Chapter 71
Towards Dynamic Semantics for Synthesizing Interpreted DSMLs

Peter J. Clarke
Florida International University, USA

Frank Hernandez
Florida International University, USA

Yali Wu
University of Detroit Mercy, USA

Mark Allison
Florida International University, USA

Andrew A. Allen
Georgia Southern University, USA

Robert France
Colorado State University, USA

ABSTRACT

Domain-specific languages (DSLs) provide developers with the ability to describe applications using language elements that directly represent concepts in the application problem domains. Unlike general-purpose languages, domain concepts are embedded in the semantics of a DSL. In this chapter, the authors present an interpreted domain-specific modeling language (i-DSML) whose models are used to specify user-defined communication services, and support the users' changing communication needs at runtime. These model changes are interpreted at runtime to produce events that are handled by the labeled transition system semantics of the i-DSML. Specifically, model changes are used to produce scripts that change the underlying communication structure. The script-producing process is called synthesis. The authors describe the semantics of the i-DSML called the Communication Modeling Language (CML) and its use in the runtime synthesis process, and briefly describe how the synthesis process is implemented in the Communication Virtual Machine (CVM), the execution engine for CML models.

INTRODUCTION

Research in model-driven software development (MDD) (France & Rumpe, 2007; Hermans, Pinzger, & Deursen, 2009) and domain-specific modeling languages (DSMLs) (DSM Forum 2010, Sprinkle, Mernik, Tolvanen, & Spinellis, 2009) focuses on how models that provide good abstractions of complex software behaviors can be used to significantly improve software productivity and quality. In this work we differentiate between the two categories of domain-specific languages described in the literature; these are (1) text-based languages, referred to as DSLs (Mernik, Heering,

DOI: 10.4018/978-1-4666-6042-7.ch071

& Slone, 2005), and (2) graphical modeling languages, DSMLs. It should be noted that graphical languages are usually processed in an equivalent text representation, usually based on some form of XML. The use of DSMLs continues to grow in both academia and industry, resulting in a spectrum of DSMLs. This spectrum includes DSMLs that are used in the development of software artifacts which are then translated into a general-purpose high-level programming language, as well as DSMLs that are used to model the domain application and that are directly executed by a model execution engine. In this chapter we focus on the latter class of DSMLs, referred to as *interpreted DSMLs* (i-DSMLs), i.e., those that do not transformed models into the source code of another language. As we will demonstrate in this chapter, i-DSMLs are well-suited for creating and changing models at runtime.

Creating a DSML involves defining: a metamodel (abstract syntax and static semantics), one or more concrete syntaxes, and the dynamic semantics (Stahl et al., 2003). There are several tools that can be used to define the metamodel and concrete syntaxes for DSMLs, however there is little support for defining the dynamic semantics. Support for the definition of dynamic semantics for i-DSMLs is needed if they are to be used to produce models used at runtime to modify behavior, particularly in distributed runtime environments.

In this chapter, we describe an approach we used to develop the dynamic semantics of an i-DSML for the communication services domain, the *Communication Modeling Language* (CML) (Clarke, Hristidis, Wang, Prabakar, & Deng, 2006). The execution engine, the *Communication Virtual Machine* (CVM) (Deng et al., 2008), which is used to directly interpret CML models in a distributed environment, is also introduced. Although the communication services domain is the focus of this chapter, we are currently applying the approach to other domains, such as microgrid energy management (Allison, Allen, Yang, & Clarke, 2011). That is, we are in the process of developing the *Microgrid Modeling Language (MGridML)* and the *Microgrid Virtual Machine (MGridVM)* using the approach describe here. By applying our approach to multiple domains, we hope to eventually show how the approach can be generalized as a new way for defining semantics for i-DSMLs that can be directly executed by a virtual machine.

The main objective of this chapter is to describe the dynamic semantics to synthesis the i-DSML CML. CML model instances are used to synthesize control scripts (commands passed to the middleware of the execution engine – virtual machine) based on the state of the running system. Key aspects of the synthesis process are the identification of model changes and the interpretation of these changes at runtime. Using these model changes and the current state of the system, events are generated that trigger script generation tasks. The dynamic semantics of CML use labeled transition systems (LTSs) to describe how generated events are handled in the synthesis process. Details of the synthesis process in the CVM are provided, including details on how CML models are compared at runtime and how the model changes are used to execute negotiation and media transfer tasks, the two key aspects in user-defined communication. We will show how the LTS semantics for CML supports the synthesis process.

The chapter is organized as follows. The next section presents a literature review on DSMLs and the semantics of DSMLs. The following section motivates the need for i-DSMLs to support the use of models at runtime, introduces the abstract syntax for CML, the execution engine for CML, and defines a CML semantics that supports the use of CML model instances in the synthesis of control scripts. Some of the benefits, limitations and challenges of i-DSMLs are also presented in this section. Finally, we present future research directions and conclude the chapter.