CS 427 BookReaders Jenkins IRC Bot Improvements

December 9, 2015

Contents

1 Introduction

Internet Relay Chat (IRC) is a ubiquitous tool among open-source developers. Unfortunately, the existing IRC interface to Jenkins is not very useful for anything beyond trivial interaction; checking on jobs, determining the recent build status, and determining if Jenkins is "on". The Jenkins IRC interface through the Jenkins plugin ecosystem only provides a limited set of tools and commands for interacting with an instance of Jenkins. We sought to extend the usability of the underlying Jenkins bot; providing developers with a powerful Jenkins interaction environment. Specifically our goals were to add the following improvements:

1. add user customized colors to Jenkins messages;
2. provide a mechanism to query Jenkins for files and URLs;
3. provide a query engine to receive a filtered set of build messages;
4. provide a syntax to direct Jenkins to chat with other, specific IRC users;
5. provide general usability improvements.

This document provides an overview necessary to understand the basic operation, design, and context of the CS427 BookReaders Team plugin improvements for Jenkins.

The general intent of our team effort was to substantially improve upon the functionality of the IRC Jenkins interface environment. It is within the IRC context that all of our work was conducted; however, our efforts span multiple existing plugins. The existing IRC plugin is dependent upon the Instant Messaging (IM) plugin, which provides a bot mechanism to communicate between Jenkins and various chat services (e.g. IRC, Twitter, etc.). Many of our improvements added new commands to the bot, which improves the overall IM-based plugin ecosystem in Jenkins. We only made modifications to the IRC plugin that were specific to the IRC interface environment. Taken together, our changes to both plugins—outlined here—will improve the user experience for IRC users seeking to interact with Jenkins, and provide a broader repertoire of tools for dependent plugins of the IM plugin as well.

1.1 Existing Architecture

Our efforts began by investigating available plugins to leverage for this effort. We chose the IRC plugin[1] as a starting point because it provided an IRC client. Once the IRC plugin is installed, its IRC connection configuration is setup using the main Configuration

[1] https://wiki.jenkins-ci.org/display/JENKINS/IRC+Plugin

page within Jenkins. The Jenkins instance is assigned an IRC nickname, password, and channel to communicate on. For our project we used the freenode IRC server network to host our channel: `##427BookReaders`. Preconfigured Post-build Actions setup within the configuration of each Jenkins project can send IRC messages automatically when a build is complete. These messages will indicate to all users of the IRC channel what the final build status of a job is.

The IRC plugin acts as a thin wrapper client constantly listening on the IRC channel for incoming messages denoted by a prefix (usually !jenkins) that is configured in the main configuration page of Jenkins. The IRC plugin will see all messages on the channel, but only those with the appropriate Jenkins prefix will be communicated to the IM bot. Once the message is determined as being relevant to Jenkins it communicates the message to an instance of the IM plugin's `Bot` class. If !jenkins is succeeded by a command, say, `!jenkins botsnack foo`, then the Jenkins processes the botsnack command with the argument "foo" and returns the result. Jenkins is able to communicate publicly or privately with users of a specific IRC channel. The basic communication pattern exhibited by the existing architecture is diagrammed in Figure 2. The IRC channel also supports terminal colors so Jenkins is able to return messages over IRC that are color coded for the user. The existing plugin highlights "SUCCESS" in the color green and "FAILURE" in the color red for various job statuses.

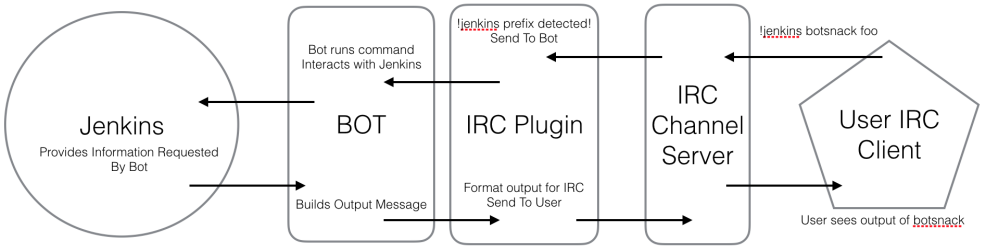

Figure 1: The basic communication model of the existing architecture.

The underlying IM plugin[2] is extremely powerful and is the engine upon which the IRC plugin is able to operate. The `Bot` class within the IM plugin comes with a series of commands that are available to the user. A listing of these commands is given by passing `!jenkins help` over the IRC channel. Commands can be listed with multiple command names. For example, "si" is a shorthand name for another command, "showIf", which is also listed. The commands of the `Bot` class are defined through a series of Abstract Java classes that are implemented by various Command classes. These abstract classes govern the way the command class implementations interact with Jenkins. For example, the `SnackCommand` class that handles the "botsnack" command implements the `AbstractTextSendingCommand`. Ultimately the bot's commands must be based upon the `BotCommand` class, which the abstract classes extend. `BotCommand` has a series of methods that govern how command names are made known

[2]https://wiki.jenkins-ci.org/display/JENKINS/Instant+Messaging+Plugin

`getCommandNames()` and provide a mechanism to execute the command through the `executeCommand(...)` method. The abstract command classes each provide slightly different methods for returning messages to the user. However, it is commonly done through an overriden `getReply(...)` method.

While the existing architecture does provide some functionality for interacting with Jenkins, our product builds upon the available infrastructure to provide substantial improvements in functionality and user customization. The remaining sections of this document will overview the general modifications that were made to each plugin and how to use the added functionality.

2 Instant Messaging Plugin

2.1 Functionality

The IM plugin provides a substantial portion of the functionality of our product. Specifically, we added a series of commands and made them available to the `Bot` class to mediate interaction between a user and Jenkins.

2.2 Improvements

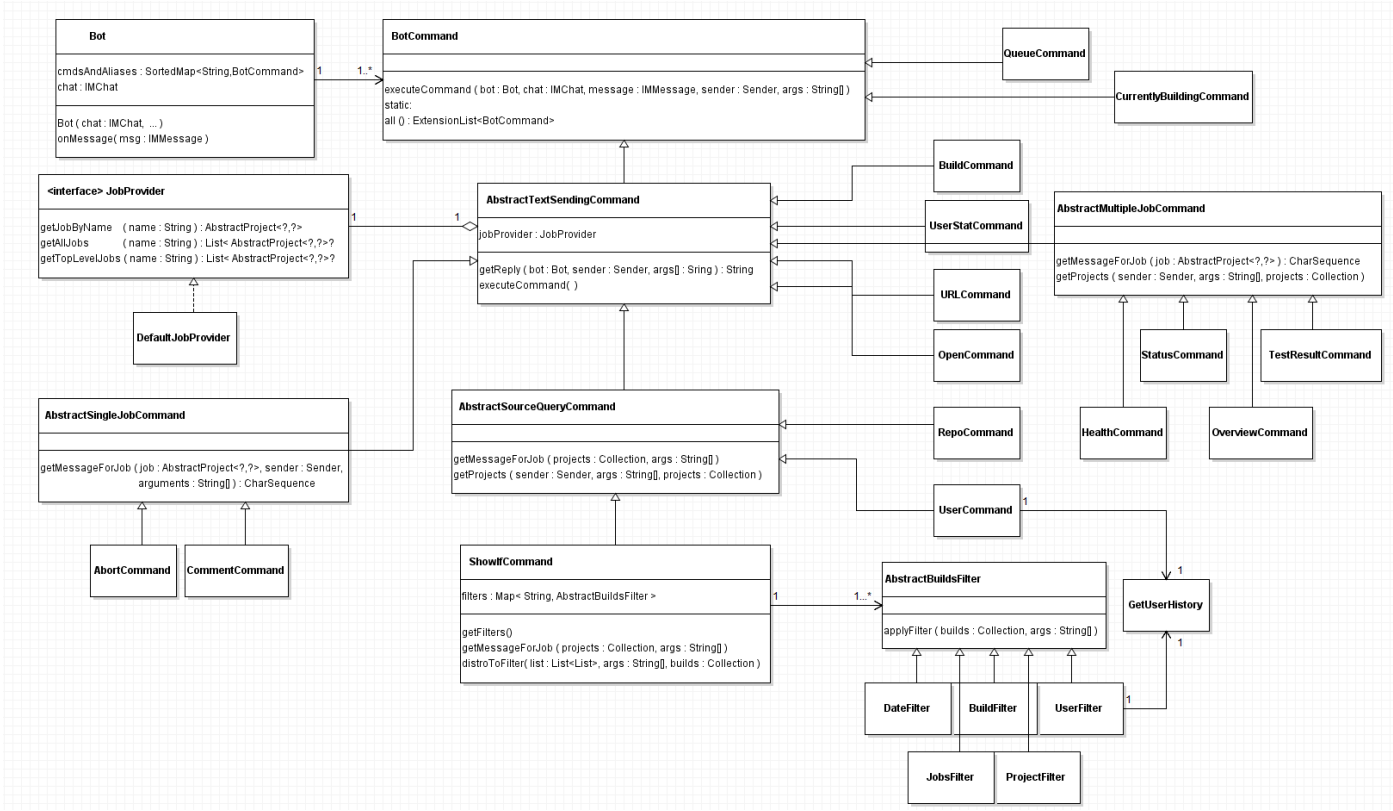

Figure 2: The class diagram for bot package after modification

2.2.1 Show If Command

The Show If Command adds a query language syntax to the Jenkins bot system. The Show If Command is interacted with through the IRC channel by typing `!jenkins showIf` followed by a series of queries delimited by the pipe (|) character. A Show If Command query can take the form of:

- `user username` This filters all of the build jobs to return only started by a specific user.

- `date (<,=,>) YYYY-MM-DD-HH-mm` This filters all of the build jobs to return the statuses of only those built before, on, or after a date based on the boolean expression.

- `project projectname` This filters all of the build jobs to return the statuses of only those builds of a given project, specified by a project's name.

- `job (<,=,>) number` This filters the build jobs to return the statuses of only those builds in relation to their job numbers.

- `build number` This query acts as a governor returning a given `number` of build results.

When the queries are appended using the pipe character as a delimiter, the returned output is the logical intersection of all of the results. So `!jenkins showIf user abcuser | date < 2015-12-15-08-00` will return all of the build jobs made by abcuser before December 15, 2015 at 8:00AM. However, if a `build` query is not specified it will append a `build 10` query to ensure that no channel flooding occurs. If a user uses a `build` *N* query, it will use the value of *N*, preempting the flooding limiter.

The basic operation of the Show If Command was built by first extending `AbstractTextSendingCommand` to create the `AbstractSourceQueryCommand` class, which has a `getProjects()` method that returns all of the Jenkins projects. This method is then used by the `ShowIfCommand` class to populate a collection of build jobs from Jenkins using the Iterator design pattern. The Show If Command class parses the user's queries, creates a collection of all of the built jobs, filters the jobs based on the supplied queries, and returns the result in a series of formatted messages to the user.

2.2.2 Overview Command

The **Overview command** allows the user to issue a command to the Jenkins bot and receive an overview of information about projects, such as build number, current status, url, last commit users and messages, etc. The **Overview command** takes the form:

```
!jenkins overview [<project>] , or
!jenkins o [<project>]
```

- `!jenkins overview`: returns the overview status and information of all the projects within the Jenkins instance.

- `!jenkins o projectname`: returns the status and information associated with a

the specific projectname.

2.2.3 Open Command

The **Open command** implements the `"ls"`-like functionality from a Unix shell, which enables the user to get the hierarchical file structure of each specific project and traverse the work-space. This command is typically formatted as:
`!jenkins open <Project Name> <Path to Directory/File>`

- `!jenkins open`: this basic command will return a hint of the complete command with a list of available projects within the Jenkins instance.
- `!jenkins open BookReaders`: returns specific information related to the current project `BookReaders`, this command renders the content in the root directory of the BookReaders project, which is [ircbot-plugin-master], `[]` indicates the directory name while file name doesn't have the bracket.
- `!jenkins open BookReaders /ircbot-plugin-master`: returns all the directories and files under `/ircbot-plugin-master` (i.e. `IRCBot-command.md IRC_help.txt [instant-messaging-plugin-master] [ircbot-plugin-master] [iteration_1] [trunk] jenkins-url.txt vmid.txt`).
- `!jenkins open BookReaders /ircbot-plugin-master/IRCBot-command.md`: if the last argument is a file, the IRCBot will return a url link directly to the file that user can click and access, (i.e. *https://fa15-cs427-032.cs.illinois.edu:8083/job/BookReaders/ws/ircbot-plugin-master/IRCBot-command.md*).

Further, the**Open command** supports incomplete argument names for `file / directory` and has a *"find"*-like feature within the user specified scope

- `!jenkins open BookReaders /ircbot-plugin-master/instant-messaging-plugin-master/trunk/po`, finds the files name starting with "po" (which only matches "pom.xml" in our case) and returns the url: *https://fa15-cs427-032.cs.illinois.edu:8083/job/BookReaders/ws/plugin-master/instant-messaging-plugin-master/trunk/pom.xml*
- `!jenkins open mp1_maven /src/Sequence`, finds all the files whose names start with "Sequence" under the directory or sub-directory of `/src` within the project `mp1_maven`. (i.e. *https://fa15-cs427-032.cs.illinois.edu:8083/job/mp1_maven/ws/src/main/java/ https://fa15-cs427-032.cs.illinois.edu:8083/job/mp1_maven/ws/src/test/java/pkg/SequenceUtilT ttps://fa15-cs427-032.cs.illinois.edu:8083/job/mp1_maven/ws/src/main/java/pkg/SequenceUtil.*

  Output files and directories are displayed in the alphabetic order.

2.2.4 Repo Command

The **Repo command** enables users to issue Jenkins a command and retrieve the repository information of builds where the code is under version control. The return information also identifies which files have been changed and the commit comment of the corresponding build(s), which enables developers to communicate effectively in some situations; such as, merging commits from different branches. The **Repo command** takes the form:

`!jenkins repo [show <# of jobs to show> | reponumber <revision number to show> | all ]`

- `!jenkins repo show`, returns a default number of the most recent build information, filenames that have been changed, and the commit comment.
- `!jenkins repo show 1`, returns the most recent build info, filenames that have been changed, and the commit comment.
- `!jenkins repo reponumber 20268`, returns specific build information from revision number 20268.
- `!jenkins all`, this directive renders the entire list of build information history. Be cautious to use this directive, especially when the project is large with a lot of build history.

2.2.5 Build Command

The **Build command** allows users to schedule a build request on Jenkins from a chat room directly. Users can control a specific project or all the projects on Jenkins to execute a build with the additional option to build immediately or schedule to build with a user-specified time-delay. The **Build command** takes the form:

`!jenkins build <job> [now | <delay> [s|m|h]] [<parameterkey> = <value>]` or `!jenkins build -all`, for generic build of all the projects on Jenkins

- `!jenkins build BookReaders`, schedules to build the job BookReaders on Jenkins in the default time of 5 seconds, the returned message from Bot will notify the user once the job build has started on Jenkins, as well as the *build status* when the build finishes. Below is an example of a typical return message:
  ```
  taofeng1:  job mp1_maven build scheduled with a quiet period of 5 seconds
  netID-zsong12:  Starting build #41 for job mp1_maven (previous build:
  SUCCESS)
  netID-zsong12:  Project mp1_maven build #41: SUCCESS in 27 sec:
  ```

`https://fa15-cs427- 082.cs.illinois.edu:8083/job/mp1_maven/41/`

- `!jenkins build BookReaders now` will start the build immediately, with the similar process as above.

- `!jenkins build BookReaders 1h` , will schedule the job BookReaders to build on Jenkins after 1 hour.

- `!jenkins build -all`, a generic build of all the projects on Jenkins, use caution when using this command because it builds every project.

2.2.6 User History Command

The **User History command** enables users to query the `build` history from *Jenkins* through the IRC chatting room directly. This feature adds many conveniences for developers to communicate their build history/log through the chat room directly. The **User History command** takes the form:

`!jenkins userHistory <user name> <(Optional) # of builds to show>`,

then it returns a list of Jenkins instance URLs of the build history. The number builds to show is defaulted to 2 if last argument is not specified.

- `!jenkins userHistory username1 3`, this returns the information from the 3 most recent builds associate with username1. For example:

  BookReader-instant-messanger #355 (17 hr ago): SUCCESS: https://fa15-cs427-032.cs.illinois.edu:8083/job/BookReader-instant-messanger/355/

  BookReader-instant-messanger #338 (2 days 22 hr ago): SUCCESS: https://fa15-cs427-032.cs.illinois.edu:8083/job/BookReader-instant-messanger/338/

  BookReader-instant-messanger #337 (2 days 22 hr ago): SUCCESS: https://fa15-cs427-032.cs.illinois.edu:8083/job/BookReader-instant-messanger/337/

- `!jenkins userHistory username1 Default 6`, this directive resets the default number of builds to show from 2 to 6 for a given username. This command renders the 6 most recent builds of username1 directly, and a subsequent query of `!jenkins userHistory username1`. It will always return a build history of 6 by default until the default is changed again.

2.2.7 Get URL Command

The **Get URL command** can help the user to query or redirect to specific project(s) or sub-directories of project URLs directly from the chat room. The **Get URL command** takes the form:

`!jenkins geturl [specific keyword to the URLs]`, and returns the specific URL associated with the keyword.

- `!jenkins geturl help`, returns a list of available directives: `users, root, plugins, conf, security, script, log, plugin, stat, statistic, scripts, node, nodes, sec, base, user, configure`

- `!jenkins geturl configure`, returns the URL of the sub-directory for the configuration page of the main project:

  jnknsbt: https://fa15-cs427-032.cs.illinois.edu:8083/configure/

3 IRC Plugin

3.1 Functionality

The IRC plugin is responsible for communication between the bot established by the instant messaging plugin and the IRC channel. The functionality extends strictly to message delivery. At the start of our project the text colorizing was very limited in functionality.

3.2 Improvements

3.2.1 Notification Scoping

This addition was deemed a necessity by the team when the channel began to get flooded with showIf queries with a high build filter. The user will be able to have the jenkins responses sent directly to them or a group of people. The command must be structured as follows:

- `!jenkins <command>` A normal command dispatched to the jenkins bot. The branch to the computation and parsing for the only command is not taken.
- `!jenkins <command> only <user>,<user>,...` A public command to the channel where the jenkins bot resides. This will produce private responses to the users listed.
- `/msg <jenkinsbot> !jenkins <command>` A private command to jenkins which will result in the jenkins bot privately messaging the user who sent the command. The branch and parsing does not occur.
- `/msg <jenkinsbot> !jenkins <command> only <user>,<user>,...` A private command to jenkins which dispatches private messages to all users listed. The message after the command does not reach the bot.

The user who dispatches the command will be automatically be associated with the group when using the only command. This, like the custom message style, was implemented by detecting the command and restructuring from within the PircListener class. From there the private message is dispatched as normal to the bot and sent back to the users. The instant messaging plugin sees no difference between the messages presented.

3.2.2 Custom Message Style & Themes

Initially, the custom messaging style was done but was limited in functionality to colorizing the messages that the bot would send to the channel. The underlying structure to the program was such that the message information was piped to a colorize function that changed the colors to a pre-set theme.
PircListener was altered to detect a command `set color` from the user message and dispatch to the IRCColorizer class which colored a pattern that the user provided. For instance, if the user input `!jenkins set color foo BLUE` then every time the jenkins bot prints the pattern "foo" it will be in blue.
Themes were also added to the IRCColorizer class in the form of a HashMap for easy switching between major themes. This makes it much easier for users to change commonly colored words such as "FAILURE" and "UNSTABLE" all at once. This also had the side effect of cleaning up quite a bit of code by getting rid of the switch statements for changes in other parts of the code.