Case Study: Domain-Specific Code Generators for EMF

The previous chapters mainly focused on the (self-)application of Genesys in the context of jABC. Although this aspect has been investigated most intensively, Genesys is not limited to the construction of code generators for jABC. Accordingly, the case study presented in this chapter is supposed to illustrate, among others, the feasibility of Genesys for other source languages and platforms (Requirement G1 - Platform Independence).

For this purpose, the case study is concerned with the Eclipse Modeling Framework (EMF) [Ste+09], which is part of the Eclipse Modeling Project (EMP) [Gro09], and which allows modeling based on its metamodel Ecore. With respect to their basic structure, models specified by Ecore are very similar to UML class diagrams.

EMP contains a large number of projects, frameworks and tools that deal with generating code from Ecore models and their instances. For example, for an arbitrary Ecore model, EMF itself is able to generate an Eclipse plugin which provides a tree-based graphical editor for corresponding model instances. The Graphical Modeling Framework (GMF) [Ecl11a] even generates editors for graphical notations. The template language Xpand [Ecl11f] enables the development of code generators for model instances. The specification of corresponding Xpand templates is supported by special editors in Eclipse, that provide, e.g., features such as code completion or static checking on the basis of a given Ecore model. Further examples of tools that are based on EMF are Acceleo, AndroMDA, MOFScript and Xtext (cf. Sect. 2.3.3 for the first three and Sect. 2.3.5 for the latter).

Although there are plenty of code generation solutions around EMF, none of them (to the author’s knowledge) allows the construction of code generators like Genesys, in a model-driven and service-oriented way. Consequently, the case study in this chapter presents an approach for integrating EMF and Genesys. The central objective of this approach is to utilize the domain knowledge specified in a metamodel as a basis for generating domain-specific SIBs. Such generated SIBs can in turn be used in Genesys in order to create code generators for any models that conform to (i.e., are instances of) the
given metamodel. By this means, a generator developer is able to resort to the specific concepts and terminology of the source language when constructing code generators, thus meeting Requirement S1 - Domain-Specificity. Fig. 7.1 depicts this approach.

![Diagram](image)

**Fig. 7.1.** Approach for constructing code generators for EMF with Genesys

Initially, a metamodel\(^1\) is created with Ecore. This metamodel establishes relevant concepts and notions for corresponding models of the desired domain (cf. Sect. 2.2). The central goal is enabling generator developers to apply Genesys for constructing code generators which support any models conforming to the metamodel. Sect. 4.1.1 mentioned that the development of a code generator in Genesys requires a corresponding SIB bundle for processing the desired source language. In the case of jABC, the “Graph Model SIBs” are used for processing input SLGs. Accordingly, in order to support another source language, another specifically dedicated SIB bundle has to be employed in place of the “Graph Model SIBs”.

As visible in Fig. 7.1, a special code generator, the EMF SIB Generator, is used to automatically generate such a SIB bundle based on the given metamodel. The EMF SIB Generator itself was also built and generated with Genesys. The resulting SIBs are able to process any models that conform to the metamodel, and thus can be considered on a par with a domain-specific “model API” for those models. Subsequently, the generated SIBs serve as a basis for constructing further Genesys code generators, that translate any instances of the metamodel into a desired target language (bottom part of Fig. 7.1). This way, a generator developer profits from the advantages of Genesys without having to relinquish EMF’s strengths in domain-specificity.

---

\(^1\) For the sake of simplicity, this chapter uses the notion “metamodel” synonymously with the term “domain model”.