System design document for falcons (SDD)

Contents

Version: 1.0
Date: 2011-04-13
Authors: Marika Hansson, Gustav Mörtberg, Juliuz Printz, Jack Pettersson

1 Introduction

This section gives a brief overview of the project.

1.1 Design goals

The program will have a slim design, since this is not the main focus or most important part of the system. The GUI that will be implemented, mainly on the client side, will have usability and simplicity as main focus. Future goals with the design is to, as much as possible, create possibilities to make phone applications and maybe also write extensions in other languages than Java.

1.2 Denitions, acronyms and abbreviations

- GUI, graphical user interface.
- Java, platform independent programming language.
- Client, the device the user is communication with the server on
- MVC, a way to partition an application with a model (containing data), view (renders the GUI) and controller (manages communication).

1.3 References

MVC, see http://en.wikipedia.org/wiki/Model-view-controller.

2 System overview

The purpose of the system is to create a communication framework to enable the user to easily manage information and tasks between different units in the network (e.g. servers and desktop computers).

The system is built in an entirely modular fashion, split into three seperate libraries which

handles different parts of the system. We have a library that defines a client and handles all of the operations related to the client, the server library works in a similar fashion. Theres also an library which provides all the extension points for plugins and plugin creation.

The code is written using an modified version of MVC, with an anemic model. This means that we have a conventional controller and view but the model is seperated into data and logic. We have chosen to go this route because it provides us with a safer and more modular implementation.

3 System architecture

3.1 Software decomposition

3.1.1 General

- Falcons Client
    - falcons.client - Implementation of the client library.
    - falcons.client.view - The view associated with the client implementation.
- Falcons Server
    - falcons.server - Implementation of the server library.
    - falcons.server.view - The view associated with the server implementation.
- Falcons Library
    - falcons.client.controller - Contains all controllers that communicate between client view and client model
    - falcons.client.model - Manages the client logic and data
    - falcons.client.network - Manages all the communication between the client and the server (client-side).
    - falcons.pluginmanager - Manages the loading of all plug-ins, both client and server-side.
    - falcons.server.controller - Contains all controllers that communicate between server view and server model
    - falcons.server.model - Manages the server logic and stores its data.
    - falcons.server.network - Manages all the communication between the client and the server (server-side).
- Falcons Pluginlibrary
    - falcons.plugin - The abstraction of plug-ins.

3.1.2 Communication

The communication works by sending objects, through sockets, between the client and the server. The objects that are sent are called PluginCall, these objects contain the call's final destination, the information about the plugin that is used and its version along with an object called AbstractPluginData. These contains all the information that the plugin itself needs to do what it should.

3.1.3 Decomposition into subsystems

Our system consists of three subsystems. These are the “client-system”, the “server-system” and the “plugin-system”.

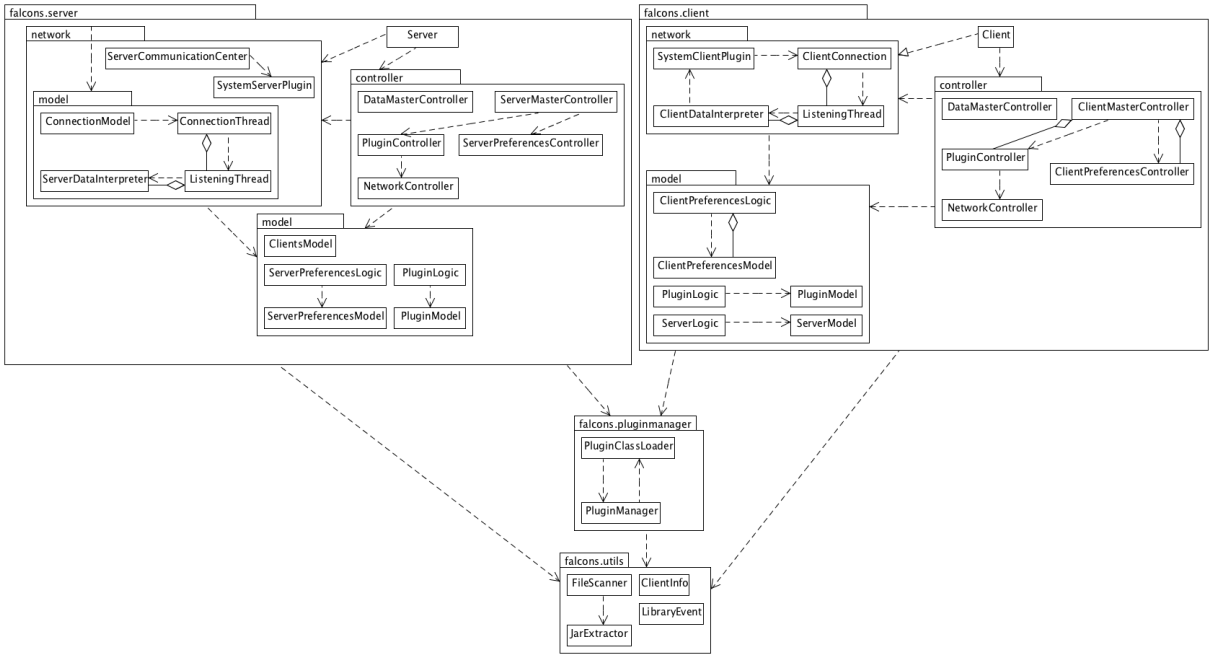
The client-system handles everything that the client would need to do. For example sending and recieveing PluginCalls and managing all relevant data.

The server-system handles everything that the server would need to do. For example sending and recieveing PluginCalls and managing all relevant data.

The plugin-system contains all the abstractions of plugins; abstractions of a plugin and a basic class for creating PluginDatas (that can be extended if more complex objects are to be sent).

3.1.4 Dependency analysis

As one can see in the figure below, the extended MVC design pattern is followed quite well in the main library and it has very few circular dependencies. The view is not visible here though, as it is a separate project since it should be easy to use the library to write your own front-end.



3.2 Concurrency issues

The threads we have don’t interact with the same objects, just the other way around (i.e. the same object interact with many different threads).

3.3 Persistent data management

The data is saved in the server’s and the client’s models, respectively. Preferences between sessions is saved in XML-files.

3.4 Access control and security

Plugins can’t access the rest of the application at all, except for a few methods given in the plugin library such as getting some information about the connected clients.

4 Detailed System Design

This section provides a more detailed discussion about the components introduced in the System Architecture section.

4.1 The client

4.1.1 Definition

This is an implementation that is the part that the user communicates with the server through. It runs as a system tray icon that shows GUI with settings-view (to connect) and main-view (to use plugins) when clicked.

4.1.2 Responsibilities

Simply works as the GUI and implementation for the user to use and communicate with the system. The client doesn’t manage much code or methods, this is provided by the libraries, works as a black-box between the user and the library.

4.1.3 Constraints

The client doesn't have constrains in it’s own but since it communicates with library that contains all the implementation that actually does something, it simply has the same limitations as the library.

4.1.4 Uses/interactions

The library executes and counts everything that the clients need to be done so it has a very strong interactions with the library.

4.1.5 Processing

Since the client can be seen more like an interface to the library it doesn’t do much calculation and executing. It sends all this work to the library.

4.2 The server

4.2.1 Definition

The entity of the library that handles the central communication between all clients and their respective plugins. It runs in the systemtray.

4.2.2 Responsibilities

The server is responsible for sending the right PluginCalls to the right places and executing some PluginCalls in itself. Also it handles the storage of all clients and their plugins.

4.2.3 Constraints

The server can't initiate any connections on it's own. It won't perfom anything on it's own. It needs either a client to tell it what to do or a user implementation that tells it what to do.

4.2.4 Uses/interactions

Uses all of the server-library to execute the commands it receives. It has references to all of the server-library, directly or indirectly.

4.2.5 Processing

The server receives commands in the form of LibraryEvents that tells it what needs to be done. It delegates all the neccessary commands to the correct parts of the server-library.

4.3 Server/client library

4.3.1 Definition

This is the actual framework that the server and client implements and uses. It handles all the communication and all the storage of clients and plugins.

4.3.2 Responsibilities

Handles all the loading of plugins. Handles communication between server and clients. Handles all of the PluginCalls, and makes sure that they are sent to the correct location.

4.3.3 Constraints

The library can handle everything we think it needs.

4.3.4 Uses/interactions

The library only uses the components within itself, a framework called SimpleXML aswell as the pluginlibrary.

4.3.5 Processing

See above and code.

4.4 Plugin library

4.4.1 Definition

The Library that defines the abstractions for plugins and the data they require. It also defins all of the Events that can occur inside a library that's relevent for the client or server

4.4.2 Responsibilities

It is responsible for all the pluginabstractions and it's interactions with the server or client. It defines all the types of events that is relevant to the client or server aswell.

4.4.3 Constraints

One of the constraints that the Plugin Library has is that the implementation requires certain things from every java file, which makes it a hassle to include external libraries in plugin creation.

4.4.4 Uses/interactions

This library doesn't use any external libraries. It is self-dependant

4.4.5 Processing

The pluginlibrary isn't responsible for any processing. It only provides a framework that enable a plugin to be loaded into the system.

4.5 Plugins

4.5.1 Definition

This is the flexible part the uses the framework so you can code a plugin that does almost anything you want it to.

4.5.2 Responsibilities

This is very free since every plugin has it's own tasks to manage but there are some things that a plugin have to implement and provide to the framework.
Every plugin has to return a JPanel where the plugin later will be used from in the client.
Other responsibilities is the mandatory implementation of the pluggable interface in all plugin-class files, this is so that the classloader can unpack the .jar file containing the plugin.

4.5.3 Constraints

The main idea for the framework is to make it language independent so you could code your plugin in any language and not necessarily Java, but as the implementation is today it has some connections to Java that makes this hard.

4.5.4 Uses/interactions

Every plugin uses the framework (of course) which provides the plugin with all necessary implementations for eg sending pluginData between client and server which is critical for the plugins to have a meaning.

4.5.5 Processing

This part will not be covered in this SDD since every plugin has it’s own executions and workflows.