Chapter IV

Improving the Understandability of Dynamic Semantics: An Enhanced Metamodel for UML State Machines

Eladio Domínguez, Universidad de Zaragoza, Spain

Angel Luis Rubio, Universidad de La Rioja, Spain

María Antonia Zapata, Universidad de Zaragoza, Spain

ABSTRACT

A clear understanding of the dynamic semantics of languages involved in the representation of behavior is essential for a large and varied audience such as final users of these languages, CASE tool builders or method engineers. This chapter introduces a proposal aimed at achieving such an understanding by suggesting a different metamodeling approach. This approach is based on a two layer architecture which puts forward the explicit distinction between the generic behavior represented in a dynamic model (Base Layer) and the behavior represented in relation to a particular situation (Snapshot Layer). Using this architecture as a starting point, a metamodel of UML State Machines is proposed, which consists basically of two UML class diagrams (one diagram for each layer of the architecture) and two maps. These maps represent, respectively, the determination of the initial status and the process performed by a run to completion step as defined in the UML semantics.
INTRODUCTION

The statechart technique is a visual formalism defined as an enhancement of finite-state machines, originally developed by D. Harel (Harel, 1987) to specify complex reactive systems. Much literature has been written on this topic in recent years and in particular, a large number of variants of the technique have been proposed (Beek, 1994). More recently, the success of the statechart formalism has received a major boost since an object-oriented adaptation of the technique, namely State Machines, has been adopted as part of the Unified Modeling Language (OMG, 2003; Rumbaugh, Jacobson & Booch, 1999).

There are many works which define a complete formal semantics of Harel’s Statecharts (see, for example, Ehrig, Geisler, Klar & Padberg, 1997; Harel, Pnueli, Schmidt & Sherman, 1987; Harel & Politi, 1998; Hong, Kim, Cha & Kwon, 1995; Hooman, Ramesh & Roever, 1992; Beek, 1994; Maggiolo-Schettini, 2003). However, a known shortcoming of UML State Machines is that in the UML specification document (OMG, 2003), although the syntax and static semantics of State Machines are precisely stated, the dynamic semantics is not rigorously defined (Engels, Haussmann, Heckel & Sauer, 2000; Latella, Majzik & Massink, 1999; Lilius & Paltor, 1999). Undoubtedly, a precise specification of the behavior of State Machines is essential for a large and varied audience. For instance, the final users of the language (such as system analysts and designers) need at least an overall but accurate idea of how a state machine behaves. Secondly, CASE tool builders interested in supporting State Machines greatly benefit from having an unambiguous specification of the language. Finally, method engineers would use a precise specification of State Machines to analyze issues such as language adaptability, comparison with other behavioral approaches, transformation, and so on. This complex situation has resulted in the definition of a precise dynamic semantics of State Machines being the subject of recent intensive research (Borger, Cavarra & Riccobene, 2000; Engels et al., 2000; Jin, Esser & Jannec, 2002; Latella et al., 1999; Lilius & Paltor, 1999; Mann & Klar, 1998; Reggio, 2002; Reggio, Knapp, Rumpe, Selic & Wieringa, 2000; Varro, 2002).

The problem is that the majority of approaches that try to establish a precise dynamic semantics of State Machines make use of formal notations such as Rewrite Rules (Kwon, 2000; Lilius & Paltor, 1999), Hierarchical Automata (Latella et al., 1999), Abstract State Machines (Borger et al., 2000) or Object Z (Mann & Klar, 1998). However, like other authors (Engels et al., 2000; Reggio, 2002), we think that these approaches have the drawback of being difficult to read and understand, and therefore they are not wholly suitable since dynamic semantics must be precisely established but in such a way that the understandability and readability of the specification is facilitated.

Without neglecting the need for formal notations when issues such as verification or model checking have to be dealt with, we propose to adopt a metamodeling approach which is a widely accepted way of improving the properties of understandability and readability (Hofstede & Verhoef, 1997; Verhoef, 1993). This proposal is based on a two-layer architecture we outlined in Domínguez, Rubio and Zapata (2000a, 2000b). This architecture makes explicit the distinction between the generic behavior represented in a dynamic model (Base Layer) and the behavior represented in relation to a particular situation (Snapshot Layer). In addition, the concept of movement from a current situation to another is captured by using the notion of mapping. Using this architecture as a starting point, a metamodel of UML State Machines is proposed, which consists basically of two UML class diagrams (one diagram for each layer of the architecture) and two maps. These maps represent, respectively, the