Modeling, reverse-engineering, and testing variability : The Jhipster case study

Jhipster is a tool for generating a complete and modern Web stack, at the back and front-end level (server side with Spring Boot, clientside with AngularJS and Bootstrap, and Yeoman, Bower, Grunt and Maven for builiding the application).

Jhipster is quite popular with more than 3000 stars in Github. It is available online : https://jhipster.github.io/

During the courses we have illustrated the use of variability techniques with Jhipster. We have seen how users can configure Jhipster and how a derivation process produces different artefacts (e.g., Java classes).
We now want to go further. We will investigate variability modeling techniques to reason about the configuration space of Jhipster, with the long-term goal of verifying all configurations of Jhipster.

The exercises below can be seen as preliminary steps before comprehensively conducting an empirical study of a variability-intensive system. It can contribute to the field of product lines and thus some questions are open and non trivial[1]*.*

We ask you a series of questions. The answers should be written and organized in a technical document (PDF format). Feature models should be provided as well.

1. Write a technical explanation (2 pages max.) describing how Jhipster is configured -- from the configuration process in the console to the actual execution of a variant of Jhipster. A reader should understand the whole process and the artefacts involved/generated. Please also emphasize the concepts seen during the courses (domain-specific languages, model transformations, variability, etc.)
2. We want to model the configurations authorized by Jhipster. Explain why playing with the configurator and tries every possible combination of options is not a viable solution.
3. Write a feature model that characterizes the configurations of the Jhipster configurator. (We will use FAMILIAR for this purpose.) A valid configuration of the resulting feature model should be authorized by the configurator (for instance, Java 8 and Java 7 are mutually exclusive). Due to the limits of a

[1] Some researchers in DiverSE are currently working on this subject

manual strategy based on playing the configurator (see point 2), it is highly recommended to have a look at some artefacts of Jhipster for specifying the feature model.
4. Compute the number of valid configurations of Jhipster
5. Explain the principles of an automated, reverse-engineering procedure for obtaining the feature model of Jhipster
6. Using github, report some exitsing bugs that have been related to configuation issues. For each bug, explain the consequence for maintainers of Jhipster
7. Explain how we could re-find existing bugs of Jhipster using a feature model
8. Using the history of git and this automated procedure, what insights could be inferred ?
9. Propose a strategy for testing the configurations of Jhipster at each commit or release

Please send the document to mathieu.acher@irisa.fr
A defense of your work will be done the 8th january 2016 (20' of presentation + 10' of questions).
The presentation is typically structured as follows

- 5' about Jhipster (addressing point 1)
- 5' about modeling variability of Jhipster and concrete feature models obtained (adressing points 2, 3 and 4)
- 7' about reverse engineering and testing (points 5, 6, 7, 8, and 9)
- 3' for concluding and opening perspectives

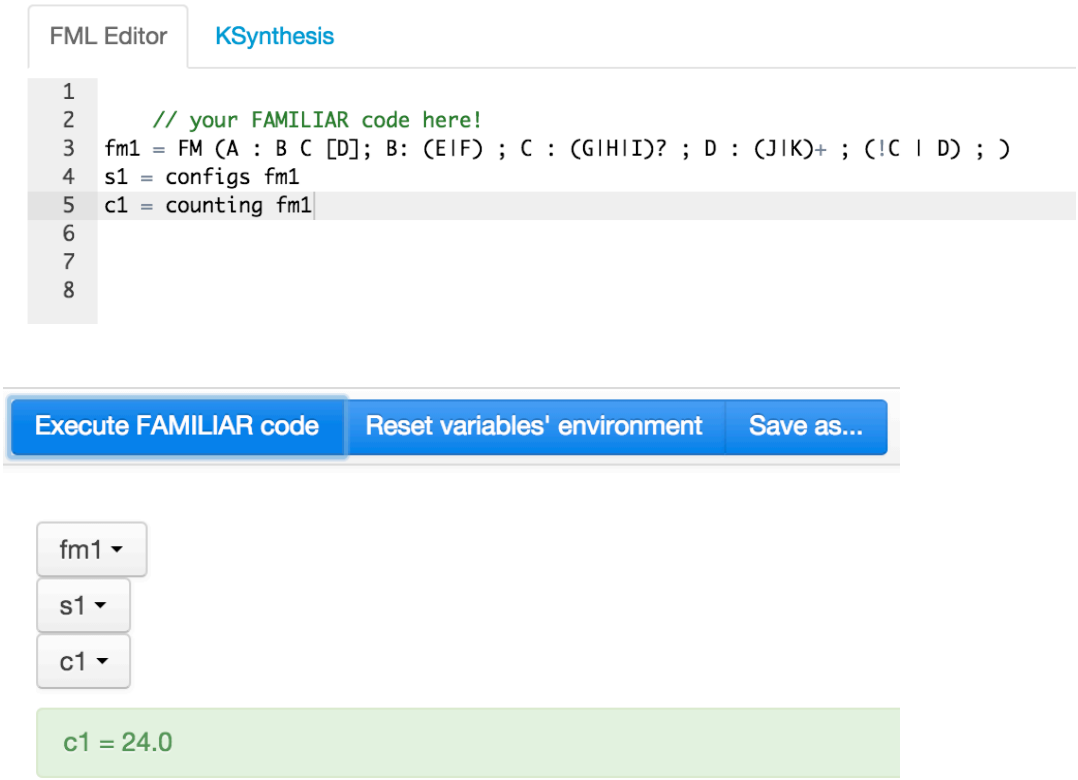
FAMILIAR

FAMILIAR is available as a Web application written using the Play framework (https://www.playframework.com). The app starts a Web server and is accessible here:
http://127.0.0.1:9000/ide/familiar

For getting FAMILIAR : http://mathieuacher.com/teaching/MDI/fmlapp-1.0-SNAPSHOT.tgz

There is an fmlapp.bat (./fmlapp in Linux/MacOS) under the folder bin/

You can then write a FAMILIAR script with a textual editor and then execute it.



FAMILIAR operations (configs, cores, counting, isValid, etc.) are documented here: https://github.com/FAMILIAR-project/familiar-documentation/blob/master/manual/