Chapter 14

Android Executable Modeling: Beyond Android Programming

Olivier Le Goaer
University of Pau, France

Franck Barbier
University of Pau, France

Eric Cariou
University of Pau, France

ABSTRACT

Within the model-driven engineering field, model execution consists in interpreting the model through a dedicated execution engine instead of executing a code based on, or generated from, the model. The class of modeling languages endowed with such executability is called i-DSML (interpreted Domain-Specific Modeling Language). This is an important development shift because a modeling effort seamlessly substitutes to a programming effort. This alternative way for building increasingly complex software is particularly beneficial to the mobile applications market where fast development and agility are recognized as key factors of success. This chapter illustrates how parts of an Android mobApp can be modeled and executed by leveraging a well-known i-DSML, namely UML 2 State Machine Diagrams and the PauWare engine thereof. Beyond this specific case, the proposed installation of PauWare on Android OS sets up the foundation for a whole range of mobApps, provided that they are modeled with the Statecharts formalism.

INTRODUCTION

Among the manifold forms taken by abstraction, modeling has proved its efficiency for handling complexity of software development, contrasting with classical programming all focused on source code. Indeed, models abstract away details to concentrate on particular, high-level, viewpoints on the system to be built. As such, they offer good reasoning supports to designers. Models are so powerful that they have earned their place in the software engineering through a dedicated sub field called Model-Driven Engineering.
Android Executable Modeling

Engineering (MDE). After having been intensively used as contemplative assets until the mid 2000, models have been turned into productive assets, relying on automated transformation chains ending predominately to source code. It is worthwhile mentioning that this evolution owes much to the OMG’s MDA initiative (Miller et al., 2003). A more recent trend is to see a model as an end in itself by directly executing it (Lehmann et al., 2010 and Combemale et al., 2012). The analogy could be now that of an “animated” or “enacted” blueprint; it may serve as a way of simulation of course, but also as a full-fledged executable system so that the implementation stage is totally skipped. This vision shift from static (albeit productive) model to dynamic model tends to abolish the boundaries between modeling and programming; its slogan might be “What you model is what you get” (WYMIWYG).

The entirely model-centered and hence fast development allowed by interpretable models is particularly interesting when focusing on tiny devices or on embedded software like Smart-* (Phones, Watches, Glasses, TVs…) equipment. Indeed, these applications are characterized by a high time-to-market pressure and a rapid fluctuation of user’s requirements, while they run on top of fast-paced platforms. This situation is going to explode with the future Internet of Things (IoT), and the arrival of a multitude of connected objects. Because a growing number of these systems are running with Android, we choose this operating system to put into practice our ideas.

This chapter is organized as follows: we first provide an overview on what is the recent i-DSML approach. Then, we focus on the Statecharts i-DSML and the PauWare engine and how to take advantage of it when building Android mobApps. Next, we exemplify on a toy mobApp that provides energy-saving assistance, and give all implementation details on a separate section. We also explain how it is possible to cope with changes once the mobApp is deployed. Finally, related works are discussed before we conclude.

i-DSML AT A GLANCE

It is worth recalling that modeling is all about abstraction. Modeling challenges programming in the sense that it requires different skills, neither more nor less valuable, just different. A smart software engineer makes routinely and fluently the round-trip between these two levels of abstraction.

While the models, as outputs of a modeling process, were often considered as contemplative artifacts before the 2000s (intended to be printed and pinned to wall, to be caricatured), the situation has deeply changed since: everyone is nowadays making a productive use of models. This shift has been logically accompanied with a “modeling liberation/emancipation” supported by the concept of domain-specific modeling languages (DSML) (Fowler, 2010): everyone was encouraged to use multiple modeling languages, for each particular aspects of software, not only relying on mainstream, general-purpose modeling languages (GPML). The productive usage of models written with a DSML requires executability, which is thereby no longer the exclusive attribute of programming languages. As noted by Mernik (Mernik et al., 2005), model execution can be achieved in two ways:

- Compiled DSML: DSML constructs are translated to base language constructs and library calls. People are mostly talking about code generation when pointing at this approach;
- Interpreted DSML: DSML constructs are recognized and interpreted using an operational semantics processed by an execution engine. With this approach, no transformation takes place; the model is directly executable.