Declarative User Interfaces : specifying UI variabilities

Sofie Goderis Dirk Deridder
sgoderis@vub.ac.be dderidder@vub.ac.be
Programming Technology Lab
Vrije Universiteit Brussel, Pleinlaan 2, B-1050 Brussels, Belgium

1 Introduction

In our research we apply the principle of separation of concerns onto User Interfaces. The goal is to get a clean separation in the source code between code responsible for UI logic and code responsible for the application logic. Since both are tightly coupled (i.e. the application logic needs to interact closely with UI logic and vice versa), this results in code entanglement which reduces the evolvability of the software. Therefore it is difficult to vary the way a user interface is layed out (e.g. using a tabbed, multi-column, or multi-windows strategy) or how it should be handled on different platforms (e.g. PDA , internet based).

In our approach we use SOUL, a declarative meta programming environment, to describe the different UI concerns. This declarative specification is taken as input by our SoGUI reasoner, that transforms it into a working user interface (where application logic and UI logic are once again woven together).

This way it is possible to vary the UI visualisation and behaviour by merely changing the declarative specifications (facts and rules) that describe these UI concerns.

Each variability is captured by a set of rules. The reasoning mechanism combines the different sets to generate the UI variant emerging from the chosen variabilities.

For specifying the UI we anticipate the use of visual tools. Once the UI is modelled visually, the tool translates the specification into a declarative specification for this is the base modelling language in our approach. Next the reasoning mechanism transforms this model into the actual working UI (i.e. the actual code).

2 Separating UI concerns with a declarative approach

As said before, we use a declarative environment for describing the different UI concerns. For now we focus on the UI concerns related to the UI visualisation and the UI behaviour. Visualisation is what the UI looks like and what widgets are provided. Behaviour specifies how widgets relate to and influence each other.

Furthermore we distinguish different levels of abstractions within these concerns, going from high level to lower level to low level output (i.e. device or platform) dependent specifications. For example the high level specification gives an abstract description of the UI using 'questions'. The lower level indicates that a 'question' is a combination of a label, some radio-buttons and an action button while the low level output refers to the actual label and button widgets of a Java GUI.

A reasoning mechanism is responsible to compose the different sets of rules and generate the actual UI, which is the UI code woven together with the application code.

3 UI variabilities

Since we have a high level declarative specification that describes the abstract UI and a low level specification that describes the actual implementation details, it is possible to vary the UI visualisation and UI behaviour and therefore create several variants for a certain UI.

Consider a common application to be deployed to different devices. The high level UI specification (e.g. the different components) for this application does not change. The low level implementation however is different for the different devices (e.g. what a component actually looks like). When replacing the set of rules responsible for transforming the high level specification into a certain low level implementation with another set, a variant of the same high level UI is created and the variabilities are captured within the rule sets.
The same methodology of replacing one rule set by another applies for other UI variabilities. For instance when varying the UI visualisation by replacing one layout strategy (e.g. one-column) with another one (e.g. two-column).

4 Composition of UI variabilities

Since we use a transformation step that combines different sets of rules together into an actual UI, one can combine different sets to achieve different UIs.

For instance UI components or UI behaviour are grouped together by means of rules and facts. Such as a save-edit-cancel buttons group that combines three buttons with a certain enable/disable behaviour. This behavioural group can be used throughout different UIs because of the possibility to compose UI variabilities. A user selects all necessary elements and variabilities of the UI, the reasoning mechanism combines then into the actual UI variant.

5 Model to SOUL to Code

Currently we are implementing tool support for visualising parts of the UI specifications. For a programmer it is easier to specify a UI graphically, nevertheless we do not want to loose any of the declarative expressiveness. These visualisations are transformed into the declarative facts and rules to be used by the declarative reasoning engine. Together with the more general set of rules (such as the device specific rules) the reasoning engine then creates the final UI, and therefore the UI code and the linking code with the application.

6 Conclusion

User Interfaces can vary in different ways. In our research we use a declarative approach to specify UIs, which provides us with a way to specify and combine UI variabilities. Furthermore, because of the use of a declarative reasoner we can combine several variabilities. Finally the declarative UI specification is a way of modelling a UI and is used in model to model and model to code transformations.