TDA 594/DIT 593 - Assignment 3 - Preprocessor-Based Implementation

Due Date: Sunday, December 4, 11:59 PM
Submission: Via Gitlab and Canvas

Overview

After the last two exercises, your company is now convinced it should adopt a software platform. Specifically, you learned that (in group assignment 1) SPLE can be highly beneficial for companies, as long as proper engineering effort is made. You also analyzed (in group assignment 2) the existing products (i.e., example bots) with a domain analysis and now have a feature model that can be used to identify valid (and invalid) robot configurations.

In the future, it would be ideal to allow any new customer to get a highly customized bot by selecting a valid set of features. However, this now requires a proper implementation that supports variability. Now, you have decided to move forward and attempt a first integration of three software products (i.e., bots) into a common platform. You have decided to start with a particularly simple variability mechanism, using a preprocessor to selectively manipulate the source code on demand to produce a lean compiled unit with only the requested functionality.

Your goals in this assignment are to reengineer existing software variants using a diffing technique, identify features from code that represent variation and that help you distinguish two variants, and to establish a simple product line, relying on a configurable preprocessor-driven platform and your feature model.

Your Tasks

Your task is to integrate three bots BasicSurfer, BasicGFSurfer, and GFTargetingBot into a common codebase that can be extended in the future with additional bots. We provide the source code of the bots in a zip file in the file area on Canvas[1].

1. Do a pairwise diff among the three bots. This means that you take one bot as the base and examine the differences with each of the other ones. Observe the differences and try to understand why these differences are there, in order to identify features that are already accounted for in your feature model or missing from the model.
2. Extend and refine your feature model from group assignment 2. Ensure that your existing features are modelled correctly. Add any newly identified features into the relevant places in the feature model. After you've identified all the features from the three bots, create a second, simpler feature model where you have removed unnecessary features (e.g., those you identified from another bot previously, but are not present in any of the three bots that you are integrating). Define dependencies, if necessary, among the

[1] https://chalmers.instructure.com/courses/16077/files/folder/Assignments?preview=1757335

features, as additional cross-tree constraints. Follow the hints about feature identification below.

3. Integrate the codebases by establishing a platform with variation points using the Antenna preprocessor within FeatureIDE. Follow the FeatureIDE tutorial[2] for Antenna (select the composer “Antenna” when creating a new FeatureIDE project and then proceed in the section “Antenna”). Refactor the three bots into this newly created platform.
4. After you have created a basic platform by refactoring these three bots, use Antenna to produce concrete variants (i.e., select features and product code containing only those features).
    a. Produce three variants representing the original three bots.
    b. Produce two new variants. These bots must not be equivalent to the original bots, i.e., they must differ in their functionality.
5. Ensure that the variants (both the representations of the original bots and the new bots) all run in the simulator (you could even let your variants fight against each other).

Hints

You can use any diff tool to compare the source code of the bots. Some examples include the built-in tools in Eclipse, Notepad++ with the Compare plugin, or the tool Meld (https://meldmerge.org/). You will notice that two of the bots are rather similar, while one differs more from the other two. Therefore, you may find it easier to first integrate the two similar ones. Verify the integration by checking that the two robots correspond to a valid configuration. Generate each bot, then see if they yield the same behavior as the original bot that had no variability mechanisms by running them in the simulator. If it behaves the same as before, then the integration was valid. Check these two before you proceed to the third.

A feature should not represent a variant (a bot). So, if you have a feature that is named “BasicGFSurfer,” then you have not identified features properly. It is important that you can use the new platform to derive variants that did not exist before by flexibly combining features.

It is not required that you modify the Java code beyond what is required for integration (primarily, add variation points, i.e., Antenna conditional compilation directives). However, you are welcome to make changes or refactor code if it improves readability or maintainability of the newly integrated codebase. Detail any changes you have made in the report.

We recommend that you use the same combination of tools and versions as in Assignment 2: FeatureIDE 3.9, Java 17, and Eclipse 4.20. If you encounter issues, report them to your supervisor.

In addition to online tutorial material, you may find the following book helpful in understanding how to work with FeatureIDE: *Jens Meinicke, Thomas Thum, Reimar Schröter, Fabian*

[2] https://github.com/FeatureIDE/FeatureIDE/wiki/Tutorial

Benduhn, Thomas Leich, Gunter Saake. Mastering Software Variability with FeatureIDE. This book is available free electronically from the Chalmers library.

It is also possible to use Antenna outside Eclipse and FeatureIDE, if you would prefer to edit your code in another IDE like IntelliJ. We recommend against that, and do not provide support for this case, but essentially, you would run Antenna from the command line and provide some configuration (which you created by configuring the feature model within Eclipse) to it. FeatureIDE exports a `.properties` file when you select "Runtime Parameters" as the composer that would be useful for this purpose.

We recommend that you enable continuous integration to avoid making commits that break the build. Continuous integration is a process where developers are encouraged to integrate their changes into a shared code repository on a frequent basis. Often, a set of scripted actions take place automatically each time you commit changes to the code repository[3]. This can be used to compile the project each time you commit to ensure that the project still builds successfully. If you choose to add test cases (this is optional during this assignment, but can help), you can also execute those to assess whether your code works. The following tutorial explains to enable continuous integration[4].

Deliverable

Submit, via Canvas, the following (one submission per team):

- The source code of your refactored platform. Submit this on Canvas as a link to a "release" of your GitLab repository[5].
- A document in PDF format, containing:
    - A description of your integration strategy, including explanation of how you integrated and refactored the code and the challenges encountered.
    - The modified feature model from Assignment 2 and the simplified feature model as high-resolution images.
    - For each variant (version with non-selected features commented out), include the code. State the feature selection for that variant.
    - Screenshots of your variants operating in the simulator (screenshots can be individual or show combinations of the bots). One suggestion is to have the new variants battle the original ones and document the results (the winning rate should be about 50-50).

[3] For more information, see https://en.wikipedia.org/wiki/Continuous_integration
[4] https://chalmers.instructure.com/courses/16077/pages/continuous-integration-with-gitlab-ci
[5] For information on creating a release, see https://docs.gitlab.com/ee/user/project/releases/

Grading Guidelines

Note, these guidelines are intended to give some guidance, but are not exhaustive. Each supervisor will assign a grade based on the correctness and quality of your work.

Grade	Guidelines
5	● Consistent and well-explained integration strategy that thoroughly describes the process that you followed to integrate the different code into this version, including challenges encountered. ● Correct use of preprocessor directives to produce variants. ● Feature model has been updated and is consistent with the implementation. ● Features correspond correctly to functionality variation, and features do not directly represent single variants. ● New variants are not equivalent to original bots, and are consistent with the feature model. ● Refactored bots still function as expected in the simulator. ● New variants function as expected in the simulator.
4	● Consistent integration strategy. Detailed explanation of the integration process. ● Correct use of preprocessor directives to produce variants. ● Feature model is consistent with the implementation. ● New variants are not equivalent to original bots, and are consistent with the feature model. ● Refactored bots still function as expected in the simulator. ● New variants function as expected in the simulator.
3	● Some explanation is made of the integration strategy. ● Correct use of preprocessor directives to produce variants. ● Feature model is consistent with the implementation. ● At least one new variant exists, is not equivalent to original bots, and is consistent with the feature model. ● Refactored bots still function as expected in the simulator. ● New variants function as expected in the simulator.
U	● Integration strategy not properly explained. ● Implementation inconsistent with feature model. ● New bots do not exist, are equivalent to original bots, are not integrated successfully. ● Refactored or new bots do not function in the simulator.