Brahma Project

Luke Kennedy and Andrew Hopkins

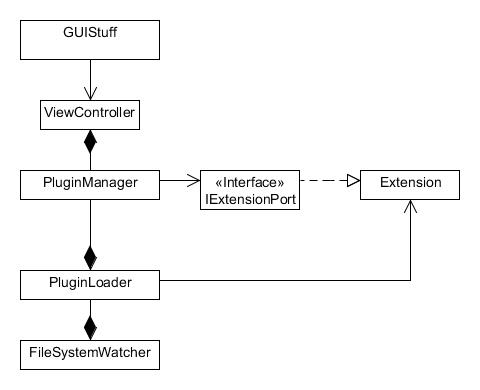
# Design Diagrams

## Model Decomposition Diagram

## S:\CSSE 477\Brahma Project\Diagrams\Module Decomposition PNG.png

## S:\CSSE 477\Brahma Project\Diagrams\Dependency Diagram PNG.pngDependency Diagram

## Class Diagram



## Interaction Diagrams of Important Components

# S:\CSSE 477\Brahma Project\Diagrams\Interaction Diagram PNG.png

# Test Specifications

Our testing strategy for this project will be to utilized unit testing to verify the correctness and accuracy of the individual components and integration testing, using mocks, to show that our modules work together correctly.

# Code Repository

The code for this project can be found on Github at <https://github.com/LukeKennedy/freezing-hipster>.

# Quality Attributers

Extendibility: Our application allows users to create or find any plugin that supports our plugin architecture.

Modifiability: This application uses a set of cohesive classes that allow a clear divide of what code is located where. Because of this, it is simple to modify the source and add to it. Our application takes advantage of separating logic and user interfaces, so parts can be modified without worrying about affecting others in an extremely detrimental way.

Usability: The application uses commonly understood user interface paradigms that will allow users to easily how to interact with the application. The application provides feedback for every user action to show the user that an event is occurring.

Testability: Mock plugins may be injected into our plugin core to test the proper functionality of the host application. The PluginTimer can be tested using a mock plugin and JPanel. The Controller class has view objects passed to it to allow unit testing without the need for a proper interface on top of it.

# Creating Plugins

To create a plugin for this Brahma application, start by creating a class that extends the abstract plugin.Plugin class. Be sure to implement all methods provided by the abstract class. In the start(), pause(), and resume() methods, call the superclass method to insure proper functionality.

After creating your Plugin extension, create a manifest file that includes a “Plugin-Class” property with a value of the Java class of your extension. For example, if your Plugin class is “com.example.foo.java”, your manifest would include the entry “Plugin-Class: com.example.foo”.