SEP3 – Group 4

13 December 2012

Advisors:
Mona W. Andersen
Hans Søndergaard
Jan M. Pedersen

Team:
Galin Kostov – 166698
Martin Kyukov – 166703
Janis Bruno – 166732
Mikkel Cortnum – 171640

Table of Contents

Table of Figures

Abstract

The purpose of the project was to build a distributed system, utilizing at least two different types of middleware, and including legacy functions written in the C programming language, using JNI. This goal was achieved by designing a system that utilizes CORBA and RMI to communicate between each of its servers. The last goal was achieved by implementing a calculator server, which utilizes a native library of C functions to do all calculations. The resulting application fulfilled the requirements set forth by the assignment; the system utilizes both CORBA and RMI as middleware, and includes a native library, of C functions, on one of the servers. All in all, the final system is more than satisfactory, and the requirements of the project were fully met.

Introduction

Online users have access to many web services, which provide simple every day-use features with a touch of entertainment such as cleverbot.com and Google calculator, but they are spread across multiple websites. The team's interest is to develop a distributed system with multiple servers to provide users with the possibility to access different services and features as one service.

During the project period the objective was to create a complete distributed client-server system with the features of scrabble, jokes, calculator and music. To fulfill the objective each features was deployed on a different server so the client requests an action from main server and gets reply from one particular feature server. Furthermore client-requests are handled by an intelligent string reader to provide a feel of human-like conversations. To fulfill all objectives the team implemented the calculator feature in the C programming language. Different types of middleware are used to establish connection between servers, both RMI and CORBA.

Further in the report follows detailed description of the most important parts of the system. For example: how the system was developed (including diagrams), how analyses were carried out, how the designing stages went off and the implementation of the system.

Analysis

The requirements for the system are listed in the following table, and are based on the user stories chosen by the product owner[1].

- Functional Requirements:
  - The user should write an ordinary message in order to invoke one of the services.
  - The user can use different features.
  - The user can only be human.
  - The user can invoke more than one feature per session.
- Non-Functional Requirements:

[1] User stories can be found in the product backlog, on the CD: Appendixes/SCRUM/SCRUMworkbook.

- Each service is supported by one of the dedicated servers.
- The main server redirects the user to the correct server, according to the corresponding service.
- The calculator will be written in C.
- The main server will be written in Java.
- Multi-client, more than one client can be connected to the main server at the same time.
- The user connects to the main server only.
- Stand-alone system (no active administrators needed.)

Figure I - System Requirements

As also mentioned in the introduction, the system will consist of five separate servers, a main server and a server for each feature; music, jokes, scrabble and calculations. The use cases available to the client are depicted in the figure II.

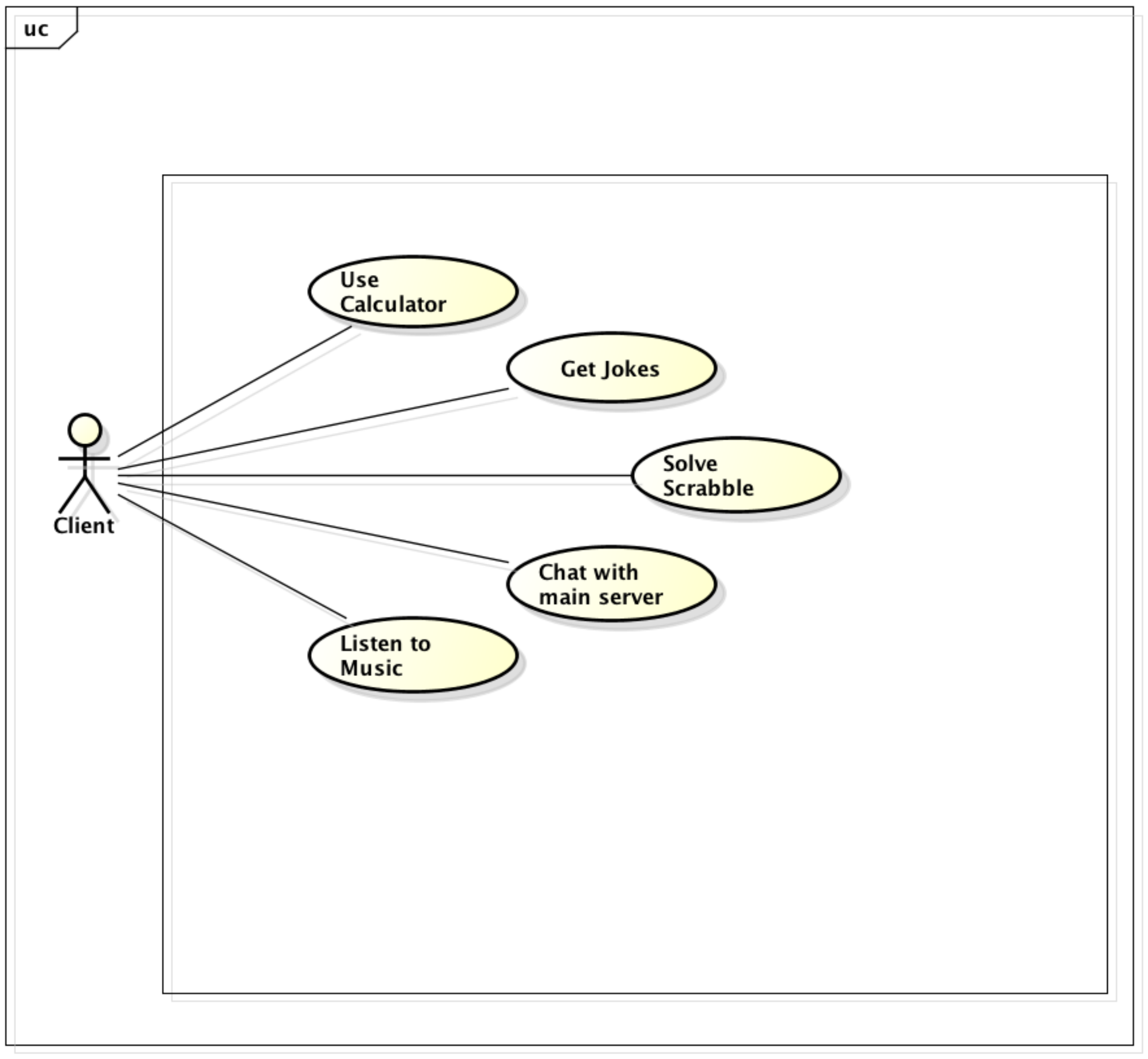

Figure II - Use Case Diagram

Each use case happens through the client application. Before any of the use cases can take place the client has to connect to the server. When the client opens their client application they will first be asked to input their screen name, and then to choose which representative of the server the want to talk to, Marcus or Jane. If the server does not recognize the selected representative the client will chat with a standard representative with no special comebacks. When connected, the client can type their request and the intelligent string recognition class will analyze the client's message. If any word in the message is recognized as a command, the main server will take the client to the appropriate feature. Alternatively the client can select a specific feature by clicking one of the "feature buttons" in the GUI. This flow of activities is depicted in the activity diagram for overall use of the system, which can be seen in figure III.

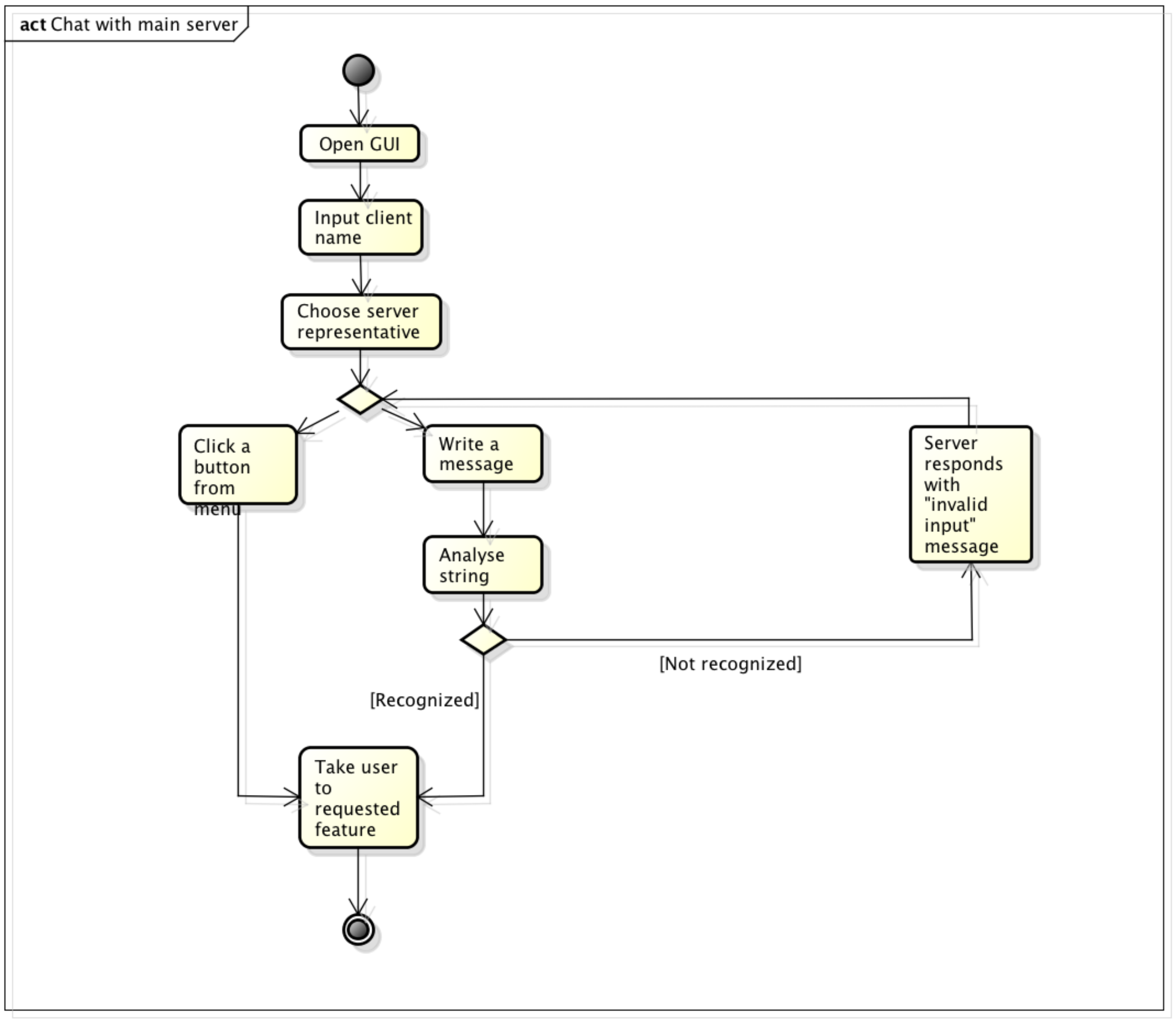
act Chat with main server
Open GUI
Input client name
Choose server representative
Click a button from menu
Write a message
Server responds with "invalid input" message
Analyse string
[Not recognized]
[Recognized]
Take user to requested feature

Figure III - Activity Diagram for Overall Use of System

After connecting the client can select one of four features, which will be described in the following section of this report.

One of the features the client can use is the jokes feature, this use case can be invoked either by typing a command like "tell me a joke" or "crack me up" or by clicking the jokes button in the GUI. After this the server will connect to the jokes server and return a random joke to the client. The flow of activities in this use case is depicted in figure IV.

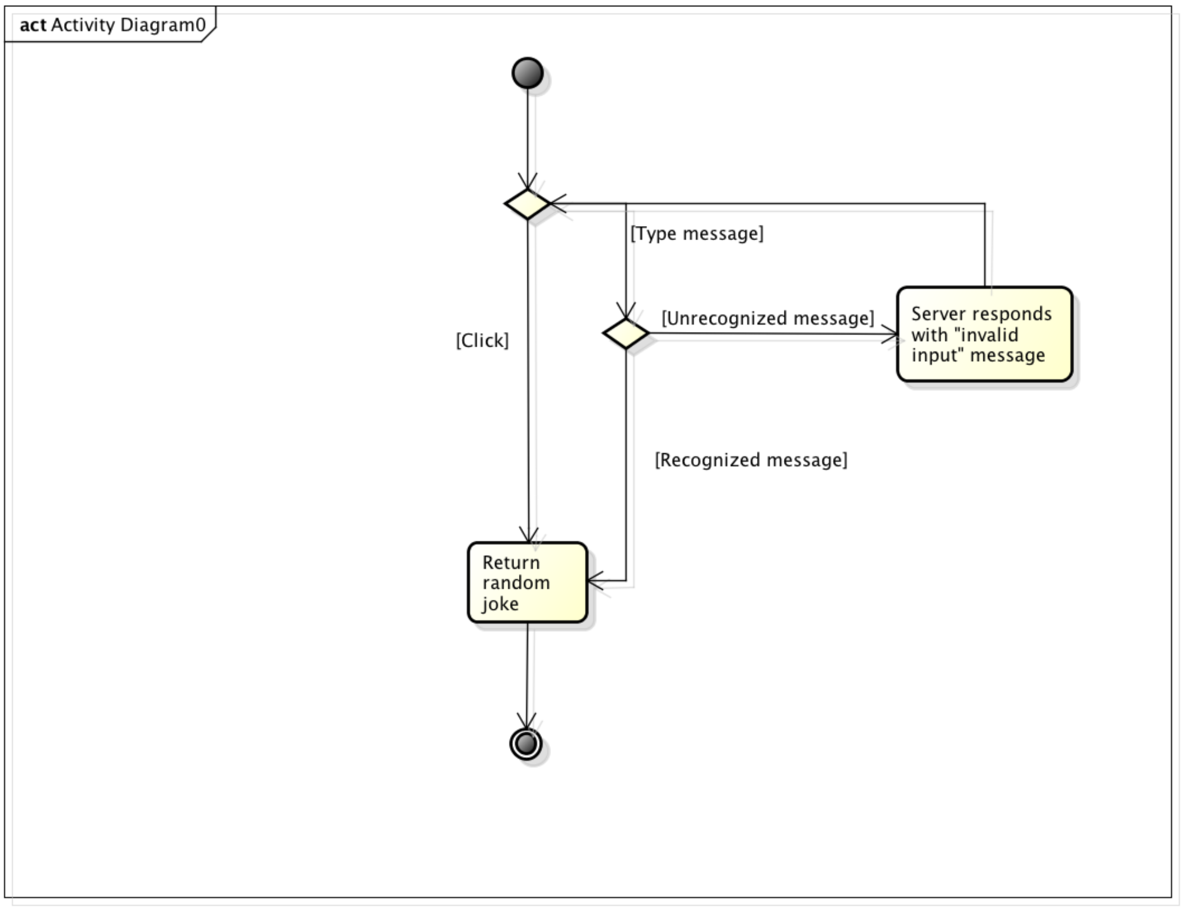

Figure IV - Activity Diagram for Jokes Use Case

If for some reason the client inputs a message, which is not recognized as a command, the "chat with main server" use case is invoked. In this use case the server will return a random comeback to the client depending on which server representative was chosen when the client first connected; for example, "Marcus: Try asking for a joke." Or "Jane: Sorry Sir, I do not know what you mean."[2] Also if the client writes a message that is recognized as a thank you message the server will return a random thank you response also depending on the server representative; for example, "Marcus: NP bro." or "Jane: My pleasure."[3]

Another feature the client can use is the scrabble feature, in the same way as the jokes use case the scrabble use case can be invoked either by typing or by clicking the scrabble button. After the server recognizes that the client wishes to use the scrabble feature the server will ask for the letters to be scrabbled. If no letters are typed i.e. "?09+32?-", the server will respond with an invalid input message and ask the client to try again. When the client has input at least one letter the main server will send the input to the scrabble server, where the scrabble solver will find all the legal English combos from the letters and return them to the client.

[2] CD: Appendixes/Source Code/src/MainServer/Model/MainServer.java
[3] CD: Appendixes/Source Code/src/MainServer/Model/MainServer.java

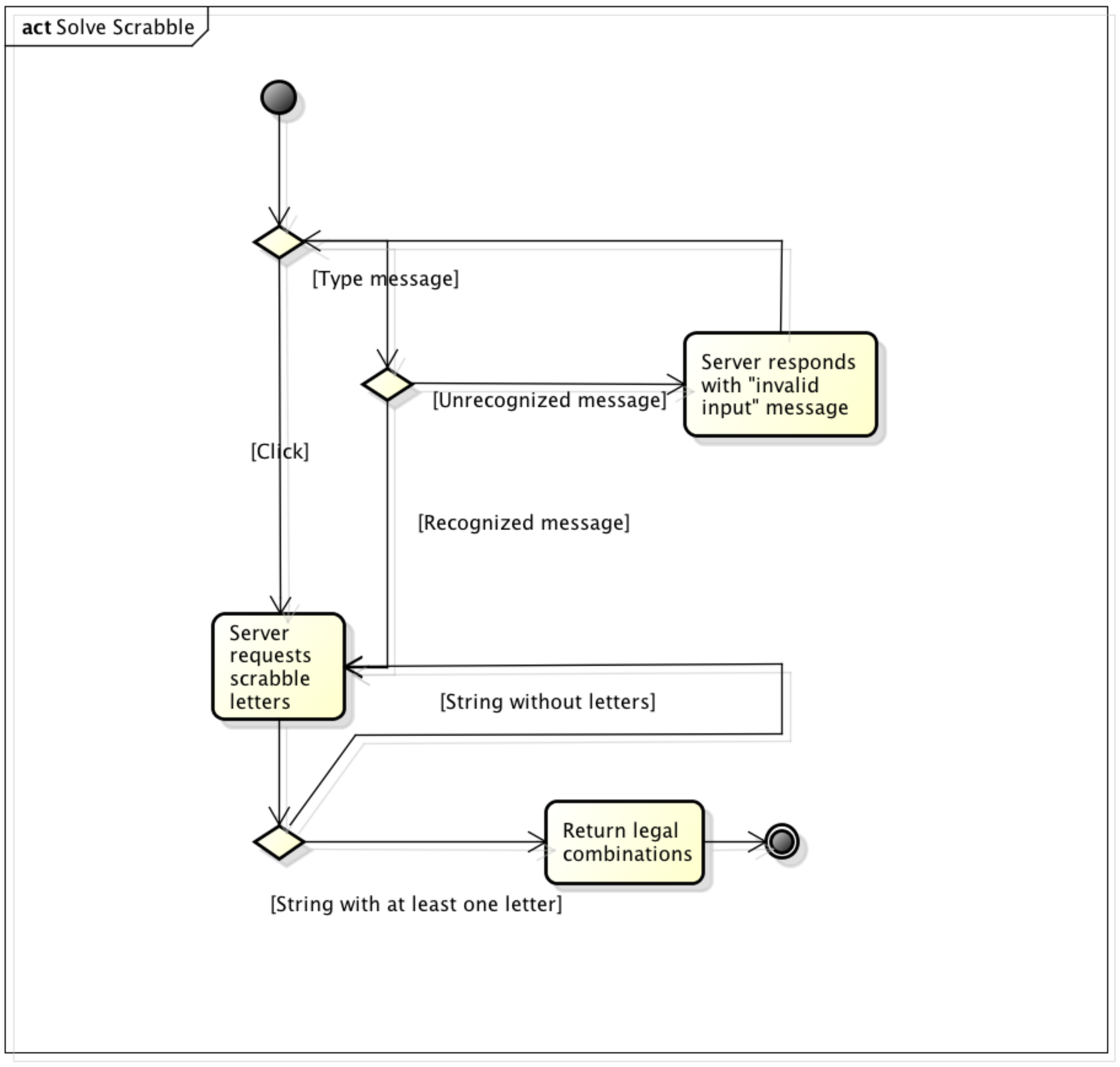
act Solve Scrabble
[Type message]
Server responds with "invalid input" message
[Unrecognized message]
[Click]
[Recognized message]
Server requests scrabble letters
[String without letters]
Return legal combinations
[String with at least one letter]

Figure V - Activity Diagram for Scrabble Use Case

Yet another feature the client can use is the music feature, once more like the scrabble and the joke use cases the music use case can be invoked either by typing or by clicking the music button. If a song is already playing on the server, the server will tell the client what is playing. The server will then ask the client what they want to listen to. The server then forwards this message to the music server, where it will first be treated as a song, if a matching son is found this I returned to the main server. However, if no song match is found the music server will treat the message as an artist name. If an artist match is found the music server will return a random song by that artist to the main server. If no song match and no artist match are found, no song will be returned to the main server. The main server will set the received song as now playing on the server. If no song is received the server will stop playing music until any one client invokes the music feature. The flow of activities in this use case is described in figure VI.

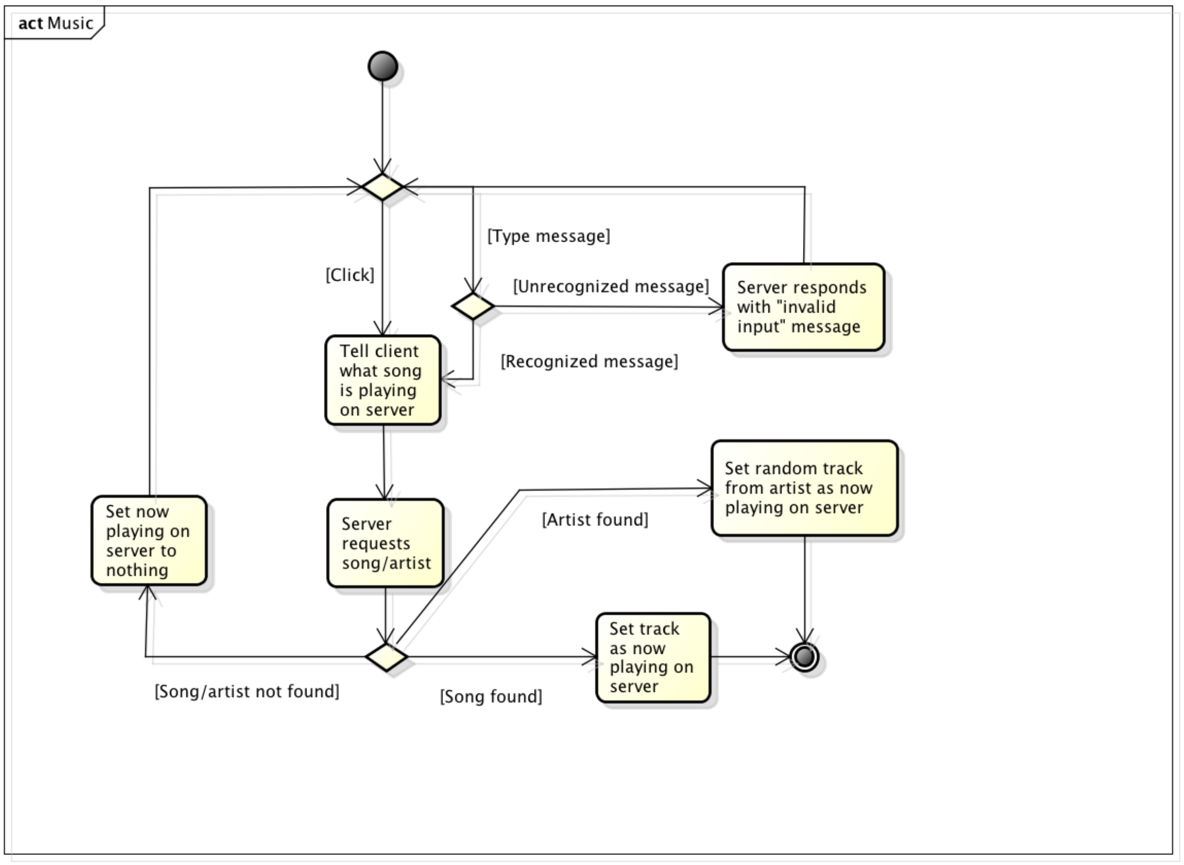
act Music
[Type message]
[Click]
[Unrecognized message]
Server responds with "invalid input" message
Tell client what song is playing on server
[Recognized message]
Set random track from artist as now playing on server
[Artist found]
Set now playing on server to nothing
Server requests song/artist
Set track as now playing on server
[Song/artist not found]
[Song found]

Figure VI - Activity Diagram for Music Use Case

The last feature that the user can use is the calculator feature, once more like the scrabble, the joke and the music use cases the calculator use case can be invoked either by typing or by clicking the calculator button. The server then asks for the equation to be solved, if no legal math chars are entered i.e. "lk_o!id.," the server will, like in scrabble use case, respond with an invalid input message and ask the client to try again. If at least one legal math char is entered the main server will forward the expression to the calculator server, which will solve the expression and return the result to the client. The flow of activities in this use case is depicted in figure VII.

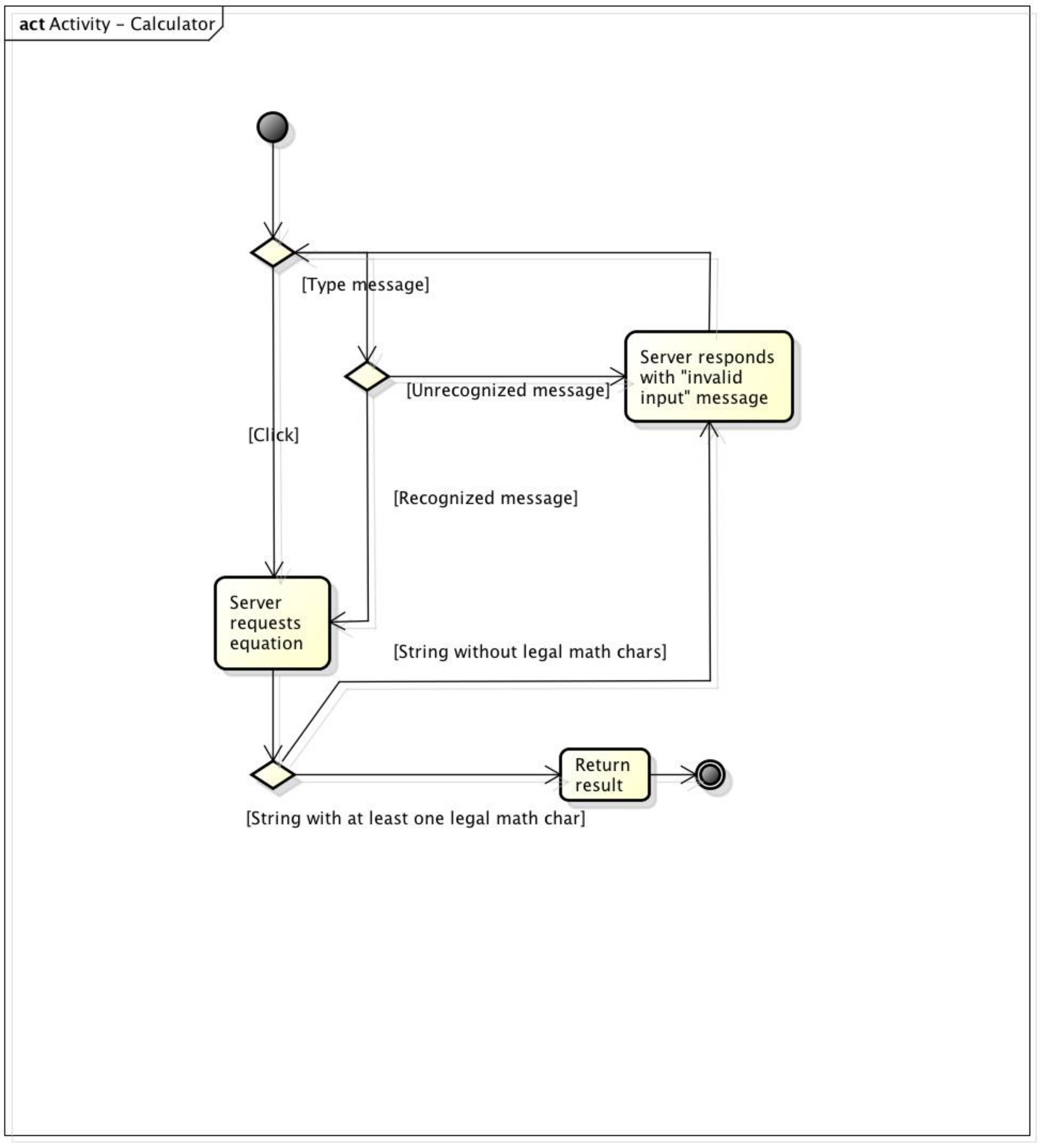
act Activity – Calculator
[Type message]
Server responds with "invalid input" message
[Unrecognized message]
[Click]
[Recognized message]
Server requests equation
[String without legal math chars]
Return result
[String with at least one legal math char]

Figure VII - Activity Diagram for Calculator Use Case

The following figure, figure VIII, shows a simple class diagram for the entire system. In this simple class diagram only the most important classes are included, such as the different servers, and only the most important methods and attributes are included, such as sendMessage() for the main server and the attributes of the message class.

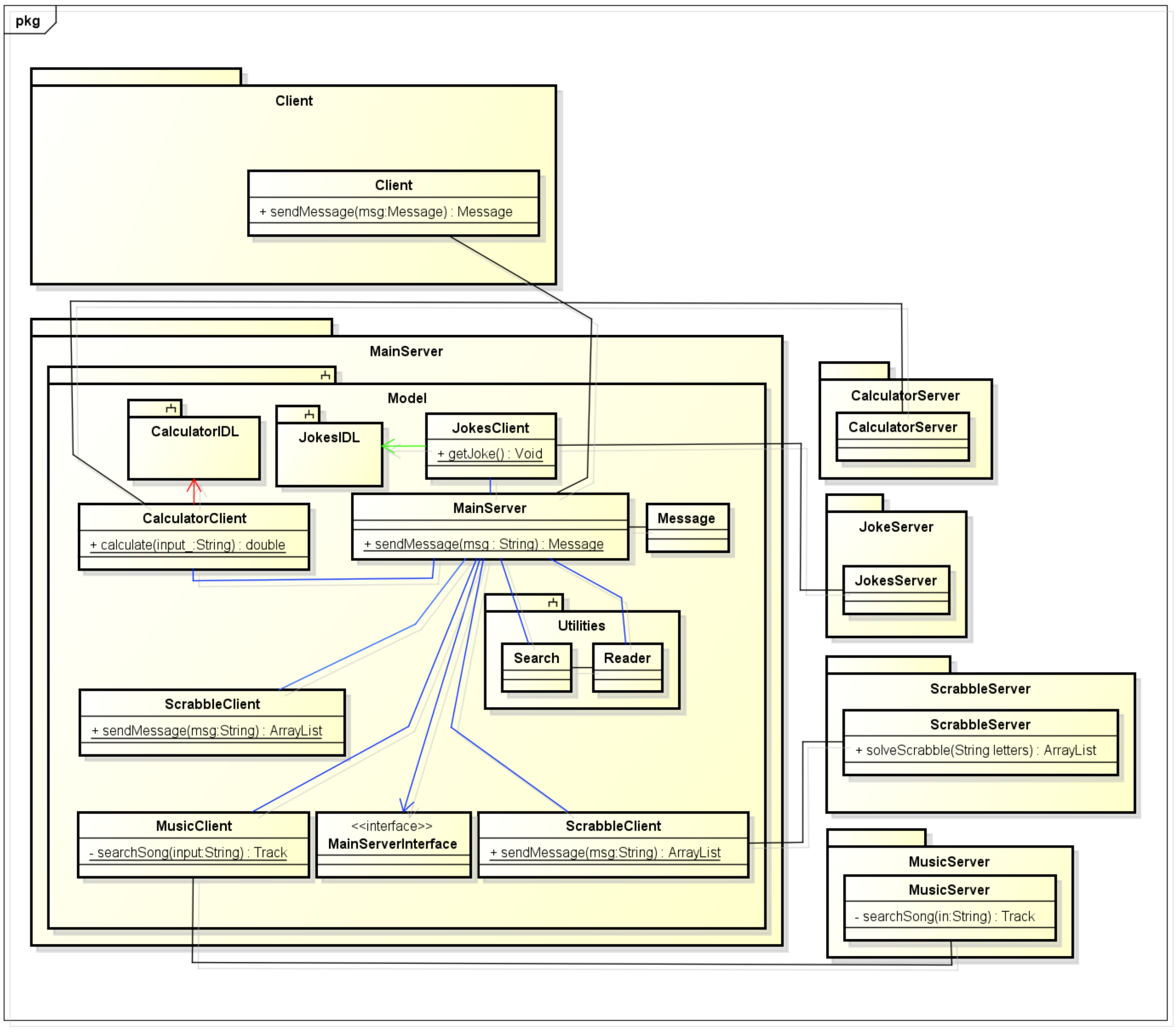

Figure VIII - Simple Class Diagram for Entire System

As depicted in the class diagram the main server is connected to each of the separate servers through a separate client class for each separate server. The communication between the client and the main server happens by sending a message object to the main server and returning a message object to the client through the main servers sendMessage() method.

Design

The system is designed to be understandable and easy to use. For this reason the graphical user interface was designed to be as simple as possible. The GUI includes an uneditable text field for displaying the communication between the client and the server, the message window, and an editable text field for where the client can write their chat message to the server, the chat window. The GUI also features a send button, and five buttons for each feature the system provides to the user. The first draft of the GUI can be seen in the following picture as a paper mockup, figure IX.

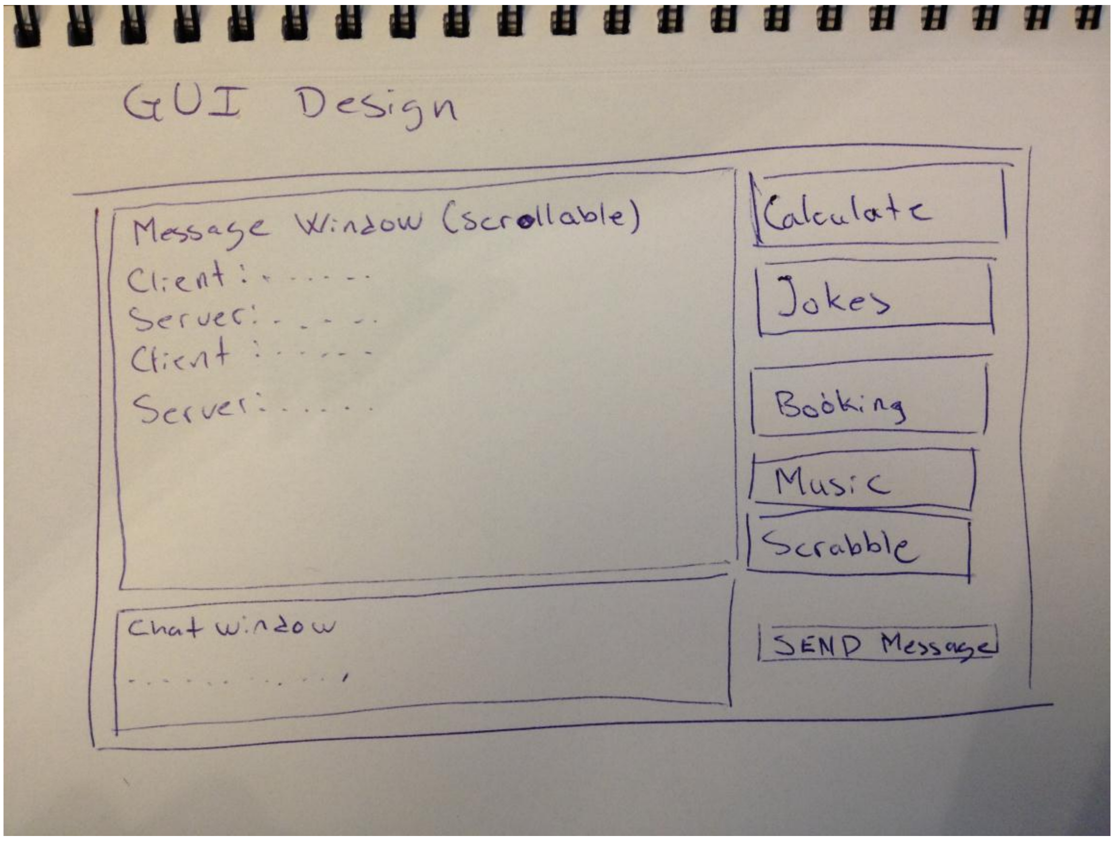

Figure IX - GUI Paper Mockup

The design of the GUI is of course made so that in order to send a message the user should be able to do so either by pressing enter, or by clicking the send message button.

The system itself was designed to meet the criteria for the SEP3 project; to have more than one type of middleware, to use JNI and the C programming language and to be a distributed system. Each feature was designed so that it can be implemented on a separate server, all connected to a main server. The middleware used for these connections are RMI and CORBA. The design pattern applied to keep order within the system itself is model-view-controller. MVC is applied to each separate server, even if it only has model classes, except for the calculator server[4]. This decision was made to ensure the same organization on each separate server. The use of MVC is depicted in the below class diagram, figure X. This class diagram does not show all methods and attributes, for full class diagram see footnote 5.

[4] See the section on implementation.

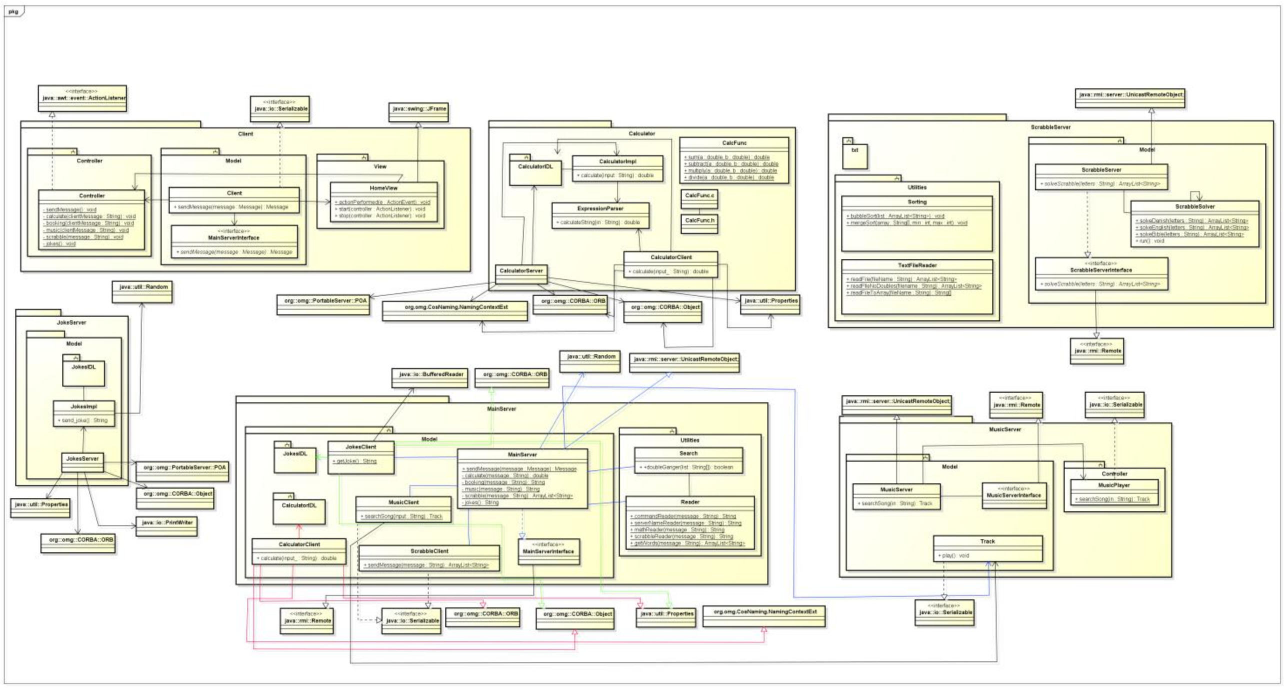

Figure X - Class Diagram for Entire System[5]

The main server is designed with a MainServer class that has the primary method for communicating with the client, namely the SendMessage() method. This method takes a message object as an argument and returns a message object as well. The message object has five attributes, the message itself, the serverName and three boolean arguments, scrabble, music and calculate. These attributes are necessary for the server to have more than one client connected at one time. The serverName will tell the server which representative the specific client is talking to and each boolean argument will tell the server, if the client has already invoked either the music, calculator or scrabble feature. The connection between the main server and the client is RMI. The choice of RMI for the connection between the main server and the client was made so that the client could be implemented with as little code as possible. The client is designed so that all the work takes place in the controller and not in the class called Client. This is done to honor the choice of MVC. The client class only has one method, which, unless the client is in the startup phase, simply forwards the message from the controller to the main server and the returned message from the main server back to the client. If in the startup phase, the client will invoke to different methods on the main server, namely the setClientName() and the serServerName() methods. These methods could be executed locally without connecting to the server, but by connecting to the server, the startup message that the client receives can be changed without having to update the client's application. The flow of the sendMessage() method is depicted in the below activity diagram, figure XI.

[5] Full class diagram can be found on CD: Appendixes/Class Diagrams/FullClassDiagram.asta

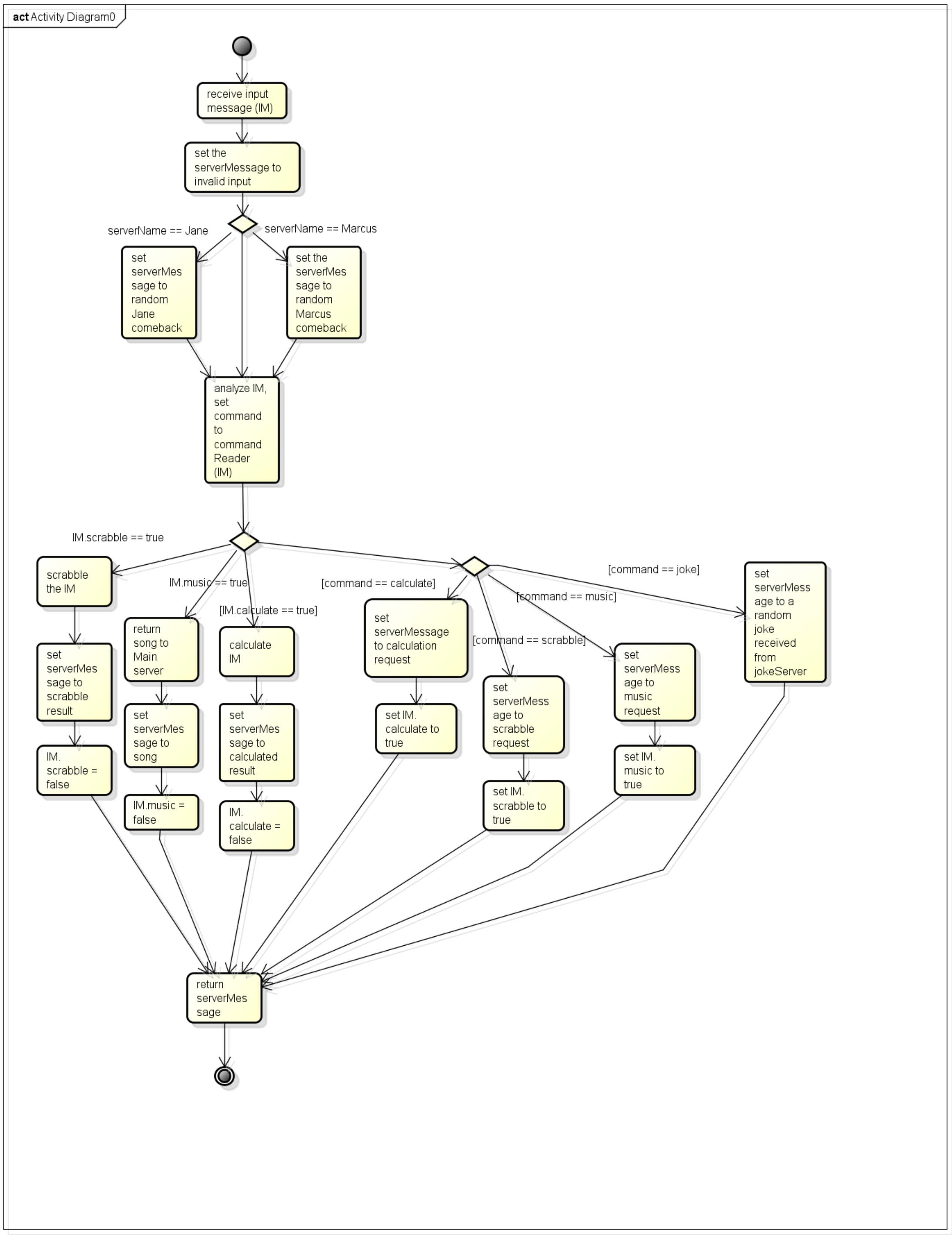
act Activity Diagram0
receive input message (IM)
set the serverMessage to invalid input
serverName == Jane
serverName == Marcus
set serverMessage to random Jane comeback
set the serverMessage to random Marcus comeback
analyze IM, set command to commandReader (IM)
IM.scrabble == true
IM.music == true
[IM.calculate == true]
[command == calculate]
[command == joke]
[command == music]
[command == scrabble]
scrabble the IM
set serverMessage to scrabble result
IM.scrabble = false
return song to Main server
set serverMessage to song
IM.music = false
calculate IM
set serverMessage to calculated result
IM.calculate = false
set serverMessage to calculation request
set IM.calculate to true
set serverMessage to scrabble request
set IM.scrabble to true
set serverMessage to music request
set IM.music to true
set serverMessage to a random joke received from jokeServer
return serverMessage

Figure XI - Activity Diagram for sendMessage() Method

The connection between the main server and each separate feature's server is designed to go through a separate client class that is kept on the main server. By doing this, the main server simply has a method for connecting to each feature, which starts the separate client and saves the result of this method call in an appropriate data holder. This way the amount of code in the MainServer class is kept at a minimum. The middleware between the main server and the separate servers for each feature is equally divided between RMI and CORBA. This decision was made to ensure that the requirements for the project were fulfilled. The RMI connections exist between the main server and the music server, and between the main server and the scrabble server. The CORBA connections exist between the main server and the jokes server, and between the main server and the calculator server.

The jokes server is designed as a very simple CORBA server, without naming service. This decision was made so that both CORBA servers can run on the same computer, at the same time. When the MainServer calls the getJoke() method from the JokesClient, the jokes server simply returns a random joke from an ArrayList<String> of jokes on the jokes server[6].

The calculator server is designed as a CORBA server that utilizes a naming service. Furthermore all the calculations done on the calculator server is done through a JNI library of files written in the C programming language. When the MainServer calls the calculate() method from the CalculatorClient, the input is sent to the calculateString() method in the ExpressionParser class on the calculator server. This calculateString method analyses the string and sends the separate calculations of that string to the appropriate methods in the C library (SUM, SUBTRACT, MULTIPLY, DIVIDE.) The result of the entire expression is then returned to the client. The flow of this sequence is depicted in the below sequence diagram, figure XII.

[6] See sequence diagram on CD: Appendixes/Activity Diagrams/JokesServer

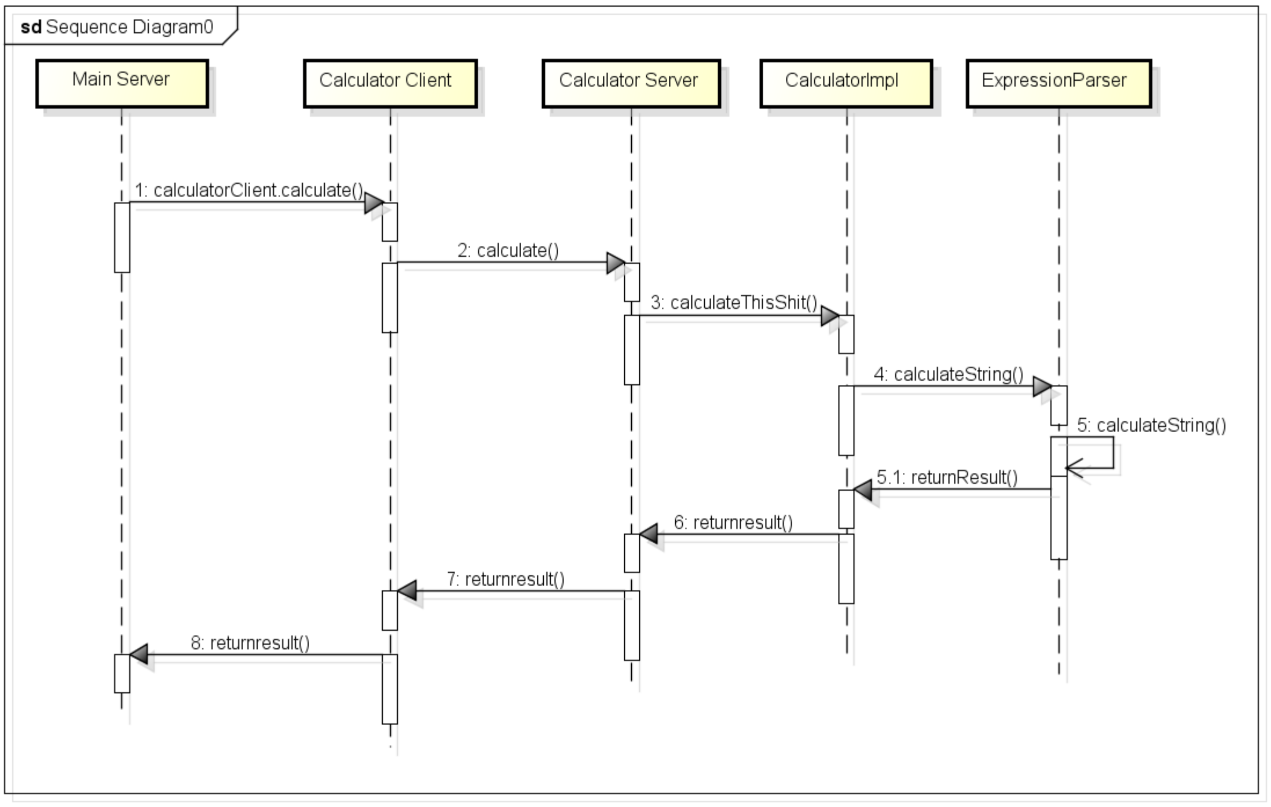

Figure XII - Sequence Diagram for Calculator Server

The music server is designed as an RMI server with a music player object that has an ArrayList<Track> of songs. This is done to simulate the playback/fetching of actual songs from the music server to the MainServer. The sequence of communication between the main server and the music server is depicted in the below sequence diagram, figure XIII.

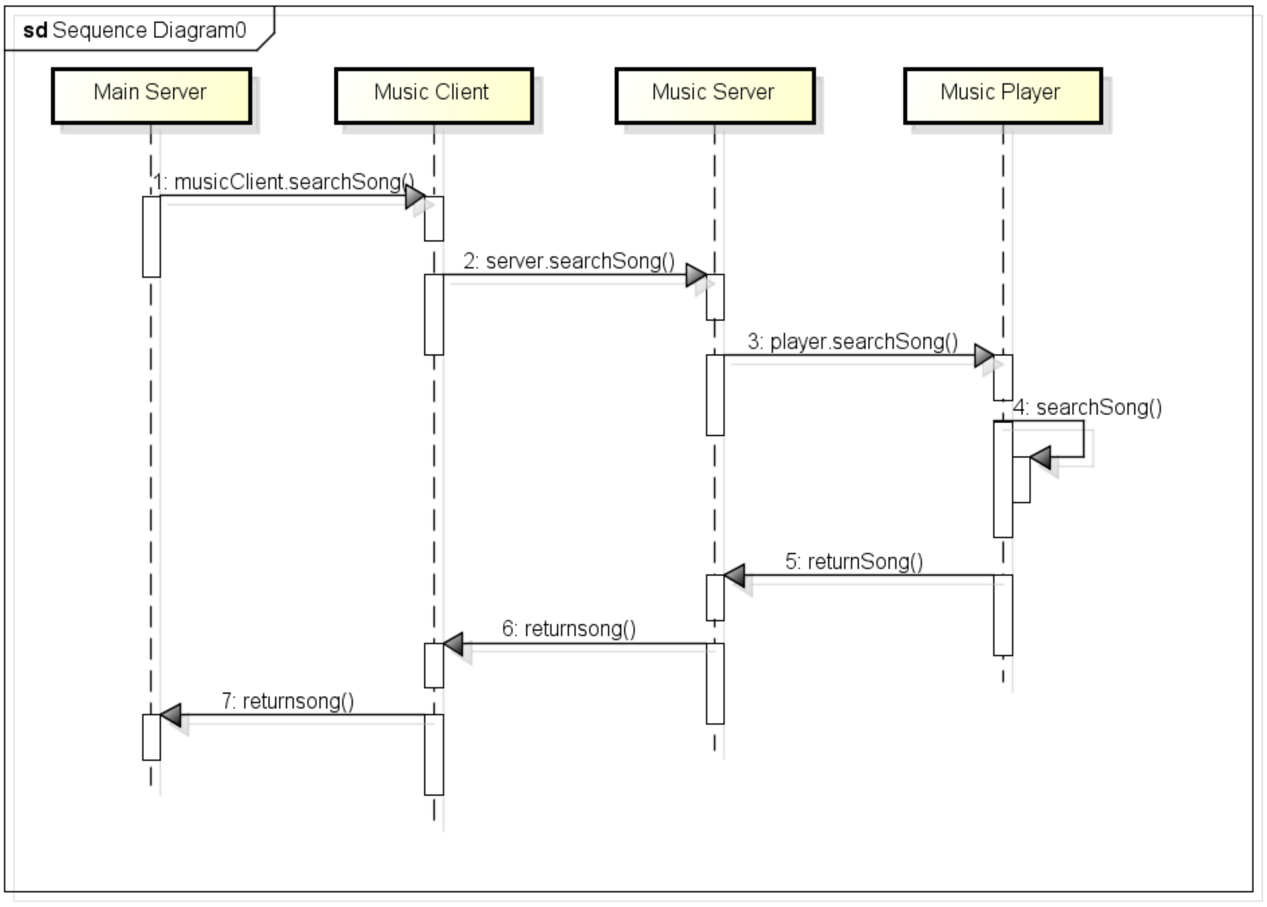

Figure XIII - Sequence Diagram for Music Server

The scrabble server is designed as an RMI server that uses the ScrabbleSolver class on the scrabble server to find all possible combinations of the letters the client wants to scrabble. These letters are then matched with a text document containing all the English letters. All searching is designed to be binary for maximum performance. The design of the ScrabbleSolver class is limited to solving eight letters[7]. One of the limitations with this design is that the implementation needs to be modified a bit, if it should be deployed on a different computer[8]. The flow of this sequence is depicted in the below sequence diagram, figure XIV.

[7] See the section on implementation.
[8] See the section on implementation.

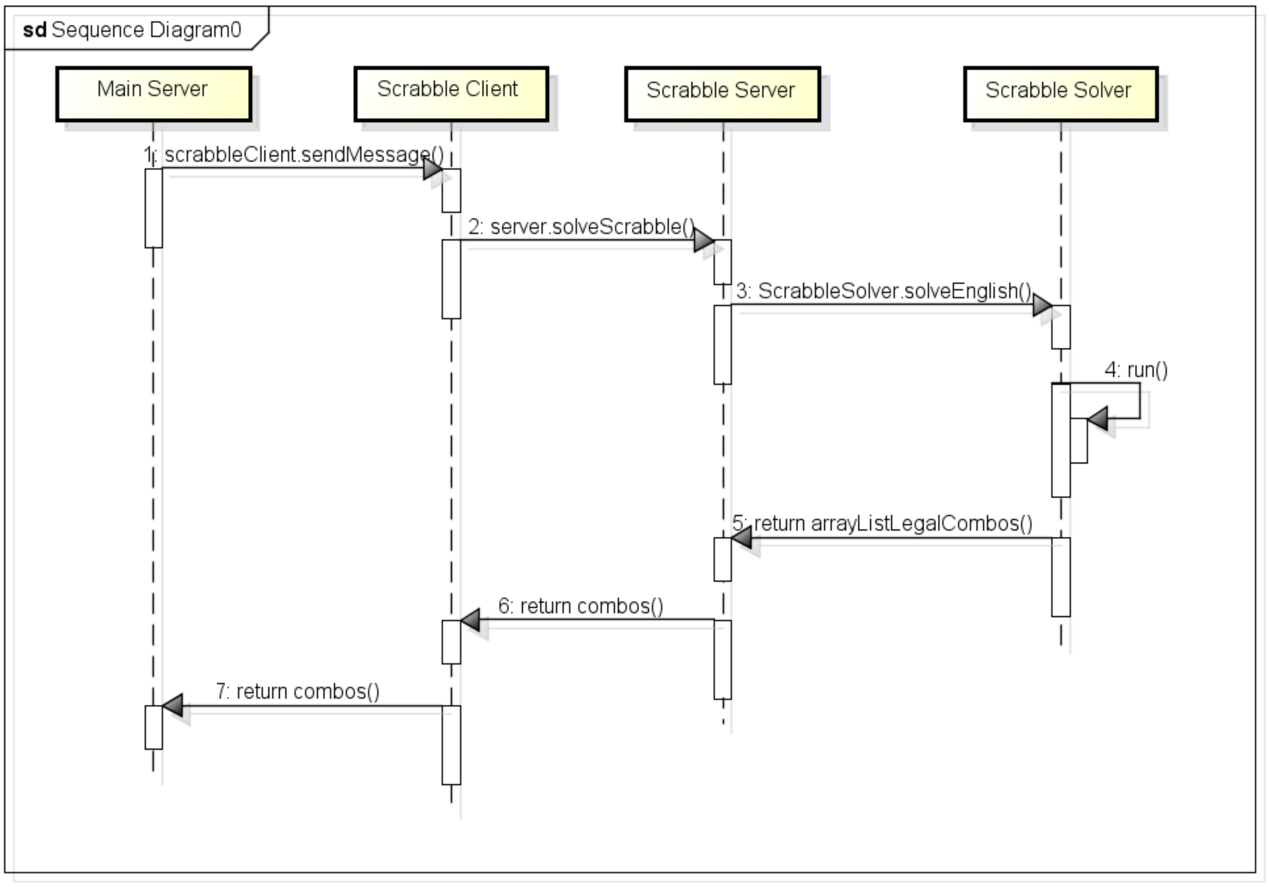
sd Sequence Diagram0
Main Server
Scrabble Client
Scrabble Server
Scrabble Solver
1: scrabbleClient.sendMessage()
2: server.solveScrabble()
3: ScrabbleSolver.solveEnglish()
4: run()
5: return arrayListLegalCombos()
6: return combos()
7: return combos()

Figure XIV - Sequence Diagram for Scrabble Server

The last important feature of the system is the intelligent string recognition class. This class analyzes all the messages from the client. It is designed with separate ArrayList<String>s for each command on the MainServer. The Reader class has a mathReader() method, which takes out all characters from the message that are not legal math chars; for example the message "9+lk9/.9" will be converted to "9+9/9." It also has a scrabbleReader() method which does the exact same thing, but only leaves letters in the message("9ku 98 jh" will be converted to "kujh".) It also has a serverNameReader() method, which will look at the message and check if it contains a keyword that would suggest that it was supposed to be a choice of server representative; for example "I wanna talk to my homie" will be converted to "Marcus" and "secretary" will be converted to "Jane." Similar to the serverNameReader() it also has a commandReader(), which will check for any keyword in the message that would suggest a specific command; for example "scrab" will be converted to "scrabble" and "tell me something funny" will be converted to "joke."

Implementation

The implementation part of this report will go through some important pieces of the code including parts of the intelligent string recognition class, the scrabble solver and parts of the implementation of the calculator in the C programming language.

The scrabbleReader() method in the intelligent string recognition class is implemented to take each char from the input string and search binary through an ArrayList<String> of illegal chars, if one is encountered the char is not added to the string that is returned. The implementation is shown in the below code snippet, figure XV.

```
private static String[] illegalStrings = {"!", "#", "$", "%", "&", "(", ")",
"*", "+", ",", "-", ".", "/",
            "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", ":", ";", "=",
"?", "@", "[", "]", "_", "{", "}"};

public static String scrabbleReader(String message)
      {
            boolean illegal = false;
            String letters = "";
            for(int i = 0; i < message.length(); i++)
            {
                  String letter = "";
                  letter += message.charAt(i);
                  if(message.charAt(i) == ' ')
                  {
                        illegal = true;
                  }
                  else if(Search.doubleGanger(illegalStrings, letter))
                  {
                        illegal = true;
                  }
                  if(!illegal)
                  {
                        letters += letter;
                  }
                  illegal = false;
            }
            return letters;
      }
public static boolean doubleGanger(String[] list, String word)
      {
            boolean found = false;
            int min, max, mid;
            min = 0;
            max = list.length - 1;
                  while (min <= max)
                  {
                        mid = (min + max) / 2;
                        if (list[mid].compareToIgnoreCase(word) == 0)
                        {
                              found = true;
                              break;
                        }
                        else if (list[mid].compareToIgnoreCase(word) < 0)
                        {
                              min = mid + 1;
                        }
                        else
                        {
                              max = mid - 1;
                        }
            }
                  return found;
      `
```

Figure XV - scrabbleReader() code snippet

The scrabbleSolver class on the scrabble server is implemented with a run method that first initializes the dictionaries from local text files into an ArrayList<String>, then it finds all the possible combinations of the input letters using backtracking without a guard for double gangers. It then searches for each of the combinations in each of the three dictionaries (Danish, English and the bible) using a binary search. Finally it cleans up the ArrayList<String> of legal combos by copying all legal combos into a new ArrayList<String> called legalCombosNoDoubles, exactly once. A code snippet of the backtracking algorithm and the clean up algorithm is shown in the below code snippet, figure XVI. As mentioned in the design part of the report, a few changes needs to be made to the code if deployed on a different computer. The part that needs to be changed is the directory address pointing to the local text documents containing the dictionaries and the bible. Furthermore, due to a memory problem with the ArrayList<String> after finding combos of more than eight letters, the algorithm was implemented to only take the first eight letters of the input string. The problem was that it would run out of heap space while finding all the combos of nine or more letters.

```java
private static void findCombos(String string)
        {
                int length = string.length();
                if(length > 8) length = 8;
                char[] result = new char[length];
                char[] stringArray = new char[length];
                boolean[] used = new boolean[length];
                for(int i = 0; i < length; i++)
                {
                        stringArray[i] = string.charAt(i);
                }

                findCombos(0, stringArray, used, result);
        }
        private static void findCombos(int position, char[] string, boolean[]
used, char[] result)
        {
                if(position == result.length)
                {
                        String combo = "";
                        combo += result[0];
                        for(int i = 1; i < result.length; i++)
                        {
                                combo += result[i];
                                allCombos.add(combo);
                        }
                }
                else
                {
                        for(int i = 0; i < used.length; i++)
                        {
                                if(!used[i])
                                {
                                        used[i] = true;
                                        result[position] = string[i];

                                        findCombos(position + 1, string, used, result);

                                        used[i] = false;
                                }
                        }
                }
        }
private static void cleanUp(ArrayList<String> list, ArrayList<String>
newList)
        {
                String[] array = new String[list.size()];
                list.toArray(array);
                Sorting.mergeSort(array, 0, array.length - 1);
                boolean doubleGanger = false;
                for(int i = 0; i < array.length; i++)
                {
                        list.add(i, array[i]);
                }
                for(int i = 1; i < list.size(); i++)
                {
                        doubleGanger = doubleGanger(newList, list.get(i));
                        if(!doubleGanger)
```

Figure XVI - ScrabbleSolver code snippet

The Calculator feature is implemented both in Java and C. The calculation algorithm itself is in Java, but it uses functions for calculation that are implemented in C. For this purpose JNI was used: the native methods that will be implemented in C were declared in a class and then a header file for these methods is generated, in order to connect the Java and C parts of the code. After that, the functionality of the methods, declared in the header file, are simply implemented in C programming language. Here is some code of the:

```c
#include <stdio.h>
#include "CalcFunc.h"

JNIEXPORT jdouble JNICALL Java_CalcFunc_sum(JNIEnv *env, jclass cls,
jdouble a, jdouble b)
{
	double sum = a + b;
	return sum;
}

JNIEXPORT jdouble JNICALL Java_CalcFunc_subtract(JNIEnv *env, jclass
cls, jdouble a, jdouble b)
{
	double subtract = a - b;
	return subtract;
}

JNIEXPORT jdouble JNICALL Java_CalcFunc_multiply(JNIEnv *env, jclass
cls, jdouble a, jdouble b)
{
	double multiply = a * b;
	return multiply;
}

JNIEXPORT jdouble JNICALL Java_CalcFunc_divide(JNIEnv *env, jclass cls,
jdouble a, jdouble b)
{
	double divide = a / b;
	return divide;
}
```

Figure XVII - C functions code snippet

And some parts of the algorithm in Java that uses the functions from C:

```
public double calculateString(String in)
   {

      char[] actions = getActions(in);
      double[] params = parseNumbers(in);
      double result = params[0];

      for (int i = 0; i < actions.length; i++)
      {
         char act = actions[i];
         if (isAction(act))
         {
            switch (act)
            {
               case '+':
                  result = CalcFunc.sum(result, params[i + 1]);
                  break;
               case '-':
                  result = CalcFunc.subtract(result, params[i + 1]);
                  break;
               case '/':
                  if (params[i + 1] != 0)
                     result = CalcFunc.divide(result, params[i + 1]);
                  break;
               case '*':
                  result = CalcFunc.multiply(result, params[i + 1]);
                  break;
            }

         }
      }
      return result;
   }
```

Figure XVIII - Calculation algorithm code snippet

The calculator feature was not implemented using the MVC design pattern, because the team encountered problems when implementing the JNI part and the certain utility's implementation was left in a separate project folder.

Test

Unit testing

Unit testing of software applications is done during the development (coding) of an application.

The objective of unit testing is to isolate a section of code and verify its correctness. In procedural programming a unit may be an individual function or procedure

The goal of unit testing is to isolate each part of the program and show that the individual parts are correct. Unit testing is usually performed by the developer.

Under the automated approach-

- A developer could write another section of code in the application just to test the function. They would later comment out and finally remove the test code when the application is done.

- They could also isolate the function to test it more rigorously. This is a more thorough unit testing practice that involves copy and pasting the function to its own testing environment to other than its natural environment. Isolating the code helps in revealing unnecessary dependencies between the code being tested and other units or data spaces in the product. These dependencies can then be eliminated.

A coder may use a Unit Test Framework to develop automated test cases. Using an automation framework, the developer codes criteria into the test to verify the correctness of the unit. During execution of the test cases, the framework logs those that fail any criterion.

Throughout the project we carried out unit testing to verify that most significant parts of code implementation are working correctly in isolation.

Test of C functions

```
JNIEXPORT jdouble JNICALL Java_CalcFunc_sum(JNIEnv *env, jclass cls, jdouble a,
jdouble b)
{
	double sum = a + b;
	return sum;
}
```

Was tested trough Eclipse, because it’s JNI, but not a pure C code. It’s a simple function. I had no problems with it. There was a little frustration about generating the header file from the native methods class, when we use packages, but the problem was solved. All the results were as expected.

Use Case Testing

Use cases tell the story of how someone interacts with a software system to achieve a goal either successfully or abandoning the goal. The use case describes multiple paths that the user can follow within the use case. Each path that can be followed within the use case is a use case scenario. Each use case follows with one or more scenarios.

Use Case Name	Get Jokes
Context Of Use	Client wants to read a joke
Scope	System
Level	Primary Task
Primary Actor	Client
Preconditions	Clients is connected to server, screen names is selected, representative has been chosen.
Success End Conditions	Joke is returned to client UI
Failed End Conditions	Jokes is not returned
Trigger	Client types in recognizable word or Jokes button

	is pressed	
Basic Flow	Steps	Action
	B1	Client types command or clicks Joke button
	B2	System returns jokes
Alternative Flow	Steps	Action
	A1	Client types unrecognizable command
	A2	System displays “Invalid input”
	A3	Client types recognizable command
	A4	System returns jokes

Figure XIX - Use case test table for Jokes feature

Use Case testing was done manually following Basic and alternative flows, what represents different scenarios.
For each scenario each step was tested multiple times. During the test, where the system is interacting with variables, test was done with different values.
All scenarios were successfully tested and proven to work as expected.

In order to see the full testing documentation of the system – refer to the Testing Document in the appendixes[9].

Results

The project turned out well. The results are as expected – the whole system works properly, all the needed methodologies were followed and the team work was on a high level.
The study helped to find a way to connect multiple client-server systems, using different middleware, into one whole system; to construct a program consisting of two or more programming languages; to develop various algorithms. All the work that was conducted really contributed in hardening the gained knowledge and expanding it further by discovering new features and possibilities, concerning the development of distributed systems, team work, time and plan management.

[9] Full documentation of the system testing process:
Appendixes/TestDocumentation/TestDocumentation – MS Word.docx

Discussion

The main focus of the project was to create a distributed system with multiple features installed on different servers, with various middleware, that can simulate famous and massive-used features among the Internet (like Google calculator or CleverBot) but with the very better detail that all these features will be on one place, easy to access just in seconds, unlike the above stated utilities, that are spread among different websites in the world wide net.

As stated in the Results section, the calculator works fine, but not exactly as initially intended. At first it was planned that the utility should be able to solve complicated equations like "(9-4)*((56/7) +3)" but the team encountered problems with coming up with a working algorithm and it was concluded as working just with the four general mathematical functions. Now the feature is able to solve an equation like "9*5+4 - 2.6" as it also recognizes the fractional numbers. Further development of the algorithm may continue in the future.

The scrabble feature works as initially intended, and the team did not encounter any problem implementing it. Though, as mentioned earlier in the report, this feature has a memory limitation that limits it to solve only a maximum of eight letters as input.

Conclusion

In conclusion the system was implemented successfully without any major bugs that comprise the integrity of the system. Furthermore, the system runs smoothly, without any errors, when testing it using the use case descriptions. Also, the few minor problems that are present, does not affect the system as a whole; for example, the memory problem with the scrabble, is in fact not a problem, since it is unusual to have more than seven tiles in a game of scrabble. Likewise, the limitation of the calculator does not turn out to be much of a problem, since it is only limited by not being able to solve mathematical expressions containing parentheses. All in all, the outcome of this project was a successful and well-working distributed system, utilizing JNI and the C programming language, along with two different types of middleware.

References

[1] CleverBot, *http://cleverbot.com*, December 2012

[2] Google, *www.google.com*, December 2012

Appendices

All appendices can be found in the Appendix folder on the CD.
This folder includes:

- Test descriptions
- UML diagrams
- Use case descriptions
- User guide
- Diary
- Source code
- Scrum workbook
- Daily scrum notes