Chat Plugin Design Document

Timothy Madigan and Joseph Ciurej
University of Illinois: Urbana-Champaign (CS427)

December 12, 2013

Contents

1 A Brief Overview of the Chat Plugin

The purpose of this project is to create an extension to the continuous integration server Jenkins that facilitates chat communication between users of the server. The main feature of this extension is an augmentation to the standard Jenkins user web pages that allows for messages to be transferred between users connected to one or more of these pages through a dynamically updating web interface.

1.1 Motivation for Chat Plugin

The primary purpose of this plugin is to enable quick and easy communication between Jenkins users through the provision of an integrated chat client as an augmentation to the Jenkins web interface. While other chat clients exist in abundance, the Jenkins chat plugin allows for easy integration into a familiar environment with minimal installation difficulties.

1.2 Goals of Chat Plugin

The main goal of this project is to provide an intuitive, feature-rich chat client within the Jenkins web interface. By integrating a chat client into the Jenkins interface, we aim to remove the need for extraneous chat applications, thus providing a more seamless workflow. Additionally, eliminating the need for extra applications minimizes dependencies on additional exterior applications, decreasing both maintenance and integration costs for users.

In an attempt to enhance user satisfaction, we built the application with a variety of features, including:

- Peer-to-peer communication between users via a familiar chat interface.
- Group chat support to allow users to communicate with multiple team members simultaneously.
- Identification system that allows for user-specified names by both system administrators (via an admin file) and chat clients (via a user interface in the chat client).
- Persistence of chat logs to allow for users and management to cross-reference past communications.

Our hope is that these features will serve to facilitate efficient communication between users and provide near-complete coverage of desired functionality.

2 Chat Plugin Architecture and Design

The overall architecture of the Jenkins chat plugin follows a traditional client-server model where clients to the chat plugin send messages to a back-end server and this back-end server forwards incoming messages based on provided recipient information. The implementation for the server component of the model (and the implementation for supporting back-end functionality) is programmed in Java while the client component is programmed with a combinaion of Javascript and HTML. These two components communicate using websocket technology (provided by the Netty-SocketIO library), which allows for seamless and asynchronous client-server interaction. The plugin extends Jenkins to closely integrate the start-up and shut-down processes of the server component with the Jenkins server and to provide server access to clients through the Jenkins web interface.

2.1 Primary Plugin Library Components

The core of the Jenkins chat plugin consists of the types used to implement the server and client components of the client-server model. For each of these primary components, the listing below describes the major functionality encapsulated by that primary component and provides a visual aid for its role in the project through a UML diagram.

2.1.1 The `ChatServer` Component

The primary type that implements the server component in the chat plugin. This type encapsulates the behavior of the server component in the client-server model for the chat plugin, transporting received client messages across the network to other designated clients.

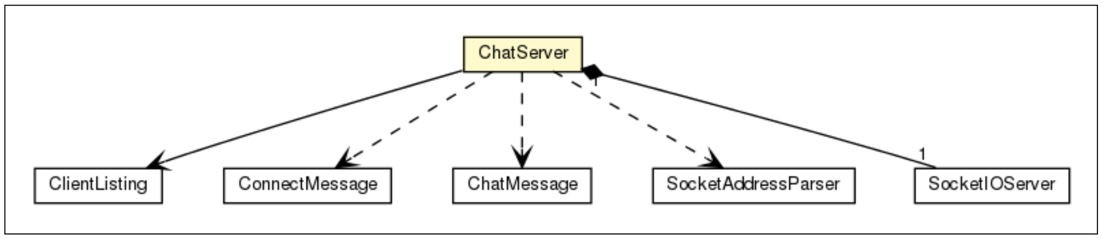

Figure 1: UML Diagram for the ChatServer Class

The primary functions of this type are client connection/disconnection broadcast updates (which are performed to inform all connected clients when a particular client connects or disconnects), client message forwarding (performed whenever a message is receieved from a client with a destination address), and back-end to front-end message forwarding.

2.1.2 The `ClientListing` Component

A supplemental type that implements a mapping from addresses of chat clients to the stored names of these clients. This type is used primarily by the `ChatServer` type in order to retrieve information about a particular client given the remote address of this client, which aids the server in associating unique names with arbitrary source/destination clients. This type is also used to persist user name data between chat sessions and allow this name information to be specified through a simple format (i.e. the CSV file format).

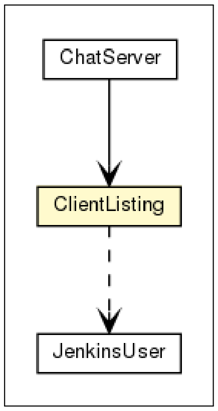

Figure 2: UML Diagram for the ClientListing Class

Essentially, the `ClientListing` type is a two-way hashtable that associates network addresses with user names and supports data export to and data import from CSV files.

2.1.3 The `UserGroup` Component

Another supplemental type to the `ChatServer` type that represents a hierarchical group ordering structure. This type is used to group chat clients within uniquely-named sets of communication groups based on user-specified group structuring. This grouping and ordering information is utilized by the `ChatServer` type to determine the recipients of a message given an identifier for a group of users. Additionally, the `GroupDatabase` type provides support to store the group structuring information contained in a `UserGroup` instance into a database for persistent storage (with similar functionality available for loading this information).

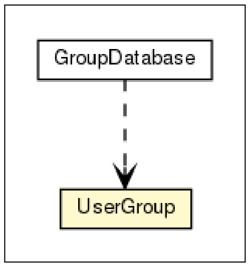

Figure 3: UML Diagram for the UserGroup Class

A single instance of the `UserGroup` type represents the subroot of a group hierarchy tree structure. Each node within this structure represents a distinct group where the proper descendents of any node represent disjoint, strict sub-groups of the group represented by the node itself.

2.1.4 The `ChatHistory` Component

A type that facilitates the storage of chat messages within a database with a scheme that supports efficient querying based on recipient group and author. The primary functionality of this type is storing given chat messages into a database and querying these messages based on author or group.

Figure 4: UML Diagram for the ChatHistory Class

While not fully integrated at the point of initial release, this type is intended to be integrated with the `ChatServer` type to support the persistence of chat logs between chat clients using the Jenkins chat plugin.

2.1.5 The `ChatClient` Component

The non-explicit, unencapsulated conglomerate of code that comprises the client portion of the client-server model. This portion of the implementation is responsible for displaying connected user information based on data sent from the `ChatServer`, properly populating text areas based on messages passed from the `ChatServer`, and sending messages to the `ChatServer` based on user input.

This code is injected into every Jenkins web view to facilitate client communication to the back-end chat server from any point in the Jenkins web interface. This injection is performed by way of a Jenkins `PageDecorator` extension, which is described in the next section.

For additional information about the client component of the client- server model, see the **plugin/src/main/webapp** subdirectory of the project source (relative to the base directory of the plugin).

2.2 Jenkins Plugin Extension Points

The initial release of the Jenkins chat plugin utilizes three Jenkins extension points in order to enable the plugin's basic chat functionality. The following list describes our plugin's extensions to Jenkins and the service that these extensions provide:

- `ChatClientPlugin` extends the `Plugin` extension point of Jenkins in order to define start-up and shut-down functionality for the chat plugin, which allows for a `ChatServer` instance to be created on Jenkins initialization.
- `ChatClientDecorator` extends the `PageDecorator` extension point of Jenkins to facilitate the injection of arbitrary Javascript and HTML into the Jenkins web interface pages. Exercising this feature allows the chat client interface to be displayed on all Jenkins web pages.
- `ChatClientRunListener` extends the `RunListener` extension point to capture build submission and completion information, which the type relays to the running `ChatServer` as a message to be forwarded to all connected chat clients (though it's important to note that this second function isn't implemented for the initial release).

2.3 Example of Chat Plugin Control Flow

In order to aid the reader in understanding the flow of the chat plugin and how exactly it's integrated into Jenkins, this section contains a high-level step-by-step example execution of the plugin. By stepping through this example, the user should be able to get a better idea about control flow within the plugin without needing to open up a debugger and step through the execution manually.

2.3.1 Step-by-Step Plugin Execution

The following is a high-level step-by-step execution of the logic within the chat plugin starting from Jenkins server initialization and ending at Jenkins shut down:

- Upon instantiating an instance of the Jenkins server, the Jenkins core scans through all installed plugins for extensions to the `Plugin` type. As an extension to this type, the `ChatClientPlugin` type will be discovered and its `start` function will be invoked.
  - Within the `start` function, an instance of the `ChatServer` type is instantiated, which causes a server socket to be opened on a port on the local machine that listens for messages and connection requests from clients.
- When a Jenkins user opens up a Jenkins web interface view, the `ChatClientDecorator` extension intercepts the view before it's serviced and injects the chat client interface into the view.

- After receiving the Jenkins web view, the injected web code performs a connection request to the running `ChatServer` instance.
  - Upon receiving a client connection request, the `ChatServer` resolves the name of the remote host by querying its `ClientListing`, sending the client's associated name as a connection response. Additionally, the `ChatServer` will broadcast a message to all connected clients indicating that the new client has connected.
  - On getting a connection message, the chat client web interface updates the view to indicate that the client referenced in the message has connected.
- When a client closes its serviced Jenkins web view, the injected web code sends a disconnect message to the `ChatServer`.
  - Upon receiving a client disconnect message, the `ChatServer` first associates the remote host with a name by queryings its `ClientListing`, then immediately broadcasts a message to all connected clients that indicates that the given client has disconnected.
  - On getting a disconnect message, the chat client web interface updates the view to indicate that the client referenced in the message has disconnected.
- When a client enters a message into a chat box, the chat client interface code sends the entered text and destination client as a packet to the `ChatServer`.
  - When the `ChatServer` receives a client message packet, it determines the host name of the destination client and relays the given packet to this client.
  - Upon receiving a message from the `ChatServer`, the client interface populates the corresponding chat box with the contents of the given message.
- At the time the Jenkins server instance is shut down, it again searches through all installed plugins for `Plugin` extensions, calling the `stop` method for each such extension.
  - During its own `stop` method invocation, the `ChatClientPlugin` type calls the `shutdown` method for the `ChatServer`. This action causes a broadcast to be sent out to all connected clients indicating that the server has been terminated, followed by the closure of the server socket on the local machine.

2.3.2 Important Notes about the Execution Process

Only the most important features of the chat plugin have their corresponding executions detailed in the step-by-step explanation in the previous section. Other more minor features (such as chat client build notifications and user interface functionality) haven't been detailed, but their role in the control flow of the chat plugin can be gleaned by skimming the documentation for the files associated with these features.

Also, important features added after the initial release are not included in the execution diagram in this document, but the control flow explanation in future documentation will be amended to account for this fact.

3 Plans for the Future of the Chat Plugin

During our initial development period, we found that we were not able to fully realize many of the features we wished to incorporate into the Jenkins chat plugin. The following section details the major additions we wish to integrate into the chat plugin in the future as well as future plans we have for the chat plugin project.

3.1 Listing of Additional Chat Plugin Features

There were many features that the team considered during the initial development period that couldn't be fully implemented due to time constraints. Many of these features enhance user experience by providing a more streamlined interface, additional communication features, and superior data persistence.

The following is a listing of all the additional chat plugin features that we hope to implement in the future:

- Complete integration of group chat functionality into the front-end web interface.
- Automatic group recognition for chat clients and automated joining of group chats on page load for the connecting user.
- Adding functionality to the front-end interface to allow for user name and contained group specification for unrecognized clients.
- Transferring user name to client IP address mappings to a database storage scheme on the Jenkins-hosting machine from current local file storage scheme.
- Full integration of the database storage functionality within the plugin to support efficient persistence of chat, user, and group information.
- Notification system that sends build messages to relevant clients whenever a Jenkins build is submitted or completed.
- Extension of the build notification functionality to allow for more granular specification of build information recipients.
- Adding the ability for users to set their status so that other connected clients have the ability to gauge their availability at a glance.
- Automatic updating of user status based on client activity within the Jenkins web interface.
- Redesign of the front-end interface to more closely resemble that of Facebook with a perpetual bottom action bar that contains open chats and connected clients.

3.2 Goals for Releasing the Chat Plugin

Down the road, we hope to publicly release the source for the chat plugin by uploading the project to GitHub. This way, any team member or curious open-source contributor may build upon the project in any way they wish. Additionally, we asprie to eventually submit the chat plugin as an official Jenkins extension, most likely a few weeks after the project becomes open-source.

4 Miscellanous Chat Plugin Information

The following section contains extra information about the chat plugin, such as installation steps and steps to execute various other plugin utilities. This section should be referenced when searching for extra instructions for the installation and maintenance of the chat plugin.

4.1 Steps for Chat Plugin Installation

Installation for the chat plugin has the following dependencies, which must be installed before the plugin itself can be installed:

Java Development Kit (v1.7+)
A version of the standard Java development kit at least as recent as version 1.7 is needed primarily to import extended networking functionality only in later JDK versions.

Jenkins (v1.509.3+)
A version of Jenkins more recent than version 1.509.3 is needed to support the chat server plugin (older versions may work, but they have not been tested).

Netty-SocketIO (v1.5.2+)
An open source Java library that facilitates network communication via websockets in Java. This library is used by the chat plugin to aid in the transferral of mesages to and from chat clients.

After these dependencies are installed, installing the chat plugin itself is relatively straightforward. The steps for performing this installation are as follows:

1. Navigate to the **plugin** subdirectory from the chat plugin source base directory.
2. Run the script within this directory by the name of **export.sh**.

After performing the second step, a sequence of installations should follow. After all these subsequent installations are complete, the plugin should be fully installed and accessible through Jenkins.

4.2 Steps for Chat Plugin Execution

By the nature of the plugin, the majority of the execution occurs when booting up a Jenkins server instance and opening up one of the Jenkins user web pages. That being said, there are no explicit steps for executing the plugin apart from the installation, which is detailed in the previous section.

4.3 Steps for Running Implementation Tests

Running the automated tests for the chat plugin can be performed by following these steps:

1. Navigate to the **lib** subdirectory from the chat plugin source base directory.
2. Perform a Maven installation within this directory. This can be performed on Unix machines by running the command '`mvn install`'.

The manual tests for the chat plugin can be found within the **lib/test** subdirectory of the base plugin directory. These tests can be executed by simply following the instructions contained within each test document.

4.4 Steps for Generating Implementation Documentation

Generating the Javadoc documentation files for the chat plugin can be performed by following these steps:

1. Navigate to the **lib** subdirectory from the chat plugin source base directory.

2. Perform a Maven Javadoc generation within this directory. This can be performed on Unix machines by running the command '`mvn javadoc:javadoc`'.

The Javadoc documentation files will be placed within the subdirectory **target/site/apidocs** relative to the **lib** directory. These files are most easily viewed by opening the file **target/site/apidocs/index.html** within a web browser.