Kellen Donohue (kellend@{cs,uw})
Zach Stein (steinz@{cs,uw})

VERB Life-Cycle Objectives Document (LCO)

Vision

VERB - View, Edit, and Run in Browser - is a lightweight development environment that can be accessed from any device with a web browser. We want to make it quick and easy to code a short program or make a quick change to an existing program and see the results. Users can view and edit their code using a JavaScript-enhanced text editor. VERB will compile and run code files, accepting command line arguments, and return the contents of stdout and stderr. Programs can also be "servicized" - VERB can generate a URL that runs an instance of a program when called, effectively turning any command line program into a web service (think AJAX).

VERB also provides users with an opportunity to experiment with new languages with no program installation or download required. VERB is a language agnostic tool. Priority languages to support include C/C++, Java, Ruby, Python, PHP, Perl, and a functional language. Support for long-running processes (such as web servers) and GUI based apps could be added. Interpreter access for interpreted languages would also be a very useful extension (similar to http://tryhaskell.org/). VERB can be extended to include a variety of collaboration tools, including version control integration, file sharing, and collaborative editing.

Architecture

Server:

VERB could either be deployed on private servers or in the cloud. VERB as a cloud service would be useful to a wide audience under a Software as a Service model. Having users run VERB on their own servers would eliminate security concerns and allow users to easily add their own modules (additional language support, custom run commands, etc.). The VERB server (or cluster) runs a database (probably mySQL) for storing metadata and a web stack (Ruby on Rails, LAMP, etc.) serving as a front-end to VERB's Run framework.

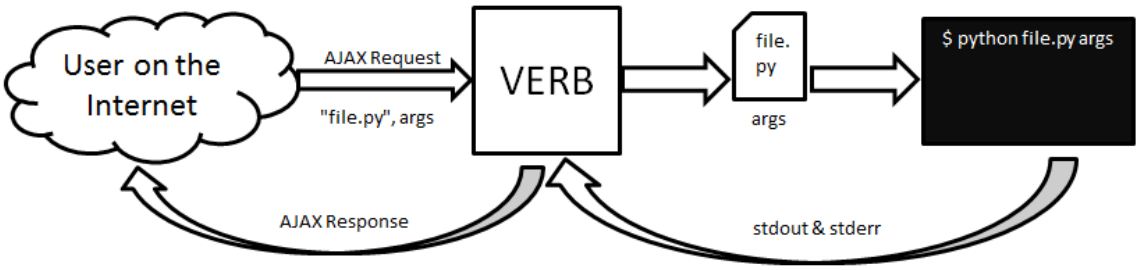
Client:

VERB takes advantage of an open source JavaScript text editor to provide users with the features they're used to, such as syntax highlighting and auto-indentation. ACE (http://ace.ajax.org/) is a good candidate for this text editor.

Web servicizer:

The web servicizer is a framework allowing developers to execute code, that they either create in VERB or upload, as if the code was a web-service, without requiring any code changes. One way to utilize the web servicizer is to add an AJAX request in a page's JavaScript code to VERB's servicized program, passing an identification token, the name of the file they want to run, and any program arguments. VERB will then execute the program with the given argument, and return the command line output of the program as the AJAX response. The web servicizer model would also be used to run code edited directly in VERB, except with the requests coming from the VERB front-end instead of from users' websites. This

diagram demonstrates the flow:



An example use case is the word guessing game jotto. Suppose you already implemented the game logic in Ruby (this was a UW CSE 341 assignment). With the web servicizer you could easily make a web interface for this game. The Ruby code would handle all the game logic, the web servicizer would communicate between your website and the Ruby code, and only the user interaction code would need to be written in JavaScript.

Alternatives

There are a few fully-featured cloud IDE's. Among them are Cloud9 (http://cloud9ide.com/) and CodeRun studio (http://www.coderun.com/ide/). VERB focuses on providing a more lightweight experience. VERB doesn't offer extensive debugging support, for instance, but VERB can be run in a smart-phone browser. Also, Cloud9 and CodeRun are more focused on web developers, while VERB supports more traditional languages such as C/C++.

WebSSH clients (such as Anyterm) also provide some similar functionality, but limit users to in-terminal applications and carry security risks. However, it would be possible to incorporate an SSH client into VERB if desired.

The web servicizer is a tool for web service developers that supports existing server-side command-line programs and scripts. The alternative to using the web servicizer is using sockets (or something similar) in the existing server-side code. Using the web servicizer is advantageous to developers desiring to use existing code in any language and developers who want to take advantage of domain-specific languages, tools, or libraries.

Risks/Challenges

If VERB is offered as a cloud service a primary concern will be ensuring the integrity of the server and security of user files. We will need some method for protecting the server from malicious code. One potential solution is utilizing the .NET framework. Many languages have CLI implementations (http://en.wikipedia.org/wiki/List_of_CLI_languages). By utilizing these implementations, we would have the added protection of the code being run in a virtual machine.

We will need a process for allocating server disk space to users' code and a framework for executing user's code and returning the results. The server will also have to handle the concurrency issues of running multiple users' code.