A News System Implementation
EDA031 Project

Erik Molin, erik.johan.molin@gmail.com
Matilda Hjälle, tpi11mhj@student.lu.se
Nina Castor, nod09nca@student.lu.se

April 13, 2015

System Design

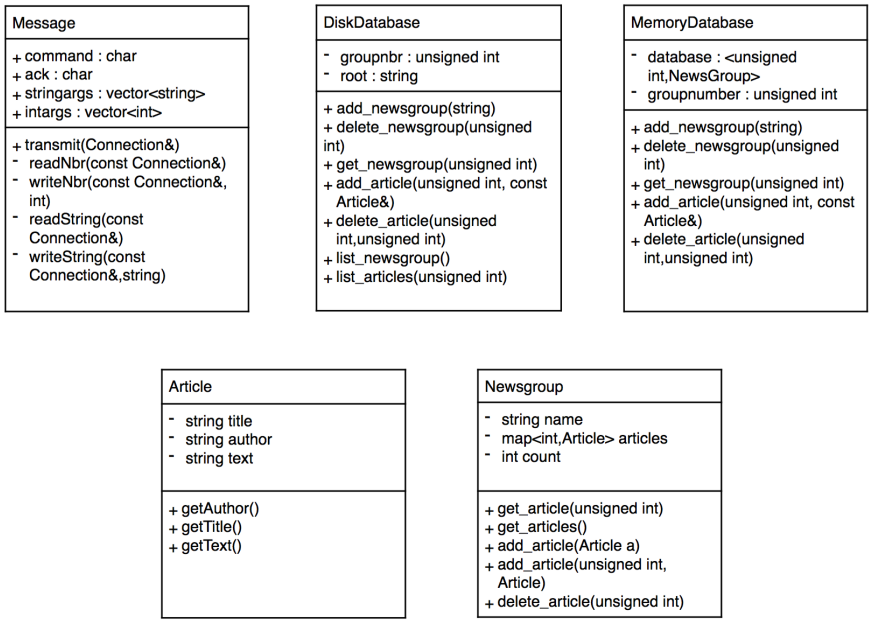

Figure 1: Class Diagram

Newsgroup and article representation

The class `Article` represents the articles in a newsgroup. It contains three attributes of type `string` to represent the title, the author and the text. All attributes are private and the class contains get-methods for all three attributes. The class `NewsGroup` contains a map of type `map<unsigned int,Article>` that keep track of the articles in that newsgroup. A newsgroup gives each article a unique key that represents the identification number and make sure that each article have a unique number. Every newsgroup have a string attribute for the name of the newsgroup. Each newsgroup have a unique name and is given a unique identification number by the database.

Database

The system uses two different databases. One that uses a database stored in primary memory, `MemoryDatabase`, and one that stores the database on the computer, `DiskDatabase`. The methods in the databases are called by the server.

MemoryDatabase

The class `MemoryDatabase` stores the database in the primary memory and starts from scratch each time it is invoked. The has two private attributes, one map of type `map<unsigned int,NewsGroup>` to store the newsgroup and an attribute of type `unsigned int` to make sure that the newsgroups get a unique identifications number.

DiskDatabase

The class `DiskDatabase` creates a directory on the computer in which the database is stored. Each newsgroup is represented by a directory and articles are stored as separate *.txt*-files in the corresponding newsgroup directory. The Diskdatabase contains two private attributes, one of type `unsigned int` to count the number of newsgroup and give each newsgroup a unique identification number, and one of type `string` that represents the path to the database directory.

Message

The class `Message` implements the message protocol specified by the project description. The server and the client communicates by sending messages over a connection. Every time the server or the client wants to send new command a new message is created. A message can contain up to four parameters, two parameters of type `char`, one for sending the commands and one for sending an answer to the client if the command succeeded or not, and two vectors, one of type `vector<int>` for sending the identification numbers and one of type `vector<string>` to send the names of newsgroups and articles.

Server

The systems uses two different servers depending on which database it uses. The main-method creates an object of type `Server` and an object of type `DiskDatabase` or an object of type `MemoryDatabase` depending on which database the server uses. The server then awaits activity from the client and performs the commands from the client on the database. This is done in the two runnable files *DiskServer* and *MemoryServer*.

Client

The client reads input from the keyboard and sends the commands as a message to the server. It then receives answer from the server and presents them in the terminal. The client connects to the server and the server can be connected to several clients.

Sequence diagram of the system

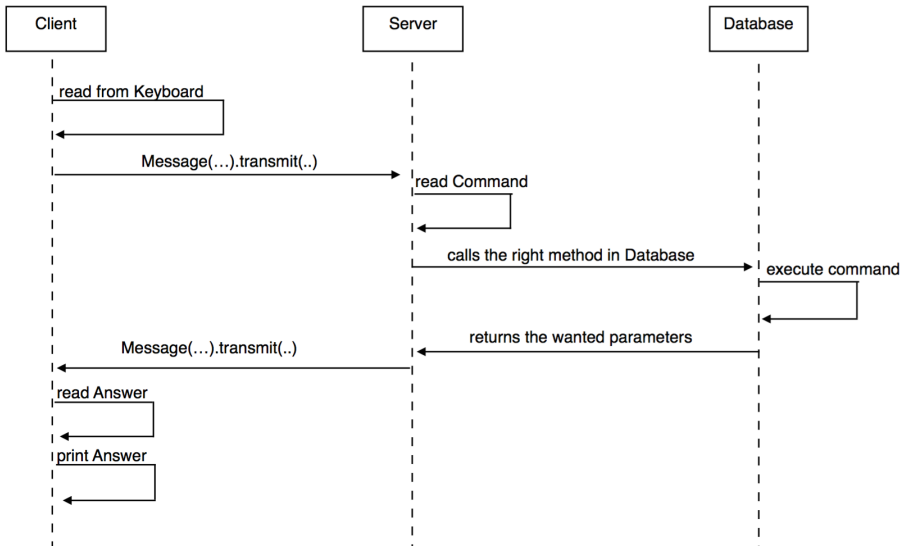

Figure 2: Sequence Diagram

Conclusions

Our system doesn't work. We tried to locate the error and think that it is in our implementation of the message protocol and how we use the class `Connection`. It is therefor difficult to determine which of the requirements our system full-fills. We need to carefully go through our implementation of the message protocol again to make sure the implementation is correct. We think that our databases works as it should, but that the connection between the server and the client is were the problems originate. Our work process wasn't optimal, since we worked separate on different parts of the projects and put it all together a bit to late.