Chapter XV
Compiling Business Process Models into Executable Code

Cesare Pautasso
University of Lugano, Switzerland

ABSTRACT

Model-driven architecture (MDA), design and transformation techniques can be applied with success to the domain of business process modeling (BPM) with the goal of making the vision of business-driven development a reality. This chapter is centered on the idea of compiling business process models for executing them, and how this idea has been driving the design of the JOpera for Eclipse workflow management tool. JOpera presents users with a simple, graph-based process modeling language with a visual representation of both control and data-flow aspects. As an intermediate representation, the graphs are converted into Event-Condition-Action rules, which are further compiled into Java bytecode for efficient execution. These transformations of process models are performed by the JOpera process compiler in a completely transparent way, where the generated executable artefacts are kept hidden from users at all times (i.e., even for debugging process executions, which is done by augmenting the original, high level notation). The author evaluates his approach by discussing how using a compiler has opened up the several possibilities for performing optimization on the generated code and also simplified the design the corresponding workflow engine architecture.

INTRODUCTION

The goal of this chapter is to present how model transformation and refinement techniques can be applied to produce executable code out of business process models. The chapter shows how model-driven architecture (MDA) techniques have been applied with success to the domain of business process modeling. More in detail, once a business process has been modeled using some language, there are two main alternatives to be considered in order to run the process model

using a workflow execution engine (Figure 1). The first involves the direct interpretation of the model, the second the compilation of the model into a lower-level representation amenable to more efficient execution.

As an example case study, the chapter shows how the idea of compiling business process models has been driving the design of the JOpera for Eclipse workflow management tool. JOpera presents users with a simple, graph-based process modeling language with a visual representation of both control and data-flow aspects. As an intermediate representation, the graphs are converted into Event-Condition-Action rules, which are further compiled into Java bytecode for execution.

These transformations have been fully implemented in the JOpera process compiler in a completely transparent way, where the generated Java executable artifacts are kept hidden from users at all times (i.e., even for debugging process executions, which is done using the original, high level notation). We evaluate our approach by discussing how using a compiler has opened up the several possibilities for performing optimization on the generated code and also simplified the design and positively impacted the quality of the corresponding workflow engine architecture.

This chapter introduces with an example a hierarchy of business process meta-models, leading from abstract, high level and graphical representations suitable for human consumption, down to lower-level languages geared towards efficient execution by a machine. Whereas for didactical purposes (and space limitations) the example presented in this chapter is focused on representations for modeling control-flow aspects, JOpera follows a similar approach with respect to the data flow and the resource perspective of the workflow models. We define relationships and transformations between the representations, in order to support the automatic refinement, optimization and compilation of models in one direction. We also present the abstraction operations required in the reverse direction in order to provide support for "source-level" monitoring and interactive debugging of the execution of business process models.

The rest of this chapter is structured as follows. A motivation for introducing process com-

Figure 1. Interpreted (left) vs. compiled (right) process execution