CS2IRC:
Bringing Counter-Strike Statistics
And
Server Control to IRC

Mike O’Laughlen
Jiunwei Chen

Abstract

Purpose
The growth of the Internet has not only brought together information from many varying
sources but communities of online users as well. The community that this project helps
to bring together is a growing population of online gamers, more precisely players of the
game called Counter-Strike.

Services Used
The way in which this project allows users to collaborate more effectively is through the
use of Internet Relay Chat (IRC), a text based chatting system that has been established
for a long period of time on the Internet. Users of the chatting system may join a channel
and “chat” with other users in that channel. The client of the project places an automated
user, a bot, to listen in on the communication in a channel. Users may request commands
from it in which it may service depending on rights of the user.

The other system that the client of the project communicates with is the Counter-Strike
server that hosts a game for players to join. The client listens for events on the server and
relays them to the IRC channel and the statistics server of the project, which collects
stores the statistics from the events. The communication to the stats server through RMI
allows for dynamic generation of statistics that the users on IRC may query.

Objective
The reason for developing such a system, in relation to distributed systems, is to learn
more about Java’s RMI along with the various protocols that Internet clients have to use
in communicating with others. In addition, learning how to develop a centralized data
system effectively would be beneficial in both the academic and private sectors.
Description

The system was divided into two parts when it was being developed and likewise it will be described in the same way.

Client Architecture

The client of the project communicates with 3 different services: IRC server, Counter-Strike game server, and the project’s statistics server.

1.0 Client Design

1.1 IRC Subsystem

The IRC portion of the client deals with connecting to the IRC server, relaying events from the game server to the IRC channel, parsing messages from the server, deciphering the intent of the message (request for a command), and execution of the command. It is essentially the front end of the system to the users in the IRC server.
The client uses a thread for IRC subsystem with the users. When an event is sent from
the Counter-Strike subsystem of the client to the IRC subsystem, the event is relayed to
the IRC server to allow users to monitor the progress of the game on the game server.
When a message is sent to the client, it is parsed to identify the intent of the message.

The following can be requested of the client:

Statistics Request
If the user on the IRC server was requesting a query of statistics, then a request is
generated by the client and sent along to the statistics server for generation of results
based on the request, this is done through RMI. When the results are returned, the client
formats the results to be displayed to the user and are relayed to the IRC server.

Formats of Stats Requests

<table>
<thead>
<tr>
<th>Query format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.stats nick</td>
<td>general stats query</td>
</tr>
<tr>
<td>.stats weapons pistols nickname</td>
<td>pistol stats query</td>
</tr>
<tr>
<td>.stats weapons shotguns nickname</td>
<td>shotgun stats query</td>
</tr>
<tr>
<td>.stats weapons subs nickname</td>
<td>submachine gun stats</td>
</tr>
<tr>
<td>.stats weapons assault nickname</td>
<td>assault rifle stats</td>
</tr>
<tr>
<td>.stats weapons sniper nickname</td>
<td>sniper rifle stats</td>
</tr>
<tr>
<td>.stats weapons special nickname</td>
<td>special stats</td>
</tr>
<tr>
<td>.stats maps kills nickname</td>
<td>maps sorted by kills</td>
</tr>
<tr>
<td>.stats maps mins nickname</td>
<td>maps sorted by minutes</td>
</tr>
<tr>
<td>.stats servers kills nickname</td>
<td>servers by kills</td>
</tr>
<tr>
<td>.stats servers mins nickname</td>
<td>servers by minutes</td>
</tr>
<tr>
<td>.stats top map map_name</td>
<td>top players by map</td>
</tr>
<tr>
<td>.stats top server ip:port</td>
<td>top players by server</td>
</tr>
</tbody>
</table>

Manage Request
If the user was requesting a manage command to be performed, then the client will check
in its internal list of users that may be authorized to use a manage command. The client's
host, “real” name on the IRC server, and password provided with the command are
checked with the users in the list and authorization is determined. The manage
commands are used for administration of “rcon” users and other managers.
Formats of Manage Requests

<table>
<thead>
<tr>
<th>Command format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.manage list manager_password</td>
<td>lists users</td>
</tr>
<tr>
<td>.manage stats manager_password start</td>
<td>connects to stats server</td>
</tr>
<tr>
<td>.manage stats manager_password stop</td>
<td>disconnects from stats server</td>
</tr>
<tr>
<td>.manage remove manager_password nickname_to_remove</td>
<td>removes a user</td>
</tr>
<tr>
<td>.manage add regular mgr_password new_nick new_pass</td>
<td>adds a regular user</td>
</tr>
<tr>
<td>.manage add manager mgr_pass new_nick new_pass</td>
<td>add a manager user</td>
</tr>
<tr>
<td>.manage change password mgr_pass nick new_pass</td>
<td>changes a user’s password</td>
</tr>
<tr>
<td>.manage change nick mgr_pass nick new_nick</td>
<td>changes a user’s nickname</td>
</tr>
<tr>
<td>.manage change level mgr_pass nickname new_level</td>
<td>changes a user’s access level</td>
</tr>
</tbody>
</table>

Rcon Request
If the user request to administrate the Counter-Strike server, then like the manage request the user is checked with a list of users stored by the client. If authorized, the client will relay the remote administration command to the Counter-Strike server.

Format of Rcon Request
.rcon user_password rcon_command //sends rcon command to CS server

Counter-Strike Player Communication Request
If the user requests to say something to the players in the Counter-Strike game server, then the user is authorized by checking if the user is an IRC channel operator. The operator status of the user is determined by an event sent from the IRC server when a user has been “opped” by another user. The status is then stored in a list of users that are in the channel. If the user is an IRC channel operator, then the message to the players is relayed to the Counter-Strike server.

Format of Communication Request
.say message_to_players //sends a message to the players

1.2 Counter-Strike Subsystem
The Counter-Strike portion of the client deals with connection to the game server, receiving event messages from the game server, parsing the messages for use with the IRC subsystem and statistics server, sending the events to the statistics server, sending the events to the IRC subsystem, and relaying administration and player communication commands to the game server.

Some of the types of events from the Counter-Strike game server are:
Player Joined Game/Team
  • A player has joined the game server.
  • A player has joined a team to start playing.

Player killed another Player on the [team] with weapon
  • A player has killed another player on the opposite team with a particular weapon.

Player touched/rescued/killed a hostage
  • A counter-terrorist player has touched a hostage in order to bring the hostage to the rescue point.
  • A counter-terrorist player has rescued a hostage at the rescue point.
  • A player has killed a hostage (probably by accident).

Player planted the bomb
  • A terrorist player has planted the C4 in order to bomb a target of the mission.

Player defused the bomb
  • A counter-terrorist player has defused the planted C4 in order to rescue the target

Team wins
  • A status event that the round of play has ended and either the counter-terrorists, terrorists, or neither team wins the round
2.0 Server Design

The StatsServer class is the central component of the server. Upon start up, the application registers a copy of the public interface RMIServer (see section 2.1) with the rmiregistry. The client interacts with the server by invoking methods on this object. RMIServerImpl, the actual implementation of RMIServer, then passes all the interaction data to the StatsServer. From there, StatsServer will decide what needs to be done.

Game events passed from client are sent to the RoundParser. These events will eventually be parsed and written to the database. Before this can be done, however, a Monitor (see section 2.2) checks to see if another process is currently writing to the database. If one is, then the current process waits until the database is available again. Once it can continue, the events from the client are parsed and finally entered into the database.

Queries for various statistics, in the form of a StatRequest object, are sent to the QueryParser. The request is parsed and subsequently, the corresponding data is pulled.
from the database. The results are put into a QueryResults object that is sent to the StatsServer and then through RMIServer, back to the client. No interaction with the Monitor is needed (as was the case with RoundParser) since a read action can occur concurrently with writing and other reading events.

2.1 RMIServer Interface

There are four methods that the client can invoke upon RMIServer.

void connectNewServer(ServerInfo server)
• Connects a new game server to the central stats server. This must be called before any other method.

void retrieveStats(ServerInfo server, Vector vStats)
• Sends a batch of game events, held in a Vector, to the central stats server to write to the database. A game event can be a player death, map change, player changing nickname, etc. The client should call this method at the end of every round of a game.

void disconnectServer(ServerInfo server)
• Disconnects the game server from the central stats server.

QueryResults queryServer(StatsRequest query)
• Queries the stats server for certain statistics. The server will return a QueryResults object which contains the results of the query.

2.2 Monitor

A simple Monitor limits write access to the database. There can only be one thread writing to the database at any given time. Any other threads that attempt to write are forced to wait in a queue. Once a thread finishes writing, it notifies the first element in the queue and releases the lock. The awoken thread then acquires the lock and continues.
3.0 Summary

We learned a great deal from the project. Through CS2IRC, we gained quite a bit of experience working with Java RMI, multi-threaded design, client/server design, and synchronization schemes using monitors.

Working with RMI was initially frustrating, as neither of us had used it before. It took us a while to get the client and server to interface with each other, and even longer to crack down on all the bugs that popped up. However, it was satisfying in the end to see everything work out on the end.

We both plan on continuing work on CS2IRC, primarily on the application-specific side of things (changes in the way events are parsed, what statistics are stored, etc.). The client/server interface may also change, so that the client no longer has to explicitly connect and disconnect from the stats server. This would force the server to be more robust, handling situations where the client or even the server may suddenly crash.