Final Project Draft

Sabbir Ahmed

November 10, 2021

1 Introduction

In software engineering, design patterns are general, reusable solutions to commonly occurring problems [1]. It is generally considered good practice to integrate design patterns into software products, especially large projects, since it allows the developers to focus their time and attention towards specific implementations. The purpose of this draft is to introduce an open-source project and present analysis on the software patterns within the implementation.

2 Structural Simulation Toolkit (SST)

The software that is being focused on in the final project is Structural Simulation Toolkit (SST). It is a simulation framework that prioritizes high performance computing (HPC) models [3]. Since SST is a large scale project with many stable extensions implemented for its kernel, the scope of the project will be limited to specific sections of the core repository. The repository is hosted on GitHub [2].

3 Software Patterns Present in SST

The following patterns can be observed to have been already implemented in the project:

1. Abstract factory pattern
2. Factory method pattern
3. Singleton pattern
4. Strategy pattern

Other patterns are present in the project, such as C++ idioms (Include Guard Macro).

3.1 Abstract Factory/Factory Method

The abstract factory and factory method patterns are present in the `SST::Factory` class. In the repository, the class can be located at `factory.h`. In the repository, it is used to create several concrete classes, including `Component` and `Module` objects. The class also provides templated variadic methods to create concrete classes of generic classes, such as

```
/*
 * General function to create a given base class.
 *
 * @param type
 * @param params
 * @param args Constructor arguments
 */
template<class Base, class ... CtorArgs>
Base* Create(const std::string& type, CtorArgs&& ... args)
```

3.2 Singleton

The singleton pattern is present in the `SST::Factory` class. In the repository, the class can be located at `factory.h`. The class is used to instantiate other concrete simulation classes. SST requires simulation objects to be synchronized throughout the kernel, especially since they can be running on a distributed system where race conditions can become major issues. The software forces these simulation objects to be singletons.

3.3 Strategy pattern

The strategy pattern is present in the `SST::Core::Serialization::serializer` class. The class is implemented throughout multiple files in `serialization`, where it is overloaded in the files with various parameter types, with all the various versions of the class simply overloading the function call operator (`operator()`).

References

[1] Design patterns. `https://sourcemaking.com/design_patterns`.

[2] sstsimulator/sst-core. `https://github.com/sstsimulator/sst-core`.

[3] The structural simulation toolkit. `http://sst-simulator.org/`.