ENG 3050 – Software Engineering Design I

Winter 2015

Artificial Intelligence Literature Searching

Assignment 2 – Software Specification

Group #3

Contributing Members:

John Simko, Brandon Stanley, Shahood Mirza

Date Submitted: 2/9/2015

Approval

This document has been read and approved by the following team members responsible for its implementation:

Print Name	Signature	Comments

Print Name	Signature	Comments

Print Name	Signature	Comments

Revision History

Date	Author	Comments
Feb 03	Brandon Stanley	Created specification document
Feb 06	Brandon Stanley	Laid out document framework, added use case diagram, created table of contents, performed COCOMO estimation and calculation, added GUI from previous document
Feb 06	John Simko	Assisted with creation of class diagram, developed rough outline for communication protocol, created rough sequence diagram.
Feb 08	Brandon Stanley	Created, modified, and added class diagrams. Created all use case descriptions. Added to risks and risk mitigation.
Feb 08	John Simko	Wrote Introduction, created UML sequence diagram.
Feb 08	Shahood Mirza	Added project plan, roles and gantt chart
Feb 09	John Simko	Added CRC cards as Software Architecture
Feb 09	Brandon Stanley	Fixed missing use-cases, fixed missing methods in class diagram, build communications protocol section.
Feb 09	Shahood Mirza	Added GUI and database models
Feb 15	Brandon Stanley	Updated Intermediate COCOMO, made mention of threading to Login Use Case, fixed confusing class names, and mistyped multiplicity, added the word “program” to the end of all references to Client and Server, made mention of input of IP and port number to connect to the server.
Feb 16	Brandon Stanley	Updated GUI in Client Class Diagram, added admin-user based client/server communication codes, added login sequence diagram
Feb 27	John Simko	Updated Introduction and CRC Cards to reflect changes to class diagram. Reorganized sections to be more understandable. Corrected typos.
Mar 1	John Simko	added Server side function response code for indicating successful admin login and for log out
March 2	Brandon Stanley	Re-did GUI to more accurately reflect product, minor changes to client diagram., added to CRC cards

Table of Contents

1. Introduction

This document outlines the specifications of a client-server system for searching and sorting through a database of resources. The resources consist of articles and standards relating to the field of artificial intelligence.

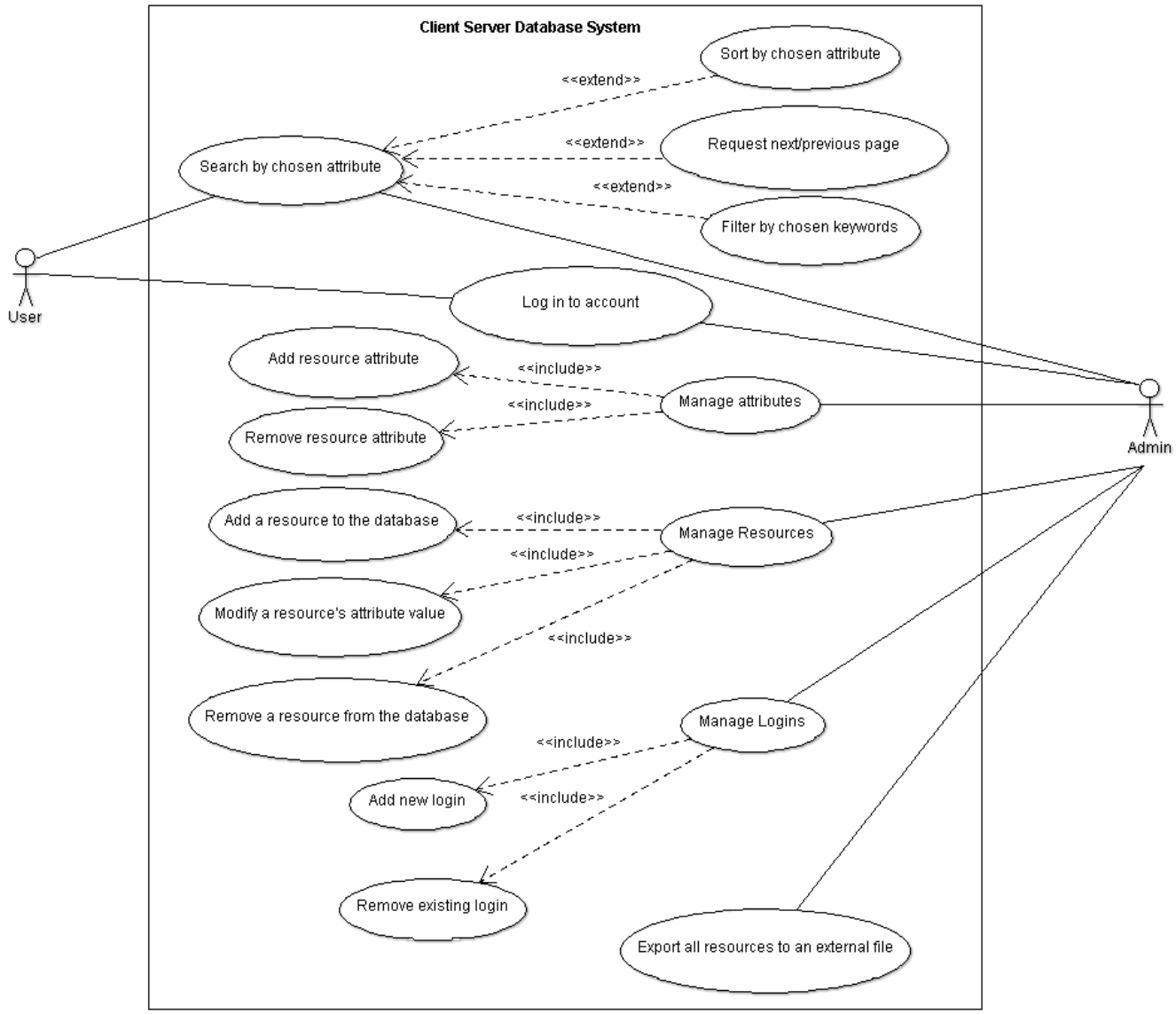
The database will be stored on the server in CSV format, and imported into a C++ vector format upon startup of the server program. Once the server is prepared for requests, it will listen for TCP socket connection requests. Upon receiving a connection request, a client connection object is created at the server to manage all data transfers with that instance of the client.

The client program displays a GUI for a login screen on startup, and sends a socket connection request once the user has entered a username and password and clicked the “login” button. Upon successful login, the user is given a GUI search window from which they are able to search the server database for any literature contained within it. Users are always able to search resources by Title, Author, Date of Publication, and Keyword. Users will also be able to sort by each of those, as well as sorting by a hidden relevance value generated by the search.

The server’s client connection object handles search requests by calling the servers search function, which returns a vector of references to resources in the database. This vector also has a relevance number matched to each reference, representing how likely that resource is relevant to the users search. Client connection sends a specific amount of results to the user (called a “page”), and stores which page was last sent. If the client selects a filter, the client connection generates a vector subset of resource references (modifiedSearch) which only has results containing the filtered keywords.

2. Product Specification
2.1 Use Case Model

2.1.1 Use Case Diagram



2.1.2 Use Case Description

Brief Description

In the *search by chosen attribute* use case either a user or admin requests the server side database to search for resources based on a given attribute and a given search string. The server program returns a page of resources, and the user or admin can request other pages at any given time. The user or admin can then filter, or sort the results by keywords and attributes respectively.

Step by step description

1. User/admin searches by a chosen attribute.

1.1. An attribute is selected from a drop down menu of currently existing attributes, and a search phrase is entered to the search text field. Attributes searchable include:

- Title
- Author
- Date of publication
- Keywords
- Abstract
- Any further attributes added by an admin

1.2. Search details are sent from the client program to the server program.

1.3. Server program parses the search request, stores a copy of the results tied to that particular client program in a Server_Side_Connection_To_Client object that runs on its own thread, and returns a page of the results to the client program.

1.4. Client program receives a page of searched results, generates a list of filter keywords, and displays both the search results and the filter words in the GUI.

2. User/admin requests next or previous page.

2.1. The user/admin presses the next or previous button, the client program sends a request to the server program for another page.

2.2. The server program generates the next set of pages, and sends the resources to the client program.

2.3. The client program displays the new page of results.

3. The user/admin sorts by attribute

3.1. User/admin selects an attribute to sort by, and whether it is ascending or descending.

3.2. Client program sends this information to the server.

3.3. The server program parses this information.

3.4. The server program sorts by given requirements, returns the first page of sorted resources to client.

3.5. The client program displays returned list of resources.

4. The user/admin filters by keywords.

4.1. The user/admin selects one or more keywords to filter results by.

4.2. The client program sends this information to the server.

4.3. The server program parses this information, and filters the results based on the given requirements.

4.4. The server program returns the first page of the new set of results to the client.

4.5. The client program displays the returned list of resources.

Brief Description

In the *manage attributes* use case an admin adds or removes an attribute to all current and future resources.

Set by step description

1. Admin adds a new resource attribute

1.1. An admin enters the name of a new attribute, and default value for all current resources this attribute is being added to (can be an empty string). This information is sent to the server program.

1.2. The server program adds the attribute

1.2.1. The server program parse the string, getting the attribute's name and default.

1.2.2. The server program sets two mutex locks, one for accessing the resources for all other users and admins, and one for modifying the local database files.

1.2.3. The server program then adds the new attribute and the default value to all current resources, adds the new attribute to the list of attributes

1.2.4. The server program then releases the mutex lock on the resources.

1.2.5. The server program then updates the local database files.

1.2.6. The server program releases the mutex lock on local database files.

1.2.7. The server program sends a confirmation that the attribute has been added.

1.3. The client program displays to the admin if the attribute has been successfully added or not.

2. Admin removes an existing resource attribute

2.1. An admin enters the name of a new attribute, and default value for all current resources this attribute is being added to (can be an empty string). This information is sent to the server program.

2.2. The server program removes the attribute

2.2.1. The server program checks to see that the attribute to be removed exists, and if it does it is not protected. Protected attributes are:

- Title
- Author
- Keywords
- Abstract

2.2.1.1. If the attribute is write protected, an error message signifying it is write protected is returned to the client program.

2.2.1.2. If the attribute does not exist, an error message signifying it does not exist is returned to the client program.

2.2.1.3. If the attribute exists and is not write protected, continue with use case description.

2.2.2. The server program sets two mutex locks, one for accessing the resources for all other users and admins, and one for modifying the local database files.

2.2.3. The server program then removes the attribute from all current resources, and removes the attribute to the list of attributes.

2.2.4. The server program then releases the mutex lock on the resources.

2.2.5. The server program then updates the local database files.

2.2.6. The server program releases the mutex lock on local database files.

2.2.7. The server program sends a confirmation that the attribute has been removed..

2.3. The client program displays to the admin if the attribute has been successfully added or not.

Brief Description

In the *manage resources* use case an admin adds, removes, or modifies a resource in the database.

Set by step description

1. Admin adds a resource to the database.
1.1. Admin enters the information on the new resource, and clicks the send button.
1.2. The client program converts the resource to a string, and sends it to the server program.
1.3. The server program creates a resource object using this data it receives.
1.4. The server program adds the new resource to the list of resources.
1.5. The server program sets a mutex lock on the database resource files, adds the new resource and then releases the mutex lock.
1.6. The server program sends a pass/fail message to the client.
1.7. The client program displays the pass/fail message.
2. Admin modifies a resource in the database.
2.1. The admin selects the resource to modify, makes changes, and presses the send button.
2.2. The client program converts the resource to a string, and sends it to the server program.
2.3. The server program looks up the resource to modify by using its resource ID number.
2.4. The server program makes the changes to the resource.
2.5. The server program sets a mutex lock on the database resource files, modifies the resource and then releases the mutex lock.
2.6. The server program sends a pass/fail message to the client.
2.7. The client program displays the pass/fail message.
3. Admin removes a resource in the database.
3.1. The admin selects the resource to remove
3.2. The client program converts the resource to a string, and sends it to the server.
3.3. The server program looks up the resource to remove by using its resource ID number.
3.4. The server program removes the resource from the list of resources.
3.5. The server program sets a lock on the database resource files, removes the resource and then releases the mutex lock.
3.6. The server program sends a pass/fail message to the client program.
3.7. The client program displays the pass/fail message.

Brief Description

In the *export all resources to an external file* use case an admin creates a local copy of the entire database to a local file on their system.

Set by step description

1. Admin exports all resources to an external file
1.1. The admin sends a request to the server program for all resources
1.2. The server program sends all resources to the client program.
1.3. The client program pipes all the resources to a local file.

Brief Description

In the *log in to account* use case a user or admin logs in to the server program through the client program protocol.

Set by step description

1. User or admin attempts to log into their account
1.1. User or admin enters username and password, enters the server's IP and port #, and presses the send button. The client program sends this information to the server program.
1.2. The server receives the login information, looks up the user information, and compares the provided password against the stored password.
1.3. The server creates a new Server_Side_Connection_To_Client object, and runs it in a new thread.
1.4. The server program sends the client program a validation code of
    0: Failed
    1: User
    2: Admin
1.5. The client program receives the validation code, and spawns the main GUI for searching.

Brief Description

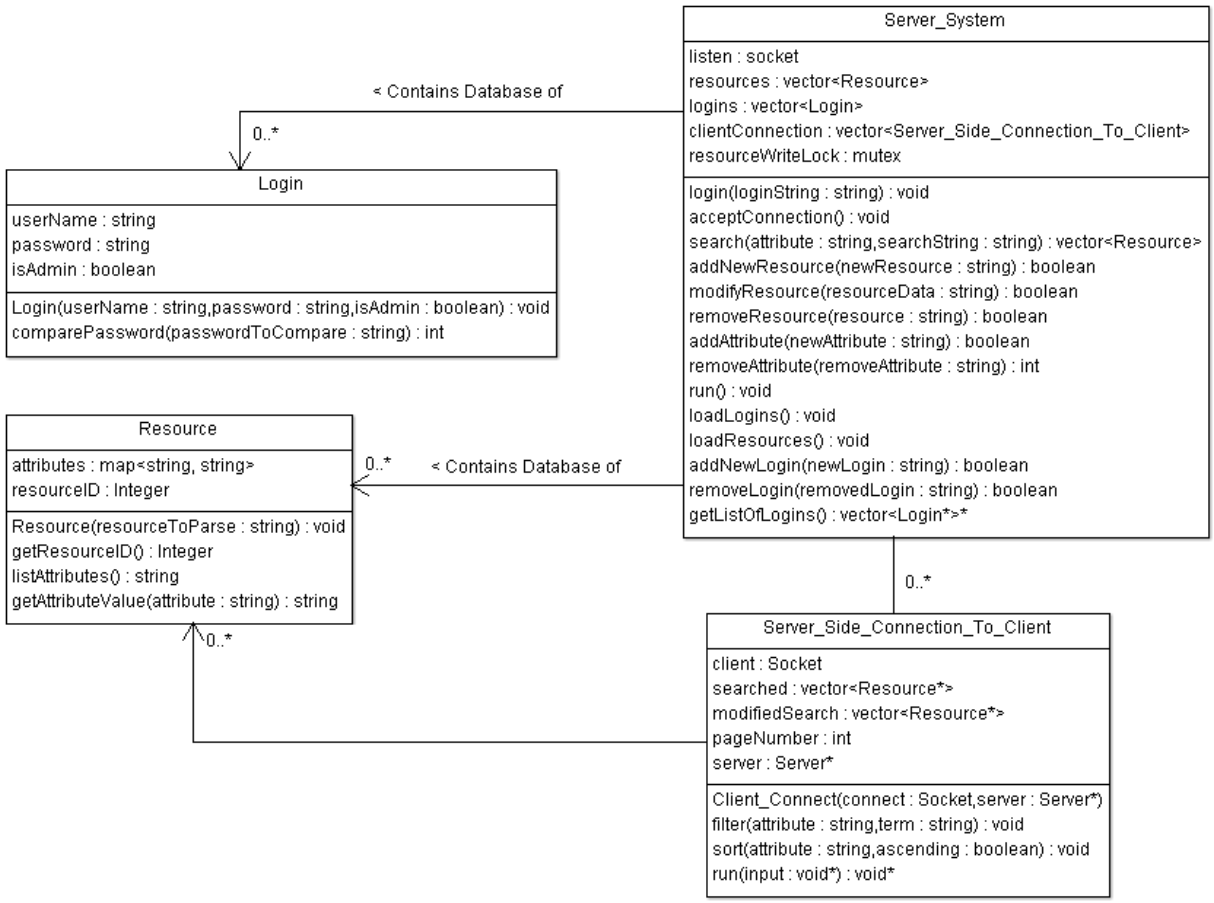
In the *manage logins* use case an admin adds a new login or removes an existing login from the database.

Set by step description

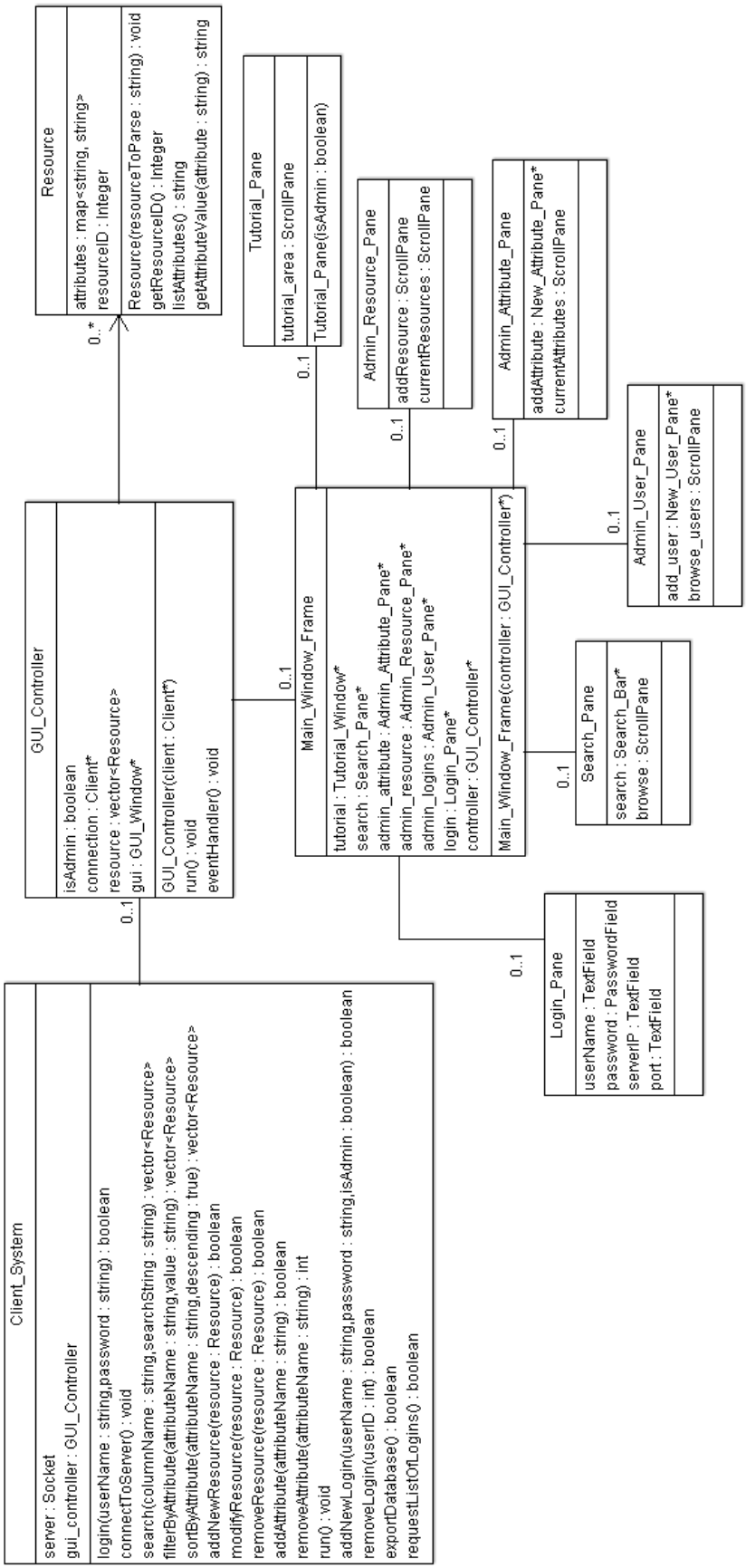
1. Admin adds a new login.
1.1. Admin enters new login information and clicks send
1.2. Client program sends information to the server program for processing.
1.3. Server program parses information
1.4. Server program adds login to vector of logins.
1.5. Server program sets a mutex lock on the login database file
1.6. Server program adds new login account to login database file.
1.7. Server program releases mutex lock on login database file.
1.8. Server program returns a pass/fail message to the client program.
1.9. Client program displays pass/fail message to the admin.
2. Admin removes an existing login.
2.1. Admin selects and existing login and clicks the delete button.
2.2. Client program sends information to the server for processing.
2.3. Server program parses information
2.4. Server program removes login from vector of logins.
2.5. Server program sets a mutex lock on the login database file
2.6. Server program removes login account from login database file.
2.7. Server program releases mutex lock on login database file.
2.8. Server program returns a pass/fail message to the clien programt.
2.9. Client program displays pass/fail message to the admin

2.2 UML Class Diagrams

2.2.1 Server Class Diagram



2.2.2 Client Class Diagram



2.3 Software Architecture

2.3.1 Server Side CRC Cards

CLASS **Server_System**
RESPONSIBILITY 1. Listen for new connections from listen socket 2. Create **Server_Side_Connection_To_Client** in its own thread. 3. Import resources table 4. Import user logins table 5. Search resources vector for search matches 6. Add or Remove **Resource** 7. Modify existing **Resource** 8. Send login information to **Login** 9. Add and remove valid logins 10. Mutex lock resources/logins when being managed
COLLABORATION 1. **Server_Side_Connection_To_Client** 2. **Resource** 3. **Login** 4. **Client_System**

CLASS **Server_Side_Connection_To_Client**
RESPONSIBILITY 1. Hold connection to **Client_System** running in its own thread. 2. Store vector of **Resource** search results 3. Send search/sort/filter results over socket to **Client_System** 4. Parse requests from **Client_System** 5. Send search requests to **Server_System** 6. Create vector subset on filtered keyword(s) 7. Send login information to **Server_System** 8. Send add/remove resource requests to **Server_System** 9. Send add/remove login requests to **Server_System**
COLLABORATION 1. **Server_System** 2. **Client_System** 3. **Resource**

CLASS **Login**
RESPONSIBILITY 1. Verify login credentials are valid 2. Return login request results 3. Identify client as user or admin
COLLABORATION 1. **Server_System**

CLASS **Resource**
RESPONSIBILITY 1. Store a resource 2. Return list of current resource attributes 3. Return resource's specified attribute 4. Return ID of selected resource
COLLABORATION 1. **Server_System** 2. **Server_Side_Connection_To_Client** 3. **GUI Controller**

2.3.2 Client Side CRC Cards

CLASS **Client_System**
RESPONSIBILITY 1. Connect to the **Server_System** 2. Send login information over server socket to **Server_System** 3. Send search information over server socket 4. Send filter information over server socket 5. Send sorting information over server socket 6. Send add/remove resource information over server socket 7. Send modify resource information over server socket 8. Send add/remove attribute information over server socket 9. Send add/remove login information over server socket 10. Send export database request over server socket 11. Send request for a list of usernames over server socket
COLLABORATION 1. **Server_System** 2. **GUI_Controller**

CLASS **GUI_Controller**
RESPONSIBILITY 1. Parse **GUI window** inputs 2. Send input data with relevant function code to **Client_System** 3. Display **Login Screen** on startup 4. After successful login, display **Main Screen** 5. If isAdmin true, display button to access **Admin Screen**
COLLABORATION 1. **Client_System** 2. **GUI_Window** **3. Resource**

CLASS **Main_Window_Frame**
RESPONSIBILITY 1. Spawn **Login_Pane**, **Tutorial_Pane**, and **Search_Pane** 2. If user is admin, also spawn **Admin_User_Pane**, **Admin_Attribute_Pane**, and **Admin_Resource_Pane** 3. Send user input to GUI Controller
COLLABORATION 1. **GUI_Controller** 2. **Login_Pane** 3. **Search_Pane** 4. **Tutorial_Pane** 5. **Admin_User_Pane** 6. **Admin_Attribute_Pane** 7. **Admin_Resource_Pane**

CLASS **Tutorial_Pane**
RESPONSIBILITY 1. Display simple text instructions for how to use the client to search, sort, filter, get other pages, etc.
COLLABORATION 1. **Main_Window_Frame**

CLASS **Search_Pane**
RESPONSIBILITY 1. Show user input box for search string. 2. Show selectable options for what field to search by 3. Show page of search results 4. Show buttons for next page and previous page 5. Show list of valid keywords to filter results by 6. Indicate which field search results are being sorted by
COLLABORATION 1. **Main_Window_Frame**

CLASS **Login_Pane**
RESPONSIBILITY 1. Show user input box for username, password, and the desired servers IP address and port
COLLABORATION 1. **Main_Window_Frame**

CLASS **Admin_User_Pane**
RESPONSIBILITY 1. Add new users to the database 2. Remove users from the database
COLLABORATION **1. GUI_Controller** **2. Search_Pane** **3. Main_Window_Frame**

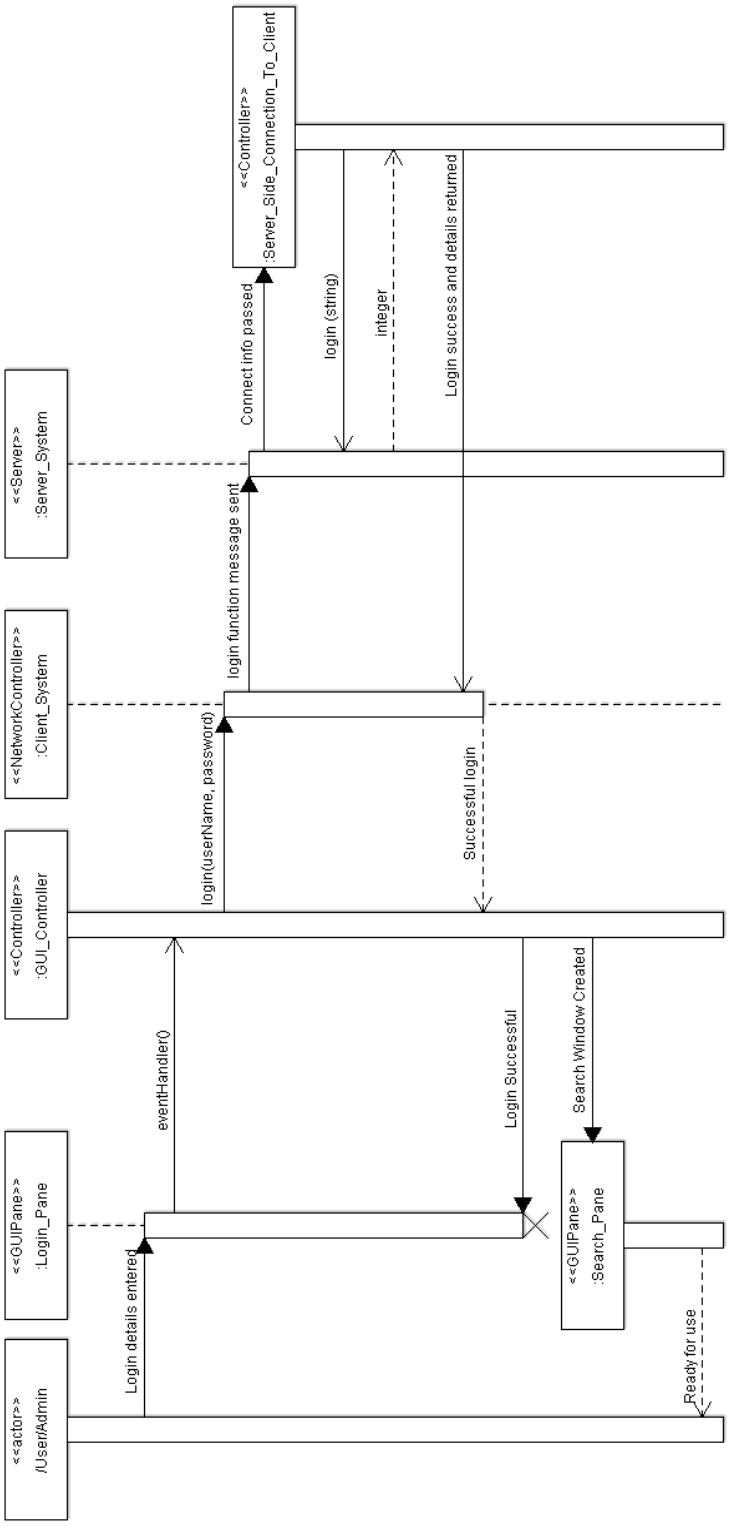
CLASS **Admin_Resource_Pane**
RESPONSIBILITY 1. Add new resources to the database 2. Modify existing resources. 3. Delete existing resources.
COLLABORATION **1. GUI_Controller** **2. Main_Window_Frame**

CLASS **Admin_Attribute_Pane**
RESPONSIBILITY 1. Add new attributes to the database 2. Remove attributes from the database
COLLABORATION 1. **GUI_Controller** **2. Main_Window_Frame**

2.4 Sequence Diagrams

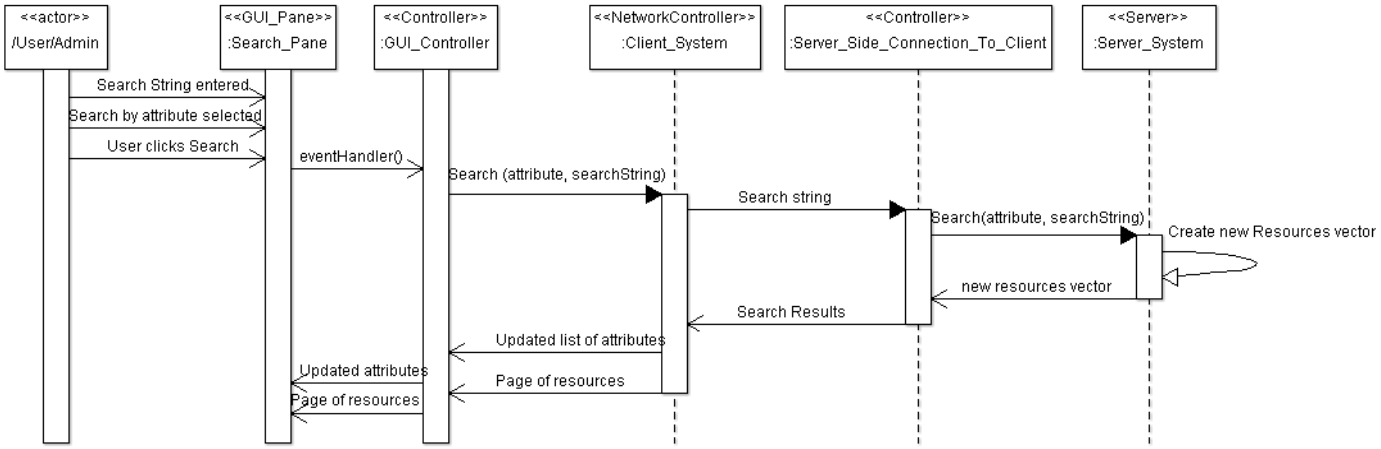
2.4.1. Login sequence diagram

This diagrams shows the login process of a user to gain access to the database. A user types their login information into the client program's GUI, presses the submit button, and that data is sent to the server program. The server spawns a Server_Connection_To_Client object, and logs the user in. The server program then sends a confirmation and all relevant login information to the client program.



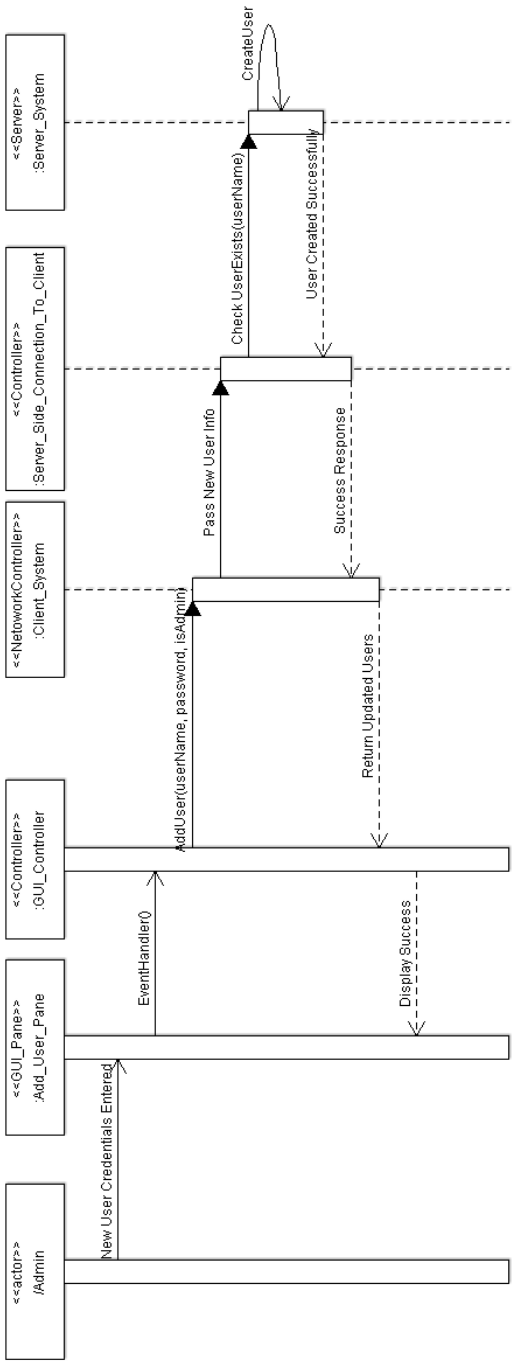
2.4.2. Search Sequence Diagram

This diagram shows a logged in user performing a search by some attribute. The user types in their search string, selects the attribute they want to search by, and clicks the search button. This gets caught by the GUI_Controller's event handler, and sends the search string and attribute to the Client_System. Client_System converts the search information to match the communication protocol, appending the search control code to the front. The Server_Side_Connection_To_Client checks the control code, and calls the Server_System Search function with the search string. A new vector of resources is created and returned to Server_Side_Connection_To_Client, which sends the first page of results back to the Client_System along with the current list of attributes. Client_System restores data into resources vector and sends to GUI_Controller to display.


<<actor>>
/User/Admin
<<GUI_Pane>>
:Search_Pane
<<Controller>>
:GUI_Controller
<<NetworkController>>
:Client_System
<<Controller>>
:Server_Side_Connection_To_Client
<<Server>>
:Server_System
Search String entered
Search by attribute selected
User clicks Search
eventHandler()
Search (attribute, searchString)
Search string
Search(attribute, searchString)
Create new Resources vector
new resources vector
Search Results
Updated list of attributes
Updated attributes
Page of resources
Page of resources

2.4.3. Create User Sequence Diagram

The diagram shows the creation process of a new user. As the credentials are added and submitted to the server via the GUI, a connection is created with the server and the data is submitted. The server checks to ensure the user does not already exist, and if not creates the new user. Finally, it sends a success response to the client and updates the user table.



2.5 Communication Protocols

All numbers represent a number of bytes.

2.5.1. Client sends function code and logic

2.5.1.0. General Template

Function logic where fields are of unknown length are comma separated lists of data. Data is transferred through the socket until the end of data is sent. Function logic is built up in a string before being passed to the appropriate method to parse the logical string.

1	**2**	**…**	**n**
Function code	Function logic		

2.5.1.1. Send current list of attributes

1	**n/a**
0x00	Blank

2.5.1.2 Next page

1	**n/a**
0x01	Blank

2.5.1.3 Previous page

1	**n/a**
0x02	Blank

2.5.1.4. Search function

1	**2**	**…**	**n**
0x03	Attribute, search string		

2.5.1.5. Sort function

1	**2**	**…**	**n**
0x04	Up/down, Attribute to sort by		

2.5.1.6. Filter function

1	**2**	**…**	**n**

0x05	Keyword

2.5.1.7. Add attribute function

1	**2**	**…**	**n**
0x06	Attribute name, default value		

2.5.1.8. Remove attribute function

1	**2**	**…**	**n**
0x07	Attribute name		

2.5.1.9. Add new resource function

1	**2**	**…**	**n**
0x08	Attribute1, attribute2, …, attributeN		

2.5.1.10. Modify existing resource function

1	**2**	**…**	**n**
0x09	Attribute 1, attribute 2, …, attribute n		

2.5.1.11. Remove existing resource function

1	**2**	**…**	**n**
0x0A	Attribute ID		

2.5.1.12. Request existing users function

1	**n/a**
0x0B	Blank

2.5.1.12. Add new user function

1	**2** … **n**
0x0C	Admin status, Username, password

2.5.1.13. Remove existing user function

1	**2** … **n**
0x0D	Username

2.5.1.14. Login function

1	**2** … **n**
0x0E	Username, password

2.5.2. Server attribute response

The server will respond to an attributes request by sending a comma separated version of the current list of attributes

1 … **n**
Attribute 1, Attribute 2, Attribute 3, …, Attribute n

2.5.3. Search, filter, sort, next page, and previous page response

The server will respond with a list of resources which are separated by braces, and within each resource is the list of attributes which are comma separated. A ‘page’ of up to 10 resources will be transferred at a time.

1 … **n**
{Rsc1Att1, …, Rsc1Attn} … {Rsc10Att1, …, Rsc10Attn}

2.5.4. Server side function response code

This is the response code the server sends whenever the client program requests a server side method. These methods include: add new resource, modify existing resource, remove existing resource, add new attribute, remove existing attribute, add new login, remove existing login, and login.

1	**…**	**2**
{0:fail}/{1: pass}/{2: write protected}/{3: admin login pass}/{9:close connection and logout}		

2.5.4. Request for existing users response

This is the server's response to a request for existing users. A comma separated list of usernames is sent to the client for the admin to view.

1	**…**	**n**
Username1, Username2, …, Usernamen		

2.6 Interfaces

2.6.1 Graphical User Interfaces

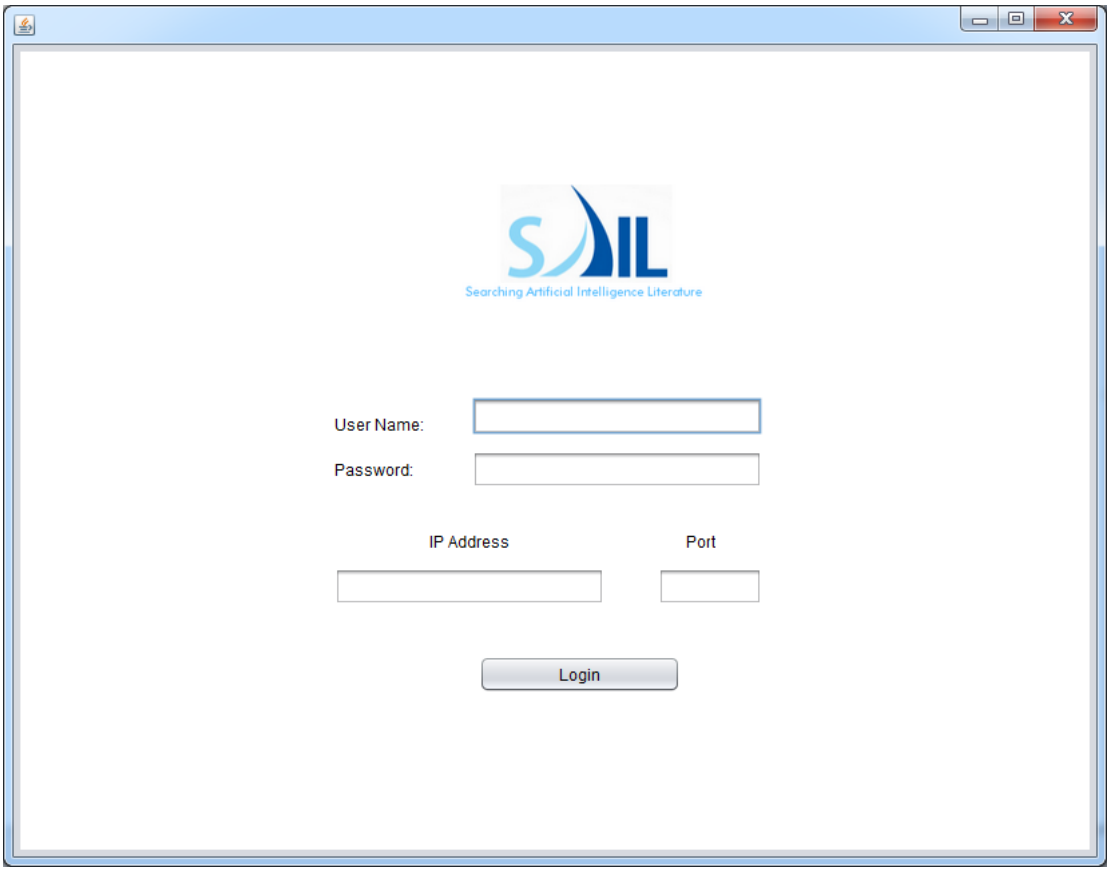


Figure1: Login screen

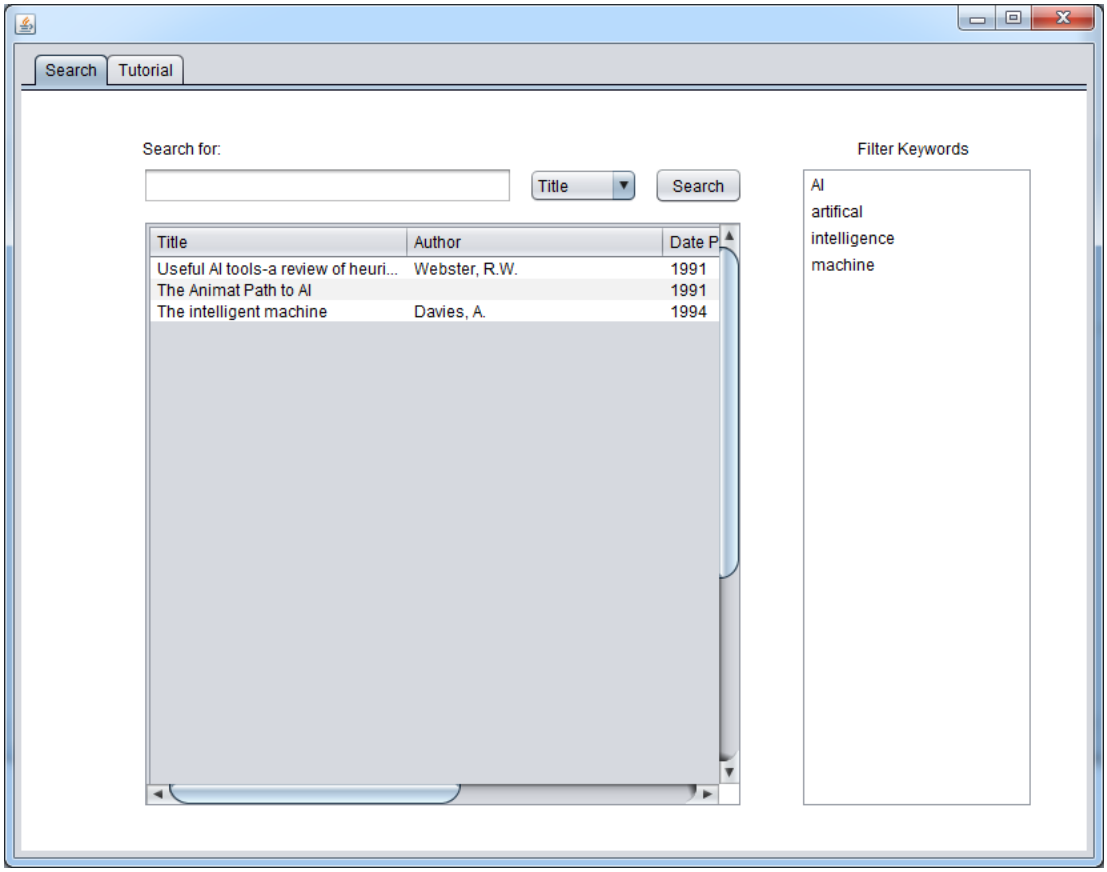


Figure 2: Search Window (Common to both users)

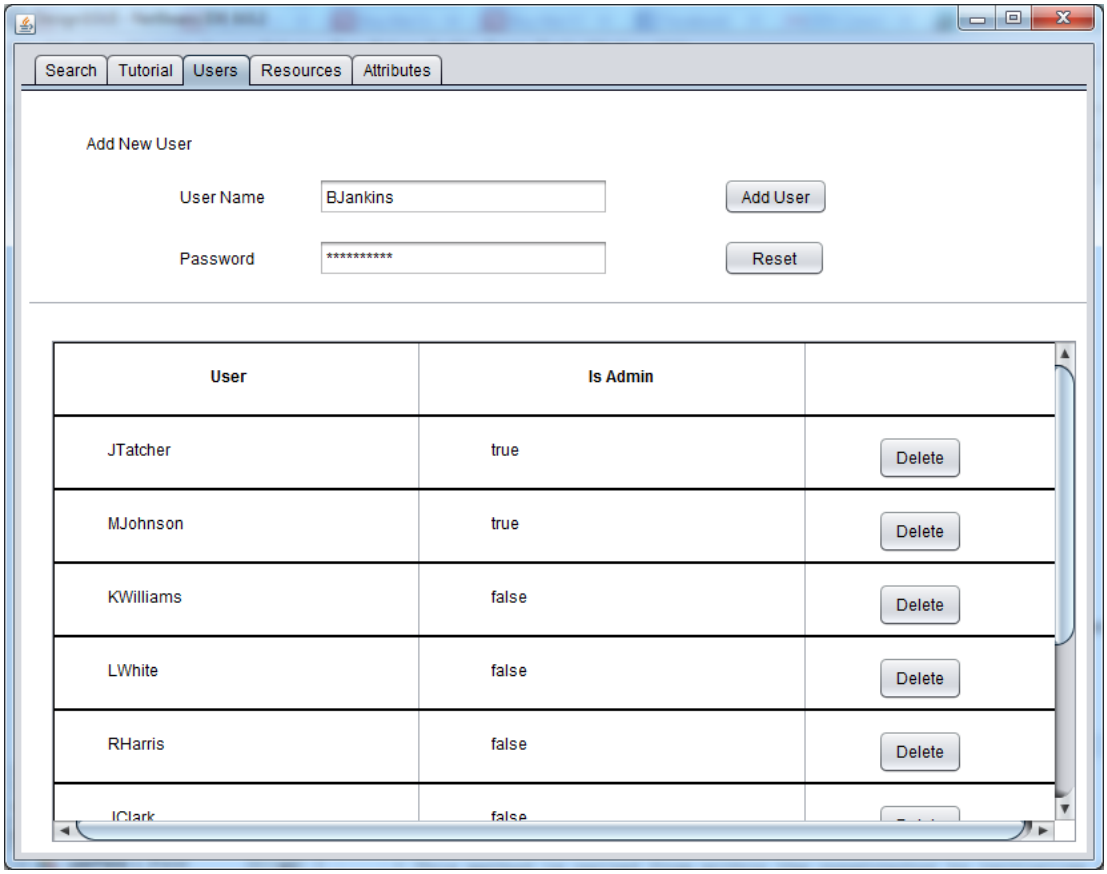


Figure 3: Administration panel for adding and removing users

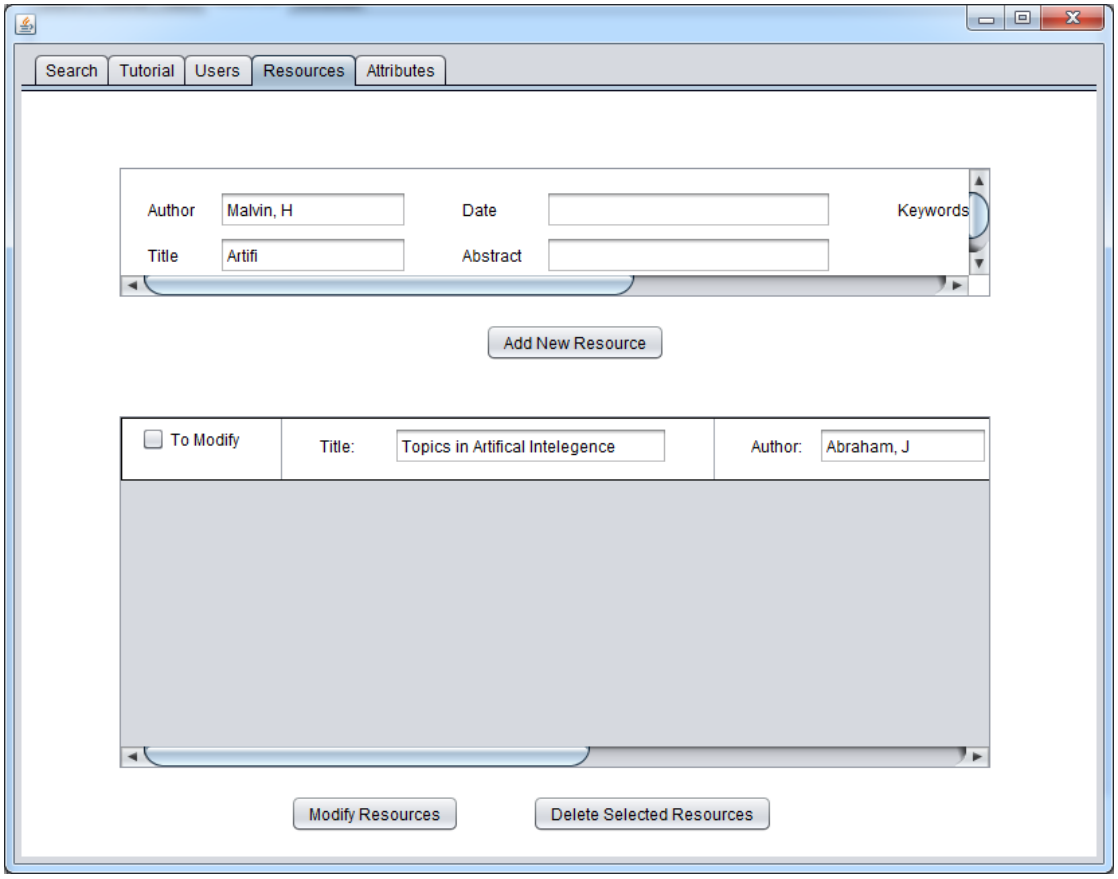


Figure 4: Administration panel for adding, modifying, and removing resources

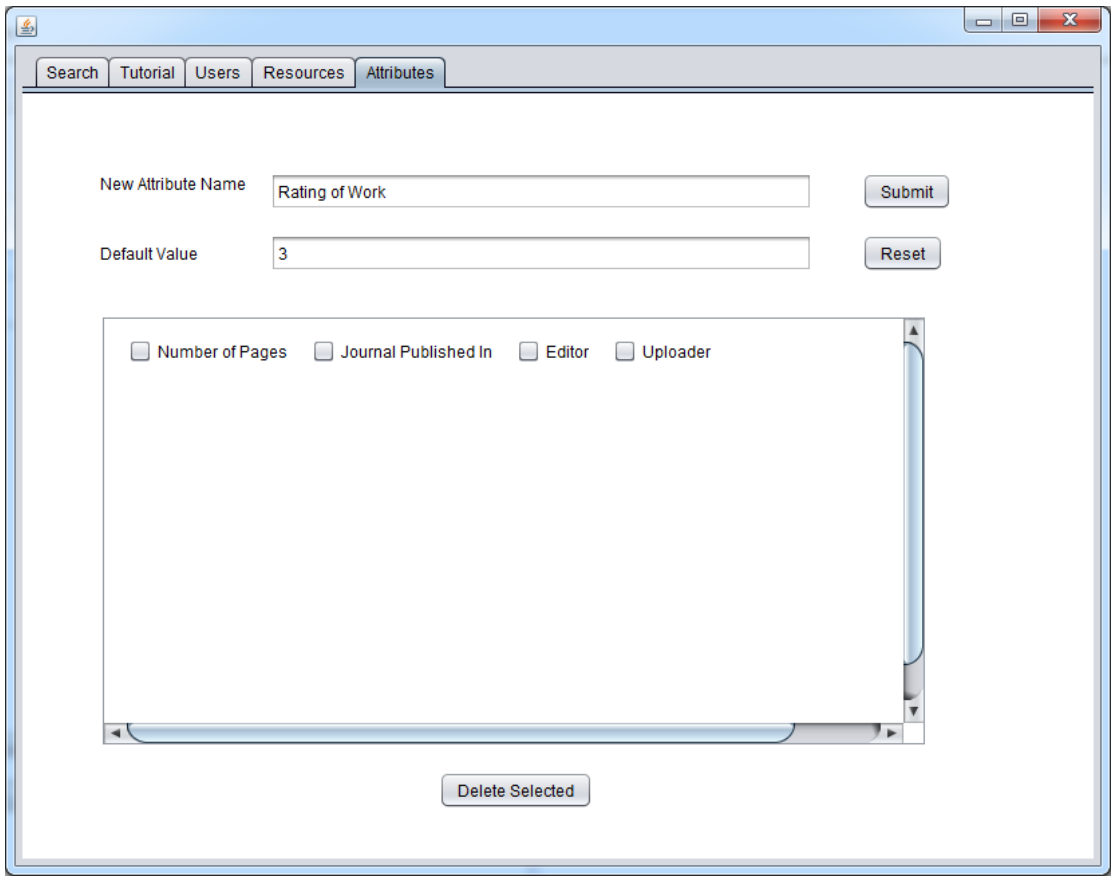


Figure 5: Administration panel for adding, and removing attributes

2.6.2 Hardware Interfaces

Not applicable.

2.6.3 Software Interfaces

Not applicable.

3. Software Project Management Plan

3.1 Software Process Model

Our client/server software will be developed using the Unified Process life cycle model. Each increment of development will be followed by iterations through each workflow where the specific goals, in terms of requirements, design, implementation etc., will be considered.

Using the Unified Process model in development will allow our objectives to be clarified before committing to specific stages of the process. This way, we can ensure development is consistent with the details outlined within our specification and requirements documents.

3.2 Roles and Responsibilities

The software is divided in three primary subsystems: server side, client side and graphical user interface. Each developer will be primarily responsible for one subsystem. However, all developers will collaborate and provide input throughout the development process for each subsystem.

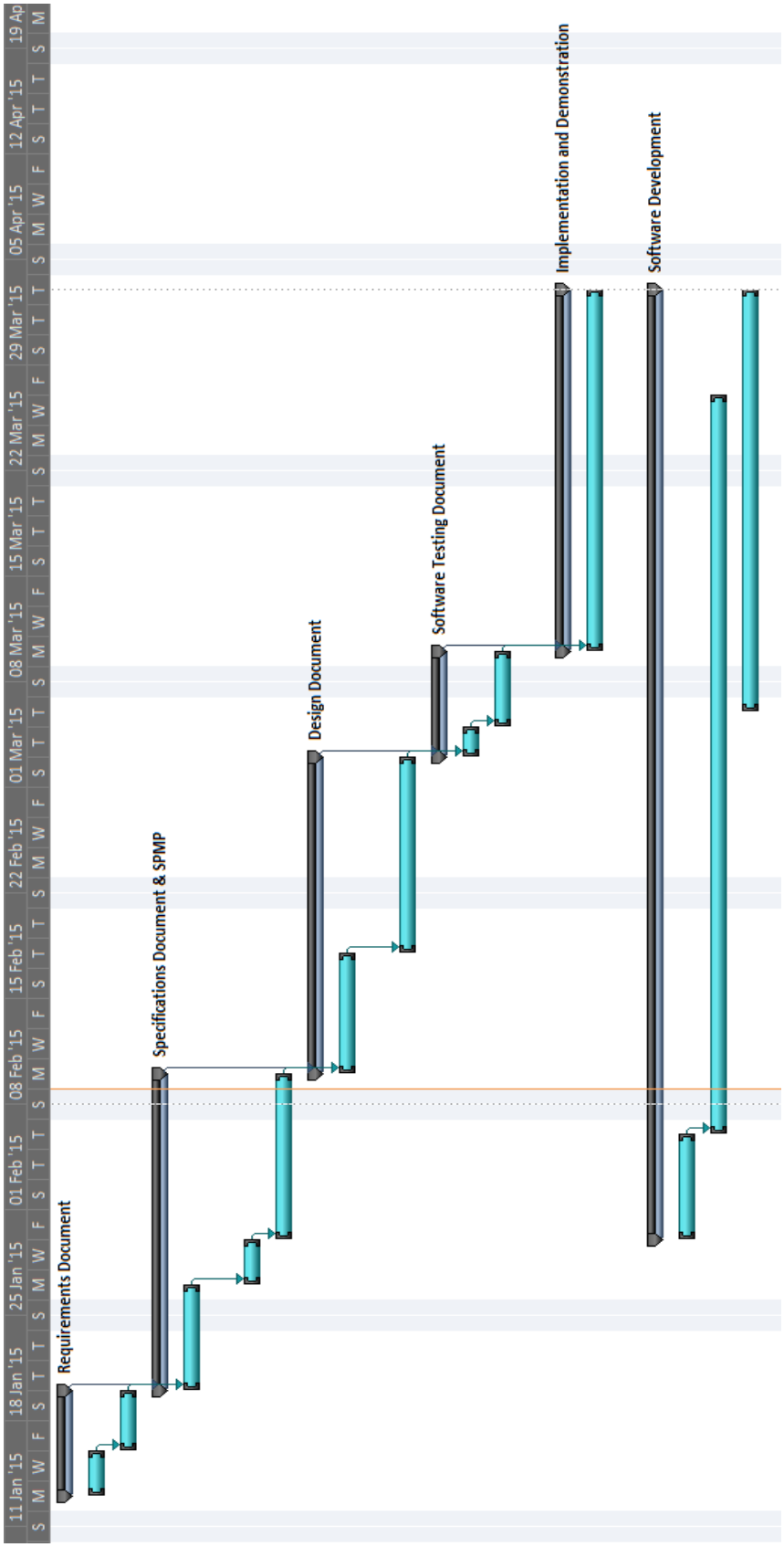
Subsystem	**Owner**
Server Side	John Simko
Client Side, Current Project Manager	Shahood Mirza
Graphical User Interface	Brandon Stanley

3.3 Project Plan

3.3.1 Deliverables and Milestones

Task Name	Duration	Start	Finish
Requirements Document	**5 days**	**Tue 13/01/15**	**Mon 19/01/15**
Assessment of client needs	3 days	Tue 13/01/15	Thu 15/01/15
Document write up	2 days	Fri 16/01/15	Mon 19/01/15
Specifications Document & SPMP	**15 days**	**Tue 20/01/15**	**Mon 09/02/15**
Outlining project objectives and goals	5 days	Tue 20/01/15	Mon 26/01/15
Division of responsibilities	3 days	Tue 27/01/15	Thu 29/01/15
Specification write up	7 days	Fri 30/01/15	Mon 09/02/15
Design Document	**15 days**	**Tue 10/02/15**	**Mon 02/03/15**
Meeting with client regarding final design	6 days	Tue 10/02/15	Tue 17/02/15
Document write up	9 days	Wed 18/02/15	Mon 02/03/15
Software Testing Document	**5 days**	**Tue 03/03/15**	**Mon 09/03/15**
Outline testing procedures	2 days	Tue 03/03/15	Wed 04/03/15
Unit and regression testing procedures	3 days	Thu 05/03/15	Mon 09/03/15
Implementation and Demonstration	**18 days**	**Tue 10/03/15**	**Thu 02/04/15**
Testing, deployments and final wrap-up	18 days	Tue 10/03/15	Thu 02/04/15
Software Development	**45 days**	**Fri 30/01/15**	**Thu 02/04/15**
Initial design discussions	5 days	Fri 30/01/15	Thu 05/02/15
Coding & development	35 days	Fri 06/02/15	Thu 26/03/15
Integration/regression testing & implementation	20 days	Fri 06/03/15	Thu 02/04/15

3.3.2 Gantt Chart



3.4 Intermediate COCOMO

3.4.1 Estimate Justification:

It is estimated that the project will take approximately 3k lines of code to complete. Approximately 2.5k for the GUI, and 0.5k for the backend. This number is derived from an automatically generated GUI using Visual Studios' MFC GUI creation wizard, to get a rough idea what to expect. The GUI generated this way made approximately 2.5k lines of code.

We have determined that we perform the calculation under as a "Semi-detached" Software project, giving:

- a = 3
- b = 1.12

The COCOMO EAFs are selected as follows:

- 1.15: Required Software Reliability is High
- 1.00: Size of application database is nominal
- 1.15: Complexity of Software product is relatively high due to lack of familiarity
- 1.00: Runtime constraints nominal
- 1.00: Memory constraints nominal
- n/a : Volatility of the virtual machine environment
- 1.00: Required turnabout time
- 1.19: Analyst Capability is low due to lack of familiarity
- 1.13: Application experience is low
- 1.17: Software Engineer capability is low due to lack of experience
- n/a : Virtual machine experience
- 1.07: Programming language experience is low *due to no knowledge* of C++ **GUI** design
- 0.91: Application of Software engineering methods
- 1.10: Use of software tools is low due to restrictions of project
- 1.23: Very low required development schedule

3.4.2 Calculation:

The formula is as follows:

$$E=a(KLoC)^{(b)}.EAF$$

Giving a calculation of:

$$E = 3*3^{1.12} *1.15*1.15*1.19*1.13*1.17*1.07*0.91*1.10*1.23$$

Which results in an Effort time of:

$$E = 28.15 \text{ Man-months.}$$

This means each members has to do:

$$E/3 = 9.38 \text{ Man-months}$$

Given there is only 2 months a month and a half to code this project, this means each member has to do the work of:

$$9.38/1.5 = 6.25 \text{ Men}$$

3.5 Risks and Risk Mitigation

Risk: Scope of project changes.
Mitigation: ● A willingness to change entire parts of the product as required. ● Maintain all documentation in editable forms, allowing for rapid changes as circumstances may require. ● Modular architecture design to allow rapid changes without worry of regression faults.

Risk: GUI is not completed on time.
Mitigation: ● Other members of the project can help in development. ● Other courses could be ignored in favor of completion of this project. ● Worst case scenario, components of GUI could be autogenerated.

Risk: Client/Server is not completed on time.
Mitigation: ● Other members of the project can help in development. ● Other courses could be ignored in favor of completion of this project.

Risk: Functionality missed
Mitigation: ● Perform peer reviews of current and past documentation.

Risk: Admin/user credentials are lost or stolen
Mitigation: ● Use secure connections (SSL) to prevent attacks and data sniffing ● Users have the option to change passwords