Abstract

The principle of software generation has been getting more attention in recent years and has become an important paradigm in software engineering. Generative software development simply put means generating code using modelling languages such as UML or domain-specific languages (DSL‘s) as source input, which contributes to saving time, effort and money.
This automatic generated code is then used for all sorts of things depending on the intended purpose of the source modelling language and the target infrastructure. The task of creating this code is handled by a Generator which consists of a transformation engine (model source to code) and a runtime environment. It can in some cases even add a high amount of functionality to the target that was not “modeled” in the source.
Over time, software engineers have noticed recurring problems that can be analyzed so that we are able to conclude a general design pattern that helps along in the long run. The design pattern can accelerate the development process and provide proven development paradigms, which helps save time without having to reinvent patterns every time a problem arises whether it contributes to the reusability in the generator, or adaptability in the target code after it’s been generated.
In this thesis, we will explore generative software development in more depth as well as its patterns. We will mainly examine some of the challenges developers face in different stages and the approaches these developers take to handle them. In the process, we will consider two perspectives: reusability and adaptability