

Distributed Website Information Client & Server

Implementation Details

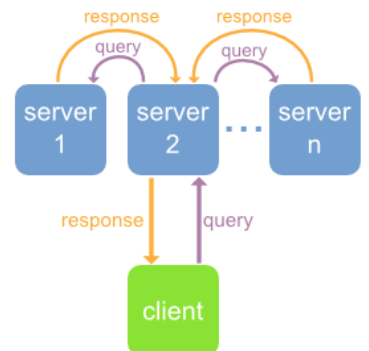
My implementation allows for any number of servers to deal with one or more clients. When a client connects to the network it will pick a single server to communicate with. Client search queries are sent to that server which will search itself and query all other servers on behalf of the client, collate their responses and return them to the client.

If a server is shut down, it will send a copy of its database to another server and if the client was communicating with that server, it will find a new one to communicate with.

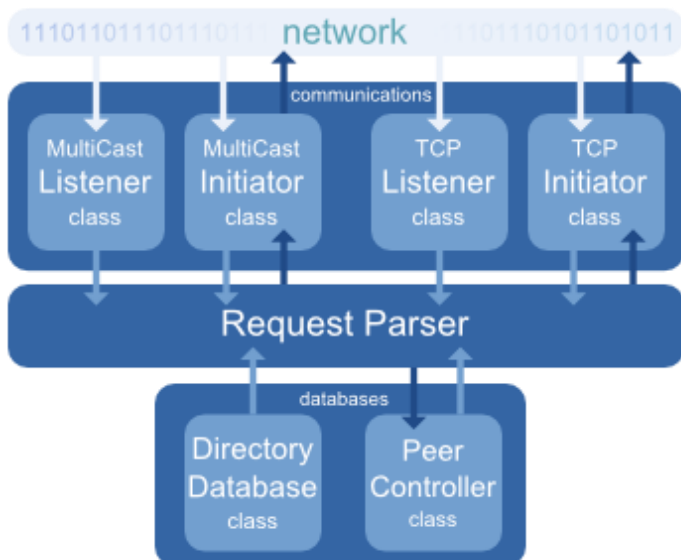
The client and server applications have a similar internal architecture. They continually listen on port 5555 TCP and UDP. When data is received through these sockets it is passed to a parser which decides on which action to take based on which command was received, or no action if no command was recognised. The parser will then perform some operation (e.g. search locally for data) and optionally return data to the sender of the original message or create a connection to another server.

Servers maintain a database of website information and also a list of online peer servers. Clients maintain a list of online servers. When new servers come online they are added automatically to existing peer/server lists and they are removed automatically when shut down or if they become unresponsive (e.g. lose network connectivity), in which case the server that discovers this will tell all others to remove it from their peer lists.

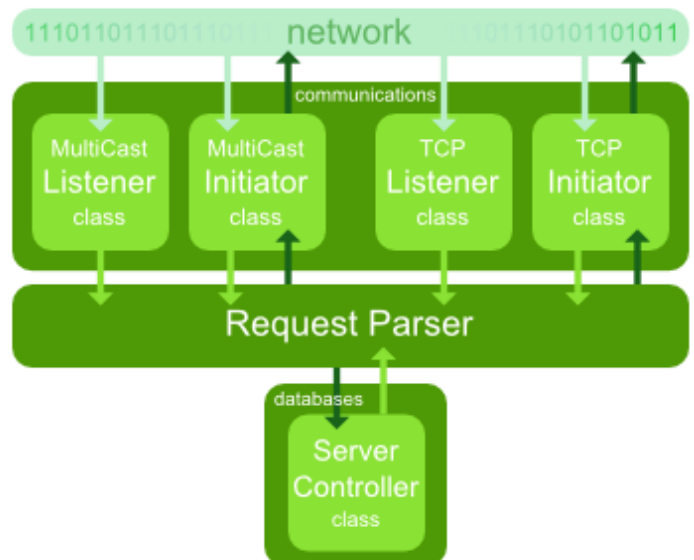
On start-up, a server can be fed a specially formatted text input file containing the website records for the database. This is interpreted by the server and added to its database. It will then signal its presence (using a multicast packet) to all other online servers, which will all reply and initiate a three-way handshake to confirm their presence to the new server. When a client is run, it will also signal its presence to all running servers, who will all respond with a three-way handshake. The client will then pick one server to communicate with.



Internal view of the server



Internal view of the client



Objectives

I have met all of the objectives using one implementation of the client and server (webDirectoryClient.java and webDirectoryServer.java). For portability reasons both the client and the server have a command-line interface.

Basic Objectives

I have met all of the basic objectives. You may run a client/server which exchange search queries and search results. An error is displayed if no matches are found.

Intermediate Objectives

I have met all of the intermediate objectives. You may run multiple instances of a server with different databases (as a specified input file) on different computers in the same subnet. Servers will query each other for a search request made by a client.

Advanced Objectives

I have met all of the intermediate objectives. Servers dynamically add each other to their peer lists when a new one comes online and they will all remove one if it is shut down or times out. When a server is shut down it will transfer its database to another server.

Quality of Coding & Implementation

- Neat code, well commented. Abstracted into separate classes and packages.
- Efficient use of data structures and control blocks.
- Minimal network traffic.
- Only requires one port to be open (TCP & UDP send & receive on port 5555).
- Simple protocol.
- Matching records are returned if they appear on any server, not just the one communicating with the client.

Robustness

- The client and server applications are stable and have error checking built in throughout the codebase. I have made use of exception handling to intelligently deal with errors.
- Unrecognised commands (via the command-line and network) are ignored.

Usability and Accessibility

To add to the usability and accessibility of the programs:

- A 'help' command is included so that the user can check which commands are available.
- On the server, the IP of the local machine is displayed on the command prompt to make it easy to determine which machine the user is interacting with (users usually interact with several concurrently)
- The commands on the server and client are short, easy to type and easy to remember.
- Useful notifications are displayed on the command prompt when important events occur e.g. a new server is detected.
- For the client, detection of servers is automatic. On the server, detection of new peers and dead peers is automatic. Database transfer on shutdown is automatic also.

Limitations

- All servers and clients must be on the same subnet. If I had more time I would create the ability to 'bridge' over multiple subnets by electing one server to communicate between subnets in each group.
- The client-server communication channels are not secure – any non-authorized program could read the traffic being sent through the system or a rouge program could masquerade as a legitimate server and provide illegitimate information to clients. If I had more time I would implement encryption and client/server authentication procedures.

22/01/2010

- New records cannot be added to running servers. To add records you must kill the server, modify the input file and re-launch it. If I had more time I would implement the ability to add records on-the-fly without having to restart servers.

Installation & Usage Instructions

Running an instance of the server

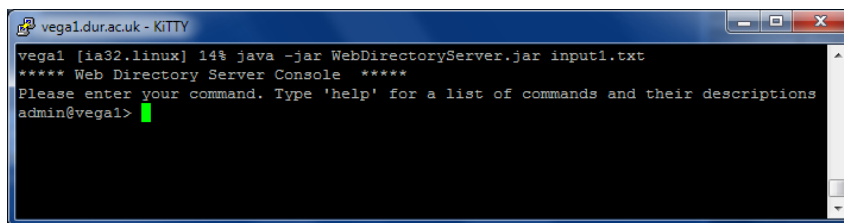
The server requires Java 1.6. I tested the implementations on the Vega linux machines at Durham University. Assuming you are in the same directory as the jar file type in a command-line:

```
Java -jar webDirectoryServer.jar databaseInput.txt
```

Where, '**databaseInput.txt**' is the database file for that server. You may run as many instances as you like but only one per unique IP on the subnet. I have provided three input files, 1.txt, 2.txt and 3.txt which you could use to test the implementation.

You may use the command-line interface to interact with the server. The following commands are available:

- **help** – show this list of commands
- **showdata** - display all the database records on this server
- **localsearch** – search this server's database for a record (this does not search its peers)
- **showpeers** – list all the known peers of this server (other online servers)
- **exit** – shut down the server (its database will be sent to a peer if it can find one)



```
vega1.duracuk - KITTY
vega1 [ia32.linux] 14% java -jar WebDirectoryServer.jar input1.txt
***** Web Directory Server Console *****
Please enter your command. Type 'help' for a list of commands and their descriptions
admin@vega1>
```

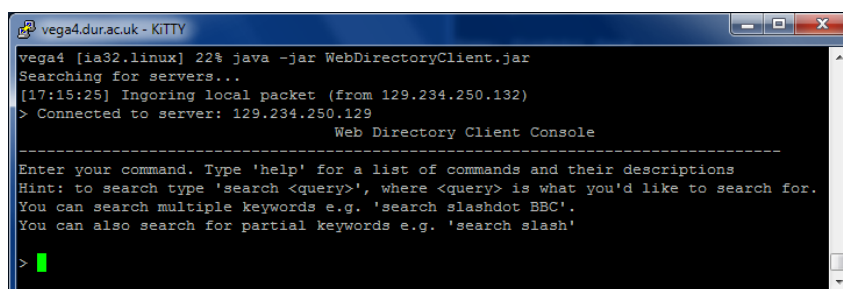
Running the client

The client requires Java 1.6. To run an instance of the client, assuming you are in the same directory as the jar file type in a command-line:

```
Java -jar webDirectoryClient.jar
```

The client will automatically attempt to find a server to communicate with

- **help** – show this list of commands
- **search <query>** - search all servers for any records containing the text 'query'
- **findnewserver** – search attempt to find a server to communicate with. If the client is already communicating with a server, another will be selected.
- **exit** – quit the client



```
vega4.duracuk - KITTY
vega4 [ia32.linux] 22% java -jar WebDirectoryClient.jar
Searching for servers...
[17:15:25] Ingoing local packet (from 129.234.250.132)
> Connected to server: 129.234.250.129
-----
Web Directory Client Console
-----
Enter your command. Type 'help' for a list of commands and their descriptions
Hint: to search type 'search <query>', where <query> is what you'd like to search for.
You can search multiple keywords e.g. 'search slashdot BBC'.
You can also search for partial keywords e.g. 'search slash'
>
```