High-Level-Design-Document.md

High Level Design Document

1. Project Overview

The aim of this project is to improve a developer's experience using Aspect Oriented Programming (AOP). AOP allows developers to modularize cross-cutting concerns into aspects, which can be reused across multiple codebases. In this project, we provide several pre-built aspects, including caching, measuring execution time, logging, parallelization, and measuring CPU and memory usage that can be accessed via command line. We also provide a Command-Line Interface (CLI) that takes the path to the jar file so that these aspects can be added to the developer's code at runtime. Byte Code Weaving is used to achieve this.

2. User Requirements

The target audience for this project is developers who are familiar with AOP and are looking for an easier way to add aspects to their codebase. The project should provide the following features:

- Pre-built aspects that can be accessed via command line
- Command-Line Interface (CLI) to add aspects to codebase at runtime
- Documentation on how to use the pre-built aspects and CLI

3. Technical Requirements

The project should meet the following technical requirements:

- Use AspectJ as the AOP framework
- Use Gradle as the build tool
- Provide a jar file that can be added as a dependency to the developer's project
- Use Byte Code Weaving to add aspects to the developer's code at runtime
- Use PicoCLI for the Command-Line Interface (CLI)

4. Design Decisions

The following design decisions were made during the project:

- AspectJ was chosen as the AOP framework due to its popularity and ease of use.
- Gradle was chosen as the build tool because it is widely used in the Java community and has good integration with AspectJ.
- Pre-built aspects were provided to make it easier for developers to add cross-cutting concerns to their codebase.
- The CLI was implemented using PicoCLI to make it easier for developers to add aspects to their codebase at runtime.
- Byte Code Weaving was used to add aspects to the developer's code at runtime because it is a more flexible and powerful approach compared to source code weaving.

5. Design Overview

5.1 Main Features

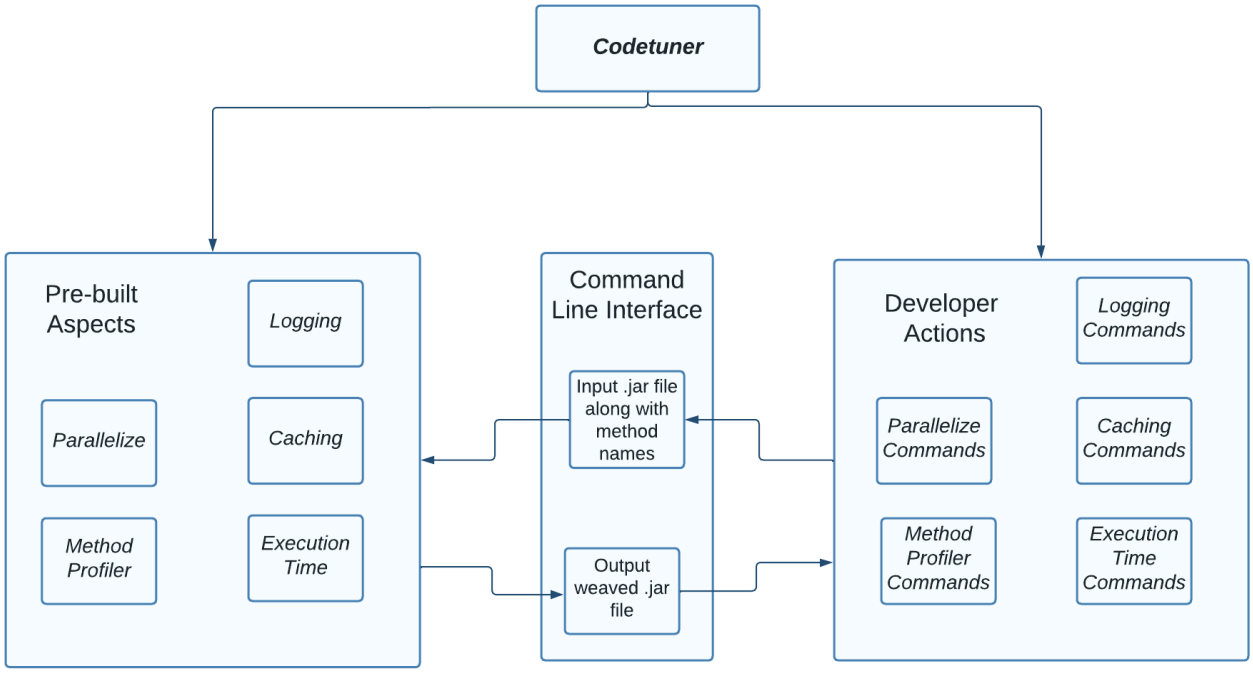
The main design features of the project are as follows:

1. Pre-built aspects that can be accessed via command line written in Aspect J
2. Command-Line Interface (CLI) to add aspects to codebase at runtime
3. Easy integration with existing projects
4. Documentation on how to use the pre-built aspects and CLI

In order to make the design easy to understand, the design has been illustrated diagrammatically in the following sections.

5.2 Application Architecture

The application architecture is shown in the following diagram:



Explanation of the diagram:

- The developer's codebase is compiled into a jar file.
- The developer then enters the path to the jar file into the CLI along with the method names in which they wish to implement the aspects.
- The CLI then accesses the pre-built aspects written in AspectJ to weave the aspects into the developer's codebase at runtime.
- The CLI then outputs the weaved jar file which can be used by the developer.

5.3 Technology Architecture

Aspect J

AspectJ is an aspect-oriented programming (AOP) framework that extends the Java programming language with additional features for modularizing crosscutting concerns, such as logging, caching, and security. It provides a range of powerful and flexible mechanisms for defining and applying aspects to Java code, including compile-time and runtime weaving, pointcuts, and advice. AspectJ is widely used in industry and has a large and active community.

PicoCLI

PicoCLI is a Java library for creating command-line interfaces (CLIs) that are easy to use and intuitive for developers. It uses annotations to define commands and options, and provides strong typing for arguments. PicoCLI is highly customizable, integrates well with Gradle and other build tools, and has good documentation and community support.

5.4 Standards

- Inputs - The CLI takes the path to the jar file and the method names as inputs.
- Outputs - The CLI outputs the weaved jar file.
- Quality - The application has a very minimalistic design and is easy to use. The application is also well documented.

5.5 User Interface

The user interface is implemented in the form of a CLI using PicoCLI. The interface allows the developer to access the following pre-built aspects:

- Caching
- Measuring Execution Time
- Logging
- Parallelization
- Method Profiling

Commands are entered in the following format:

```
codetuner [-hV] [COMMAND]
-h, --help      Show this help message and exit.
-V, --version   Print version information and exit.
```

Commands:

```
time         Log execution time of methods
profiler     Gives CPU and Memory of methods
parallelize  Parallelize the methods with annotation
logging      extensively logs program execution
caching      Cache methods with annotation
```

More details on the commands can be found in the playbook.

5.6 Tactics Employed

5.6.1 Performance

The performance of the application is improved by using Byte Code Weaving to add aspects to the developer's codebase at runtime. This allows the developer to add aspects to their codebase without having to recompile their codebase. The entire application rests locally on the developer's machine, which also improves performance.

5.6.2 Security

The application is secure because it does not access any external resources. The application also does not store any data on the developer's machine.

5.6.3 Usability

The application is easy to use because it has a very minimalistic design. The application is also well documented. Once the developer understands how to use the application, they can easily add aspects to their codebase at runtime.

5.6.4 Reliability

Any errors in the input format in the CLI are handled gracefully by the application. The application also provides a help command that can be used to understand how to use the application.

5.6.5 Maintainability

The application is well modularised into the CLI and the pre-built aspects. This makes it easy to add new aspects to the application. The application is also well documented, which makes it easy to understand the codebase and make any modifications in the future.

5.6.6 Reusability

The application is reusable because it can be used to add aspects to any codebase that uses AOP. As a note of caution, the developer must ensure that the aspects are applicable to the methods in their codebase. For example, not all methods can be parallelized.

5.7 Resource Utilization

The resource consumption of using AspectJ depends on various factors, such as the number and complexity of aspects, the size and complexity of the codebase being woven, and the environment in which the code is executed. In general, adding aspects to a codebase using AspectJ will increase the memory consumption, CPU utilization, and disk I/O of the application at runtime.

The weaving process itself requires additional computation, as AspectJ has to analyze the codebase, identify join points, and weave the advice into the appropriate locations. This can slow down the application startup time and increase the memory usage. However, the exact impact will depend on the size and complexity of the codebase and the aspects being added.