Université catholique de Louvain
Ecole Polytechnique de Louvain

LINGI2252
Software Maintenance and Evolution

Implementation of a feature-based context-oriented system

Lab Session 6

Professor: Kim Mens
(kim.mens@uclouvain.be)

Teaching assistant: Benoît Duhoux
(benoit.duhoux@uclouvain.be)

November 18, 2020

1 Approach of the lab sessions

Since the beginning of these lab sessions, we have been exploring how to design smart or context-aware software systems using context and feature modelling. We also proposed a specific development methodology to help design such systems. In the last lab session, we introduced the key notions of our feature-based context-oriented programming language. However, with only these key notions, we are not yet able to implement complex systems. For that, we need to dig a bit deeper and explore some more advanced notions of our programming language, as well as how we can build and adapt user interfaces with it, for such complex systems.

After having given you an introduction to some of the more advanced concepts of our programming language, we will first ask you to explore and understand a provided smart *e-Shop* system, to gain a better comprehension of the different notions that were introduced during the presentation. Next, we will ask you to extend this prototype system to get a first hands-on experience with the full expressive power of our programming language. Finally, we will ask you to implement a smart system that will be assigned to you. This last exercise will not be scored but will be part of our scientific study on the expressiveness of our approach and programming language. Participating will nevertheless be useful for you because it will give you a good hands-on feeling of the language, and an analysis of the implementation of the language will be the subject of the third evaluated mission.

2 Running a feature-based context-oriented application

We have already explained how we can run a feature-based context-oriented application with the command *ruby main_script.rb*. However, running the application like that does not allow us to interact with it by (de)activating some contexts at run-time to simulate how changes to the environment surrounding the application would impact its behaviour. Now we will explain how we can simulate changes to the surrounding environment through the *Context Simulator* command line tool.

First, you have to launch a server with the option `ContextSimulator` by running the command `ruby server.rb - -ContextSimulator` in the folder `tools/server`.

Then, you have to run the *Context Simulator* tool. This tool is a command line tool for which the commands must respect the format
`activate: context_name, context_name;`

`deactivate: context_name, context_name`
where the attribute `context_name` must be the name of a context in your context model (and not the instance variable of the context in your context declaration). You can either give a command for an activation/deactivation of one or several contexts, or you can give a full command for an activation and a deactivation of one or several contexts, whatever the order of your command.

Finally you must run your application with `ruby main_script.rb` in your application folder.

3 Analysing and understanding some advanced concepts of the language

This exercise will be time-boxed to 30 minutes maximum.

As a first exercise, you have to analyse the provided smart *e-Shop* system, named *smart_e-shop_v0.3.0*, in the *apps* folder of the archive *RubyCOP.zip* to better understand how you can use some advanced notions of our feature-based context-oriented programming language.

Before diving into the code, run the application as explained in section 2. By default we have activated the `Desktop` context. Now deactivate this context and activate the `Smartphone` context in a single transaction with the command `deactivate:Desktop; activate:Smartphone` in the *Context Simulator* command line tool [1]. Can you understand and explain why it is important to activate the `Smartphone` context and deactivate the `Desktop` context simultaneously? What would happen if we would run only part of the command?

Then, explore how we have implemented the user interface components of the provided application [2] and how we composed the different UI widgets thanks to the *proceed* mechanism.

During this code inspection you will probably have noticed how the *Observer* design pattern was used to implement the user interface. If you never encountered this design pattern in the past or you do not remember how it works, please consult `https://refactoring.guru/design-patterns/observer`

[1]To see the modifications, put the focus on your app.

[2]A documentation of the UI library we have used as in our program is available at `https://rubydoc.info/gems/fxruby/frames`. You can also find a reference book at `http://index-of.es/Ruby/FXRuby%20-%20Create%20Lean%20and%20Mean%20GUIs%20with%20Ruby%20(2008).pdf` to understand some unexplained concepts in the documentation.

which provides a generic explanation of it. However, you will notice that we did not follow exactly the implementation template provided in this documentation of the design pattern. Instead, we preferred to use the built-in mixin implementation of this pattern in Ruby. To get more information on this mixin, read its documentation at `https://ruby-doc.org/stdlib-2.5.5/libdoc/observer/rdoc/Observable.html`. As your own implementation, which you will have to program next, should be as maintainable and reusable as possible, it could be interesting to consider using this design pattern for your implementation of the UI components.

4 Extending the smart *e-Shop* system

After having investigated how we implemented our prototype of a smart *e-Shop* system, you are now asked to practice these advanced concepts yourself, by extending our smart *e-Shop* system to order products. You can reuse your previous implementation for the core logic of ordering products, but you will now have to implement the user interface components as well. To implement this extension try to adhere to the wireframes illustrated in Fig. 1 and in Fig. 2, for a desktop and a smarpthone context, respectively.

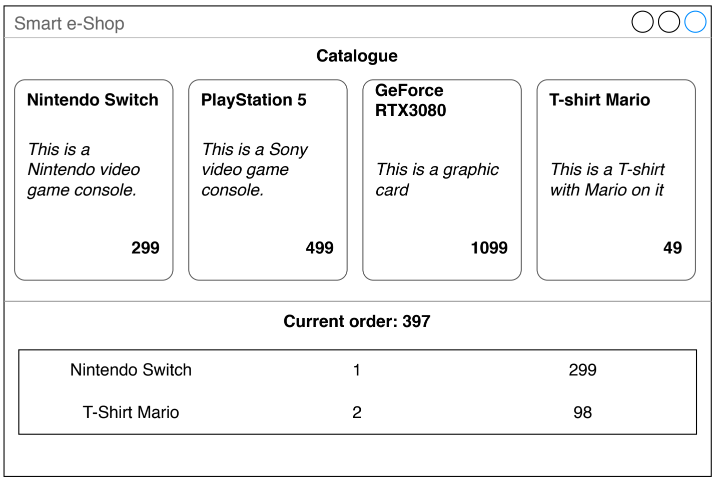
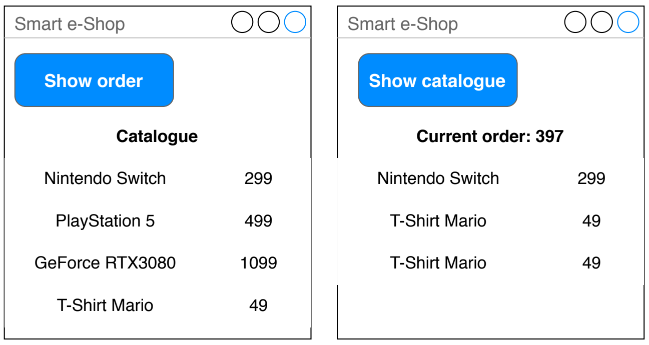


Figure 1: Wireframe for a desktop.



Catalogue

Nintendo Switch	299
PlayStation 5	499
GeForce RTX3080	1099
T-Shirt Mario	49

Current order: 397

Nintendo Switch	299
T-Shirt Mario	49
T-Shirt Mario	49

Figure 2: Wireframe for a smartphone.

5 Developing your assigned smart system

During the two last lab sessions we have introduced and explained our feature-based context-oriented programming language through presentations and exercises. Now that you have gained some expertise in the development of smart systems with our programming language, we ask you to implement the case study you have designed with our feature-based context-oriented methodology during lab session 3. In other words, **all B.xx groups must implemented the smart messenger application while the A.xx groups have to develop a smart calculator system**. **This last exercise is entirely dedicated to our scientific research**. In this experiment, we will ask you to develop your smart system based on your context and feature model. Of course, you can update your design if you had some design issues in your models. In your implementation you should develop both **the core logic and the user interfaces**. You have to submit your implementation, in whatever state it is, by **November 27th, 2020 at 11:55 pm**. The submission platform will be provided later by an announcement on Moodle. For this submission, you **have to submit an archive named "smart_X_Y.zip"** where 'X' is either "messenger" or "calculator" and 'Y' is your same anonymous identifier you have used since the beginning of the semester. This archive must contain a report illustrating your context and feature model, as well as the mapping model between these two models and a folder containing all your source code of your smart system. For guaranteeing anonymity please make sure that your real names are is not in the metadata of any file or directory.

As this last exercise is dedicated to our scientific research, you will not be graded for this submission but please play the game as real developers using our programming language. Even if you would not have managed to fully implement your smart system, even submitting your non-complete submission is relevant to us. This will allow us to analyse how you have used our programming language, and perhaps what problems you encountered. We count on you!