ÉCOLE POLYTECH'NICE SOPHIA
SI5/M2 INFO
2021-2022

Julien Deantoni

Domain Specific Language

UXifier

Project delivery

This project concerns the second delivery of the DSL course. It will include the implementation of a DSL focused on the specification of a specific family of UI (to be defined by yourself), in either an internal **or** an external DSL. Deliveries are expected by email (to Julien Deantoni: firstname.lastname@univ-cotedazur.fr, with [DSL] as object prefix) followed by "team X lab 2" where X is the name of your team (as used in the slack dedicated channel). The delivery is expected before the 1st of March 2022 at 10:00PM Paris Time. The delivery is expected as a PDF report (please let your report be succinct and rigorous).
The report must contain :

- the name of the members of your team
- a link to the code of your DSL (typically a link to the git repo)
- a description of the language proposed:
  - the domain model represented as a class diagram;
  - the concrete syntax represented in a BNF-like form;
  - a description of your language and how it was implemented;
  - a simple description on how you wrote the compiler to obtain executable code
- a set of relevant scenarios implemented by using your language;
- a critical analysis of (i) DSL implementation with respect to the UXifier use cases and (ii) the technology you chose to achieve it;
- responsibility of each member in the team with respect to the delivered project.
- a rationalization of its main usage and all of its features. The associated script and materials will be provided.

Objectives: Define the UXifier language

Your objective here is to define a DSL allowing the definition of various, possibly complex web app user interface. The language main focus on one (or several) family of application. For instance one may focus on the definition of Blog (which will not be allowed since given as an example ;). Classic functionalities contain creation of the app, its menus, the different zones where actual content will be put. The DSL should not cover the functional part of the app that is not generalizable to the entire family of web app user interface. It can bring some functionalities like for instance the *responsive* aspect.

From your programs written in the $UXifier$ language, you will generate executable code. The target language is your choice, but it should be usable under different operating systems and used for web app. I propose to use Grommet (`https://v2.grommet.io/`) framework but you can use anyone as long as you I do not have to do much to have it on my machine (running linux OpenSuse Tumbleweed).

Usually, basic scenarios are provided to let you understand what is expected for your DSL. Since it depends on the family of web app your focusing on, these basic scenario should be defined by each team. The goal is to illustrate through simple scenarios the basic functionalities that are expected to be supported by your DSL.

Remember that a DSL is not *only* a language. It should come with a set of services to help the acceptance of the end user. In your delivery these services will also be illustrated through basic scenario to rationalize their usefulness.

Since the functional part will not be part of your DSL (e.g., what happens when clicking a specific button will not be part of the DSL), you should support the user to avoid deleting her/his code when a change is done on the UI by using the DSL. This means that the functional part should be clearly identified in a way or another to avoid the code generation to remove them.

Finally, you will propose (at least) two *killer features* that provide great plus value to the use of your DSL.

Any of the points above may be discussed between you and me during the course hours or on the slack.

Commmon Parts

The following parts must be available in your DSL:

- **Domain Model**:

  The domain model (a.k.a. abstract syntax) should be clearly identified in the delivered code. It will be provided as a class diagram, together with explanation about the main choices you did.

- **Concrete syntax**:

  The concrete syntax, in a eBNF form, must be clearly identified and used by a relevant set of scenarios. The syntax must leverage the tool chosen to implement it to make it clear and easy to use.

- **Validation**:

  Support your end-user by checking that a model is realizable on the expected platform. For instance you can enforce good practices to avoid overloaded menu, difficult to use UI, etc.

- **Code generation**:

  Provide a generator producing code whose execution results in a usable web app. This code can be directly compiled by a script generation (if needed).