three classes:

- `MessageHandler`
- `MessageViewHandler`
- `MessageViewController`

Other handlers and controllers that are also of interest for message view modeling but have already been tested are these 12 classes (ten handlers and two controllers):

- `AssignmentAssignToHandler`
- `AssignmentStatementHandler`
- `AssignToHandler`
- `CombinedFragmentHandler`

- `ExecutionStatementHandler`
- `InteractionConstraintHandler`
- `LifelineViewHandler`
- `MessageAssignToHandler`
- `ReferenceAndValueHandler`
- `ValueSpecificationHandler`
- `FragmentsController`
- `MessageController`

The handlers can be found in the *ca.mcgill.sel.ram.ui.views.message.handler.impl* package, while the controllers can be found in the *ca.mcgill.sel.ram.controller* package.

You need to decide on an ***integration testing strategy*** (that was discussed in the course) based on a ***dependency graph*** that covers all 15 classes (the three you are required to test and the 12 that have already been tested). You are required to hand in a ***report*** detailing your integration testing strategy. You have to include your dependency graph, and you have to justify why you have chosen the integration testing strategy you have chosen compared to other integration testing strategies discussed in the course. Furthermore, you have to describe which drivers, mocks, and stubs you need for each of the four classes you are required to test.

Part 3 – Unit Testing

Based on your chosen integration testing strategy, you are required to automate your tests by implementing a JUnit 4 ***test suite*** for the unit and integration tests of each of the three classes you are required to test. All of your tests must be saved in the folder /tests/groupNN (replace NN with your myCourses group number) in the respective project. In addition, the tests for one class must be in the same package as the class under test. See Figure 6 for an illustration of the folder and package structure.

```
Controller Project Root (ca.mcgill.sel.ram.controller)
└── src
    └── ca.mcgill.sel.ram.controller
        └── MessageViewController.java
        └── ...
└── tests
    └── groupNN
        └── ca.mcgill.sel.ram.controller
            └── TestMessageViewController.java
            └── ...
GUI Project Root (ca.mcgill.sel.ram.gui)
└── src
    └── ca.mcgill.sel.ram.ui.views.message.handler.impl
        ...
└── tests
    └── groupNN
        ... (follow same principle as above)
```

Figure 6. Organization of Test Suite

You have to apply a ***white-box testing*** technique of your choice (but one that was discussed in the course) to each class to thoroughly test the class based on a coverage criteria of your choice (but discussed in the course).

You are not allowed to change the source code of the TouchCORE tool, i.e., you are not required to fix any bugs that you may find while testing the tool.

You are required to hand in the ***source code of your test suite*** and a ***report*** detailing your white-box testing technique(s) (i.e., how you determined which test cases are needed). You have to justify why you have chosen the technique(s) and coverage criteria you have chosen and discuss the coverage you have achieved with your chosen technique for each of the three classes under test. Furthermore, you have to include a coverage report from a coverage tool for all three classes under test.

Submission

Combine the three reports into one single document, i.e., the ***project report***. Clearly state the course name and number, term, group number, and team members on the title page of the project report. The project report (as well as each individual report) is to be formatted with single line spacing, Times New Roman 10pt font, and normal margins (2.54cm all around).

By Tuesday **November 24th, end of day**, you are required to hand in a ***single zip file*** of your two ***groupNN folders***. Before zipping up this folder, add the ***project report*** (which includes the three individual reports) to the root of your controller's groupNN folder. To create the zip file, use the *Export* feature of Eclipse to create an *Archive File* (Export – General – Archive File) of the project, but only select your two groupNN folders (and keep the directory structure).

Marking Scheme

Part of Project	***Marks***
Part 1 – Acceptance Testing	30
a) justification of chosen black-box testing approach	7/30
b) discussion of achieved coverage	7/30
c) test case description (execution, expected/actual results, verdict)	16/30
Part 2 – Integration Testing Strategy	24
a) correctness of dependency graph	10/24
b) justification of chosen integration testing strategy	7/24
c) description of drivers, mocks, and stubs	7/24
Part 3 – Unit Testing	44
a) justification of chosen white-box testing technique(s) and coverage criteria	7/44
b) discussion of achieved coverage including reports from coverage tool	7/44
c) JUnit Test Suite	30/44
On-time Setup of GitHub Repository	2
Total Marks:	100
The total mark may be adjusted based on the actual contributions of a team member to the project.	