Inter-Tubes Synergistic-User Coding System

Project proposal by Michael Levine

1. Operational Concepts

Currently, there are a plethora of collaborative text editors available on the internet. There are also a large number of programming environments, source control systems, and related doodads. The idea is to merge all of these into one.

I propose an application that runs on your local machine for creating, editing, and controlling source code, all under the auspices of collaboration. It would implement syntax highlighting for supported languages (and the ability to add more languages), multiple users per file, locks, bug tracking, permissions, and cross-platform connections.

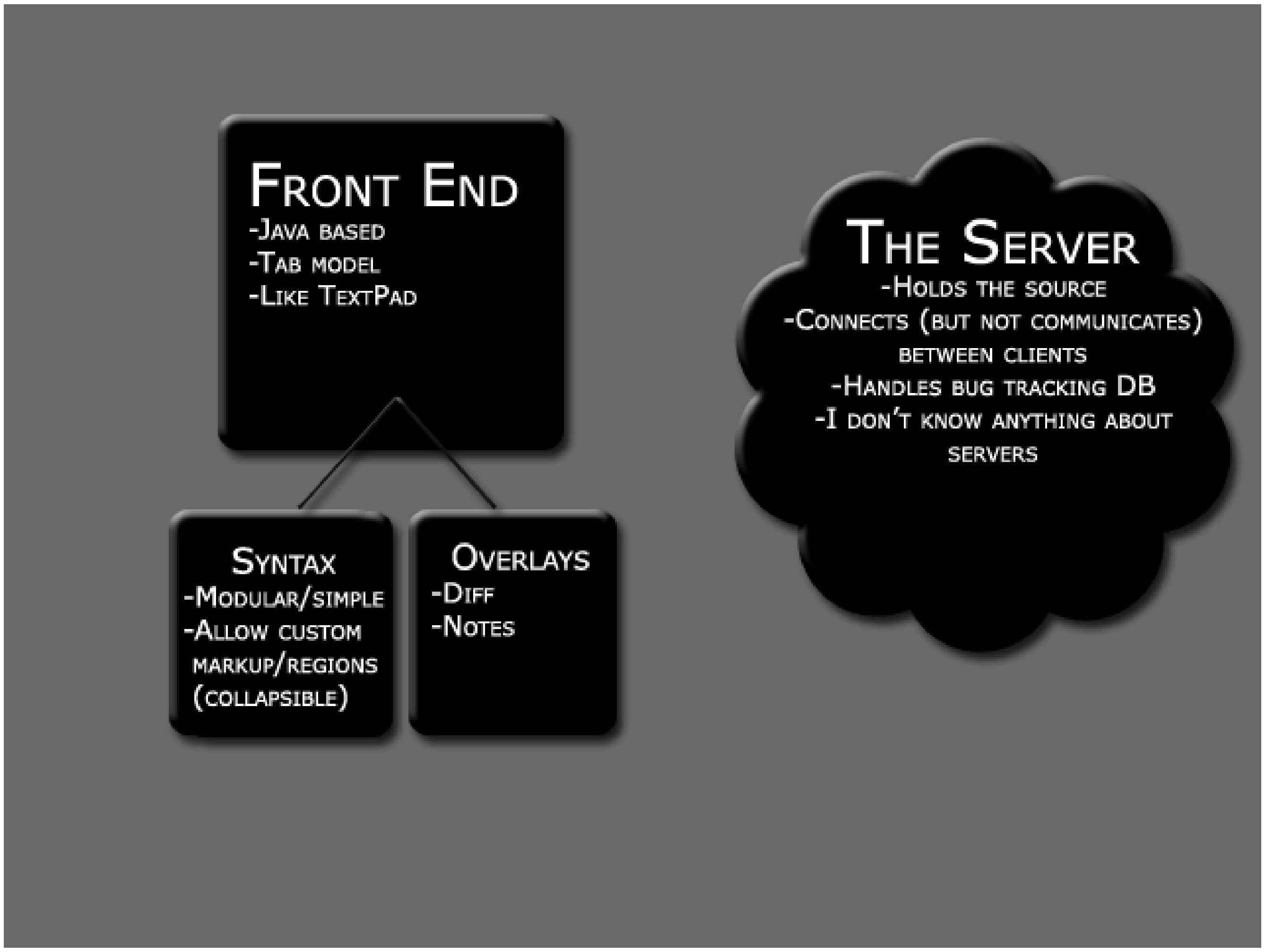
Previously, most source control systems were, single-mindedly, focused on merging changes into a file. Then, a separate program was needed to actually view the differences in any meaningful way. On top of this, a third program would be in charge of tracking why the changes were made and what they fixed. To top it all off, until the coder is ready to merge, no one can see the changes that are in store.

This application would circumvent most of these issues by unifying everything into one interface and one application. Admittedly, in the time frame, there is almost no way to get anything approaching Eclipse, Visual Studio, or even TextMate level editing out of the project. It would focus more on the integration of bug tracking, source control, diffing, and collaboration. It might even be so far removed from the code that people find it better to code in their usual environment, and then load the file in our solution to handle the rest. That would be perfectly acceptable.

2. System Requirements

The system would require an internet connection, any supported OS (or possibly even include a web-based minimal version, time allowing), and nothing special of the CPU. The database machine would have to be... well, I don't know anything about databases or servers. But I don't think any kind of major load would be put on it.

3. System and Software Architecture



The ideal goal would be to have a multi-platform piece of software. That pushes the code to something like Java or Python. The only real multi-platform GUI choices I know of are Swing and GTK; Swing is much easier to deal with and only requires the JVM to be installed.

I know next to nothing about the database. It would probably use some variant of SQL because a lot of people use that. It would only allow clients to connect to one another and handle locks/permissions/source. Actual communication between two clients would not run through the server.

The main idea to keep everything simple but expandable is to allow easy additions and modifications. Features like user-definable regions (like #region in Visual Studio) and linking would keep reading the code manageable. There could also be support for completely custom markup that the system keeps track of, but is not actually part of the source – like sticky notes attached to lines of code.

Because the source control and bug tracking are built directly into the system, calling up a diff overlay or viewing multiple versions of the file with accompanying notes would be just a button click away.

4. Lifecycle Plan

The full scope of the project would not be implementable in 8 weeks. A more realistic timetable would be to have basic interface and architecture decisions hammered out within the first week. By week 3, there should be a simple text editor and a server that supports basic source control (even if it does it by running an existing source control system). Week 5 should introduce, on the editor side, syntax highlighting and custom user markup. For the server, it should be able to have a rudimentary system in place for storing user markup along with the files, in addition to making sure the source control works. Week 6 would see the editor interfacing fully into the source control of the server – the ability to load/save/merge that brings along user markup and displays it in a sensible fashion. At week 7, there should be a system in place that allows any number of users watch (and chat) an editing session – bonus points if multiple people can edit in one session. Week 8 (the final week) would see everything getting wrapped up, with a hopefully functioning editor that allows markup, overlays of diff and notes, check in/out, and multiple users on one file.

People wise, it should take at least 6 competent coders, at least 2 of which have previous database experience. The database/server guys would work on their stuff and the other coders would work on the client-side application throughout the project, although the database guys would probably do a fair share of application coding as well.

5. Feasibility Rationale

Java already has a text editor built in, so the hard part is going to be all of the work that goes along with it. Basic things, like syntax highlighting (of just keywords) and some "smart" tabbing (based on braces and user "hints") are not very large amounts of work. The back end source control would primarily consist of a directory structure, the source files, and the diffs. Worst case (although probably preferred) would be to use SVN or CVS as part of the implementation, in addition to scripts that run on top of it to handle user mark up and notes. That seems *fairly* reasonable to get done, although having to implement out own source control could prove to be too much work.

The rest of the text editor features other than actual collaboration all feel like one person, one week of work. Once there is a basic architecture in for modifying how the code looks on screen, adding overlays and notes can build off of that and be reasonable. The collaboration I honestly have no clue on, and could prove to be a big risk – it may not be feasible in the time given, nor may it even be a feature the users want.

The server-side could also be a big risk, as I don't have any knowledge to base my assumptions in.