

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

Due Date: Wednesday, December 9, 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 (group assignment 1) SPLE can be highly beneficial for companies, as long as proper engineering effort is made. You also analyzed (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. You also understand how to perform automated analyses of this feature model using SAT solvers.

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 the simplest 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 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 Canvas¹.

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

¹ <https://chalmers.instructure.com/courses/10761/files/folder/Assignments/resources?preview=903457>

any of the three bots that you are integrating). Define dependencies, if necessary, among the 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 tutorial² for Antenna-based variability (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. Product 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. Test the integration with 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. Also remember that some features may not contribute to visible robot behavior, and may instead provide “under the hood” functionality such as debugging or logging. 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 the use of FeatureIDE 3.7. You must use Java 8 or newer (we have tested with Java 8 and 15). We have tested with Eclipse 09-2020 and 12-2018. Therefore, we recommend the combination: (FeatureIDE 3.7, Java 15, Eclipse 09-2020). If you encounter issues, report them to your supervisor.

² <https://github.com/FeatureIDE/FeatureIDE/wiki/Tutorial>

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 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 suggest you enable Continuous Integration to avoid commits that break the build. The following tutorial will help³.

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 repository⁴.
- 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 during this process.
 - 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).

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 (Chalmers)	Grade (GU)	Guidelines
5	VG	<ul style="list-style-type: none">• Consistent and well-explained integration strategy that thoroughly describes the process, including challenges encountered.

³ <https://chalmers.instructure.com/courses/7439/pages/continuous-integration-with-gitlab-ci>

⁴ For information on creating a release, see <https://docs.gitlab.com/ee/user/project/releases/>

		<ul style="list-style-type: none"> • 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	G	<ul style="list-style-type: none"> • 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		<ul style="list-style-type: none"> • 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	U	<ul style="list-style-type: none"> • 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.