Mandatory Assignment 1

The CORBA Taste Profile Service

By:

Paul Joakim Oulie Andreassen

Thomas Kristiansen

Uy Viet Tran

How to run the application

1. Start orbd with given port:

   Linux: orbd -ORBInitialPort <Port>

   Windows: start orbd -ORBInitialPort <Port>

2. Start server:

   java -jar <Full path to jarfile> -ORBInitialPort <Port> <Full path to input file (the "database file")> <Integer value to decide if cache is to be used on server or not (0 = cache off. 1 = cache on)>

   For example:

   java -jar C:\Users\server.jar -ORBInitialPort 6000 C:\Users\train_triplets.txt 0

3. Wait until server outputs “Init finished”

4. Start client:

   java -jar <Full path to jarfile> -ORBInitialPort <Port> <Full path to input file> <Full path to output file (including name of file with extension)> <Integer value to decide if cache is to be used on client or not (0 = cache off. 1 = cache on)>

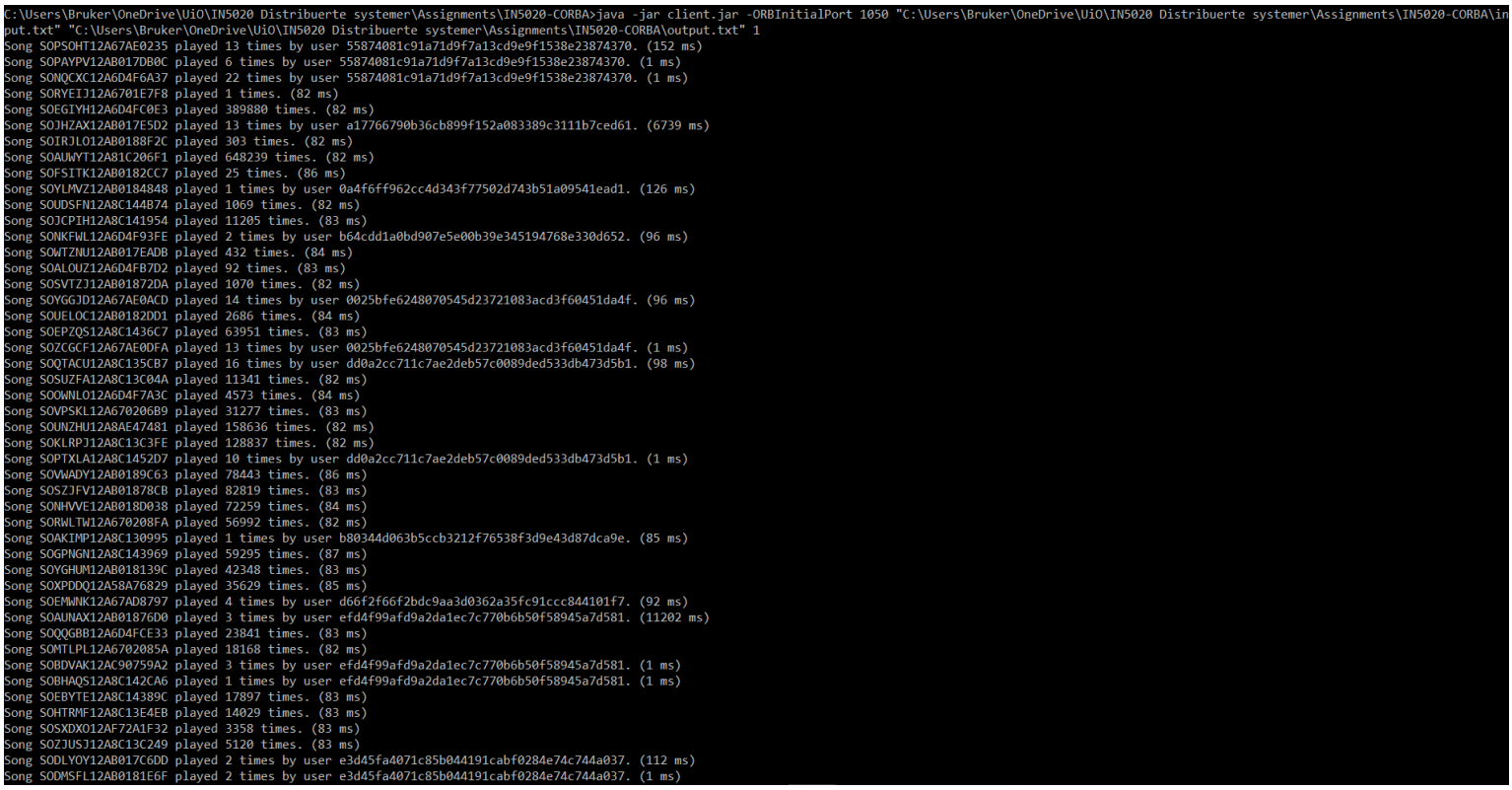
   For example:

   java -jar C:\Users\client.jar -ORBInitialPort 6000 C:\Users\input.txt C:\Users\output.txt 0

Design and Implementation

Client:

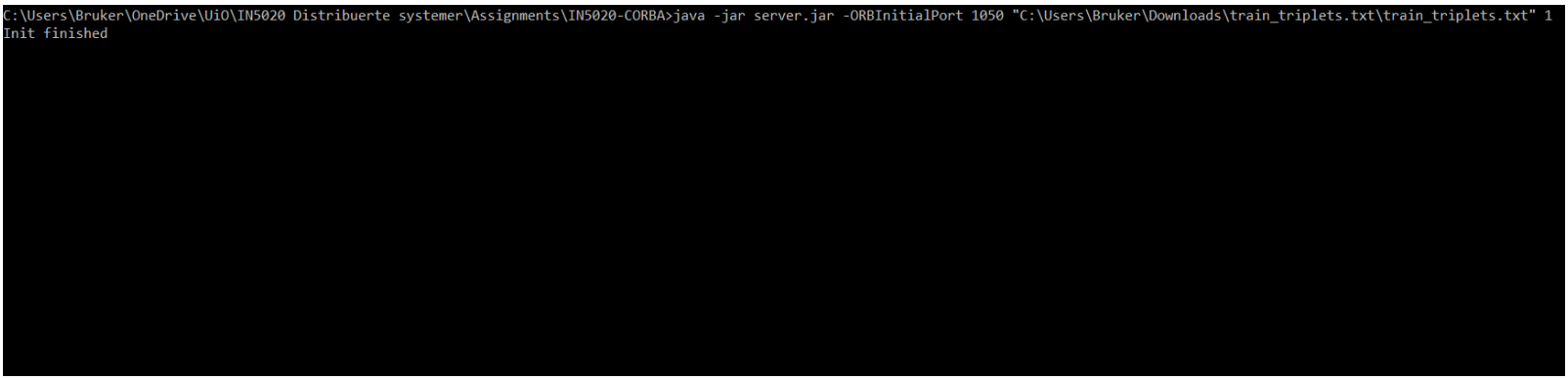
The client consist of just one pretty straight forward class. This class connects the client to the server trough the CORBA ORB, and uses this connection to call the different methods available. What methods to be called are decided by an input file. The class takes three arguments (in addition to the CORBA arguments). The first argument is a path to an input file. It is dependent on this file to know what methods to call. The input file has to consist of one or more lines, where each line holds information about the method to call and the parameters to pass to the method. The second argument is a path to an output file. This is the file where the values returned from the methods are outputted to. The last argument is an integer, which decides if client side cache is to be used or not (0 = client cache off, 1 = client cache on). More info about the cache is written under the “Cache” heading below.



Screenshot of client running

Server and servant:

The server basically consist of two classes. One class that initiates the server through the CORBA ORB, and another class that acts like the servant of the server which holds all methods that are reachable from clients. In addition to the server and servant classes, the server also uses a file holding a huge database of user and song associations. This is the file the different methods of the server uses to get the values it is going to return. The server class takes two arguments (in addition to the CORBA arguments). The first argument is a path to the database file. The second argument is an integer, which decides if server side cache is to be used or not (0 = client cache off, 1 = client cache on). More info about the cache is written under the "Cache" heading below.



Screenshot of server running

Cache:

For the implementation of the cache, we have created one interface, and two classes that implements that interface. Both classes has a HashMap which holds the different objects that are being saved in the cache.

The first cache-class is the user-cache. This cache can hold up to one thousand user profiles. If the cache is full, and you try to add another user profile to the cache, that new profile is only added if the user has played more total songs than one or more of the user profiles already in the cache. If this is the case, the user profile with the fewest amount of played songs is removed from the cache when this new user profile is added.

The second cache-class is the song-cache. This cache has no limit to the amount of songs it can hold, so there is no check executed when you try to add a song to it.

If cache is enabled on the server, the server has both a user-cache and a song-cache. To fill these caches with data, we have created an init method that goes through the database-file, and fills both caches with data. Since the database-file will stay the same through the lifetime of the server, there is no reason to do this more than the one time it is done at the startup of the server.

To fill the user-cache, we take advantage of the fact that all the lines of the database file are grouped based on the user ID. This means that we, on one read of the file, can go through all the user-song combination of one user, create a user profile, add the profile to the cache and then repeat this on the next user. By doing this, we end up with a user-cache containing the one thousand users that has played the most songs, without storing more than one thousand users in memory at all times.

The song-cache is a bit simpler, since we can store all the songs we find in the database file. This means that each time we get to a new line of the database file, we either add the song to the song-cache or update the information of the song in the song-cache, if that song already has been added to the cache.

On the client, we only use a user-cache, if cache is enabled. This cache starts of as empty, and gets filled up as the clients retrieves a user-profile from one of the methods it has called from the server. This means that only user-profiles already used by the client are stored in the cache. The same one thousand user limit also apply for this cache.