TDA/DIT 594 - Assignment 4 - Modularization and Dynamic Variability

Due Date: Sunday, December 20, 11:59 PM
Submission: Via Gitlab and Canvas

Overview

Previously, your company successfully created a small product line by integrating three bots into a preprocessor-based architecture. You have played a bit with different configurations and given the generated bots to test customers who like the idea. Your company realizes that this way, it can shorten time-to-market for new robot firmware builds, deliver more value to individual stakeholders, and offer increased customization. In summary, your company is now more confident that establishing a configurable platform is the way to go.

Unfortunately, it is quite clear that the preprocessor annotations will clutter the source code and lead to maintenance nightmares when more bots are integrated into a platform. You also believe that compile binding time is not necessary, and load-time configuration is more appropriate (perhaps, in the future, you will also support dynamic runtime binding).

Preprocessors were a great start. However, now, a proper architecture is needed, facilitating more modularity of features. Your company has decided to create a framework and refactor the existing features into modules adhering to the framework. You are shifting from a compile-time, tool-based, annotation-based implementation (preprocessors) into a load-time, language-based, composition based approach based on a framework and design patterns.

Following this assignment, you will be able to abstract lower-level variation into an object-oriented framework, judge about the applicability of design patterns and apply them when beneficial, and be able to modularize concerns (features) to the extent possible.

Your Tasks

Your task is to establish an object-oriented framework for bots and realize your platform from the previous assignment with it. Specifically, you will start with the same feature model you refined in Assignment 3 and the three bots that we provided code for.

1. Switch to dynamic variability using parameters (Lecture 6b) and design patterns (Lecture 7). This requires switching the project's FeatureIDE composer to RuntimeParameters instead of Antenna. FeatureIDE will then create a .properties file that you can use to configure a bot on launch (see hints below!).
2. Think about abstractions in terms of classes according to what you learned in this and other courses. Create a framework based on these bots and your feature model in a dedicated package (e.g., groupX.framework).

3. Integrate the codebases by refactoring the bots into your framework. Modularize your existing features as much as possible.
4. Your integrated codebase should implement at least one design pattern. This can be one of the design patterns discussed in class, or any other that supports variability in some form. As a rule of thumb, we expect to see at least one design pattern applied, but it is possible to incorporate more.
    a. Only apply design patterns if they make sense and their use can be justified. Do not keep adding design patterns if they would make the design less effective!
5. In addition to the functionality GuessFactorTargeting, implement the feature LinearTargeting within your framework based on the description in the Wiki.[1]
6. Implement one more (non-trivial) feature of your choice, either from your domain analysis (when you know the implementation exists), from the RoboWiki, or from another RoboCode source (i.e., a GitHub project).
    a. We recommend you find existing code and refactor it into your framework, rather than starting from nothing and coding yourself. Give credit to the original source in your code comments and submission.

Hints

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.

While you could start with your Assignment 3 code and try to change the FeatureIDE composer via the project properties, you may find it easier to create a new project and copy the feature model and the code over incrementally when creating the framework.

We suggest you enable Continuous Integration to avoid commits that break the build. The following tutorial will help[2].

For reading the properties file, do not use FeatureIDE's generated PropertyManager class, since that will try to read the file from the filesystem, which Robocode's security manager will not allow. Instead:

1. Use the provided class ConfigurationManager[3] to read the property file.
2. Copy the properties file into the bin/ folder as well as the folder where ConfigurationManager is stored. You can copy the file manually or use[4] a build file[5].

---

[1] https://robowiki.net/wiki/Linear_Targeting
[2] https://chalmers.instructure.com/courses/7439/pages/continuous-integration-with-gitlab-ci
[3] https://chalmers.instructure.com/courses/10761/files/folder/Assignments/resources?preview=999843
[4] https://rohitprabhakar.com/2010/02/03/how-to-run-ant-build-from-eclipse
[5] https://chalmers.instructure.com/courses/10761/files/folder/Assignments/resources?preview=999838

3. In robocode, navigate to options->preferences->development options and add your project. Alternatively, follow the instructions at https://robowiki.net/wiki/Robocode/Eclipse/Running_from_Eclipse

Deliverable

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

- A document in PDF format, containing:
    - A link to a "release" of your GitLab repository containing the source code of your refactored platform[6].
    - A description of your framework, including a class diagram outlining the architecture. Your diagram can be created in any tool of your choice, and can even be generated from your code using, for example, Eclipse plug-ins.
    - A description of your integration strategy and design decisions, including explanation of how you integrated and refactored the code, which design patterns were used, why you applied those specific design patterns, and the challenges encountered during this process.
    - A description of the two feature implementations.
    - Your updated feature model, if any further changes were made.
    - The property files for three example variant configurations (at least one should be different from the previous assignment).
    - 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	• Design is clearly described, features are cleanly separated, and a class diagram is present. • Consistent and well-explained integration strategy that thoroughly describes the process, including challenges encountered. • Consistent and correct use of design patterns, with clear and detailed rationale for their use. • LinearTargeting and another new feature are implemented and their implementation is described in clear detail (describe the functionality of the techniques, including the different outcomes of that functionality).

[6] For information on creating a release, see https://docs.gitlab.com/ee/user/project/releases/

		● Feature model is consistent with the implementation. ● New variants are not equivalent to original bots, and are consistent with the feature model. ● Refactored bots and new variants still function as expected in the simulator.
4	G	● Design is described and a class diagram is present, features are mostly separated (some overlap may be present). ● Consistent integration strategy. Detailed explanation of the integration process. ● Correct use of design patterns, with detailed rationale for their use. ● LinearTargeting and another new feature are implemented and their implementation is described. ● Feature model is consistent with the implementation. ● New variants are not equivalent to original bots, and are consistent with the feature model. ● Refactored bots and new variants still function as expected in the simulator.
3		● Design is described, features are mostly separated (some overlap may be present). ● Some explanation is made of the integration strategy. ● Correct use of design patterns, with some rationale for their use. ● LinearTargeting and another new feature are implemented and their implementation is described. ● Feature model is consistent with the implementation. ● New variants are not equivalent to original bots, and are consistent with the feature model. ● Refactored bots and new variants still function as expected in the simulator.
U	U	● Design is not described. ● Significant overlap between features. ● Integration strategy not properly explained. ● Design patterns used incorrectly, or not used at all. Lack of justification for their use. ● 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.