Chapter 13
Adaptive Software based on Correct-by-Construction Metamodels

Franck Barbier
LIUPPA, France

Pierre Castéran
LaBRI, France

Pierre Castéran
LaBRI, France

Eric Cariou
LIUPPA, France

Olivier le Goaer
LIUPPA, France

ABSTRACT
Despite significant research efforts in the last decade, UML has not reached the status of being a high-confidence modeling language. This is due to unsound foundations that result from the insufficiently formal structuring of metamodels that define the MOF/UML Infrastructure. Nowadays, UML-related metamodels are implemented in computing environments (e.g., EMF) to play the role of metadata when one seeks adaptation at runtime. To properly instrument metamodel-based adaptation, this chapter re-formalizes the core of the MOF/UML Infrastructure along with giving formal proofs that avoid ambiguities, contradictions, or redundancies. A (meta-)class creation mechanism (either by instantiation or inheritance) is based on inductive types taken from the constructive logic. Inherent proofs based on the Coq automated prover are also provided. This chapter’s contribution is aligned with a previously established metamodeling framework named “Matters of (meta-)modeling.”

INTRODUCTION
In recent years, Model Driven Development (MDD) has been considered as a suitable software engineering technology to make software potentially and really adaptive at runtime (Zhang & Cheng, 2006). This gave rise to the notion of models at runtime (Fleurey & Solberg, 2009; Morin et al., 2009). Models at runtime are embedded models within the system during its execution and aim to express adaptation policies, conditions, and rules on models representing the system under execution. On another hand, research contributions
on MDD-based software admit the unavoidable need for executable modeling languages.

Model execution consists in interpreting the model through a dedicated execution engine instead of executing a code based on, or generated from, the model. In this context, the adaptation deals with directly modifying and acting on the model under execution (Cariou et al., 2012).

Practically, based on its meta-meta-model Ecore (an implementation of the MOF) and its reflection capabilities (a.k.a. metadata), the Eclipse Modeling Framework (EMF) (Steinberg et al., 2008) has become the reference platform for developing MDD tools. EMF has then played a great role in implementing executable modeling languages. For instance, Ker-meta (www.kermeta.org), an action language relying on Ecore metamodels, facilitates the development of execution engines. Such executable modeling languages can be defined from scratch, following the concept of DSML (Domain-Specific Modeling Language).

Executable modeling languages can also be based on an effort to make existing languages like UML (or some of its constituents) executable. In this scope, SCXML (State Chart XML) and PauWare (Barbier, 2008) are two tools that offer the possibility of executing Harel’s Statecharts.

SCXML is a standard (www.w3.org/TR/scxml). Its most well-known implementation is that of the Apache foundation: Commons SCXML (commons.apache.org/scxml). SCXML possesses its own metamodel based on a lightweight XML notation, which is independent of EMF and UML in general. Differently, PauWare (www.PauWare.com) is a Java library that is properly aligned with UML State Machine Diagrams.

In contrast with EMF-based executable modeling languages defined for instance with Kermeta, SCXML models may be embedded in applications’ components and interpreted in real-time. For efficiency, PauWare prefers a Java-based representation of models that requires very little memory and an accelerated models’ interpretation. For instance, a PauWare-based application may be embedded in a tiny device (an Android or Java ME device for instance) where SCXML requires more space because of XML. PauWare code may indifferently be generated from XMI (State Machine Diagrams sub-component of UML) or from the SCXML notation. In (Ballagny et al., 2009), we proposed an adaptation-enabled version of PauWare named MOCAS (sourceforge.net/projects/mocasengine).

The common point of all these adaptation techniques and tools is that they are based on the usage of models such as UML State Machine Diagrams. Unfortunately, the continuous enhancement of MOCAS stumbles over the unsound structure of the UML Superstructure metamodel relating to state machines. A systematic restructuring of this metamodel was therefore required. A sample of a revised metamodel for state machines is proposed (in the “Example” section of this chapter) along with a method to formally prove that revisions are well-founded. This occurred by means of the Coq automated theorem prover (coq.inria.fr) to express metatypes as inductive types that come from the constructive type logic. In this chapter, we propose a more ambitious method based on the same logic that leads to build from scratch a Domain-Specific Modeling Language (DSML) dedicated to software adaptation. The definition of this DSML requires to have at one’s disposal a metamodelling framework allowing to formally define a model conforming to a metamodel.

The key issue is the definition of a minimal kernel of metatypes and metarelationships that may serve as bootstrap for the rest of the language to be defined. Precisely, adaptation requires reflective capabilities that are embodied by metacircularity in the language’s kernel. We illustrate this problem with the Class and Property metatypes of the UML Infrastructure metamodel (Figure 1). Namely, classes are structurally composed of properties (black diamond notation in UML). Chronologically speaking, mutual dependencies in definitions show cycles, namely Class is defined from Property, which is defined from