Chapter 2

Introduction to Model-Driven Development with Cinco

This chapter lays down the fundamentals of the model-driven development (MDD) using the Cinco SCCE Meta Tooling Framework, as well as the steps necessary to get up and running with the framework. The term MDD refers to a development paradigm where the functionalities of the software are first specified as models, from which then executable code can be automatically generated. To create and use models to represent the software system, a domain-specific language (DSL) that is close to the problem domain, is needed. Application domain experts possess the knowledge of the application structure, which they represent in form of models. Domain experts and application programmers have to agree on how the specification language determines syntax and semantic of the model elements.

2.1 Core Principles of Model-Driven Development

The core principles of model-driven development (MDD) reside in the fact that software development is accelerated by providing a simple, but efficient abstraction of the software structure as a model. Those model abstraction, representation of real-world objects, are transposed trough a series of model-to-model or model-to-code transformations [10]. Our work is to utilize the DSL provide by the Cinco framework to design our graphical DSL, which in turn will permit the generation of a functioning Selenium Java application.

Opposed to the common development method, applying a graphical model to layout the different user sequences allows even non-programmer (here the domain expert with

much more expertise on how to design a great software documentation) to accomplish the task of documenting the features offered by the web application. Nonetheless, the programmer has the tasks — in collaboration with the domain expert — to specify the meaning of each model element for the code generation process.

When applied correctly, the result of the model-driven development process is a tailored application to domain. This reflects one of the main advantages of MDD, the accuracy of targeting directly the specific problem. Besides, it is still possible to change the DSL so that it adapts to the new challenges emerging during the development process. This can be iterated until the specification reaches preciseness wanted to solve the problem.

2.2 Domain Specific Language

A domain-specific language (DSL), as the name suggests, is a language adapted to specific development domain. [7] gives a succinct analogy to DSLs by saying that it is comparable to a tool specially crafted only for one specific task as opposed to general programming languages that can be seen as tools for multiple different tasks. If we stick to this analogy, just as one would start with a blueprint to construction a mechanical tool, designing as DSL required similar steps. One have to conceptually lay out the behavior and eventually — in case it is a graphical language we want to design — the look of each element that can be used in our DSL.

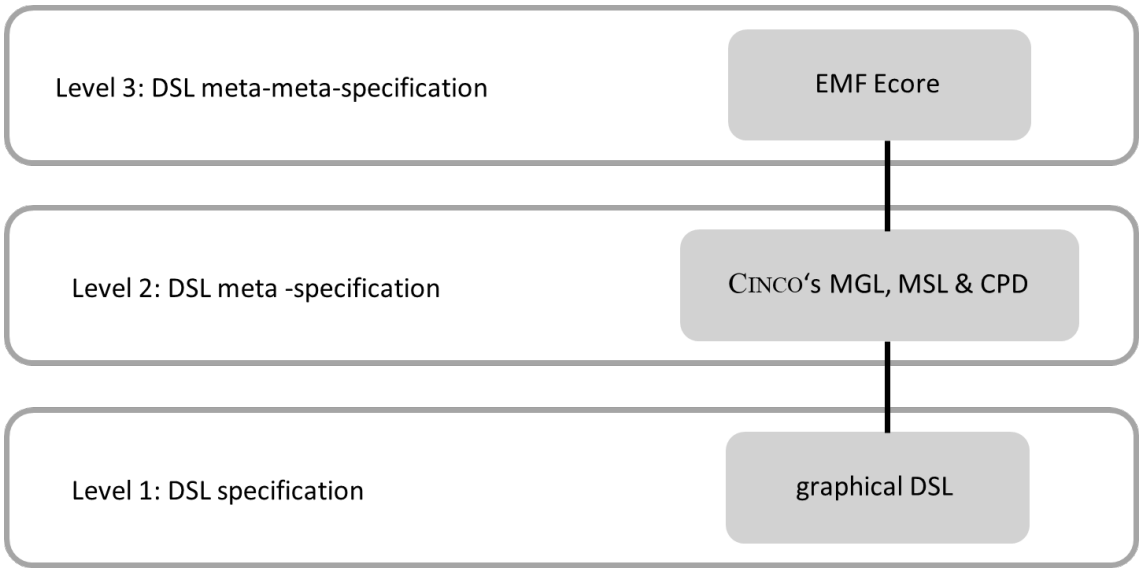

Figure 2.1: Hierarchy of our graphical DSL specification

Blueprinting our DSL is equivalent to defining a meta-language or meta-DSL to our language. *Meta* literally means *situated behind or beyond*. So the meta-language is the descriptive language coming before our language, describing the meaning (semantic) and the relationship between the objects of our target language. In other words, the meta-DSL is the abstract syntax and the resulting DSL concrete syntax. It is possible to ascend the modelling hierarchy of meta definition until we reach a self-referencing language, like it is the case Unified Modeling Language (UML).

In our case, we want a graphical language for creating models that represent the different parts of a web application and how the end-user can possibly interact with them. The meta-language to our graphical DSL is provided by the Cinco SCCE Meta Tooling framework. It comprises the Meta Graph Language (MGL), the Meta Style Language (MSL) and the Cinco Product Definition (CPD) as depicted in fig. 2.1, all has been constructed using Xtext, a language Workbench for writing textual DSLs [6]. Coming chapters will explain those concepts in detail.

2.3 Cinco SCCE Meta Tooling Framework

The Cinco Framework is a generator-driven development environment for domain-specific graphical modelling tools [1]. It is actually developed by the chair of programming systems at the Technical University of Dortmund. One of the great features of this framework is that it allows us to generate an entire editor application with just one click from a simple textual specification language — the Meta Graph Language (MGL) mentioned in the previous section.

The MGL together with the MSL form the meta-model from which Cinco generates a ready-to-run modeling tools called Cinco product. The MSL is where the Cinco developer defines the look of every node and edge element, as well as the font and color of the text to be displayed in the graphical model [6]. The created meta-model is based on Ecore, the meta-modeling language of the Eclipse Modeling Framework (EMF) and the Graphiti framework is used to generate the corresponding graphical model editor. Additionally, you find the right button to trigger code generation in the created editor. Chapter 4 is devoted to the Cinco product (the editor application), in particular section 4.3 gives an in-depth explanation of the generation process.

2.4 End-User Documentation

Our main goal is to produce an entire end-user documentation page on a graphical model. For that we have to first determine what the essential parts of a good documentation are. Before going further, we ought to precise that the type of documentation to be created will depend on how the documentation designer lays out the model, that means the model could be designed to represent a technical documentation, requiring deep knowledge of the application documented or it could also be structured to produce a documentation for simple end-user with no technical knowledge at all of the underlying web application.

Bearing that in mind, a good documentation should respond three important questions: *what* is the purpose of the application, meaning the problem the application intends to solve, *what* is the end-user supposed to do to quickly and efficiently get to the solution and *how* is it achieved [4]. The last question is better answered by providing visual help in form of screen captures or any illustration that fastens the understanding of the documentation.

Our focus is primarily on the documentation of web application, hence documenting the User Interface (UI), which is composed of web elements (like buttons, navigation links, checkboxes, etc.) the end-user can interact with. This settles implicitly the type of end-user documentation we aim to create, a step-by-step guide for navigating UI to reach the expected result.