Derek Ochal

DePaCoG Plugin White Paper

## Introduction

The program uses Typesafe Configuration Library to manage configuration files and Logback and SLFL4J for logging. Three packages represent the three basic elements of the software. Five design patterns were used during the development: abstract factory, façade, chain of responsibility, builder, and factory method.

## Abstract Factory

The main part of the program - the DePaCoG itself – is designed according to Abstract Factory pattern. This pattern is very versatile and can produce many similar products while leaving room for expansion. The products are objects containing the generated code. DePaCoG does not have a user interface. It makes debugging easier by separating front end from back end. Because it needs a way to communicate with other classes, it produces a report with a list of patterns it supports.

## Façade

DePaCoG’s user interface is an implementation of a Façade design pattern. This design allows for interchangeability with future GUI, API, etc. This façade operates only one object (the factory) but could be expanded.

## Chain of Responsibility

DePaCoG uses the Chain of Responsibility design pattern to filter and modify requests for its product. Inner abstract class Handler is implemented as three receiver objects. This design decouples error checking feature for easier debugging. It also enables modifications to requests that can be introduced and taken out as needed. Handler is an inner private class because this feature is not required, and future factories may not have it. In current release, error checking and first letter capitalization is unimplemented.

## Builder

Because the resulting code is a complex object, its creation is delegated by DePaCoG to one of its builders. A LocalBuilder first analyses the request, assigns variable values, and creates file references. It then uses Typesafe Configuration Library to fill files with code. Lastly, it assembles and returns the product. This solution allows for many different builders, each with unique generation algorithm. Future builders can use remote repositories for source code instead of local files. They can also send files to a remote machine instead of creating them locally.

## Factory Method

Most of the LocalBuilders are similar but their code generation algorithm is unique. Therefore, they use Factory Method design pattern. This design makes most of the LocalBuilder’s code reusable and leaves room during the unique phase of the generation. It also separates the most complex part of the program to be implemented in final stages of development.